Queueing Networks and Markov Chains

Modeling and Performance Evaluation with Computer Science Applications

Gunter Bolch
Stefan Greiner
Hermann de Meer
Kishor S. Trivedi

Queueing Networks
and
Markov Chains

Queueing Networks and Markov Chains

Modeling and Performance Evaluation with Computer Science Applications

Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi

A Wiley-Interscience Publication
JOHN WILEY & SONS, INC.
New York / Chichester / Weinheim / Brisbane / Singapore / Toronto

ISBN 0-471-20058-1.

This title is also available in print as ISBN 0-471-19366-6

For more information about Wiley products, visit our web site at www.Wiley.com.

Library of Congress Cataloging in Publication Data:

Queueing networks and Markov chains: modeling and performance evaluation with computer science applications / Gunter Bolch . . . [et al.].
p. cm.
"A Wiley-Interscience publication."
Includes bibliographical references and index.
ISBN 0-471-19366-6
1. Electronic digital computers — Evaluation. 2. Markov processes.
3. Queuing theory. I. Bolch, Gunter.
QA76.9.E94Q48 1998
004.2′4′01519233 — dc21 97-11959
CIP

Printed in the United States of America.

10 9 8 7 6 5 4 3 2

Contents

Preface

Queueing networks and Markov chains are commonly used for the performance and reliability evaluation of computer, communication, and manufacturing systems. Although there are quite a few books on the individual topics of queueing networks and Markov chains, we have found none that covers both of these topics. The purpose of this book, therefore, is to offer a detailed treatment of queueing systems, queueing networks, and continuous and discrete-time Markov chains.

In addition to introducing the basics of these subjects, we have endeavored to:

- Provide some in-depth numerical solution algorithms.
- Incorporate a rich set of examples that demonstrate the application of the different paradigms and corresponding algorithms.
- Discuss stochastic Petri nets as a high-level description language, thereby facilitating automatic generation and solution of voluminous Markov chains.
- Treat in some detail approximation methods that will handle large models.
- Describe and apply four software packages throughout the text.
- Provide problems as exercises.

This book easily lends itself to a course on performance evaluation in the computer science and computer engineering curricula. It can also be used for a course on stochastic models in mathematics, operations research and industrial engineering departments. Because it incorporates a rich and comprehensive set of numerical solution methods comparatively presented, the text may also

well serve practitioners in various fields of applications as a reference book for algorithms.

With sincere appreciation to our friends, colleagues, and students who so ably and patiently supported our manuscript project, we wish to publicly acknowledge:

- Jörg Barner and Stephan Kösters for their painstaking work in keying the text and in laying out the figures and plots.
- Peter Bazan, who assisted both with the programming of many examples and comprehensive proofreading.
- Hana Ševčíková, who lent a hand in solving many of the examples and contributed with proofreading.
- János Sztrik, for his comprehensive proofreading.
- Doris Ehrenreich, who wrote the first version of the section on communication systems.
- Markus Decker, who prepared the first draft of the mixed queueing networks section.
- Those who read parts of the manuscript and provided many useful comments, including: Khalid Begain, Oliver Düsterhöft, Ricardo Fricks, Swapna Gokhale, Thomas Hahn, Christophe Hirel, Graham Horton, Steve Hunter, Demetres Kouvatsos, Yue Ma, Raymond Marie, Varsha Mainkar, Victor Nicola, Cheul Woo Ro, Helena Szczerbicka, Lorrie Tomek, Bernd Wolfinger, Katinka Wolter, Martin Zaddach, and Henry Zang.

Gunter Bolch and Stefan Greiner are grateful to Fridolin Hofmann, and Hermann de Meer is grateful to Bernd Wolfinger, for their support in providing the necessary freedom from distracting obligations.

Thanks are also due to Teubner B.G. Publishing house for allowing us to borrow sections from the book entitled **Leistungsbewertung von Rechensystemen** (originally in German) by one of the coauthors, Gunter Bolch. In the present book, these sections are integrated in Chapters 1 and 7 through 10.

We also thank Andrew Smith, Lisa Van Horn, and Mary Lynn of John Wiley & Sons for their patience and encouragement.

The financial support from the SFB (Collaborative Research Centre) 182: "Multiprocessor and Network Configurations" of the DFG (Deutsche Forschungsgemeinschaft) is acknowledged.

Finally, a web page has been set up for further information regarding the book. The URL is `http://www4.cs.fau.de/QNMC/`

GUNTER BOLCH, STEFAN GREINER, HERMANN DE MEER, KISHOR S. TRIVEDI

Queueing Networks
and
Markov Chains

1

Introduction

1.1 MOTIVATION

Information processing system designers need methods for the quantification of system design factors such as performance and reliability. Modern computer communication and production line systems process complex workloads with random service demands. Probabilistic and statistical methods are commonly employed for the purpose of performance and reliability evaluation. The purpose of this book is to explore major probabilistic modeling techniques for the performance analysis of information processing systems. Statistical methods are also of great importance but we refer the reader to other sources [Jain91 Triv82] for this topic. Although we concentrate on performance analysis we occasionally consider reliability availability and combined performance and reliability analysis. Performance measures that are commonly of interest include throughput resource utilization loss probability and delay (or response time).

The most direct method for performance evaluation is based on actual measurement of the system under study. However during the design phase the system is not available for such experiments and yet performance of a given design needs to be predicted to verify that it meets design requirements and to carry out necessary trade-offs. Hence abstract models are necessary for performance prediction of designs. The most popular models are based on discrete-event simulation (DES). DES can be applied to almost all problems of interest and system details to the desired degree can be captured in such simulation models. Furthermore many software packages are available that facilitate the construction and execution of DES models.

The principal drawback of DES models, however, is the time taken to run such models for large, realistic systems particularly when results with high accuracy (i.e., narrow confidence intervals) are desired. A cost-effective alternative to DES models, analytic models can provide relatively quick answers to "what if" questions and can provide more insight into the system being studied. However, analytic models are often plagued by unrealistic assumptions that need to be made in order to make them tractable. Recent advances in stochastic models and numerical solution techniques, availability of software packages, and easy access to workstations with large computational capabilities have extended the capabilities of analytic models to more complex systems.

Analytical models can be broadly classified into state space models and non-state space models. Most commonly used state space models are Markov chains. First introduced by A. A. Markov in 1907, Markov chains have been in use in performance analysis since around 1950. In the past decade, considerable advances have been made in the numerical solution techniques, methods of automated state space generation, and the availability of software packages. These advances have resulted in extensive use of Markov chains in performance and reliability analysis. A Markov chain consists of a set of states and a set of labeled transitions between the states. A state of the Markov chain can model various conditions of interest in the system being studied. These could be the number of jobs of various types waiting to use each resource, the number of resources of each type that have failed, the number of concurrent tasks of a given job being executed, and so on. After a sojourn in a state, the Markov chain will make a transition to another state. Such transitions are labeled either with probabilities of transition (in case of discrete-time Markov chains) or rates of transition (in case of continuous-time Markov chains).

Long run (steady-state) dynamics of Markov chains can be studied using a system of linear equations with one equation for each state. Transient (or time dependent) behavior of a continuous-time Markov chain gives rise to a system of first-order, linear, ordinary differential equations. Solution of these equations results in state probabilities of the Markov chain from which desired performance measures can be easily obtained. The number of states in a Markov chain of a complex system can become very large and, hence, automated generation and efficient numerical solution methods for underlying equations are desired. A number of concise notations (based on qucucing networks and stochastic Petri nets) have evolved, and software packages that automatically generate the underlying state space of the Markov chain are now available. These packages also carry out efficient solution of steady-state and transient behavior of Markov chains. In spite of these advances, there is a continuing need to be able to deal with larger Markov chains and much research is being devoted to this topic.

If the Markov chain has nice structure, it is often possible to avoid the generation and solution of the underlying (large) state space. For a class of queueing networks, known as product-form queueing networks (PFQN),

it is possible to derive steady-state performance measures without resorting to the underlying state space. Such models are therefore called non-state space models. Other examples of non-state space models are directed acyclic task precedence graphs [SaTr87] and fault-trees [STP96]. Other examples of methods exploiting Markov chains with "nice" structure are matrix-geometric methods [Neut81]. We do not discuss these model types in this book due to space limitations.

Relatively large PFQN can be solved by means of a small number of simpler equations. However, practical queueing networks can often get so large that approximate methods are needed to solve such PFQN. Furthermore, many practical queueing networks (so-called non-product-form queueing networks, NPFQN) do not satisfy restrictions implied by product-form. In such cases, it is often possible to obtain accurate approximations using variations of algorithms used for PFQNs. Other approximation techniques using hierarchical and fixed-point iterative methods are also used.

The flowchart shown in Fig. 1.1 gives the organization of this book. The rest of this chapter, Section 1.2, covers the basics of probability and statistics. In Chapter 2, Markov chains basics are presented together with generation methods for them. Exact steady-state solution techniques for Markov chains are given in Chapter 3 and their aggregation/disaggregation counterpart in Chapter 4. These aggregation/disaggregation solution techniques are useful for practical Markov chain models with very large state spaces. Transient solution techniques for Markov chains are introduced in Chapter 5.

Chapter 6 deals with the description and computation of performance measures for single-station queueing systems in steady state. A general description of queueing networks is given in Chapter 7. Exact solution methods for PFQN are described in detail in Chapter 8 while approximate solution techniques for PFQN are described in Chapter 9. Solution algorithms for different types of NPFQN (such as networks with priorities, non-exponential service times, blocking, or parallel processing) are presented in Chapter 10.

The solution algorithms introduced in this book can also be used for optimization problems as described in Chapter 11. For the practical use of modeling techniques described in this book, software packages (tools) are needed. Chapter 12 is devoted to the introduction of a queueing network tool, a stochastic Petri net tool, a tool based on Markov chains and a toolkit with many model types, and the facility for hierarchical modeling is also introduced. Each tool is described in some detail together with a simple example. Throughout the book we have provided many example applications of different algorithms introduced in the book. Finally, Chapter 13 is devoted to several large real-life applications of the modeling techniques presented in the book.

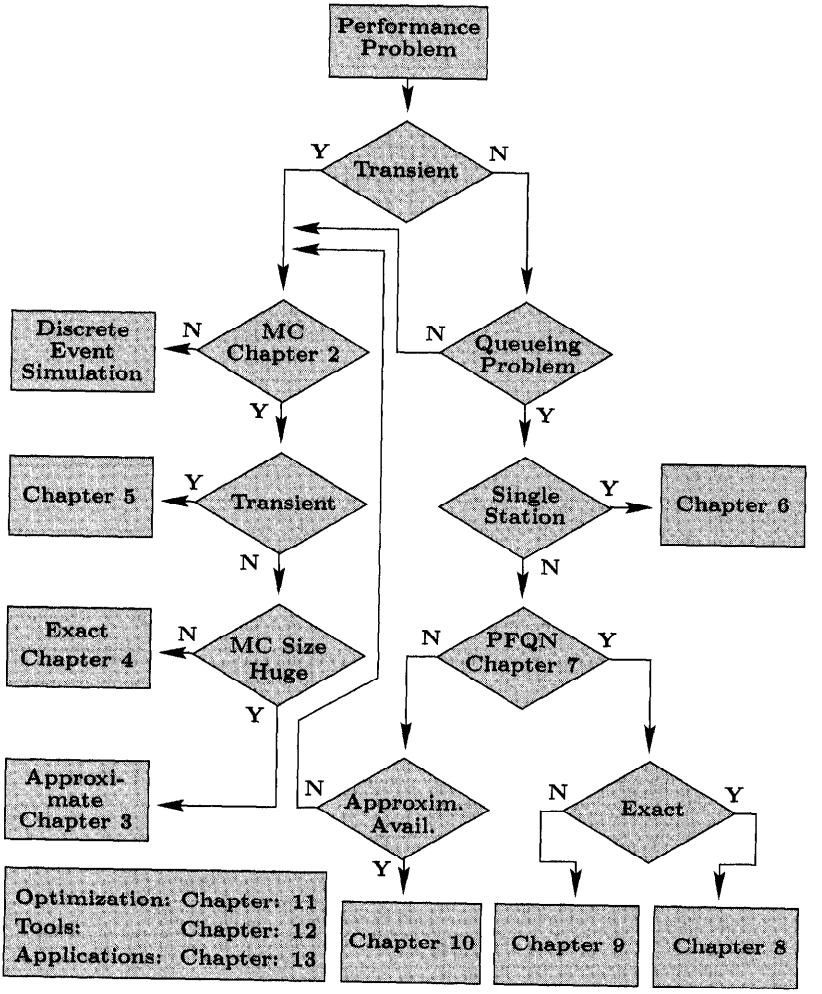


Fig. 1.1 Flowchart describing how to find the appropriate chapter for a given performance problem.

1.2 BASICS OF PROBABILITY AND STATISTICS

We begin by giving a brief overview of the more important definitions and results of probability theory. The reader can find additional details in books such as [Alle90, Fell68, Koba78, Triv82]. We assume that the reader is familiar with the basic properties and notations of probability theory.

1.2.1 Random Variables

A random variable is a function that reflects the result of a random experiment. For example, the result of the experiment "toss a single die" can be described by a random variable that can assume the values one through six. The number of requests that arrive at an airline reservation system in one hour or the number of jobs that arrive at a computer system are also examples of a random variable. So is the time interval between the arrivals of two consecutive jobs at a computer system, or the throughput in such a system. The latter two examples can assume continuous values, whereas the first two only assume discrete values. Therefore, we have to distinguish between continuous and discrete random variables.

1.2.1.1 Discrete Random Variables A random variable that can only assume discrete values is called a *discrete random variable*, where the discrete values are often non-negative integers. The random variable is described by the possible values that it can assume and by the probabilities for each of these values. The set of these probabilities is called the *probability mass function* (pmf) of this random variable. Thus, if the possible values of a random variable X are the non-negative integers, then the pmf is given by the probabilities:

$$p_k = P(X = k), \quad \text{for} \quad k = 0, 1, 2 \ldots, \tag{1.1}$$

the probability that the random variable X assumes the value k.

The following is required:

$$P(X = k) \geq 0,$$
$$\sum_{\text{all } k} P(X = k) = 1.$$

For example, the following pmf results from the experiment "toss a single die":

$$P(X = k) = \frac{1}{6}, \quad \text{for } k = 1, 2, \ldots, 6.$$

The following are other examples of discrete random variables:

- Bernoulli random variable: Consider a random experiment that has two possible outcomes, such as tossing a coin ($k = 0, 1$). The pmf of the

random

random variable X is given by:

$$P(X=0)=1-p \quad \text{and} \quad P(X=1)=p, \quad \text{with } 0<p<1. \tag{1.2}$$

- Binomial random variable: The experiment with two possible outcomes is carried out n times where successive trials are independent. The random variable X is now the number of times the outcome 1 occurs. The pmf of X is given by:

$$P(X=k)=\binom{n}{k}p^k(1-p)^{n-k}, \quad k=0,1,\ldots,n. \tag{1.3}$$

- Geometric random variable: The experiment with two possible outcomes is carried out several times, where the random variable X now represents the number of trials it takes for the outcome 1 to occur (the current trial included). The pmf of X is given by:

$$P(X=k)=p(1-p)^{k-1}, \qquad k=1,2,\ldots. \tag{1.4}$$

- Poisson random variable: The probability of having k events (Poisson pmf) is given by:

$$P(X=k)=\frac{(\alpha)^k}{k!}\cdot e^{-\alpha}, \qquad k=0,1,2,\ldots;\alpha>0. \tag{1.5}$$

The Poisson and geometric random variables are very important to our topic; we will encounter them very often. Several important parameters can be derived from a pmf of a discrete random variable:

- Mean value or expected value:

$$\overline{X}=E[X]=\sum_{\text{all } k} k\cdot P(X=k). \tag{1.6}$$

The function of a random variable is another random variable with the expected value of:

$$E[f(X)]=\sum_{\text{all } k} f(k)\cdot P(X=k). \tag{1.7}$$

- nth moments:

$$\overline{X^n}=E[X^n]=\sum_{\text{all } k} k^n\cdot P(X=k), \tag{1.8}$$

that is, the nth moment is the expected value of the nth power of X. The first moment of X is simply the mean of X.

- nth central moment:

$$\overline{(X-\overline{X})^n} = E[(X-E[X])^n] = \sum_{\text{all } k}(k-\overline{X})^n \cdot P(X=k). \tag{1.9}$$

The nth central moment is the expected value of the nth power of the difference between X and its mean. The first central moment is equal to zero.

- The second central moment is called the variance of X:

$$\sigma_X^2 = \text{var}(X) = \overline{(X-\overline{X})^2} = \overline{X^2} - \overline{X}^2, \tag{1.10}$$

where σ_X is called the standard deviation.

- The coefficient of variation is the normalized standard deviation:

$$c_X = \frac{\sigma_X}{\overline{X}}. \tag{1.11}$$

Information on the average deviation of a random variable from its expected value is provided by c_X, σ_X, and var(X). If $c_X = \sigma_X = \text{var}(X) = 0$, then the random variable assumes a fixed value with probability one.

Table 1.1 Properties of several discrete random variables

Random Variables	Parameter	$\overline{X}$	var(X)	c_X^2
Bernoulli	p	p	$p(1-p)$	$\frac{1-p}{p}$
Binomial	n,p	np	$np(1-p)$	$\frac{1-p}{np}$
Geometric	p	$\frac{1}{p}$	$\frac{1-p}{p^2}$	$1-p$
Poisson	α	α	α	$\frac{1}{\alpha}$

Table 1.1 gives a list of random variables, their mean values, variances, and the squared coefficients of variation for some important discrete random variables.

1.2.1.2 Continuous Random Variables A random variable X that can assume all values in the interval $[a,b]$, where $-\infty \le a < b \le +\infty$, is called a *continuous random variable.* It is described by its *distribution function* (also called CDF or cumulative distribution function):

$$F_X(x) = P(X \le x), \tag{1.12}$$

which specifies the probability that the random variable X takes values less than or equal to x, for every x.

From Eq. (1.12) we get for $x < y$:

$$F_X(x) \leq F_X(y),$$
$$P(x < X \leq y) = F_X(y) - F_X(x).$$

The *probability density function* (pdf) $f_X(x)$ can be used instead of the distribution function, provided the latter is differentiable:

$$f_X(x) = \frac{dF_X(x)}{dx}. \tag{1.13}$$

Some properties of the pdf are:

$$f_X(x) \geq 0 \quad \text{for all } x,$$
$$\int_{-\infty}^{\infty} f_X(x)dx = 1,$$
$$P(x_1 \leq X \leq x_2) = \int_{x_1}^{x_2} f_X(x)dx,$$
$$P(X = x) = \int_{x}^{x} f_X(x)dx = 0,$$
$$P(X > x_3) = \int_{x_3}^{\infty} f_X(x)dx.$$

Note that for so-called *defective* random variables [STP96] we have:

$$\int_{-\infty}^{\infty} f_X(x)dx < 1.$$

The density function of a continuous random variable is analogous to the pmf of a discrete random variable. The formulae for the mean value and moments of continuous random variables can be derived from the formulae for discrete random variables by substituting the pmf by the pdf and the summation by an integral:

- Mean value or expected value:

$$\overline{X} = E[X] = \int_{-\infty}^{\infty} x \cdot f_X(x)dx \tag{1.14}$$

and:

$$E[g(X)] = \int_{-\infty}^{\infty} g(x) \cdot f_X(x) dx. \tag{1.15}$$

- nth moment:

$$\overline{X^n} = E[X^n] = \int_{-\infty}^{\infty} x^n \cdot f_X(x) dx. \tag{1.16}$$

- nth central moment:

$$\overline{(X - \overline{X})^n} = E[(X - E[X])^n] = \int_{-\infty}^{\infty} (x - \overline{X})^n f_X(x) dx. \tag{1.17}$$

- Variance:

$$\sigma_X^2 = \text{var}(X) = \overline{(X - \overline{X})^2} = \overline{X^2} - \overline{X}^2, \tag{1.18}$$

with σ_X as the standard deviation.

- Coefficient of variation:

$$c_X = \frac{\sigma_X}{\overline{X}}. \tag{1.19}$$

A very well known and important continuous distribution function is the *normal distribution.* The CDF of a normally distributed random variable X is given by:

$$F_X(x) = \frac{1}{\sqrt{2\pi\sigma_X^2}} \int_{-\infty}^{x} \exp\left(-\frac{(u - \overline{X})^2}{2\sigma_X^2}\right) du, \tag{1.20}$$

and the pdf by:

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma_X^2}} \exp\left(-\frac{(x - \overline{X})^2}{2\sigma_X^2}\right).$$

The standard normal distribution is defined by setting $\overline{X} = 0$ and $\sigma_X = 1$:

CDF:
$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \cdot \int_{-\infty}^{x} \exp\left(-\frac{u^2}{2}\right) du, \tag{1.21}$$

pdf:
$$\phi(x) = \frac{1}{\sqrt{2\pi}} \cdot \exp\left(-\frac{x^2}{2}\right).$$

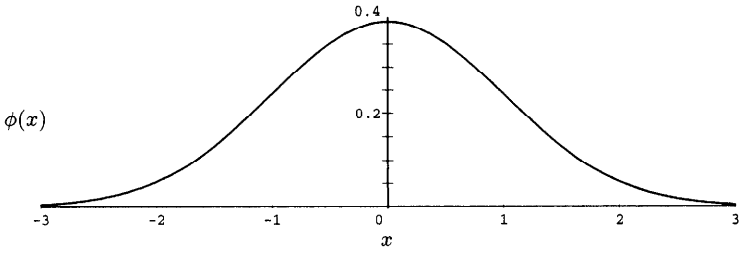


Fig. 1.2 pdf of the standard normal random variable.

A plot of the preceding pdf is shown in Fig. 1.2.
For an arbitrary normal distribution we have:

$$F_X(x) = \Phi\left(\frac{x-\overline{X}}{\sigma_X}\right) \quad \text{and} \quad f_X(x) = \phi\left(\frac{x-\overline{X}}{\sigma_X}\right),$$

respectively.

Other important continuous random variables are described as follows.

(a) Exponential Distribution

The exponential distribution is the most important and also the easiest to use distribution in queueing theory. Interarrival times and service times can often be represented exactly or approximately using the exponential distribution. The CDF of an exponentially distributed random variable X is given by Eq. (1.22):

$$F_X(x) = \begin{cases} 1 - \exp\left(-\dfrac{x}{\overline{X}}\right), & 0 \le x < \infty, \\ 0, & \text{otherwise.} \end{cases}$$

$$\text{with } \overline{X} = \begin{cases} \dfrac{1}{\lambda}, & \text{if } X \text{ represents interarrival times,} \\ \dfrac{1}{\mu}, & \text{if } X \text{ represents service times.} \end{cases} \tag{1.22}$$

Here λ or μ denote the parameter of the random variable. In addition, for an exponentially distributed random variable with parameter λ the following relations hold:

pdf:	$f_X(x) = \lambda e^{-\lambda x}$,
mean:	$\overline{X} = \dfrac{1}{\lambda}$,
variance:	$\mathrm{var}(X) = \dfrac{1}{\lambda^2}$,
coefficient of variation:	$c_X = 1$.

Thus the exponential distribution is completely determined by its mean value.

The importance of the exponential distribution is based on the fact that it is the only continuous distribution that possesses the memoryless property:

$$P(X \leq u + t \mid X > u) = 1 - \exp\left(-\frac{t}{\overline{X}}\right) = P(X \leq t). \tag{1.23}$$

As an example for an application of Eq. (1.23), consider a bus stop with the following schedule: Buses arrive with exponentially distributed interarrival times and identical mean $\overline{X}$. Now if you have already been waiting in vain for u units of time for the bus to come, the probability of a bus arrival within the next t units of time is the same as if you had just shown up at the bus stop, that is, you can forget about the past or about the time already spent waiting.

Another important property of the exponential distribution is its relation to the discrete Poisson random variable. If the interarrival times are exponentially distributed and successive interarrival times are independent with identical mean $\overline{X}$, then the random variable that represents the number of buses that arrive in a fixed interval of time $[0, t)$ has a Poisson distribution with parameter $\alpha = t/\overline{X}$.

Two additional properties of the exponential distribution can be derived from the Poisson property:

1. If we merge n Poisson processes with distributions for the interarrival times $1 - e^{-\lambda_i t}$, $1 \leq i \leq n$, into one single process, then the result is a Poisson process for which the interarrival times have the distribution $1 - e^{-\lambda t}$ with $\lambda = \sum_{i=1}^{n} \lambda_i$ (see Fig. 1.3).

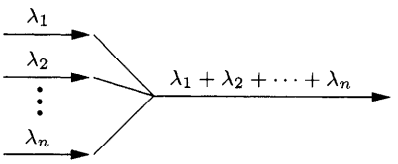


Fig. 1.3 Merging of Poisson processes.

2. If a Poisson process with interarrival time distribution $1 - e^{-\lambda t}$ is split into n processes so that the probability that the arriving job is assigned

to the ith process is q_i, $1 \leq i \leq n$, then the ith subprocess has an interarrival time distribution of $1 - e^{-q_i \lambda t}$, i.e., n Poisson processes have been created, as shown in Fig 1.4.

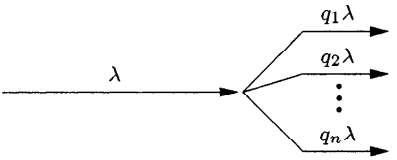


Fig. 1.4 Splitting of a Poisson process.

The exponential distribution has many useful properties with analytic tractability, but is not always a good approximation to the observed distribution. Experiments have shown deviations. For example, the coefficient of variation of the service time of a processor is often greater than one, and for a peripheral device it is usually less than one. This observed behavior leads directly to the need to consider the following other distributions:

(b) Hyperexponential Distribution, H_k

This distribution can be used to approximate empirical distributions with a coefficient of variation larger than one. Here k is the number of phases.

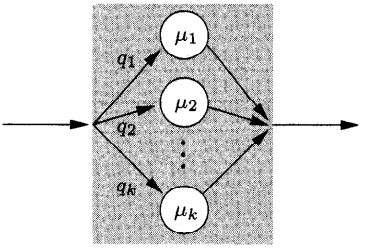


Fig. 1.5 A random variable with H_k distribution.

Figure 1.5 shows a model with hyperexponentially distributed time. The model is obtained by arranging k phases with exponentially distributed times and rates $\mu_1, \mu_2, \ldots, \mu_k$ in parallel. The probability that the time span is given by the jth phase is q_j, where $\sum_{j=1}^{k} q_j = 1$. However, only one phase can be occupied at any time. The resulting CDF is given by:

$$F_X(x) = \sum_{j=1}^{k} q_j (1 - e^{-\mu_j x}), \qquad x \geq 0. \tag{1.24}$$

pdf: $$f_X(x) = \sum_{j=1}^{k} q_j \mu_j e^{-\mu_j x}, \quad x > 0,$$

mean: $$\overline{X} = \sum_{j=1}^{k} \frac{q_j}{\mu_j} = \frac{1}{\mu}, \quad x > 0,$$

variance: $$\text{var}(X) = 2 \sum_{j=1}^{k} \frac{q_j}{\mu_j^2} - \frac{1}{\mu^2},$$

coefficient of variation: $$c_X = \sqrt{2\mu^2 \sum_{j=1}^{k} \frac{q_j}{\mu_j^2} - 1} \geq 1.$$

For example, the parameters μ_1, μ_2 of an H_2 distribution can be estimated to approximate an unknown distribution with given mean $\overline{X}$ and sample coefficient of variation c_X as follows:

$$\mu_1 = \frac{1}{\overline{X}} \left[1 - \sqrt{\frac{q_2}{q_1} \frac{c_X^2 - 1}{2}} \right]^{-1}, \tag{1.25}$$

$$\mu_2 = \frac{1}{\overline{X}} \left[1 + \sqrt{\frac{q_1}{q_2} \frac{c_X^2 - 1}{2}} \right]^{-1}. \tag{1.26}$$

The parameters q_1 and q_2 can be assigned any values that satisfy the restrictions $q_1, q_2 \geq 0, q_1 + q_2 = 1$ and $\mu_1, \mu_2 > 0$.

(c) Erlang-k Distribution, E_k

Empirical distributions with a coefficient of variation less than one can be approximated using the Erlang-k distribution. Here k is the number of exponential phases in series. Figure 1.6 shows a model of a time duration that has an E_k distribution. It contains k identical phases connected in series, each with exponentially distributed time. The mean duration of each phase is $\overline{X}/k$, where $\overline{X}$ denotes the mean of the whole time span.

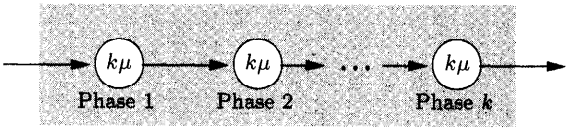


Fig. 1.6 A random variable with E_k distribution.

If the interarrival times of some arrival process like our bus stops are identical exponentially distributed, it follows that the time between the first arrival and the $(k+1)$th arrival is Erlang-k distributed.

The CDF is given by:

$$F_X(x) = 1 - e^{-k\mu x} \cdot \sum_{j=0}^{k-1} \frac{(k\mu x)^j}{j!}, \quad x \geq 0, k = 1, 2 \ldots . \tag{1.27}$$

pdf: $f_X(x) = \dfrac{k\mu(k\mu x)^{k-1}}{(k-1)!} e^{-k\mu x}$, $x > 0$, $k = 1, 2, \ldots,$

mean: $\overline{X} = \dfrac{1}{\mu},$

variance: $\text{var}(X) = \dfrac{1}{k\mu^2},$

coefficient of variation: $c_X = \dfrac{1}{\sqrt{k}} \leq 1.$

If the sample mean $\overline{X}$ and the sample coefficient of variation c_X are given, then the parameters k and μ of the corresponding Erlang distribution are estimated by:

$$k = \left\lceil \frac{1}{c_X^2} \right\rceil, \tag{1.28}$$

and:

$$\mu = \frac{1}{c_X^2 k \overline{X}}. \tag{1.29}$$

(d) Hypoexponential Distribution

The hypoexponential distribution arises when the individual phases of the E_k distribution are allowed to have assigned different rates. Thus, the Erlang distribution is a special case of the hypoexponential distribution.

For a hypoexponential distributed random variable X with two phases and the rates μ_1 and μ_2 ($\mu_1 \neq \mu_2$), we get the CDF as:

$$F_X(x) = 1 - \frac{\mu_2}{\mu_2 - \mu_1} e^{-\mu_1 x} + \frac{\mu_1}{\mu_2 - \mu_1} e^{-\mu_2 x}, \qquad x \geq 0. \tag{1.30}$$

pdf: $f_X(x) = \dfrac{\mu_1 \mu_2}{\mu_1 - \mu_2}(e^{-\mu_2 x} - e^{-\mu_1 x}), \quad x > 0,$

mean: $\overline{X} = \dfrac{1}{\mu_1} + \dfrac{1}{\mu_2},$

variance: $\text{var}(X) = \dfrac{1}{\mu_1^2} + \dfrac{1}{\mu_2^2},$

coefficient of variation: $c_X = \dfrac{\sqrt{\mu_1^2 + \mu_2^2}}{\mu_1 + \mu_2} < 1\ .$

The values of the parameters μ_1 and μ_2 of the hypoexponential CDF can be estimated given the sample mean $\overline{X}$ and sample coefficient of variation by:

$$\mu_1 = \frac{2}{\overline{X}}\left[1 + \sqrt{1 + 2\,(c_X^2 - 1)}\right]^{-1},$$

$$\mu_2 = \frac{2}{\overline{X}}\left[1 - \sqrt{1 + 2\,(c_X^2 - 1)}\right]^{-1},$$

with $0.5 \leq c_X^2 \leq 1$.

For a hypoexponential distribution with k phases and the phase rates μ_1, μ_2, ..., μ_k, we get [Bega93]:

pdf:
$$f_X(x) = \sum_{i=1}^{k} a_i \mu_i e_1^{-\lambda_i x}, \quad x > 0,$$
$$\text{with } a_i = \prod_{j=1, j\neq i}^{k} \frac{\mu_j}{\mu_j - \mu_i}, \quad 1 \leq i \leq k,$$

mean:
$$\overline{X} = \sum_{i=1}^{k} \frac{1}{\mu_i},$$

coefficient of variation:
$$c_X = \left(1 + 2 \frac{\sum_{i=1}^{k} \left(\mu_i \sum_{j=i+1}^{k} \mu_j \right)}{\sum_{i=1}^{k} \mu_i^2} \right)^{-\frac{1}{2}}.$$

(e) Gamma Distribution

Another generalization of the Erlang-k distribution for arbitrary coefficient of variation is the gamma distribution. The distribution function is given by:

$$F_X(x) = \int_0^x \frac{\alpha\mu \cdot (\alpha\mu u)^{\alpha-1}}{\Gamma(\alpha)} \cdot e^{-\alpha\mu u} du, \quad x \geq 0, \alpha > 0,$$
$$\text{with } \Gamma(\alpha) = \int_0^\infty u^{\alpha-1} \cdot e^{-u} du, \quad \alpha > 0. \tag{1.31}$$

If $\alpha = k$ is a positive integer, then $\Gamma(k) = (k-1)!$

Thus the Erlang-k distribution can be considered as a special case of the gamma distribution:

pdf: $$f_X(x) = \frac{\alpha\mu \cdot (\alpha\mu x)^{\alpha-1}}{\Gamma(\alpha)} e^{-\alpha\mu x}, \quad x > 0, \alpha > 0,$$

mean: $$\overline{X} = \frac{1}{\mu},$$

variance: $$\mathrm{var}(X) = \frac{1}{\alpha\mu^2},$$

coefficient of variation: $$c_X = \frac{1}{\sqrt{\alpha}}.$$

It can be seen that the parameters α and μ of the gamma distribution can easily be estimated from c_X and $\overline{X}$:

$$\mu = \frac{1}{\overline{X}} \quad \text{and} \quad \alpha = \frac{1}{c_X^2}.$$

(f) Generalized Erlang Distribution

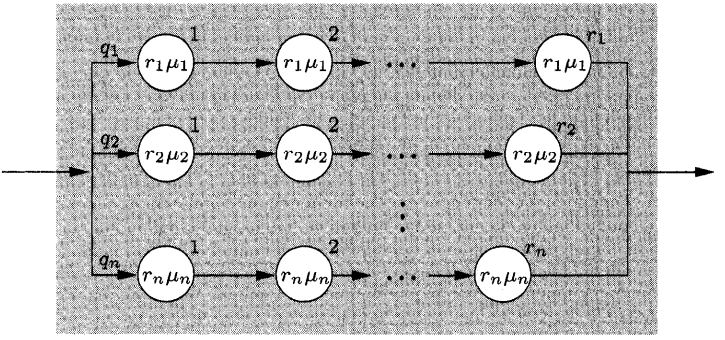


Fig. 1.7 A random variable with generalized Erlang distribution.

Rather complex distributions can be generated by combining hyperexponential and Erlang-k distributions; these are known as generalized Erlang distributions. An example is shown in Fig. 1.7 where n parallel levels are depicted and each level j contains a series of r_j phases connected, each with exponentially distributed time and rate $r_j\mu_j$. Each level j is selected with probability j. The pdf is given by:

$$f_X(x) = \sum_{j=1}^{n} q_j \cdot \frac{r_j\mu_j(r_j\mu_j x)^{r_j-1}}{(r_j-1)!} \cdot e^{-r_j\mu_j x}, \qquad x \geq 0. \tag{1.32}$$

Another type of generalized Erlang distribution can be obtained by lifting the restriction that all the phases within the same level have the same rate, or, alternatively, if there are non-zero exit probabilities assigned such that remaining phases can be skipped. These generalizations lead to very complicated equations that are not further described here.

(g) Cox Distribution, C_k (Branching Erlang Distribution)

In [Cox55] the principle of the combination of exponential distributions is generalized to such an extent that any distribution that possesses a rational Laplace transform can be represented by a sequence of exponential phases with possibly complex probabilities and complex rates. Figure 1.8 shows a model of a Cox distribution with k exponential phases.

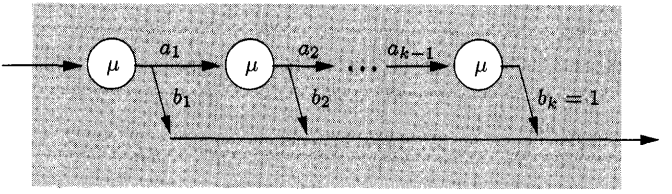


Fig. 1.8 A random variable with C_k distribution.

The model consists of k phases in series with exponentially distributed times and rates $\mu_1, \mu_2, \ldots, \mu_k$. After phase j, another phase $j+1$ follows with probability a_j and with probability $b_j = 1 - a_j$ the total time span is completed. As described in [SaCh81], there are two cases that must be distinguished when using the sample mean value $\overline{X}$ and the sample coefficient of variation c_X to estimate the parameters of the Cox distribution:

Case 1: $c_X \leq 1$

To approximate a distribution with $c_X \leq 1$, we suggest using a special Cox distribution with (see Fig. 1.9):

$$\begin{aligned} \mu_j &= \mu \quad j = 1, \ldots, k, \\ a_j &= 1 \quad j = 2, \ldots, k-1. \end{aligned}$$

From the probabilistic definitions, we obtain:

mean:
$$\overline{X} = \frac{b_1 + k(1-b_1)}{\mu},$$

variance:
$$\operatorname{var}(X) = \frac{k + b_1(k-1)\,(b_1(1-k) + k - 2)}{\mu^2},$$

squared coefficient of variation:
$$c_X^2 = \frac{k + b_1(k-1)\,(b_1(1-k) + k - 2)}{[b_1 + k(1-b_1)]^2}.$$

In order to minimize the number of stages, we choose k such that:

$$k = \left\lceil \frac{1}{c_X^2} \right\rceil. \tag{1.33}$$

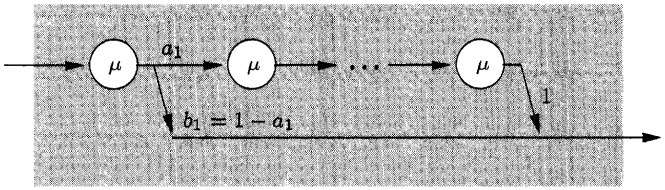


Fig. 1.9 A random variable with C_k distribution and $c_X \leq 1$

Then we obtain the parameters b_1 and μ from the preceding equations:

$$b_1 = \frac{2kc_X^2 + (k-2) - \sqrt{k^2 + 4 - 4kc_X^2}}{2(c_X^2 + 1)(k-1)}, \tag{1.34}$$

$$\mu = \frac{k - b_1 \cdot (k-1)}{\overline{X}}. \tag{1.35}$$

Case 2: $c_X > 1$

For the case $c_X > 1$, we use a Cox-2 model (see Fig. 1.10)

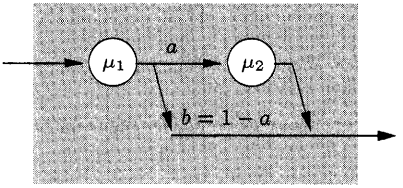


Fig. 1.10 A random variable with Cox-2 distribution.

From classical formulae, we obtain:

mean: $$\overline{X} = \frac{1}{\mu_1} + \frac{a}{\mu_2},$$

variance: $$\text{var}(X) = \frac{\mu_2^2 + a\mu_1^2(2-a)}{\mu_1^2 \cdot \mu_2^2},$$

squared coefficient of variation: $$c_X^2 = \frac{\mu_2^2 + a\mu_1^2(2-a)}{(\mu_2 + a\mu_1)^2}.$$

We have two equations for the three parameters μ_1, μ_2, and a, and therefore obtain an infinite number of solutions. To obtain simple equations we choose:

$$\mu_1 = \frac{2}{\overline{X}}, \tag{1.36}$$

and obtain from the preceding equations:

$$\mu_2 = \frac{1}{\overline{X}c_X^2} \quad \text{and} \quad a = \frac{1}{2c_X^2}. \tag{1.37}$$

(h) Weibull Distribution

The Weibull distribution is at present the most widely used failure distribution. Its CDF is given by:

$$F_X(x) = 1 - \exp(-(\lambda x)^\alpha), \quad x \geq 0. \tag{1.38}$$

pdf: $$f_X(x) = \alpha\lambda(\lambda x)^{\alpha-1}\exp(-(\lambda x)^\alpha), \quad \lambda > 0,$$

mean: $$\overline{X} = \frac{1}{\lambda}\Gamma\left(1 + \frac{1}{\alpha}\right),$$

squared coefficient of variation: $$c_X^2 = \frac{\Gamma(1+2/\alpha)}{\{\Gamma(1+1/\alpha)\}^2} - 1.$$

with the shape parameter $\alpha > 0$ and the scale parameter $\lambda > 0$.

(i) Lognormal Distribution

The lognormal distribution is given by:

$$F_X(x) = \Phi\left(\frac{\ln(x) - \lambda}{\alpha}\right), \quad x > 0. \tag{1.39}$$

pdf: $$f_X(x) = \frac{1}{\alpha\sqrt{2\pi}}\exp(-\{\ln(x) - \lambda\}^2/2\alpha^2), \quad x > 0,$$

mean: $$\overline{X} = \exp(\lambda + \alpha^2/2),$$

squared coefficient of variation: $$c_X^2 = \exp(\alpha^2) - 1,$$

where the shape parameter α is positive and the scale parameter λ may assume any real value.

The parameters α and λ can easily be calculated from c_X^2 and $\overline{X}$:

$$\alpha = \sqrt{\ln(c_x^2 + 1)},$$

$$\lambda = \ln\overline{X} - \frac{\alpha^2}{2}. \tag{1.40}$$

The importance of this distribution arises from the fact that the product of n mutually independent random variables has a log-normal distribution in the limit $n \to \infty$. In Table 1.2 the formulae for the expectation $E[X]$, the variance $var(X)$, and the coefficient of variation c_X for some important distribution functions are summarized. Furthermore, in Table 1.3 formulae for estimating the parameters of these distributions are given.

1.2.2 Multiple Random Variables

In some cases, the result of one random experiment determines the values of several random variables, where these values may also affect each other. The *joint probability mass function* of the discrete random variables

Table 1.2 Expectation $E[X]$, variance $\text{var}(X)$, and coefficient of variation c_X of important distributions.

Distribution	Parameter	$E[X]$	$\text{var}(X)$	c_X
Exponential	μ	$\frac{1}{\mu}$	$\frac{1}{\mu^2}$	1
Erlang	μ, k $k=1,2,\ldots$	$\frac{1}{\mu}$	$\frac{1}{k\mu^2}$	$\frac{1}{\sqrt{k}} \leq 1$
Gamma	μ, α $(0<\alpha<\infty)$	$\frac{\alpha}{\mu}$	$\frac{\alpha}{\mu^2}$	$0 < \frac{1}{\sqrt{\alpha}} < \infty$
Hypoexponential	μ_1, μ_2	$\frac{1}{\mu_1} + \frac{1}{\mu_2}$	$\frac{1}{\mu_1^2} + \frac{1}{\mu_2^2}$	$\frac{\sqrt{\mu_1^2+\mu_2^2}}{\mu_1+\mu_2} < 1$
Hyperexponential	k, μ_i, q_i	$\sum_{i=1}^{k} \frac{q_i}{\mu_i} = \frac{1}{\mu}$	$2\sum_{i=1}^{k} \frac{q_i}{\mu_i^2} - \frac{1}{\mu^2}$	$\sqrt{2\mu^2 \sum_{i=1}^{k} \frac{q_i}{\mu_i^2} - 1} > 1$

$X_1, X_2, \ldots, X_n$ is given by:

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) \tag{1.41}$$

and represents the probability that $X_1 = x_1$, $X_2 = x_2$, $\ldots$ $X_n = x_n$. In the continuous case, the *joint distribution function*:

$$F_{\mathbf{X}}(\mathbf{x}) = P(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_n \leq x_n) \tag{1.42}$$

represents the probability that $X_1 \leq x_1$, $X_2 \leq x_2$, $\ldots$ $X_n \leq x_n$, where $\mathbf{X} = (X_1, \ldots, X_n)$ is the n-dimensional random variable and $\mathbf{x} = (x_1, x_2, \ldots, x_n)$.

A simple example of an experiment with multiple discrete random variables is tossing a pair of dice. The following random variables might be determined:

X_1 number that shows on the first die,

X_2 number that shows on the second die,

$X_3 = X_1 + X_2$, sum of the numbers of both dice.

1.2.2.1 Independence The random variables $X_1, X_2, \ldots, X_n$ are called (*statistically*) *independent* if, for the continuous case:

$$\begin{aligned} &P(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_n \leq x_n) \\ &\quad = P(X_1 \leq x_1) \cdot P(X_2 \leq x_2) \cdot \ldots \cdot P(X_n \leq x_n), \end{aligned} \tag{1.43}$$

or the discrete case:

$$\begin{aligned} &P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) \\ &\quad = P(X_1 = x_1) \cdot P(X_2 = x_2) \cdot \ldots \cdot P(X_n = x_n). \end{aligned} \tag{1.44}$$

Table 1.3 Formulae for estimating the parameters of important distributions.

Distribution	Parameter	Calculation of the Parameters
Exponential	μ	$\mu = 1/\overline{X}$
Erlang	μ, k $k=1,2,\ldots$	$k = \text{ceil}(1/c_X^2)$ $\mu = 1/(c_X^2 \cdot k\overline{X})$
Gamma	μ, α $0<\alpha<\infty$	$\alpha = 1/c_X^2$ $\mu = 1/\overline{X}$
Hypoexponential	μ_1, μ_2	$\mu_{1/2} = \frac{2}{\overline{X}}\left[1 \pm \sqrt{1+2(c_X^2-1)}\right]^{-1}$
Hyperexponential (H_2)	μ_1, μ_2, q_1, q_2	$\mu_1 = \frac{1}{\overline{X}}\left[1 - \sqrt{\frac{q_2}{q_1}\frac{c_X^2-1}{2}}\right]^{-1}$ $\mu_2 = \frac{1}{\overline{X}}\left[1 + \sqrt{\frac{q_1}{q_2}\frac{c_X^2-1}{2}}\right]^{-1}$ $q_1 + q_2 = 1, \mu_2 > 0$
Cox ($c_X \leq 1$)	k, b_i, μ_i	$k = \text{ceil}(1/c_X^2)$ $b_1 = \frac{2kc_X^2 + k - 2 - \sqrt{k^2+4-4kc_X^2}}{2(c_X^2+1)(k-1)}$ $b_2 = b_3 = \ldots = b_{k-1} = 0, \quad b_k = 1$ $\mu_1 = \mu_2 = \ldots = \mu_k = \frac{k - b_1 \cdot (k-1)}{\overline{X}}$
Cox ($c_X > 1$)	k, b, μ_1, μ_2	$k = 2$ $b = c_X^2\left[1 - \sqrt{1 - \frac{2}{1+c_X^2}}\right]$ $\mu_{1/2} = \frac{1}{\overline{X}}\left[1 \pm \sqrt{1 - \frac{2}{1+c_X^2}}\right]$

Otherwise, they are (*statistically*) *dependent.* In the preceding example of tossing a pair of dice, the random variables X_1 and X_2 are independent, whereas X_1 and X_3 are dependent on each other. For example:

$$P(X_1 = i, X_2 = j) = \frac{1}{36}, \qquad i, j = 1, \ldots, 6,$$

since there are 36 different possible results altogether that are all equally probable. Because the following is also valid:

$$P(X_1 = i) \cdot P(X_2 = j) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36},$$

$P(X_1 = i, X_2 = j) = P(X_1 = i) \cdot P(X_2 = j)$ is true, and therefore X_1 and X_2 are independent. On the other hand, if we observe the dependent variables X_1 and X_3, then:

$$P(X_1 = 2, X_3 = 4) = P(X_1 = 2, X_2 = 2) = \frac{1}{36},$$

since X_1 and X_2 are independent. Having:

$$\begin{aligned} P(X_1 = 2) &= \frac{1}{6} \quad \text{and} \\ P(X_3 = 4) &= P(X_1 = 1, X_2 = 3) + P(X_1 = 2, X_2 = 2) \\ &\quad + P(X_1 = 3, X_2 = 1) = \frac{3}{36} = \frac{1}{12} \end{aligned}$$

results in $P(X_1 = 2) \cdot P(X_3 = 4) = \frac{1}{6} \cdot \frac{1}{12} = \frac{1}{72}$, which shows the dependency of the random variables X_1 and X_3:

$$P(X_1 = 2, X_3 = 4) \neq P(X_1 = 2) \cdot P(X_3 = 4).$$

1.2.2.2 Conditional Probability A *conditional probability*:

$$P(X_1 = x_1 \mid X_2 = x_2, \ldots, X_n = x_n)$$

is the probability for $X_1 = x_1$ under the conditions $X_2 = x_2$, $X_3 = x_3$, etc. Then we get:

$$\begin{aligned} &P(X_1 = x_1 \mid X_2 = x_2, \ldots, X_n = x_n) \\ &\quad = \frac{P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)}{P(X_2 = x_2, \ldots, X_n = x_n)}. \end{aligned} \tag{1.45}$$

For continuous random variables we have:

$$\begin{aligned} &P(X_1 \leq x_1 \mid X_2 \leq x_2, \ldots, X_n \leq x_n) \\ &\quad = \frac{P(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_n \leq x_n)}{P(X_2 \leq x_2, \ldots, X_n \leq x_n)}. \end{aligned} \tag{1.46}$$

We demonstrate this also with the preceding example of tossing a pair of dice. The probability that $X_3 = j$ under the condition that $X_1 = i$ is given by:

$$P(X_3 = j \mid X_1 = i) = \frac{P(X_3 = j, X_1 = i)}{P(X_1 = i)}.$$

For example, with $j = 4$ and $i = 2$:

$$P(X_3 = 4 \mid X_1 = 2) = \frac{1/36}{1/6} = \frac{1}{6}.$$

If we now observe both random variables X_1 and X_2, we will see that because of the independence of X_1 and X_2,

$$\begin{aligned} P(X_1 = j \mid X_2 = i) &= \frac{P(X_1 = j, X_2 = i)}{P(X_2 = i)}, \\ &= \frac{P(X_1 = j) \cdot P(X_2 = i)}{P(X_2 = i)}, \\ &= P(X_1 = j). \end{aligned}$$

1.2.2.3 Important Relations The following relations concerning multiple random variables will be used frequently in the text:

- The expected value of a sum of random variables is equal to the sum of the expected values of these random variables. If $c_1, \ldots, c_n$ are arbitrary constants and $X_1, X_2, \ldots, X_n$ are (not necessarily independent) random variables, then:

$$E\left[\sum_{i=1}^{n} c_i X_i\right] = \sum_{i=1}^{n} c_i E[X_i]. \tag{1.47}$$

- If the random variables $X_1, X_2, \ldots, X_n$ are stochastically independent, then the expected value of the product of the random variables is equal to the product of the expected values of the random variables:

$$E\left[\prod_{i=1}^{n} X_i\right] = \prod_{i=1}^{n} E[X_i]. \tag{1.48}$$

- The *covariance* of two random variables X and Y is a way of measuring the dependency between X and Y. It is defined by:

$$\begin{aligned} \operatorname{cov}[X, Y] &= E\left[(X - E[X])(Y - E[Y])\right] = \overline{(X - \overline{X})(Y - \overline{Y})} \\ &= E[X \cdot Y] - E[X] \cdot E[Y] = \overline{XY} - \overline{X} \cdot \overline{Y}. \end{aligned} \tag{1.49}$$

If $X = Y$, then the covariance is equal to the variance:

$$\text{cov}[X, X] = \overline{X^2} - \overline{X}^2 = \text{var}(X) = \sigma_X^2.$$

If X and Y are statistically independent, then Eq. (1.48) gives us:

$$E[X \cdot Y] = E[X] \cdot E[Y],$$

and Eq. (1.49):

$$\text{cov}(X, Y) = 0.$$

In this case, X and Y are said to be uncorrelated.

- The *correlation coefficient* is the covariance normalized to the product of standard deviations:

$$\text{cor}[X, Y] = \frac{\text{cov}[X, Y]}{\sigma_X \cdot \sigma_Y}. \tag{1.50}$$

Therefore, if $X = Y$, then:

$$\text{cor}[X, X] = \frac{\sigma_X^2}{\sigma_X^2} = 1.$$

- The variance of a sum of random variables can be expressed using the covariance:

$$\text{var}\left[\sum_{i=1}^{n} c_i X_i\right] = \sum_{i=1}^{n} c_i^2 \,\text{var}(X_i) + 2 \cdot \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_i c_j \,\text{cov}[X_i, X_j]. \tag{1.51}$$

For two independent (uncorrelated) random variables we get:

$$\text{var}\left[\sum_{i=1}^{n} c_i X_i\right] = \sum_{i=1}^{n} c_i^2 \,\text{var}(X_i). \tag{1.52}$$

1.2.2.4 The Central Limit Theorem Let $X_1, X_2, \ldots, X_n$ be independent, identically distributed random variables with an expected value of $E[X_i] = \overline{X}$, and a variance of $\text{var}(X_i) = \sigma_X^2$. Then their arithmetic mean is defined by:

$$S_n = \frac{1}{n} \sum_{i=1}^{n} X_i.$$

We have $E[S_n] = \overline{X}$ and $\text{var}(S_n) = \sigma_X^2 / n$. Regardless of the distribution of X_i, the random variable S_n approaches, with increasing n, a normal distribution with a mean of $\overline{X}$ and a variance of $\text{var}(X)/n$:

$$f_{S_n}(x) \underset{n \to \infty}{\longrightarrow} \frac{1}{\sigma_X \sqrt{2\pi/n}} \cdot \exp\left(-\frac{(x - \overline{X})^2}{2\sigma_X^2/n}\right). \tag{1.53}$$

1.2.3 Transforms

Determining the mean values, variance, or moments of random variables is often rather tedious. However, these difficulties can sometimes be avoided by using transforms, since the pmf of a discrete or the density function of a continuous random variable are uniquely determined by their z- or Laplace transform, respectively. Thus if two random variables have the same transform, they also have the same distribution function and vice versa. In many cases, the parameters of a random variable are, therefore, more easily computed from the transform.

1.2.3.1 z-Transform Let X be a discrete random variable with the probability mass function $p_k = P(X = k)$ for $k = 0, 1, 2, \ldots$, then the *z-transform* of X is defined by:

$$G_X(z) = \sum_{k=0}^{\infty} p_k \cdot z^k \quad \text{with } |z| \leq 1. \tag{1.54}$$

The function $G_X(z)$ is also called the *probability generating function.* Table 1.4 specifies the probability generating functions of some important discrete random variables. If the z-transform $G_X(z)$ of a random variable is given, then the probabilities p_k can be calculated immediately:

$$p_0 = G_X(0), \tag{1.55}$$

$$p_k = \frac{1}{k!} \cdot \left. \frac{d^k G_X(z)}{dz^k} \right|_{z=0}, \qquad k = 1, 2, 3, \ldots. \tag{1.56}$$

And for the moments:

$$\overline{X} = \left. \frac{dG_X(z)}{dz} \right|_{z=1}, \tag{1.57}$$

$$\overline{X^2} = \left. \frac{d^2 G_X(z)}{dz^2} \right|_{z=1} + \overline{X}, \tag{1.58}$$

$$\overline{X^k} = \left. \frac{d^k G_X(z)}{dz^k} \right|_{z=1} + \sum_{i=1}^{k-1} a_{k,i} \cdot \overline{X^i}, \qquad k = 3, 4, \ldots, \tag{1.59}$$

with

$$\begin{aligned} 3a_{k,1} &= (-1)^k (k-1)!, && k = 3, 4, \ldots, \\ a_{k,k-1} &= \frac{k(k-1)}{2}, && k = 3, 4, \ldots, \\ a_{k,i} &= a_{k-1,i-1} - (k-1)a_{k-1,i}, && k = 4, 5, \ldots \text{ and } i = 2, \ldots, k-2. \end{aligned}$$

Table 1.4 Generating functions of some discrete random variables

Random Variable	Parameter	$G_X(z)$
Binomial	n, p	$[(1-p)+pz]^n$
Geometric	p	$\frac{pz}{1-(1-p)z}$
Poisson	α	$e^{\alpha(z-1)}$

The generating function of a sum of independent random variables is the product of the generating functions of each random variable, so for $X = \sum_{i=1}^{n} X_i$ we get:

$$G_X(z) = \prod_{i=1}^{n} G_{X_i}(z). \tag{1.60}$$

1.2.3.2 Laplace Transform The Laplace transform plays a similar role for non-negative continuous random variables as the z-transform does for discrete random variables. If X is a continuous random variable with the density function $f_X(x)$, then the *Laplace transform* (LT) of the density function (pdf) of X is defined by:

$$L_X(s) = \int_o^\infty f_X(x)e^{-sx}dx, \qquad |\, e^{-s} \,| \leq 1, \tag{1.61}$$

where s is a complex parameter. $L_X(s)$ is also called the Laplace-Stieltjes transform (LST) of the distribution function (CDF) and can also be written as:

$$L_X(s) = \int_0^\infty e^{-sx}dF_X(x). \tag{1.62}$$

LST of a random variable X will also be denoted by $X\tilde{}(s)$.

The moments can again be determined by differentiation:

$$\overline{X^k} = (-1)^k \frac{d^k L_X(s)}{ds^k}\bigg|_{s=0}, \quad k = 1, 2, \ldots. \tag{1.63}$$

The Laplace transforms of several pdfs of well-known continuous random variables as well as their moments are given in Table 1.5.

Table 1.5 Laplace transform and moments of several continuous random variables

Random Variable	Parameter	$L_X(s)=X^{\sim}(s)$	$\overline{X^n}$
Exponential	λ	$\frac{\lambda}{\lambda+s}$	$\frac{n!}{\lambda^n}$
Erlang	k,λ	$\left(\frac{\lambda}{\lambda+s}\right)^k$	$\frac{k(k+1)\ldots(k+n-1)}{\lambda^n}$
Gamma	α,λ	$\left(\frac{\lambda}{\lambda+s}\right)^\alpha$	$\frac{\alpha(\alpha+1)\ldots(\alpha+n-1)}{\lambda^n}$
Hyperexponential	λ	$\sum_{j=1}^{k} q_j \frac{\lambda_j}{\lambda_j+s}$	$\sum_{j=1}^{k} q_j \frac{n!}{\lambda_j^n}$
Hypoexponential	λ_1,λ_2	$\frac{\lambda_1\lambda_2}{\lambda_1-\lambda_2}\left(\frac{1}{\mu_2+s}-\frac{1}{\mu_1+s}\right)$	$\frac{n!}{\mu_1-\mu_2}\left(\frac{\mu_1}{\mu_2^n}-\frac{\mu_2}{\mu_1^n}\right)$

Analogous to the z-transform, the Laplace transform of a sum of independent random variables is the product of the Laplace transform of these random variables:

$$L_X(s) = \prod_{i=1}^{n} L_{X_i}(s). \tag{1.64}$$

More properties and details of Laplace and z-transforms are listed in [Klei75].

1.2.4 Parameter Estimation

Probability calculations discussed in this book require that the parameters of all the relevant distributions are known in advance. For practical use, these parameters have to be estimated from measured data. In this subsection, we briefly discuss statistical methods of parameter estimation. For further details see [Triv82].

Definition 1.1 Let the set of random variables $X_1, X_2, \ldots, X_n$ be given. This set is said to constitute a *random sample* of size n from a population with associated distribution function $F_X(x)$. It is presupposed that the random variables are mutually independent and identically distributed with distribution function (CDF) $F_{X_i}(x) = F_X(x) \quad \forall i, x$.

We are interested in estimating the value of some parameter θ (e. g. , the mean or the variance) of the population based on the random samples.

Definition 1.2 A function $\widehat{\Theta} = \widehat{\Theta}(X_1, X_2, \ldots, X_n)$ is called an *estimator* of the parameter θ, and an observed value $\hat{\theta} = \hat{\theta}(x_1, x_2 \ldots, x_n)$ is known as

an *estimate* of θ. An estimator $\widehat{\Theta}$ is considered unbiased, if:

$$E[\widehat{\Theta}(X_1, X_2, \ldots, X_n)] = \theta.$$

It can be shown that the sample mean $\overline{X}$ defined as:

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{1.65}$$

is an unbiased estimator of the population mean μ, and the sample variance S^2 defined as:

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2 \tag{1.66}$$

is an unbiased estimator of the population variance σ^2.

Definition 1.3 An estimator $\widehat{\Theta}$ of a parameter θ is said to be *consistent* if $\widehat{\Theta}$ converges in probability to θ (so-called stochastic convergence); that is:

$$\lim_{n\to\infty} P(|\widehat{\Theta} - \theta| \geq \epsilon) = 0 \forall \epsilon > 0.$$

1.2.4.1 Method of Moments The kth sample moment of a random variable X is defined as:

$$M'_k = \sum_{i=1}^{n} \frac{X_i^k}{n}, \quad k = 1, 2, \ldots. \tag{1.67}$$

To compute one or more parameters of the distribution of X, using the *method of moments,* we equate the first few population moments with their corresponding sample moments to obtain as many equations as the number of unknown parameters. These equations can then be solved to obtain the required estimates. The method yields estimators that are consistent, but they may be biased.

As an example, suppose the time to failure of a computer system has a gamma distributed pdf:

$$f_X(x) = \frac{\lambda^\alpha x^{\alpha-1} e^{-\lambda x}}{\Gamma(\alpha)}, \quad k = 1, 2, \ldots, \tag{1.68}$$

with parameters λ and α. We know that the first two population moments are given by:

$$\mu_1 = \frac{\alpha}{\lambda} \quad \text{and} \quad \mu_2 = \frac{\alpha}{\lambda^2} + {\mu_1}^2.$$

Then the estimates $\hat{\lambda}$ and $\hat{\alpha}$ can be obtained by solving:

$$\frac{\hat{\alpha}}{\hat{\lambda}} = M_1, \quad \text{and} \quad \frac{\hat{\alpha}}{\hat{\lambda}^2} + \frac{\hat{\alpha}^2}{\hat{\lambda}^2} = M_2.$$

Hence:

$$\hat{\alpha} = \frac{{M_1}^2}{M_2 - {M_1}^2}, \quad \text{and} \quad \hat{\lambda} = \frac{M_1}{M_2 - {M_1}^2}. \tag{1.69}$$

1.2.4.2 Maximum-Likelihood Estimation The principle of this method is to select as an estimate of θ the value for which the observed sample is most likely to occur. We define the likelihood function as the product of the marginal densities:

$$L(\theta) = \prod_{i=1}^{n} f_{X_i}(x_i \mid \theta).$$

where $\theta = (\theta_1, \theta_2, \ldots, \theta_k)$ is the vector of parameters to be estimated. Under some regularity conditions, the maximum-likelihood estimate of θ is the solution of the simultaneous equations:

$$\frac{\partial L(\theta)}{\partial \theta_i} = 0, \quad i = 1, 2, \ldots, k.$$

As an example, assume the interarrival time, X, of a system is exponentially distributed with arrival rate λ. To estimate the arrival rate λ from a random sample of n interarrival times, we define:

$$L(\lambda) = \prod_{i=1}^{n} \lambda \exp(-\lambda x_i) = \lambda^n \exp(-\lambda \sum_{i=1}^{n} x_i), \tag{1.70}$$

$$\frac{dL}{d\lambda} = n\lambda^{n-1} \exp(-\lambda \sum_{i=1}^{n} x_i) - \left(\sum_{i=1}^{n} x_i \right) \lambda^n \exp(-\lambda \sum_{i=1}^{n} x_i) = 0, \tag{1.71}$$

from which we get the maximum-likelihood estimator of the arrival rate that is equal to the reciprocal of the sample mean $\overline{X}$.

1.2.4.3 Confidence Intervals So far we have concentrated on obtaining the point estimates of a desired parameter. However, it is very rare in practice for the point estimate to actually coincide with the exact value of the parameter θ under consideration. In this section, we instead consider the evaluation of an *interval estimate.* We construct an interval, called the *confidence interval,* in such a way that we have a certain confidence that this interval contains the true value of the unknown parameter θ. In other words, if the estimator $\widehat{\Theta}$ satisfies the condition:

$$P(\widehat{\Theta} - \epsilon_1 < \theta < \widehat{\Theta} + \epsilon_2) = \gamma, \tag{1.72}$$

then the interval $A(\theta) = (\widehat{\Theta} - \epsilon_1, \widehat{\Theta} + \epsilon_2)$ is a $100 \times \gamma$ percent confidence interval for the parameter θ. Here γ is called the confidence coefficient (the probability that the confidence interval contains θ).

Consider obtaining confidence intervals when using the sample mean $\overline{X}$ as the estimator for the population mean. If the population distribution is normal with unknown mean μ and a known variance σ^2, then we can show that the sample mean is normally distributed with mean μ and variance σ^2/n, so that the random variable $Z = (\overline{X} - \mu)/(\sigma/\sqrt{n})$ has a standard normal distribution $N(0, 1)$. To determine a $100 \times \gamma$ percent confidence interval for

the population mean μ, then we find a number $z_{\alpha/2}$ (using $N(0,1)$ tables) such that $P(Z > z_{\alpha/2}) = \alpha/2$, where $\alpha = 1 - \gamma$. Then we get:

$$P(-z_{\alpha/2} < Z < z_{\alpha/2}) = \gamma. \tag{1.73}$$

We then obtain the $100 \times \gamma$ percent confidence interval as:

$$-z_{\alpha/2} < (\overline{X} - \mu)/(\sigma/\sqrt{n}) < z_{\alpha/2} \tag{1.74}$$

or:

$$\overline{X} - z_{\alpha/2}\frac{\sigma}{\sqrt{n}} < \mu < \overline{X} + z_{\alpha/2}\frac{\sigma}{\sqrt{n}}. \tag{1.75}$$

The assumption that the population is normally distributed does not always hold. But when the sample size is large, by central limit theorem, the statistic $(\overline{X} - \mu)/(\sigma/\sqrt{n})$ is asymptotically normal (under appropriate conditions). Also, Eq. (1.75) requires the knowledge of the population variance σ^2. When σ^2 is unknown, we use the sample variance S^2 to get the approximate confidence interval for μ:

$$\overline{X} - z_{\alpha/2}\frac{S}{\sqrt{n}} < \mu < \overline{X} + z_{\alpha/2}\frac{S}{\sqrt{n}}. \tag{1.76}$$

When the sample size is relatively small and the population is normal distributed, then the random variable $T = (\overline{X} - \mu)/(S/\sqrt{n})$ has the student t distribution with $n-1$ degrees of freedom. We can then obtain the $100(1-\alpha)$ percent confidence interval of μ from:

$$\overline{X} - t_{n-1;\alpha/2}\frac{S}{\sqrt{n}} < \mu < \overline{X} + t_{n-1;\alpha/2}\frac{S}{\sqrt{n}}, \tag{1.77}$$

where $P(T > t_{n-1;\alpha/2}) = \alpha/2$.

1.2.5 Order Statistics

Let $X_1, X_2 \ldots, X_n$ be mutually independent, identically distributed continuous random variables, each having the distribution function $F(.)$ and density $f(.)$. Let $Y_1, Y_2, \ldots, Y_n$ be random variables obtained by permuting the set $X_1, X_2, \ldots, X_n$ so as to be in increasing order. Thus, for instance:

$$Y_1 = \min\{X_1, X_2, \ldots, X_n\}$$

and:

$$Y_n = \max\{X_1, X_2, \ldots, X_n\}.$$

The random variable Y_k is generally called the *kth -order statistic.* Because $X_1, X_2, \ldots, X_n$ are continuous random variables, it follows that $Y_1 < Y_2 < \cdots < Y_n$ (as opposed to $Y_1 \leq Y_2 \leq \cdots \leq Y_n$) with a probability of one.

As examples of the use of order statistics, let X_i be the time to failure of the ith component in a system of n independent components. If the system is a series system, then Y_1 will be the system lifetime. Similarly, Y_n will denote the lifetime of a parallel system and Y_{n-m+1} will be the lifetime of an m-out-of-n system.

The distribution function of Y_k is given by:

$$F_{Y_k}(y) = \sum_{j=k}^{n} \binom{n}{j} F^j(y)[1 - F(y)]^{n-j}, \quad -\infty < y < \infty. \tag{1.78}$$

In particular, the distribution functions of Y_n and Y_1 can be obtained from Eq. (1.78) as:

$$F_{Y_n}(y) = [F(y)]^n, \quad -\infty < y < \infty,$$

and

$$F_{Y_1}(y) = 1 - [1 - F(y)]^n, \quad -\infty < y < \infty.$$

1.2.6 Distribution of Sums

Assume that X and Y are continuous random variables with joint probability density function f. Suppose we are interested in the density of a random variable $Z = \Phi(X, Y)$. The distribution of Z may be written as:

$$\begin{aligned} F_Z(z) &= P(Z \leq z) \\ &= \int\int_{A_z} f(x, y)dxdy, \end{aligned} \tag{1.79}$$

where A_z is a subset of $\mathbb{R}^2$ given by:

$$\begin{aligned} A_z &= \{(x, y) \mid \Phi(x, y) \leq z\} \\ &= \Phi^{-1}((-\infty, z]). \end{aligned}$$

A function of special interest is $Z = X + Y$ with

$$A_z = \{(x, y) \mid x + y \leq z\},$$

which is the half-plane to the lower left of the line $x+y = z$, shown in Fig. 1.11. Then we get:

$$\begin{aligned} F_Z(z) &= \int\int_{A_z} f(x, y)dxdy \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{z-x} f(x, y)dydx. \end{aligned}$$

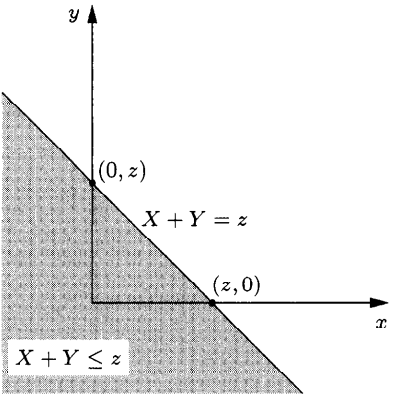


Fig. 1.11 Area of integration for the convolution of X and Y.

Making a variable substitution $y = t - x$, we get:

$$\begin{aligned} F_Z(z) &= \int_{-\infty}^{\infty} \int_{-\infty}^{z} f(x, t-x) dt dx \\ &= \int_{-\infty}^{z} \int_{-\infty}^{\infty} f(x, t-x) dx dt \\ &= \int_{-\infty}^{z} f_Z(t) dt. \end{aligned}$$

Thus the density of Z is given by:

$$f_Z(z) = \int_{-\infty}^{\infty} f(x, z-x) dx, \quad -\infty < z < \infty. \tag{1.80}$$

Now if X and Y are independent random variables, then $f(x, y) = f_X(x) f_Y(y)$, and Formula (1.80) reduces to:

$$f_Z(z) = \int_{-\infty}^{\infty} f_X(x) f_Y(z-x) dx, \quad -\infty < z < \infty. \tag{1.81}$$

Further, if both X and Y are nonnegative random variables, then:

$$f_Z(z) = \int_{0}^{z} f_X(x) f_Y(z-x) dx, \quad 0 < z < \infty. \tag{1.82}$$

This integral is known as the *convolution* of f_X and f_Y. Thus, the pdf of the sum of two non-negative independent random variables is the convolution of their individual pdfs.

If $X_1, X_2, \ldots, X_r$ are mutually independent, identically distributed exponential random variables with parameter λ, then the random variable $X_1 + X_2 + \cdots + X_r$ has an Erlang-r distribution with parameter λ. Consider an asynchronous transfer mode (ATM) switch with cell interarrival times that are independent from each other and exponentially distributed with parameter λ. Let X_i be the random variable denoting the time between the $(i-1)$st and ith arrival. Then $Z_r = X_1 + X_2 + \cdots + X_r$ is the time until the rth arrival and has an Erlang-r distribution. Another way to obtain this result is to consider N_t, the number of arrivals in the interval $(0, t]$. As pointed out earlier, N_t has a Poisson distribution with parameter λt. Since the events $[Z_r > t]$ and $[N_t < r]$ are equivalent, we have:

$$\begin{aligned} P(Z_r > t) &= P(N_t < r) \\ &= \sum_{j=0}^{r-1} e^{-\lambda t} \left[\frac{(\lambda t)^j}{j!} \right], \end{aligned}$$

which implies that:

$$\begin{aligned} F_{Z_r}(t) &= P(Z_r \leq t) \\ &= 1 - \sum_{j=0}^{r-1} \frac{(\lambda t)^j}{j!} e^{-\lambda t}, \end{aligned}$$

which is the Erlang-r distribution function. If $X \sim EXP(\lambda_1)$, $Y \sim EXP(\lambda_2)$, X and Y are independent, and $\lambda_1 \neq \lambda_2$, then $Z = X + Y$ has a two-stage hypoexponential distribution with two phases and parameters λ_1 and λ_2. To see this:

$$\begin{aligned} f_Z(z) &= \int_0^z f_X(x) f_Y(z-x) dx, \quad z > 0 \quad \text{(by Eq. (1.82))} \\ &= \int_0^z \lambda_1 e^{-\lambda_1 x} \lambda_2 e^{-\lambda_2 (z-x)} dx \\ &= \lambda_1 \lambda_2 e^{-\lambda_2 z} \int_0^z e^{(\lambda_2 - \lambda_1) x} dx \\ &= \lambda_1 \lambda_2 e^{-\lambda_2 z} \left[\frac{e^{-(\lambda_1 - \lambda_2) x}}{-(\lambda_1 - \lambda_2)} \right]_{x=0}^{x=z} \\ &= \frac{\lambda_1 \lambda_2}{\lambda_1 - \lambda_2} e^{-\lambda_2 z} + \frac{\lambda_1 \lambda_2}{\lambda_2 - \lambda_1} e^{-\lambda_1 z} \quad \text{(see Eq. (1.30)).} \end{aligned}$$

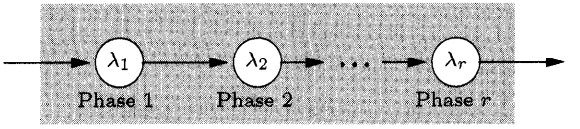


Fig. 1.12 Hypoexponential distribution as a series of exponential stages.

A more general version of this result follows: Let $Z = \sum_{i=1}^{r} X_i$, where $X_1, X_2, \ldots, X_r$ are mutually independent and X_i is exponentially distributed with parameter λ_i ($\lambda_i \neq \lambda_j$ for $i \neq j$). Then Z has a hypoexponential distribution with r phases and the density function:

$$f_Z(z) = \sum_{i=1}^{r} a_i \lambda_i e^{-\lambda_i z}, \quad z > 0, \tag{1.83}$$

where:

$$a_i = \prod_{\substack{j=1 \\ j \neq i}}^{r} \frac{\lambda_j}{\lambda_j - \lambda_i}, \quad 1 \leq i \leq r. \tag{1.84}$$

Such a phase type distribution can be visualized as shown in Fig. 1.12 (see also Section 1.2.1.2).

2

Markov Chains

2.1 MARKOV PROCESSES

Markov processes provide very flexible, powerful, and efficient means for the description and analysis of dynamic (computer) system properties. Performance and dependability measures can be easily derived. Moreover, Markov processes constitute the fundamental theory underlying the concept of queueing systems. In fact, the notation of queueing systems has been viewed sometimes as a high-level *specification* technique for (a sub-class of) Markov processes. Each queueing system can, in principle, be mapped onto an instance of a Markov process and then mathematically evaluated in terms of this process. But besides highlighting the *computational* relation between Markov processes and queueing systems, it is worthwhile pointing out also that fundamental *properties* of queueing systems are commonly *proved* in terms of the underlying Markov processes. This type of use of Markov processes is also possible even when queueing systems exhibit properties such as non-exponential distributions that cannot be represented directly by discrete-state Markov models. Markovizing methods, such as embedding techniques or supplementary variables, can be used in such cases. Here Markov processes serve as a mere *theoretical* framework to prove the correctness of computational methods applied directly to the analysis of queueing systems. For the sake of efficiency, an explicit creation of the Markov process is preferably avoided.

2.1.1 Stochastic and Markov Processes

There exist many textbooks like those from King [King90], Trivedi [Triv82], Allen [Alle90], Gross and Harris [GrHa85], Cinlar [Cinl75], Feller (his two volume classic [Fell68]), and Howard [Howa71] that provide excellent intro-

ductions into the basics of stochastic and Markov processes. Besides the theoretical background, many motivating examples are also given in those books. Consequently, we limit discussion here to the essentials of Markov processes and refer to the literature for further details.

Markov processes constitute a special, perhaps the most important, subclass of stochastic processes, while the latter can be considered as a generalization of the concept of random variables. In particular, a stochastic process provides a relation between the elements of a possibly infinite family of random variables. A series of random experiments can thus be taken into consideration and analyzed as a whole.

Definition 2.1 A *stochastic process* is defined as a family of random variables $\{X_t : t \in T\}$ where each random variable X_t is indexed by parameter $t \in T$, which is usually called the *time parameter* if $T \subseteq \mathbb{R}_+ = [0, \infty)$. The set of all possible values of X_t (for each $t \in T$) is known as the state space S of the stochastic process.

If a countable, discrete-parameter set T is encountered, the stochastic process is called a *discrete-parameter* process and T is commonly represented by (a subset of) $\mathbb{N}_0 = \{0, 1, \ldots\}$; otherwise we call it a *continuous-parameter* process. The *state space* of the stochastic process may also be continuous or discrete. Generally, we restrict ourselves here to the investigation of discrete state spaces and in that case refer to the stochastic processes as *chains*, but both continuous- and discrete-parameter processes are considered.

Definition 2.2 Continuous-parameter stochastic processes can be probabilistically characterized by the *joint (cumulative) distribution function* (CDF) $F_{\mathbf{X}}(\mathbf{s}; \mathbf{t})$ for a given set of random variables $\{X_{t_1}, X_{t_2}, \ldots, X_{t_n}\}$, parameter vector $\mathbf{t} = (t_1, t_2, \ldots, t_n) \in \mathbb{R}^n$, and state vector $\mathbf{s} = (s_1, s_2, \ldots, s_n) \in \mathbb{R}^n$, where $t_1 < t_2 < \cdots < t_n$:

$$F_{\mathbf{X}}(\mathbf{s}; \mathbf{t}) = P(X_{t_1} \leq s_1, X_{t_2} \leq s_2, \ldots, X_{t_n} \leq s_n). \tag{2.1}$$

The *joint probability density function*, pdf:

$$f_{\mathbf{X}}(\mathbf{s}; \mathbf{t}) = \frac{\partial^n F_{\mathbf{X}}(\mathbf{s}; \mathbf{t})}{\partial s_1 \partial s_2 \ldots \partial s_n}$$

is defined correspondingly if the partial derivatives exist. If the so-called Markov property is imposed on the *conditional CDF* of a stochastic process, a Markov process results:

Definition 2.3 A stochastic process $\{X_t : t \in T\}$ constitutes a *Markov process* if for all $0 = t_0 < t_1 < \cdots < t_n < t_{n+1}$ and all $s_i \in S$ the conditional CDF of $X_{t_{n+1}}$ depends only on the last previous value X_{t_n} and not on the earlier values $X_{t_0}, X_{t_1}, \ldots, X_{t_{n-1}}$:

$$\begin{aligned} &P(X_{t_{n+1}} \leq s_{n+1} \mid X_{t_n} = s_n, X_{t_{n-1}} = s_{n-1}, \ldots, X_{t_0} = s_0) \\ &\quad = P(X_{t_{n+1}} \leq s_{n+1} \mid X_{t_n} = s_n). \end{aligned} \tag{2.2}$$

This most general definition of a Markov process can be adopted to special cases. In particular, we focus here on discrete state spaces and on both discrete- and continuous-parameter Markov processes. As a result, we deal primarily with *continuous-time Markov chains* (CTMC), and with *discrete-time Markov chains* (DTMC), which are introduced in the next section. Finally, it is often sufficient to consider only systems with a time independent, i.e., *time-homogeneous*, pattern of dynamic behavior. Note that time-homogeneous system dynamics is to be discriminated from stationary system behavior, which relates to time independence in a different sense. The former refers to the stationarity of the *conditional* CDF while the latter refers to the stationarity of the CDF *itself.*

Definition 2.4 Letting $t_0 = 0$ without loss of generality, a Markov process is said to be *time-homogeneous* if the conditional CDF of $X_{t_{n+1}}$ does not depend on the observation time, that is, it is invariant with respect to time epoch t_n:

$$P(X_{t_{n+1}} \leq s_{n+1} \mid X_{t_n} = s_n) = P(X_{t_{n+1}-t_n} \leq s_{n+1} \mid X_0 = s_n). \tag{2.3}$$

2.1.2 Markov Chains

Equation (2.2) describes the well-known Markov property. Informally this can be interpreted in the sense that the whole history of a Markov chain is summarized in the current state X_{t_n}. Equivalently, given the present, the future is conditionally independent of the past. Note that the Markov property does not prevent the conditional distribution from being dependent on the time variable t_n. Such a dependence is prevented by the definition of homogeneity (see Eq. (2.3)). A unique characteristic is implied, namely, the sojourn time distribution in any state of a homogeneous Markov chain exhibits the memoryless property. An immediate, and somewhat curious, consequence is that the mean sojourn time equals the mean residual and the mean elapsed time in any state and at any time [Triv82].

If not explicitly stated otherwise, we consider Markov processes with discrete state spaces only, that is, Markov chains, in what follows. Note that in this case we are inclined to talk about probability mass functions, pmf, rather than probability density functions, pdf. Refer back to Sections 1.2.1.1 and 1.2.1.2 for details.

2.1.2.1 Discrete-Time Markov Chains We are now ready to proceed to the formal definition of Markov chains. Discrete-parameter Markov chains are considered first, that is, Markov processes restricted to a discrete, finite, or countably infinite state space, S, and a discrete-parameter space T. For the sake of convenience, we set $T \subseteq \mathbb{N}_0$. The conditional pmf reflecting the Markov property for discrete-time Markov chains, corresponding to Eq. (2.2), is summarized in the following definition:

Definition 2.5 A given stochastic process $\{X_0, X_1, \ldots, X_{n+1}, \ldots\}$ at the consecutive points of observation $0, 1, \ldots, n+1$ constitutes a *DTMC* if the following relation on the *conditional pmf*, that is, the Markov property, holds for all $n \in \mathbb{N}_0$ and all $s_i \in S$:

$$\begin{aligned} &P(X_{n+1} = s_{n+1} \mid X_n = s_n, X_{n-1} = s_{n-1}, \ldots, X_0 = s_0) \\ &\quad = P(X_{n+1} = s_{n+1} \mid X_n = s_n). \end{aligned} \tag{2.4}$$

Given an initial state s_0, the DTMC evolves over time, that is, step by step, according to *one-step transition probabilities*. The right-hand side of Eq. (2.4) reveals the conditional pmf of transitions from state s_n at time step n to state s_{n+1} at time step $(n+1)$. Without loss of generality, let $S = \{0, 1, 2, \ldots\}$ and write conveniently the following shorthand notation for the conditional pmf of the process's one-step transition from state i to state j at time n:

$$p_{ij}^{(1)}(n) = P(X_{n+1} = s_{n+1} = j \mid X_n = s_n = i). \tag{2.5}$$

In the homogeneous case, when the conditional pmf is independent of epoch n, Eq. (2.5) reduces to:

$$p_{ij}^{(1)} = p_{ij}^{(1)}(n) = P(X_{n+1} = j \mid X_n = i) = P(X_1 = j \mid X_0 = i), \quad \forall n \in T. \tag{2.6}$$

For the sake of convenience, we usually drop the superscript, so that $p_{ij} = p_{ij}^{(1)}$ refers to a one-step transition probability of a homogeneous DTMC.

Starting with state i, the DTMC will go to some state j (including the possibility of $j = i$), so that it follows that $\sum_j p_{ij} = 1$, where $0 \leq p_{ij} \leq 1$. The one-step transition probabilities p_{ij} are usually summarized in a non-negative, stochastic[1] transition matrix $\mathbf{P}$:

$$\mathbf{P} = \mathbf{P}^{(1)} = [p_{ij}] = \begin{pmatrix} p_{00} & p_{01} & p_{02} & \cdots \\ p_{10} & p_{11} & p_{12} & \cdots \\ p_{20} & p_{21} & p_{22} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

Graphically, a finite-state DTMC is represented by a state transition diagram, a finite directed graph, where state i of the chain is depicted by a vertex, and a one-step transition from state i to state j by an edge marked with one-step transition probability p_{ij}. As an example, consider the one-step transition probability matrix in Eq. (2.7) with state space $S = \{0, 1\}$ and the corresponding graphical representation in Fig. 2.1.

[1]The elements in each row of the matrix sum up to 1.

Example 2.1 The one-step transition probability matrix of the two-state DTMC in Fig. 2.1 is given by:

$$\mathbf{P}^{(1)} = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} = \begin{pmatrix} 0.75 & 0.25 \\ 0.5 & 0.5 \end{pmatrix}. \tag{2.7}$$

Conditioned on the current DTMC state, a transition is made from state 0 to state 1 with probability 0.25, and with probability0.75, the DTMC remains in state 0 at the next time step. Correspondingly, a transition occurs from state 1 to state 0 with probability 0.5, and with probability 0.5 the chain remains in state 1 at the next time step.

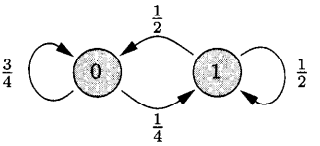


Fig. 2.1 Example of a discrete-time Markov chain referring to Eq. (2.7).

Repeatedly applying one-step transitions generalizes immediately to *n-step transition probabilities*. More precisely, let $p_{ij}^{(n)}(k,l)$ denote the probability that the Markov chain transits from state i at time k to state j at time l in exactly $n = l - k$ steps:

$$p_{ij}^{(n)}(k,l) = P(X_l = j \mid X_k = i), \quad 0 \leq k \leq l. \tag{2.8}$$

Again, the theorem of total probability applies for any given state i and any given time values k and l such that $\sum_j p_{ij}^{(n)}(k,l) = 1$, where $0 \leq p_{ij}^{(n)}(k,l) \leq 1$. This fact, together with the Markov property, immediately leads us to a procedure for computing the n-step transition probabilities recursively from the one-step transition probabilities: The transition of the process from state i at time k to state j at time l can be split into subtransitions from state i at time k to an intermediate state[2] h, say, at time m and from there, *independently of the history* that led to that state, from state h at time m to state j at time l, where $k < m < l$ and $n = l - k$. This condition leads to the well known system of *Chapman-Kolmogorov equations*:

$$p_{ij}^{(n)}(k,l) = \sum_{h \in S} p_{ih}^{(m-k)}(k,m) p_{hj}^{(l-m)}(m,l), \quad 0 \leq k < m < l. \tag{2.9}$$

Note that the conditional independence assumption, i.e., the Markov property, is reflected by the product of terms on the right-hand side of Eq. (2.9).

[2]The Markov chain must simply traverse *some* state at any time.

Similar to the one-step case, the n-step transition probabilities can be simplified for homogeneous DTMC such that $p_{ij}^{(n)} = p_{ij}^{(n)}(k,l)$ depend only on the difference $n = l - k$ and not on the actual values of k and l:

$$p_{ij}^{(n)} = P(X_{k+n} = j \mid X_k = i) = P(X_n = j \mid X_0 = i), \quad \forall k \in T. \tag{2.10}$$

Under this condition, the Chapman-Kolmogorov Eq. (2.9) for *homogeneous* DTMC simplifies to:

$$p_{ij}^{(n)} = \sum_{h \in S} p_{ih}^{(m)} p_{hj}^{(n-m)}, \quad 0 < m < n. \tag{2.11}$$

Because Eq. (2.11) holds for all $m < n$, let $m = 1$ and get

$$p_{ij}^{(n)} = \sum_{h \in S} p_{ih}^{(1)} p_{hj}^{(n-1)}.$$

With $\mathbf{P}^{(n)}$ as the matrix of n-step transition probabilities $p_{ij}^{(n)}$, Eq. (2.11) can be rewritten in matrix form for the particular case of $m = 1$ as $\mathbf{P}^{(n)} = \mathbf{P}^{(1)}\mathbf{P}^{(n-1)} = \mathbf{P}\mathbf{P}^{(n-1)}$. Applying this procedure recursively results in the following equation[3]:

$$\mathbf{P}^{(n)} = \mathbf{P}\mathbf{P}^{(n-1)} = \mathbf{P}^n. \tag{2.12}$$

The n-step transition probability matrix can be computed by the $(n-1)$-fold multiplication of the one-step transition matrix by itself.

Example 2.2 Referring back to the example in Fig. 2.1 and Eq. (2.7), the four-step transition probability matrix, for instance, can be derived according to Eq. (2.12):

$$\begin{aligned}
\mathbf{P}^{(4)} &= \mathbf{P}\mathbf{P}^{(3)} = \mathbf{P}^2\mathbf{P}^{(2)} \\
&= \begin{pmatrix} 0.75 & 0.25 \\ 0.5 & 0.5 \end{pmatrix}^2 \mathbf{P}^{(2)} = \begin{pmatrix} 0.6875 & 0.3125 \\ 0.625 & 0.375 \end{pmatrix} \mathbf{P}\mathbf{P}^{(1)} \\
&= \begin{pmatrix} 0.67188 & 0.32813 \\ 0.65625 & 0.34375 \end{pmatrix} \mathbf{P}^{(1)} = \begin{pmatrix} 0.66797 & 0.33203 \\ 0.66406 & 0.33594 \end{pmatrix}.
\end{aligned} \tag{2.13}$$

Ultimately, we wish to compute the pmf of the random variable X_n, that is, the probabilities $\nu_i(n) = P(X_n = i)$ that the DTMC is in state i, at time step n. These probabilities, called *transient state probabilities* at time n, will then allow us to derive the desired performance measures. Given the n-step transition probability matrix $\mathbf{P}^{(n)}$, the vector of the state probabilities at time

[3]It is important to keep in mind that $\mathbf{P}^{(n)} = \mathbf{P}^n$ holds only for homogeneous DTMC.

n, $\boldsymbol{\nu}(n) = (\nu_0(n), \nu_1(n), \nu_2(n), \ldots)$ can be obtained by unconditioning $\mathbf{P}^{(n)}$ on the *initial probability vector* $\boldsymbol{\nu}(0) = (\nu_0(0), \nu_1(0), \nu_2(0), \ldots)$:

$$\boldsymbol{\nu}(n) = \boldsymbol{\nu}(0)\mathbf{P}^{(n)} = \boldsymbol{\nu}(0)\mathbf{P}^n = \boldsymbol{\nu}(n-1)\mathbf{P}. \tag{2.14}$$

Note that both $\boldsymbol{\nu}(n)$ and $\boldsymbol{\nu}(0)$ are represented as row vectors in Eq. (2.14).

Example 2.3 Assume that the DTMC from Eq. (2.7) under investigation is initiated in state 1, then the initial probability vector $\boldsymbol{\nu}^{(1)}(0)=(0,1)$ is to be applied in the unconditioning according to Eq. (2.14). With the already computed four-step transition probabilities in Eq. (2.13) the corresponding pmf $\boldsymbol{\nu}^{(1)}(4)$ can be derived:

$$\boldsymbol{\nu}^{(1)}(4) = (0,1)\begin{pmatrix} 0.66797 & 0.33203 \\ 0.66406 & 0.33594 \end{pmatrix} = (0.66406, 0.33594)\,.$$

Example 2.4 Alternatively, with another initial probability vector:

$$\boldsymbol{\nu}^{(2)}(0) = \left(\tfrac{2}{3}, \tfrac{1}{3}\right) = (0.6\bar{6}, 0.3\bar{3})\,,$$

according to Eq. (2.14), we would get:

$$\boldsymbol{\nu}^{(2)}(4) = (0.6\bar{6}, 0.3\bar{3})$$

as the probabilities at time step 4.

Of particular importance are homogeneous DTMC on which a so-called *stationary probability vector* can be imposed in a suitable way:

Definition 2.6 State probabilities $\boldsymbol{\nu} = (\nu_0, \nu_1, \ldots \nu_i, \ldots)$ of a discrete-time Markov chain are said to be *stationary*, if any transitions of the underlying DTMC according to the given one-step transition probabilities $\mathbf{P} = [p_{ij}]$ have no effect on these state probabilities, that is, $\nu_j = \sum_{i \in S} \nu_i p_{ij}$ holds for all states $j \in S$. This relation can also be expressed in matrix form:

$$\boldsymbol{\nu} = \boldsymbol{\nu}\mathbf{P}, \quad \sum_{i \in S} \nu_i = 1. \tag{2.15}$$

Note that according to the preceding definition, more than one stationary pmf can exist for a given, unrestricted, DTMC.

Example 2.5 By substituting the one-step transition matrix from Eq. (2.7) in Eq. (2.15), it can easily be checked that:

$$\boldsymbol{\nu}^{(2)} = \left(\tfrac{2}{3}, \tfrac{1}{3}\right) = (0.6\bar{6}, 0.3\bar{3})\,,$$

is a stationary probability vector while $\boldsymbol{\nu}^{(1)} = (0,1)$ is not.

Definition 2.7 For an efficient analysis, we are interested in the *limiting state probabilities* $\tilde{\boldsymbol{\nu}}$ as a particular kind of stationary state probabilities, which are defined by:

$$\tilde{\boldsymbol{\nu}} = \lim_{n\to\infty} \boldsymbol{\nu}(n) = \lim_{n\to\infty} \boldsymbol{\nu}(0)\mathbf{P}^{(n)} = \boldsymbol{\nu}(0) \lim_{n\to\infty} \mathbf{P}^{(n)} = \boldsymbol{\nu}(0)\tilde{\mathbf{P}}. \tag{2.16}$$

Definition 2.8 As $n \to \infty$, we may require both the n-step transition probability matrix $\mathbf{P}^{(n)}$ and the state probability vector $\boldsymbol{\nu}(n)$ to *converge independently* of the initial probability vector $\boldsymbol{\nu}(0)$ to $\tilde{\mathbf{P}}$ and $\tilde{\boldsymbol{\nu}}$, respectively. Also, we may only be interested in the case where the state probabilities $\tilde{\nu}_i > 0, \forall i \in S$, are *strictly positive* and $\sum_i \tilde{\nu}_i = 1$, that is, $\tilde{\boldsymbol{\nu}}$ constitutes a pmf.

If all these restrictions apply to a given probability vector, it is said to be the *unique steady-state probability vector* of the DTMC.

Example 2.6 Returning to Example 2.1, we note that the n-step transition probabilities converge as $n \to \infty$:

$$\tilde{\mathbf{P}} = \lim_{n\to\infty} \mathbf{P}^{(n)} = \lim_{n\to\infty} \begin{pmatrix} 0.75 & 0.25 \\ 0.5 & 0.5 \end{pmatrix}^n = \begin{pmatrix} 0.6\bar{6} & 0.3\bar{3} \\ 0.6\bar{6} & 0.3\bar{3} \end{pmatrix}.$$

Example 2.7 With this result, the limiting state probability vector $\tilde{\boldsymbol{\nu}}$, which is independent of any initial probability vector $\boldsymbol{\nu}(0)$, can be derived according to Eq. (2.16):

$$\tilde{\boldsymbol{\nu}} = (0.6\bar{6}, 0.3\bar{3}).$$

Example 2.8 Since all probabilities in the vector $(0.6\bar{6}, 0.3\bar{3})$ are strictly positive, this vector constitutes the unique steady-state probability vector of the DTMC.

Eventually, the limiting state probabilities become *independent of time steps*, such that once the limiting probability vector is reached, further transitions of the DTMC do not change this vector, i.e., it is stationary. Note that such a probability vector does not necessarily exist for all DTMCs.

If Eq. (2.16) holds and $\tilde{\boldsymbol{\nu}}$ is independent of $\boldsymbol{\nu}(0)$, it follows that the limit $\mathbf{P}^{(n)} = [p_{ij}^{(n)}]$ is independent of time n and of index i. All rows of $\tilde{\mathbf{P}}$ would be identical, that is, the rows would match element by element. Furthermore, the jth element $\tilde{p}_{ij}$ of row i equals $\tilde{\nu}_j$ for all $i \in S$:

$$\tilde{\mathbf{P}} = \lim_{n\to\infty} \mathbf{P}^{(n)} = \lim_{n\to\infty} \mathbf{P}^n = \begin{pmatrix} \tilde{\nu}_0 & \tilde{\nu}_1 & \tilde{\nu}_2 & \cdots \\ \tilde{\nu}_0 & \tilde{\nu}_1 & \tilde{\nu}_2 & \cdots \\ \tilde{\nu}_0 & \tilde{\nu}_1 & \tilde{\nu}_2 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{2.17}$$

If the unique steady-state probability vector of a DTMC *exists*, it can be determined by the solution of the system of linear Eqs. (2.15), so that $\tilde{\mathbf{P}}$ need

not be determined explicitly. From Eq. (2.14), we have $\boldsymbol{\nu}(n) = \boldsymbol{\nu}(n-1)\mathbf{P}$. If the limit exists, we can take it on both sides of the equation and get:

$$\lim_{n\to\infty} \boldsymbol{\nu}(n) = \tilde{\boldsymbol{\nu}} = \lim_{n\to\infty} \boldsymbol{\nu}(n-1)\mathbf{P} = \tilde{\boldsymbol{\nu}}\mathbf{P}. \tag{2.18}$$

In the steady-state case no ambiguity can arise, so that, for the sake of convenience, we may drop the annotation and refer to steady-state probability vector by using the notation $\boldsymbol{\nu}$ instead of $\tilde{\boldsymbol{\nu}}$. Steady-state and stationarity coincide in that case, i.e., there is only a unique stationary probability vector.

The computation of the steady-state probability vector $\boldsymbol{\nu}$ of a DTMC is usually significantly simpler and less expensive than a time-dependent computation of $\boldsymbol{\nu}(n)$. It is therefore the steady-state probability vector of a DTMC that is preferably taken advantage of in modeling endeavors. But a steady-state probability vector does not exist for all DTMCs.[4] Additionally, it is not always appropriate to restrict the analysis to the steady-state case, even if it does exist. Under some circumstances time-dependent, i.e., transient, analysis would result in more meaningful information with respect to an application. Transient analysis has special relevance if short-term behavior is of more importance than long-term behavior. In modeling terms, "short term" means that the influence of the initial state probability vector $\boldsymbol{\nu}(0)$ on $\boldsymbol{\nu}(n)$ has not yet disappeared by time step n.

Before continuing, some simple example DTMCs are presented to clarify the definitions of this section. The following four one-step transition matrices are examined for the conditions under which stationary, limiting state, and steady-state probabilities exist for them.

Example 2.9 Consider the DTMC shown in Fig. 2.2 with the one-step transition probability matrix (TPM):

$$\mathbf{P} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.19}$$

Fig. 2.2 Example of a discrete-time Markov chain referring to Eq. (2.19).

- For this one-step TPM, an infinite number of stationary probability vectors exists: Any arbitrary probability vector is stationary in this case according to Eq. (2.15).

[4]The conditions under which DTMCs converge to steady-state is precisely stated in the following section.

- The n-step TPM $\mathbf{P}^{(n)}$ converges in the limit to:

$$\lim_{n\to\infty} \mathbf{P}^{(n)} = \tilde{\mathbf{P}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

  Furthermore, all n-step TPMs are identical:

$$\mathbf{P} = \tilde{\mathbf{P}} = \mathbf{P}^{(n)}, \quad \forall n \in T.$$

  The limiting state probabilities $\tilde{\boldsymbol{\nu}}$ do exist and are identical to the initial probability vector $\boldsymbol{\nu}(0)$ in all cases:

$$\tilde{\boldsymbol{\nu}} = \boldsymbol{\nu}(0)\tilde{\mathbf{P}} = \boldsymbol{\nu}(0).$$

- A unique steady-state probability vector does not exist for this example.

Example 2.10 Next, consider the DTMC shown in Fig. 2.3 with the TPM:

$$\mathbf{P} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{2.20}$$

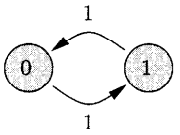


Fig. 2.3 Example of a discrete-time Markov chain referring to Eq. (2.20).

- For this one-step TPM, a stationary probability vector, which is unique in this case, does exist according to Eq. (2.15):

$$\boldsymbol{\nu} = (0.5, 0.5).$$

- The n-step transition matrix $\mathbf{P}^{(n)}$ does not converge in the limit to any $\tilde{\mathbf{P}}$. Therefore, the limiting state probabilities $\tilde{\boldsymbol{\nu}}$ do not exist.
- Consequently, a unique steady-state probability vector does not exist.

Example 2.11 Consider the DTMC shown in Fig. 2.4 with the TPM:

$$\mathbf{P} = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}. \tag{2.21}$$

- For this one-step TPM a unique stationary probability vector does exist according to Eq. (2.15):

$$\boldsymbol{\nu} = (0.5, 0.5).$$

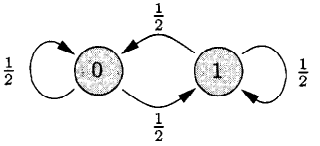


Fig. 2.4 Example of a discrete-time Markov chain referring to Eq. (2.21).

Note that this is the same unique stationary probability vector as for the different DTMC in Eq. (2.20).

- The n-step TPM $\mathbf{P}^{(n)}$ converges in the limit to:

$$\lim_{n\to\infty} \mathbf{P}^{(n)} = \tilde{\mathbf{P}} = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}.$$

Furthermore, all n-step TPMs are identical:

$$\mathbf{P} = \tilde{\mathbf{P}} = \mathbf{P}^{(n)}, \quad \forall n \in T.$$

The limiting state probabilities $\tilde{\boldsymbol{\nu}}$ do exist, are independent of the initial probability vector, and are unique:

$$\tilde{\boldsymbol{\nu}} = \boldsymbol{\nu}(0)\tilde{\mathbf{P}} = (0.5, 0.5)\,.$$

- A unique steady-state probability vector does exist. All probabilities are strictly positive and identical to the stationary probabilities, which can be derived from the solution of Eq. (2.15):

$$\boldsymbol{\nu} = \tilde{\boldsymbol{\nu}} = (0.5, 0.5)\,.$$

Example 2.12 Now consider the DTMC shown in Fig. 2.5 with the TPM:

$$\mathbf{P} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}. \tag{2.22}$$

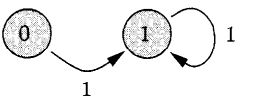


Fig. 2.5 Example of a discrete-time Markov chain referring to Eq. (2.22).

- For this one-step TPM a unique stationary probability vector does exist according to Eq. (2.15):

$$\boldsymbol{\nu} = (0, 1)\,.$$

- The n-step TPM $\mathbf{P}^{(n)}$ converges in the limit to

$$\tilde{\mathbf{P}} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}.$$

Furthermore, all n-step TPMs are identical:

$$\mathbf{P} = \tilde{\mathbf{P}} = \mathbf{P}^{(n)}, \quad \forall n \in T.$$

The limiting state probability vector $\tilde{\boldsymbol{\nu}}$ does exist, is independent of the initial probability vector, and is identical to the unique stationary probability vector $\boldsymbol{\nu}$:

$$\tilde{\boldsymbol{\nu}} = \boldsymbol{\nu} = (0, 1).$$

- A unique steady-state probability vector does not exist for this example. The elements of the unique stationary probability vector are not strictly positive.

We proceed now to identify necessary and sufficient conditions for the existence of a steady-state probability vector of a DTMC. The conditions can be given immediately in terms of properties of the DTMC.

2.1.2.1.1 Classifications of DTMC DTMCs are categorized based on the classifications of their constituent states.

Definition 2.9 Any state j is said to be *reachable* from any other state i, where $i, j \in S$, if it is possible to transit from state i to state j in a finite number of steps according to the given transition probability matrix. For some integer $n \geq 1$, the following relation must hold for the n-step transition probability:

$$p_{ij}^{(n)} > 0, \quad \exists n, n \geq 1. \tag{2.23}$$

A DTMC is called *irreducible* if all states in the chain can be reached pairwise from each other, i.e., $\forall i, j \in S, \exists n, n \geq 1 : p_{ij}^{(n)} > 0$.

A state $i \in S$ is said to be an *absorbing state*[5] if and only if no other state of the DTMC can be reached from it, i.e., $p_{ii} = 1$.

Note that a DTMC containing at least one absorbing state cannot be irreducible. If countably infinite state models are encountered, we have to discriminate more accurately how states are reachable from each other. The recurrence time and the probability of recurrence must also be taken into account.

[5]Absorbing states play an important role in the modeling of dependable systems where transient analysis is of primary interest.

Definition 2.10 Let $f_i^{(n)}$, called the *n-step recurrence probability*, denote the conditional probability of the first return to state $i \in S$ in exactly $n \geq 1$ steps after leaving state i. Then, the probability f_i of ever returning to state i is given by:

$$f_i = \sum_{n=1}^{\infty} f_i^{(n)}. \tag{2.24}$$

Any state $i \in S$ to which the DTMC will return with probability $f_i = 1$, is called a *recurrent state*; otherwise, if $f_i < 1$, i is called a *transient state.*

Given a recurrent state i, the *mean recurrence time* m_i of state i of a DTMC is given by:

$$m_i = \sum_{n=1}^{\infty} n f_i^{(n)}. \tag{2.25}$$

If the mean recurrence time is finite, that is, $m_i < \infty$, i is called *positive recurrent* or *recurrent non-null*; otherwise, if $m_i = \infty$, state i is said to be *recurrent null.* For any recurrent state $i \in S$, let d_i denote the *period* of state i, then d_i is the greatest common divisor of the set of positive integers n such that $p_{ii}^{(n)} > 0$. A recurrent state i is called *aperiodic* if its period $d_i = 1$, and *periodic* with period d_i if $d_i > 1$.

It has been shown by Feller [Fell68] that the states of an *irreducible* DTMC are all of the same type. Hence, all states are periodic, aperiodic, transient, recurrent null, or recurrent non-null.

Definition 2.11 If one of the states i of an irreducible DTMC is aperiodic then so are all the other states $j \in S$, that is, $d_j = 1, \forall j \in S$, and the DTMC itself is called *aperiodic*; otherwise it is said to be *periodic* with unique period d.
An irreducible, aperiodic, discrete-time Markov chain with all states i being recurrent non-null with finite mean recurrence time m_i is called an *ergodic* Markov chain.

We are now ready to summarize the main results for the classification of discrete-time Markov chains:

- The states of a *finite-state, irreducible* Markov chain are all recurrent non-null.
- Given an *aperiodic* DTMC, the limits $\tilde{\boldsymbol{\nu}} = \lim_{n\to\infty} \boldsymbol{\nu}(n)$ do exist.
- For any *irreducible* and *aperiodic* DTMC, the limit $\tilde{\boldsymbol{\nu}}$ exists and is independent of the initial probability vector $\boldsymbol{\nu}(0)$.
- For an *ergodic* DTMC, the limit $\tilde{\boldsymbol{\nu}} = (\tilde{\nu}_0, \tilde{\nu}_1, \tilde{\nu}_2, \ldots)$ exists and comprises the unique steady-state probability vector $\boldsymbol{\nu}$.

- The steady-state probabilities $\nu_i > 0$, $i \in S$, of an *ergodic* Markov chain can be obtained by solving the system of linear Eq. (2.15) or, if the (finite) mean recurrence times m_i are known, by exploiting the relation:

$$\nu_i = \frac{1}{m_i}, \quad \forall i \in S. \tag{2.26}$$

If *finite-state* DTMCs are investigated, the solution of Eq. (2.15) can be obtained by applying standard methods for the solution of linear systems. In the *infinite* case, either *generating functions* can be applied or special *structure* of the one-step transition probability matrix $\mathbf{P}^{(1)} = \mathbf{P}$ may be exploited to find the solution in closed form. An example of the latter technique is given in Section 3.1 where we investigate the important class of birth-death processes. The special (tridiagonal) structure of the matrix will allow us to derive closed-form solutions for the state probabilities that are not restricted to any fixed matrix size so that the limiting state probabilities of infinite state DTMCs are captured by the closed-form formulae as well.

2.1.2.1.2 DTMC State Sojourn Times The state *sojourn times* – the time between state changes – play an important role in the characterization of DTMCs. Only homogeneous DTMCs are considered here. We have already pointed out that the transition behavior reflects the memoryless property, that is, it only depends on the current state and neither on the history that led to the state nor on the time already spent in the current state. At every instant of time, the probability of leaving current state i is independently given by $(1 - p_{ii}) = \sum_{i \neq j} p_{ij}$. Applying this repeatedly leads to a description of a random experiment in form of a sequence of Bernoulli trials with probability of success $(1 - p_{ii})$, where "success" denotes the event of leaving current state i. Hence, the *sojourn time* R_i during a single visit to state i is a geometrically distributed random variable[6] with pmf:

$$P(R_i = k) = (1 - p_{ii}) p_{ii}^{k-1}, \quad \forall k \in \mathbb{N}^+. \tag{2.27}$$

We can therefore immediately conclude that the expected sojourn time $E[R_i]$, that is, the mean number of time steps the process spends in state i per visit, is:

$$E[R_i] = \frac{1}{1 - p_{ii}}. \tag{2.28}$$

Accordingly, the variance $\mathrm{var}[R_i]$ of the sojourn time per visit in state i is given by:

$$\mathrm{var}[R_i] = \frac{p_{ii}}{(1 - p_{ii})^2}. \tag{2.29}$$

[6]Note that the geometric distribution is the discrete-time equivalent of the exponential distribution, i.e., it is the only discrete distribution with the memoryless property.

2.1.2.2 Continuous-Time Markov Chains Continuous- and discrete-time Markov chains provide different yet related modeling paradigms, each of them having their own domain of applications. For the definition of CTMCs we refer back to the definition of general Markov processes in Eq. (2.2) and specialize it to the continuous parameter, discrete state space case. CTMCs are distinct from DTMCs in the sense that state transitions may occur at arbitrary instants of time and not merely at fixed, discrete time points, as is the case with DTMCs. Therefore, we use a subset of the set of non-negative real numbers $\mathbb{R}_0^+$ to refer to the parameter set T of a CTMC, as opposed to $\mathbb{N}_0$ for DTMCs:

Definition 2.12 A given stochastic process $\{X_t : t \in T\}$ constitutes a CTMC if for arbitrary $t_i \in \mathbb{R}_0^+$, with $0 = t_0 < t_1 < \cdots < t_n < t_{n+1}, \forall n \in \mathbb{N}$, and $\forall s_i \in S = \mathbb{N}_0$ for the conditional pmf, the following relation holds:

$$\begin{aligned} &P(X_{t_{n+1}} = s_{n+1} \mid X_{t_n} = s_n, X_{t_{n-1}} = s_{n-1}, \ldots, X_{t_0} = s_0) \\ &\quad = P(X_{t_{n+1}} = s_{n+1} \mid X_{t_n} = s_n). \end{aligned} \tag{2.30}$$

Similar to Eq. (2.4) for DTMCs, Eq. (2.30) expresses the *Markov* property of continuous-time Markov chains. If we further impose homogeneity, then because the exponential distribution is the only continuous-time distribution that provides the memoryless property, the state sojourn times of a CTMC are necessarily exponentially distributed.

Again, the right-hand side of Eq. (2.30) is referred to as the *transition probability*[7] $p_{ij}(u, v)$ of the CTMC to travel from state i to state j during the period of time $[u, v)$, with $u, v \in T$ and $u \leq v$:

$$p_{ij}(u, v) = P(X_v = j \mid X_u = i). \tag{2.31}$$

For $u = v$ we define:

$$p_{ij}(u, u) = \begin{cases} 1, & i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{2.32}$$

If the transition probabilities $p_{ij}(u, v)$ depend only on the time difference $t = v - u$ and not on the actual values of u and v, the simplified transition probabilities for *time-homogeneous* CTMC result:

$$p_{ij}(t) = p_{ij}(0, t) = P(X_{u+t} = j \mid X_u = i) = P(X_t = j \mid X_0 = i), \quad \forall u \in T. \tag{2.33}$$

Given the transition probabilities $p_{ij}(u, v)$ and the probabilities $\pi_i(u)$ of the CTMC at time u, the unconditional state probabilities $\pi_j(v), j \in S$ of the

[7]Note that, as opposed to the discrete-time case, there is no fixed, discrete number of transition steps considered here.

process at time v can be derived:

$$\pi_j(v) = \sum_{i \in S} p_{ij}(u,v)\pi_i(u), \quad \forall u, v \in T \ (u \leq v). \tag{2.34}$$

With $\mathbf{P}(u,v) = [p_{ij}(u,v)]$ as the matrix of the transition probabilities, for any pair of states $i, j \in S$ and any time interval $[u,v)$, $u, v \in T$, from the parameter domain, and the vector $\boldsymbol{\pi}(u) = (\pi_0(u), \pi_1(u), \pi_2(u), \ldots)$ of the state probabilities at any instant of time u, Eq. (2.34) can be given in vector-matrix form:

$$\boldsymbol{\pi}(v) = \boldsymbol{\pi}(u)\mathbf{P}(u,v), \quad \forall u, v \in T \ (u \leq v). \tag{2.35}$$

Note that for all $u \in T$, $\mathbf{P}(u,u) = \mathbf{I}$ is the identity matrix.

In the time-homogeneous case, Eq. (2.34) reduces to:

$$\pi_j(t) = \sum_{i \in S} p_{ij}(t)\pi_i(0) = \sum_{i \in S} p_{ij}(0,t)\pi_i(0), \tag{2.36}$$

or in vector-matrix notation:

$$\boldsymbol{\pi}(t) = \boldsymbol{\pi}(0)\mathbf{P}(t) = \boldsymbol{\pi}(0)\mathbf{P}(0,t). \tag{2.37}$$

Similar to the discrete-time case (Eq. (2.9)), the *Chapman-Kolmogorov* Eq. (2.38) for the transition probabilities of a CTMC can be derived from Eq. (2.30) by applying again the theorem of total probability:

$$p_{ij}(u,v) = \sum_{k \in S} p_{ik}(u,w)p_{kj}(w,v), \quad 0 \leq u \leq w < v. \tag{2.38}$$

But, *unlike* the discrete time case, Eq. (2.38) cannot be solved easily and used directly for computing the state probabilities. Rather, it has to be transformed into a system of differential equations which, in turn, leads us to the required results. For this purpose, we define the instantaneous *transition rates* $q_{ij}(t)$ $(i \neq j)$ of the CTMC traveling from state i to state j. These transition rates are related to conditional transition probabilities. Consider the period of time $[t, t+\Delta t)$, where Δt is chosen such that $\sum_{j \in S} q_{ij}(t)\Delta t + o(\Delta t) = 1$[8]. The non-negative, finite, continuous functions $q_{ij}(t)$ can be shown to exist under rather general conditions. For all states i, j, $i \neq j$, we define:

$$q_{ij}(t) = \lim_{\Delta t \to 0} \frac{p_{ij}(t, t+\Delta t)}{\Delta t}, \quad i \neq j, \tag{2.39}$$

$$q_{ii}(t) = \lim_{\Delta t \to 0} \frac{p_{ii}(t, t+\Delta t) - 1}{\Delta t}. \tag{2.40}$$

[8]The notation $o(\Delta t)$ is defined such that $\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} = 0$; that is, we might substitute any function for $o(\Delta t)$ that approaches zero faster than the linear function Δt.

If the limits do exist, it is clear from Eqs. (2.39) and (2.40) that, since $\sum_{j\in S} p_{ij}(t, t+\Delta t) = 1$, at any instant of time t:

$$\sum_{j\in S} q_{ij}(t) = 0, \quad \forall i \in S. \tag{2.41}$$

The quantity $-q_{ii}(t)$ can be interpreted as the total rate at which state i is exited (to any other state) at time t. Accordingly, $q_{ij}(t)$, $(i \neq j)$, denotes the rate at which the CTMC leaves state i in order to transit to state j at time t. As an equivalent interpretation, we can regard $q_{ij}(t)\Delta t + o(\Delta t)$ as the transition probability $p_{ij}(t, t+\Delta t)$ of the Markov chain to transit from state i to state j in $[t, t+\Delta t)$, where Δt is chosen appropriately. Having these definitions, we return to the Chapman-Kolmogorov Eq. (2.38). Substituting $v + \Delta t$ for v in (2.38) and subtracting both sides of the original Eq. (2.38) from the result gives us:

$$p_{ij}(u, v+\Delta t) - p_{ij}(u, v) = \sum_{k\in S} p_{ik}(u, w)[p_{kj}(w, v+\Delta t) - p_{kj}(w, v)]. \tag{2.42}$$

Dividing both sides of Eq. (2.42) by Δt, taking $\lim_{\Delta t\to 0}$ of the resulting quotient of differences, and letting $w \to v$, we derive a differential equation, the well known *Kolmogorov's forward equation*:

$$\frac{\partial p_{ij}(u, v)}{\partial v} = \sum_{k\in S} p_{ik}(u, v) q_{kj}(v), \quad 0 \leq u < v. \tag{2.43}$$

In the homogeneous case, we let $t = v - u$ and get from Eqs. (2.39) and (2.40) *time-independent* transition rates $q_{ij} = q_{ij}(t), \forall i, j \in S$, such that simpler versions of the Kolmogorov's forward differential equation for homogeneous CTMCs result:

$$\frac{dp_{ij}(t)}{dt} = \sum_{k\in S} p_{ik}(t) q_{kj} = \sum_{k\in S} p_{ik}(0, t) q_{kj}. \tag{2.44}$$

Instead of the forward Eq. (2.43), we can equivalently derive and use the *Kolmogorov's backward equation* for further computations, both in the homogeneous and non-homogeneous cases, by letting $w \to u$ in Eq. (2.42) and taking $\lim_{\Delta t\to 0}$ to get:

$$\frac{\partial p_{ij}(u, v)}{\partial u} = \sum_{k\in S} p_{kj}(u, v) q_{ik}(u), \quad 0 \leq u < v. \tag{2.45}$$

Differentiating Eq. (2.34) on both sides gives Eq. (2.47), using the Kolmogorov's (forward) Eq. (2.43) yields Eq. (2.48), and then, again, applying Eq. (2.34) to Eq. (2.49), we derive the differential equation for the *uncondi-*

tional state probabilities $\pi_j(v)$, $\forall j \in S$, at time v in Eq. (2.50):

$$\frac{d\pi_j(v)}{dv} = \frac{\partial \sum_{i\in S} p_{ij}(u,v)\pi_i(u)}{\partial v} \tag{2.46}$$

$$= \sum_{i\in S} \frac{\partial p_{ij}(u,v)}{\partial v}\pi_i(u) \tag{2.47}$$

$$= \sum_{i\in S} \left(\sum_{k\in S} p_{ik}(u,v)q_{kj}(v) \right) \pi_i(u) \tag{2.48}$$

$$= \sum_{k\in S} q_{kj}(v) \sum_{i\in S} p_{ik}(u,v)\pi_i(u) \tag{2.49}$$

$$= \sum_{k\in S} q_{kj}(v)\pi_k(v). \tag{2.50}$$

In the time-homogeneous case, a simpler version of Eq. (2.50) results by assuming $t = v - u$ and using time-independent transition rates q_{ij}. So we get the system of differential Eqs. (2.51):

$$\frac{d\pi_j(t)}{dt} = \sum_{i\in S} q_{ij}\pi_i(t), \quad \forall j \in S, \tag{2.51}$$

which is repeatedly used throughout this text. Usually, we prefer vector-matrix form rather than the notation used in Eq. (2.51). Therefore, for the homogeneous case, we define the *infinitesimal generator matrix* $\mathbf{Q}$ of the transition probability matrix $\mathbf{P}(t) = [p_{ij}(0,t)] = [p_{ij}(t)]$ by referring to Eqs. (2.39) and (2.40). The matrix $\mathbf{Q}$:

$$\mathbf{Q} = [q_{ij}], \quad \forall i,j \in S, \tag{2.52}$$

contains the transition rates q_{ij} from any state i to any other state j, where $i \neq j$, of a given continuous-time Markov chain. The elements q_{ii} on the main diagonal of $\mathbf{Q}$ are defined by $q_{ii} = -\sum_{j,j\neq i} q_{ij}$. With the definition in Eq. (2.52), Eq. (2.51) can be given in vector-matrix form as:

$$\dot{\boldsymbol{\pi}}(t) = \frac{d\boldsymbol{\pi}(t)}{dt} = \boldsymbol{\pi}(t)\mathbf{Q}. \tag{2.53}$$

For the sake of completeness, we include also the matrix form of the Kolmogorov differential equations in the time-homogeneous case. The Kolmogorov's *forward* Eq. (2.44) can be written as:

$$\dot{\mathbf{P}}(t) = \frac{d\mathbf{P}(t)}{dt} = \mathbf{P}(t)\mathbf{Q}. \tag{2.54}$$

The Kolmogorov's *backward* equation in the homogeneous case results in matrix form as:

$$\dot{\mathbf{P}}(t) = \frac{d\mathbf{P}(t)}{dt} = \mathbf{Q}\mathbf{P}(t). \tag{2.55}$$

As in the discrete-time case, often the *steady-state probability vector* of a CTMC is of primary interest. The required properties of the steady-state probability vector, which is also called the *equilibrium probability vector*, are equivalent to the discrete time case. For all states $i \in S$, the steady-state probabilities π_i are:

1. Independent of time t
2. Independent of the initial state probability vector $\boldsymbol{\pi}(0)$
3. Strictly positive, $\pi_i > 0$
4. Given as the time limits, $\pi_i = \lim_{t\to\infty} \pi_i(t) = \lim_{t\to\infty} p_{ji}(t)$, of the state probabilities $\pi_i(t)$ and of the transition probabilities $p_{ji}(t)$, respectively

If existing for a given CTMC, the steady-state probabilities are independent of time, we immediately get:

$$\lim_{t\to\infty} \frac{d\boldsymbol{\pi}(t)}{dt} = 0. \tag{2.56}$$

Under Condition (2.56), the differential Eq. (2.51) for determining the unconditional state probabilities resolves to a much simpler *system of linear equations*:

$$0 = \sum_{i\in S} q_{ij}\pi_i, \quad \forall j \in S. \tag{2.57}$$

In vector-matrix form, we get accordingly:

$$\mathbf{0} = \boldsymbol{\pi}\mathbf{Q}. \tag{2.58}$$

Definition 2.13 In analogy to the discrete-time case, we call a CTMC for which a unique steady-state probability vector exists, an *ergodic* CTMC.

The strictly positive steady-state probabilities can be gained by the unique solution of Eq. (2.58), when an additional normalization condition is imposed[9]. To express it in vector form, we introduce the *unit vector* $\mathbf{1} = [1, 1, \ldots, 1]^T$ so that the following relation holds:

$$\boldsymbol{\pi}\mathbf{1} = \sum_{i\in S} \pi_i = 1. \tag{2.59}$$

Another possibility for determining the steady-state probabilities π_i for all states $i \in S$ of a CTMC, is to take advantage of a well-known relation

[9]Note that besides the trivial solution $\pi_i = 0$, $\forall i \in S$, any vector, obtained by multiplying a solution of Eq. (2.58) by an arbitrary real-valued constant, would also yield a solution of Eq. (2.58).

between the π_i and the *mean recurrence time* [10] $M_i < \infty$, that is, the mean time elapsed between two successive visits of the CTMC to state i:

$$\pi_i = -\frac{1}{M_i q_{ii}}, \quad \forall i \in S. \tag{2.60}$$

In the time-homogeneous case, we can derive from Eq. (2.41) that for any $j \in S$ $q_{jj} = -\sum_{i,i\neq j} q_{ji}$, and from Eq. (2.57) we get $\sum_{i,i\neq j} q_{ij}\pi_i = -q_{jj}\pi_j$. Putting these together immediately yields the system of *global balance equations*:

$$\sum_{i,i\neq j} q_{ij}\pi_i = \pi_j \sum_{i,i\neq j} q_{ji}, \quad \forall j \in S. \tag{2.61}$$

On the left-hand side of Eq. (2.61), the *total flow* from any other state $i \in S$ *into* state j is captured. On the right-hand side, the *total flow out* of state j into any other state i is summarized. The flows are *balanced* in steady state, i.e., they are in equilibrium.

The conditions under which a CTMC is called ergodic are similar to those for a DTMC. Therefore, we can briefly summarize the criteria for classifying CTMCs and for characterizing their states.

2.1.2.2.1 Classifications of CTMC

Definition 2.14 As for DTMCs, we call a CTMC *irreducible* if every state i is *reachable* from every other state j, where $i, j \in S$; that is, $\forall i, j, i \neq j, \exists t : p_{ji}(t) > 0$. In other words, no proper subset $\hat{S} \subset S$, $\hat{S} \neq S$, of state space S exists, such that $\sum_{j\in\hat{S}} \sum_{i\in S\setminus\hat{S}} q_{ji} = 0$.

Definition 2.15 An irreducible, homogeneous CTMC is called *ergodic* if and only if the unique steady-state probability vector $\boldsymbol{\pi}$ exists.

As opposed to DTMCs, CTMCs cannot be periodic. Therefore, it can be shown that for an irreducible, homogeneous CTMC:

- The limits $\tilde{\pi}_i = \lim_{t\to\infty} \pi_i(t) = lim_{t\to\infty} p_{ji}(t)$ exist $\forall i, j \in S$ and are independent of the initial probability vector $\boldsymbol{\pi}(0)$. [11]
- The steady-state probability vector $\boldsymbol{\pi}$, if existing, can be uniquely determined by the solution of the linear system of Eq. (2.58) constrained by normalization Condition (2.59).
- A unique steady-state, or equilibrium, probability vector $\boldsymbol{\pi}$ exists, if the irreducible, homogeneous CTMC is finite.
- The mean recurrence times M_i are finite for all states $i \in S$, $M_i < \infty$, if the steady-state probability vector exists.

[10] In contrast to DTMCs, where lowercase notation is used to refer to recurrence time, uppercase notation is used for CTMCs.

[11] The limits do not necessarily constitute a steady-state probability vector.

2.1.2.2.2 CTMC State Sojourn Times We have already mentioned that the distribution of the state *sojourn times* of a homogeneous CTMC must have the memoryless property. Since the exponential distribution is the only continuous distribution with this property, the random variables denoting the sojourn times, or holding times, must be exponentially distributed. Note that the same is true for the random variable referred to as the *residual* state holding time, that is, the time remaining until the next state change occurs. [12] Furthermore, the means of the two random variables are equal to $1/(-q_{ii})$.

Let the random variable R_i denote either the sojourn time or the residual time in state i, then the CDF is given by:

$$F_{Ri}(r) = 1 - e^{q_{ii}r}, \quad r \geq 0. \tag{2.62}$$

The mean value of R_i, the mean sojourn time or the mean residual time, is given by:

$$E[R_i] = -\frac{1}{q_{ii}}, \tag{2.63}$$

where q_{ii} is defined in Eq. (2.40).

2.1.2.3 Recapitulation We have introduced Markov chains and indicated their modeling power. The most important feature of homogeneous Markov chains is their unique *memoryless* property that makes them remarkably attractive. Both continuous- and discrete-time Markov chains have been defined and their properties discussed.

The most important algorithms for computation of their state probabilities are discussed in following chapters. Different types of algorithms are related to different categories of Markov chains such as ergodic, absorbing, finite, or infinite chains. Furthermore, the algorithms can be divided into those applicable for computing the steady-state probabilities and those applicable for computing the time-dependent state probabilities. Others provide approximate solutions, often based on an implicit transformation of the state space. Typically, these methods fall into the categories of aggregation/disaggregation techniques. Note that this modeling approximation has to be discriminated from the mathematical properties of the core algorithms, which, in turn, can be numerically exact or approximate, independent of their relation to the underlying model. Typical examples include round-off errors in direct methods such as Gaussian elimination and convergence errors in iterative methods for the solution of linear systems.

[12] The residual time is often referred to as the forward recurrence time.

2.2 THE MODELING PROCESS

So far, we have limited our discussion to the mathematical foundations and properties of DTMCs and CTMCs. Since we follow an application-oriented approach, it is worthwhile to provide explicit guidelines on how to use the theoretical concepts in a practical manner. DTMCs and CTMCs are extensively used for performance and dependability analysis of computer systems, and the same fundamental techniques can be employed in different contexts. But the context strongly determines the kind of information that is meaningful in a concrete setting. Note that the context is not simply given by a plain technical system or a given configuration that is to be modeled. Additional requirements of desired qualities are explicitly or implicitly specified that need to be taken into account.

As an illustrative example, consider the outage problem of computer systems. For a given configuration, there is no ideal way to represent it without considering the application context and defining respective goals of analyses, which naturally ought to be reflected in the model structure. In a real-time context, such as flight control, even the shortest outage might have catastrophic implications for the system being controlled. Therefore, an appropriate model must be very sensitive to such a (hopefully) rare event of short duration. In contrast, the total number of jobs processed or the work accomplished during flight time is probably a less important performance measure for such a safety-critical system. If we look at a transaction processing system, however, short outages are less significant for the success but throughput is of predominant importance. So it is not useful to represent effects of short interruptions, which are of less importance in these circumstances.

There are cases where the right choice of measures is less obvious. Therefore, guidelines are equally useful for the practitioner and the beginner. Later in this chapter, a framework based on *Markov reward models* (MRMs) is presented providing recipes for a selection of the right model type and the definition of an appropriate performance measure. But before going into details of model specification, the modeling process is sketched from a global view point, thereby identifying the major phases in the modeling life-cycle and the interdependencies between these phases.

2.2.1 Modeling Life-cycle Phases

There can be many reasons for modeling a given system. Two of the important ones are:

1. Existing systems are modeled for a better understanding, for analyses of deficiencies such as identification of potential bottlenecks, or for upgrading studies.

2. Models are used during the design of future systems in order to check whether requirements are met.

Different levels of detail are commonly entailed in the two cases. Since for projection and design support fewer details are known in advance, the models tend to be more abstract so that analytical/numerical techniques are more appropriate in these instances.

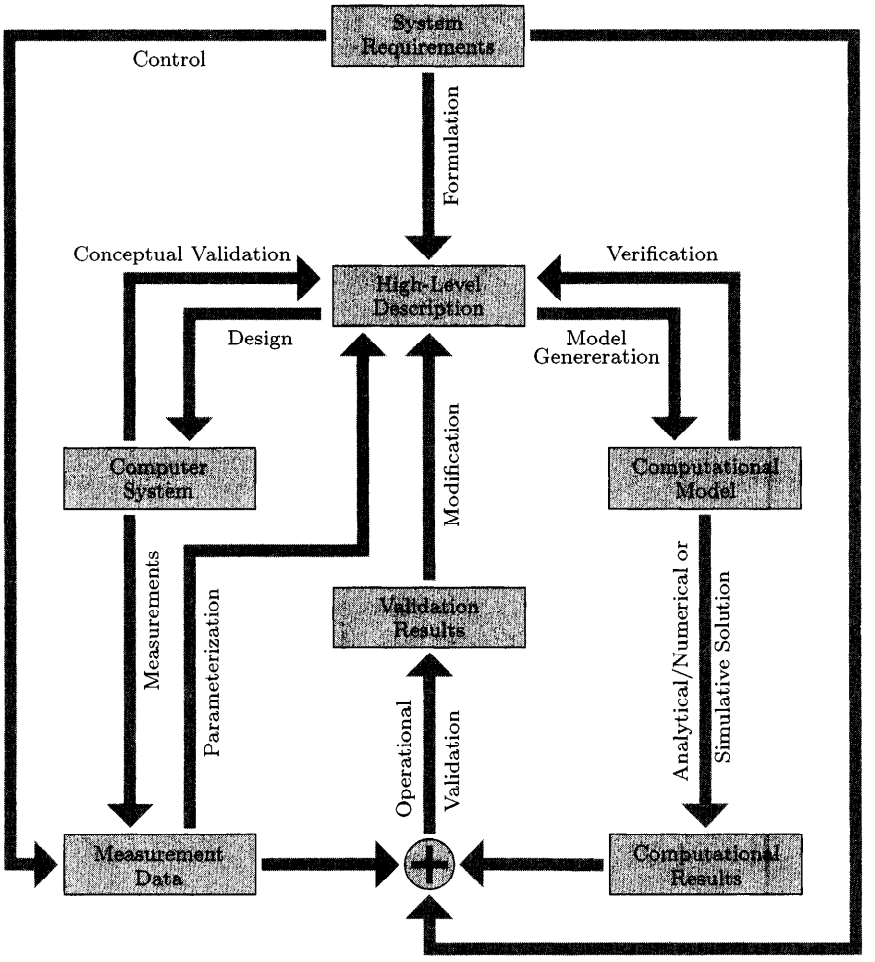


Fig. 2.6 A simplified view of the modeling process.

In general, the modeling process can be structured into phases as depicted in Fig. 2.6. First, there are certain *requirements* to be met by the application; these have an impact on what type of model is *used*. At this point we use the term "requirements" in a very broad sense and use it to refer to everything related to the evaluation of accomplishment levels, be it task or system oriented. As an example, consider the earlier discussion on the significance of outage phases for the expressiveness of the model, which cannot be

determined a priori without reference to an application context. More details related to this issue are discussed in the following sections.

A *high-level description* of the model is the first step to be accomplished. Either information about a real *computer system* is used to build the model, or experiences gained in earlier modeling studies are implicitly used. Of course, this process is rather complicated and needs both modeling and system application-specific expertise. *Conceptual validation* of the correctness of the high-level model is accomplished in an iterative process of step-wise refinement, which is not represented in full detail in Fig. 2.6. Conceptual validation can be further refined [NaFi67] into "face validation," where the involved experts try to reach consensus on the appropriateness of the model on the basis of dialogs, and into the "validation of the model assumptions," where implicit or explicit assumptions are cross-checked. Some of the crucial properties of the model are checked by answering the following questions [HMT91]:

- Is the model logically correct, complete, or overly detailed?
- Are the distributional assumptions justified? How sensitive are the results to simplifications in the distributional assumptions?
- Are other stochastic properties, such as independence assumptions, valid?
- Is the model represented on the appropriate level? Are those features included that are most significant for the application context?

Note that a *valid* model is not necessarily complete in every respect, but it is one that includes the most *significant* effects in relation to the system requirements. It leaves higher-order effects out or simplifies them drastically. In order to produce a useful model, leaving out may be of the same importance as the validity of underlying stochastic assumptions.

Once confidence is gained in the validity of the high-level model description, the next step is to generate a *computational model.* Sometimes the available solution techniques are inadequate for solving the high-level model. Some of these difficulties arise from:

- Stochastic dependencies and correlated processes.
- Non-exponential distributions.
- Effects of stiffness, where classes of events are described in a single model that occur at extremely different rates.
- System requirements that are too sophisticated leading to an overly complicated model and system measures that cannot be computed easily, such as distribution functions capturing response times or cumulative work.

Note that even discrete-event simulation (DES) techniques might suffer from difficulties such as largeness or stiffness. For instance, it is a non-trivial

problem to simulate with high confidence scenarios entailing relatively rare events while others are occurring much more often. Large or complex problems, in contrast, require the input of excessively numerous parameter sets so that correspondingly numerous simulation runs need to be executed. Many techniques have been suggested to overcome some of the problems. Among the most important ones are:

- *Hybrid* approaches that allow the combination of different solution techniques to take advantage of and to combine their strengths. Examples are mixed simulation and analytical/numerical approaches, or the combination of fault trees, reliability block diagrams, or reliability graphs, and Markov models [STP96]. Also product-form queueing networks and stochastic petri nets or non-product-form networks can be combined. More generally, this approach can be characterized as intermingling of *state-space* based and *non-state-space* based methods [STP96].

- *Phase-type* expansions are used to replace and approximate non-exponential distributions. As a trade-off, model size and computational complexity need to be taken into account.

- Sometimes *aggregation/disaggregation* techniques can be used to eliminate undesirable model properties. An approach to eliminate stiffness for transient analysis is introduced and discussed in detail in Chapter 5.

- Transformations can be applied leading from one domain of representation to another. Space, for example, is sometimes traded with time to replace one cumulative measure by another that is easier to calculate.

- DES of rare events is achieved by artificially speeding up these events in a controlled manner and taking some correction measures afterwards. This technique is called *importance sampling* [NNHG90].

Finally, the computational model may be prohibitively large due to the complexity of the problem at hand so as to preclude a direct use of the solution techniques. A low-level description is often out of the question due to the sheer size of the model and the error-prone process of creating it by hand. What is needed are techniques allowing for a compact or high-level representation of the lower-level model to be analyzed. Usually the higher-level representation languages are referred to as specification techniques and come with paradigms providing application specific structures that can be of help to practitioners.

Two primary methods to deal with large models:

- Many high-level specification techniques, queueing systems, generalized stochastic Petri nets (GSPNs), and stochastic reward nets (SRNs), being the most prominent representatives, have been suggested in the literature to automate the model generation [HaTr93]. While GSPNs/SRNs are covered in more detail in Section 2.2.3, our major theme is queueing systems. Both approaches can be characterized as *tolerating* largeness

of the underlying computational models and providing effective means for generating them.

- Another way to deal with large models is *to avoid* the creation of such models from the beginning. The major largeness-avoidance technique we discuss in this book is that of product-form queueing networks. The main idea is that the structure of the underlying CTMC allows an efficient solution that obviates the need of the generation, storage, and solution of the large state space. The second method of avoiding largeness is to separate the originally single large problem into several smaller problems and to combine submodel results into an overall solution. Both approximate and exact techniques are known for dealing with such multilevel models. The flow of information needed among submodels may be acyclic, in which case we have a hierarchical model [STP96]. If the flow of needed information is not acyclic, a fixed-point iteration may be necessary [CiTr93]. Other well-known techniques applicable for limiting model sizes are *state truncations* [BVDT88, GCS+86] and *state lumping* [Nico90].

Ideally, the computational model is automatically generated from a high-level description based on a formal and well-understood approach such as GSPNs or SRNs. In this case, *verification* is implicitly provided through the proven correctness and completeness of the techniques and tools employed. In practice, heuristics are often applied and, even worse, model generation is carried out manually. In this case, verification of the correctness of the computational model with respect to a given high-level description is of importance. But because approximations are often used, correctness is not given by a one-to-one relation between computational model and high-level description. Thus errors incurred need to be explicitly specified or bounded.

Note that the distinction between a high-level description and a computational model may sometimes be conceptual when the two representations coincide. Both the issues of model generation and analytical/numerical solution methods are the main topics of this book.

Once *computational results* have been produced, they need to be *validated* against data collected through *measurements*, if available, and against system requirements. The *validation results* are used for *modification* of the existing (high-level) model or for a proof of its validity. Of course, many iterations may be required for this important task. Note that a change in the model might ultimately result in a change in the system design.

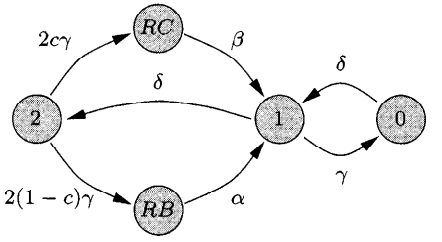
There is a strong relation between *measurements* and modeling. Measurement data are to be used for model validation. Furthermore, model *parameterization* relies heavily on input from measurement results. In case of a system design where the computer system does not yet exist, input usually comes from measurements of earlier studies on similar systems. Conversely, measurement studies are better planned and executed if guided by a model and by requirements placed on the computer system to be measured. In prac-

tice, measurements and models are not always applied on the same system. Of course, measurements cannot be used directly for the design of computer systems. Consequently, models-based solutions ought to be applied. Measurements, on the other hand, can provide the most accurate information on already existing systems. Measurement studies are often resource demanding, both with respect to hardware and software, and may require a great deal of work. The best approach, as always, is to combine the strengths of both techniques as much as possible.

In the following, we examine the issue of model formulation and the formal definition of measures based on given system requirements. While the general discussion of this section is not limited to a specific model type, in what follows we narrow our view and concentrate on CMTCs.

2.2.2 Performance Measures

We begin by introducing a simple example and then provide an introduction to Markov reward models as a means to obtain performance measures.



Parameter	Meaning
$1/\gamma$	mean time to failure
$1/\delta$	mean time to repair
$1/\alpha$	mean time to reboot
$1/\beta$	mean time to recover
c	coverage probability

State	Meaning
$i \in \{0, 1, 2\}$	i working processors
RC	recovery
RB	reboot

Fig. 2.7 A simple model of a multiprocessor system.

2.2.2.1 A Simple Example As an example adapted from Heimann, Mittal and Trivedi [HMT91], consider a multiprocessor system with n processor elements processing a given workload. Each processor is subject to failures with a *mean time to failure* (MTTF), $1/\gamma$. In case of a failure, recovery can successfully be performed with probability c. Typically, recovery takes a brief period of time with mean $1/\beta$. Sometimes, however, the system does not successfully recover from a processor failure and suffers from a more severe impact. In this case, we assume the system needs to be rebooted with longer average duration of $1/\alpha$. Probability c is called the *coverage factor* and is usually close

to 1. Unsuccessful recovery is most commonly caused by error propagation when the effect of a failure is not sufficiently shielded from the rest of the system. Failed processors need to be repaired, with the *mean time to repair* (MTTR) of $1/\delta$. Only one processor can be repaired at a time and repair of one processor does not affect the proper working of the remaining processors. If no processor is running correctly, the whole system is out of service until first repair is completed. Neither reboot nor recovery is performed when the last processor fails. If all times are assumed to be independent, exponentially distributed random variables, then a CTMC can be used to model the scenario. In Fig. 2.7 an example is given for the case of $n = 2$ processors and state space $S = \{2, RC, RB, 1, 0\}$.

Since CTMC in Fig. 2.7 is ergodic, the unique steady-state probability vector $\boldsymbol{\pi} = (\pi_2, \pi_{RC}, \pi_{RB}, \pi_1, \pi_0)$ is the solution of the Eqs. (2.58) and (2.59). From Fig. 2.7 the infinitesimal generator matrix can be derived:

$$\mathbf{Q} = \begin{pmatrix} -2\gamma & 2c\gamma & 2(1-c)\gamma & 0 & 0 \\ 0 & -\beta & 0 & \beta & 0 \\ 0 & 0 & -\alpha & \alpha & 0 \\ \delta & 0 & 0 & -(\gamma+\delta) & \gamma \\ 0 & 0 & 0 & \delta & -\delta \end{pmatrix}.$$

Clearly, the model in Fig. 2.7 is already an abstract representation of the system. With Eqs. (2.53) and (2.58), transient and steady-state probability vectors could be computed, but it is not yet clear what kind of measures should be calculated because we have said nothing about the application context and the corresponding system requirements. We take this model as a high-level description, which may need further elaboration so that a computational model can be generated from it. We consider four classes of system requirements for the example.

1. *System availability* is the probability of an adequate level of service, or, in other words, the long-term fraction of time of actually delivered service. Usually, short outages can be accepted, but interruptions of longer duration or accumulated outages exceeding a certain threshold may not be tolerable. Accordingly, the model in Fig. 2.7 must be evaluated with respect to requirements from the application context. First, tolerance thresholds must be specified as to what extent total outages can be accepted. Second, the states in the model must be partitioned into two sets: one set comprising the states where the system is considered "up," i.e., the service being actually delivered, and the complementary set comprising the states where the system is classified as "down." In our example, natural candidates for down states are in the set: $\{0, RC, RB\}$. But not all of them are necessarily classified as down states. Since reconfiguration generally takes a very short time, applications may well not be susceptible to such short interruptions. As a consequence, the less significant state RC could even be eliminated in the generation of the

computational model. Finally, the measures to obtain from the computational model must be decided. For example, the transient probability of the system being in an up state at a certain time t conditioned on some initial state, the proportion of time being in one of the up states in a given finite time horizon $[0, T)$ conditioned on an initial state, or the long-term proportion of time being in up states are some of the desired measures.

2. *System reliability* is the probability of uninterrupted service exceeding a certain length of time. By definition, no interruption of service at all can be tolerated. But note that it still needs to be exactly specified as to what kind of event is to be considered as an interruption. In the most restrictive application context, reconfiguration (RC) might not be acceptable. In contrast, nothing prevents the assumption to be made that even a reboot (RB) can be tolerated and only the failure of the last component leads to the single down state (0). As a third alternative, reconfiguration may be tolerated but not reboot and not the failure of the last component. The three possible scenarios are captured in Fig. 2.8. The model structures have also been adapted by introducing absorbing down states that reflect the fact that down states are considered as representing catastrophic events.

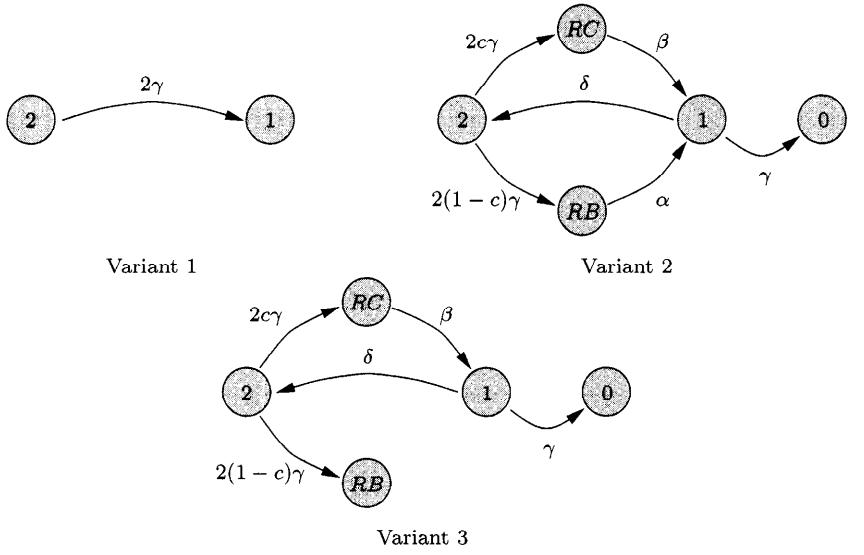


Fig. 2.8 Model variants with absorbing states capturing reliability requirements.

3. *System performance* takes the capacity of different configurations into account. Typical measures to be calculated are the utilization of the resources or system throughput. Other measures of interest relate to

the frequency with which certain incidents occur. With respect to the model in Fig. 2.7, the individual states need to be characterized by their contribution to a successful task completion. The higher the degree of parallelism, the higher the expected accomplishment will be. But it is a non-trivial task to find the right performance indices attributable to each particular state in the computational model. The easiest way would be to assign a capacity proportional to the number of working processors. But because each such state represents a whole configuration, where each system resource and the imposed workload can have an impact on the overall performance, more accurate characterization is needed. One way would be to execute separate modeling studies for every possible configuration and to derive some more detailed measures such as state-dependent effective throughputs or response time percentiles. Another way would be to expand the states and to replace some of them with a more detailed representation of the actual configuration and the workload. This approach will lead to a model of enormous size.

4. *Task completion* is reflected in the probability that a user will receive service at the required quality, or in other words, in the proportion of users being satisfied by the received service and its provided quality. Many different kinds of measures could be defined in this category. It could be the proportion of tasks being correctly processed or the probability that computation-time thresholds are not exceeded. With advanced applications such as continuous media (e.g., audio and video streams), measures are investigated relating the degree of user's satisfaction to the quality of the delivered service. Usually, such application-oriented measures are composed of different constituents such as timeliness, delay-variation, loss, and throughput measures.

In this subsection, we have indicated how system requirements affect model formulation. Furthermore, different types of performance measures were motivated and their relation to the model representation outlined. Next, we will show how the performance measures derived from system requirements can be explicitly specified and integrated into the representation of the computational model.

2.2.2.2 Markov Reward Models MRMs provide a unifying framework for an integrated specification of model structure and system requirements. We consider the explicit specification of system requirements as an essential part of the computational model. Once the model structure has been defined so that the infinitesimal generator matrix is known, the basic equations can be written depending on the given system requirements and the structure of the matrix. For the sake of completeness, we repeat here the fundamental Eqs. (2.53),

(2.58) and (2.59) for the analysis of CTMCs:

$$\frac{d\boldsymbol{\pi}(t)}{dt} = \boldsymbol{\pi}(t)\mathbf{Q}, \quad \boldsymbol{\pi}(0) = \boldsymbol{\pi}_0,$$
$$\mathbf{0} = \boldsymbol{\pi}\mathbf{Q}, \qquad \boldsymbol{\pi}\mathbf{1} = 1,$$

where $\boldsymbol{\pi}(t)$ is the transient state probability vector of the CTMC and $\boldsymbol{\pi}$ is the steady-state probability vector (assuming it exists). In addition to the transient state probabilities, sometimes *cumulative* probabilities are of interest. Let:

$$\mathbf{L}(t) = \int_0^t \boldsymbol{\pi}(u)du, \tag{2.64}$$

then $\mathbf{L}_i(t)$ denotes the *expected total time* the CTMC spends in state i during the interval $[0, t)$. A more convenient way to calculate the cumulative state probabilities is by solution of differential Eq. (2.65) [RST89]:

$$\frac{d\mathbf{L}(t)}{dt} = \mathbf{L}(t)\mathbf{Q} + \boldsymbol{\pi}(0), \quad \mathbf{L}(0) = \mathbf{0}. \tag{2.65}$$

Closely related to the vector of cumulative state probabilities is the vector describing the time-average behavior of the CTMC [SmTr90]:

$$\mathbf{M}(t) = \frac{1}{t}\mathbf{L}(t). \tag{2.66}$$

With $\mathbf{I}$ denoting the identity matrix, $\mathbf{M}(t)$ can be seen to satisfy the differential Eq. (2.67):

$$\frac{d\mathbf{M}(t)}{dt} = \mathbf{M}(t)\left(\mathbf{Q} - \frac{1}{t}\mathbf{I}\right) + \frac{1}{t}\boldsymbol{\pi}(0), \quad \mathbf{M}(0) = \mathbf{0}. \tag{2.67}$$

With these definitions, most of the interesting measures can be defined. But the special case of models containing absorbing states, like those in Fig. 2.8, deserves additional attention. Here, it would be interesting to compute measures based on the time a CTMC spends in non-absorbing states before an absorbing state is ultimately reached. Let the state space $S = A \cup N$ be partitioned into the set A of absorbing and the set N of non-absorbing (or transient) states. Then, the time spent before absorption can be calculated by taking the limit $\lim_{t\to\infty} \mathbf{L}_N(t)$ restricted to the states of the set N. Note that, in general, unless very special cases for the initial probability vector are considered, the limit does not exist for the states in A. Therefore, to calculate $\mathbf{L}_N(\infty)$, the initially given infinitesimal generator matrix $\mathbf{Q}$ is restricted to those states in N, so that matrix $\mathbf{Q}_N$ of size $|N| \times |N|$ results. Note that $\mathbf{Q}_N$ is *not* an infinitesimal generator matrix. Restricting also the initial probability vector $\boldsymbol{\pi}(0)$ to the non-absorbing states N results in $\boldsymbol{\pi}_N(0)$ and allows the

computation of $\lim_{t\to\infty}$ on both side of differential Eq. (2.65) so that following linear Eq. (2.68) results [STP96]:

$$\mathbf{L}_N(\infty)\mathbf{Q}_N = -\boldsymbol{\pi}_N(0). \tag{2.68}$$

To give an example, consider the variant three in Fig. 2.8. Here the state space S is partitioned into the set of absorbing states $A = \{RB, 0\}$ and non-absorbing states $N = \{2, RC, 1\}$ so that $\mathbf{Q}$ reduces to:

$$\mathbf{Q}_N = \begin{pmatrix} -2\gamma & 2c\gamma & 0 \\ 0 & -\beta & \beta \\ \delta & 0 & -(\gamma+\delta) \end{pmatrix}.$$

With $\mathbf{L}_N(\infty)$, the mean time to absorption (MTTA) can then be written as:

$$\text{MTTA} = \sum_{i\in N} L_i(\infty). \tag{2.69}$$

MRMs have long been used in Markov decision theory to assign cost and reward structures to states of Markov processes for an optimization [Howa71]. Meyer [Meye80] adopted MRMs to provide a framework for an integrated approach to performance and dependability characteristics. He coined the term *performability* to refer to measures characterizing the ability of fault-tolerant systems, that is, systems that are subject to component failures and that can perform certain tasks in the presence of failures.

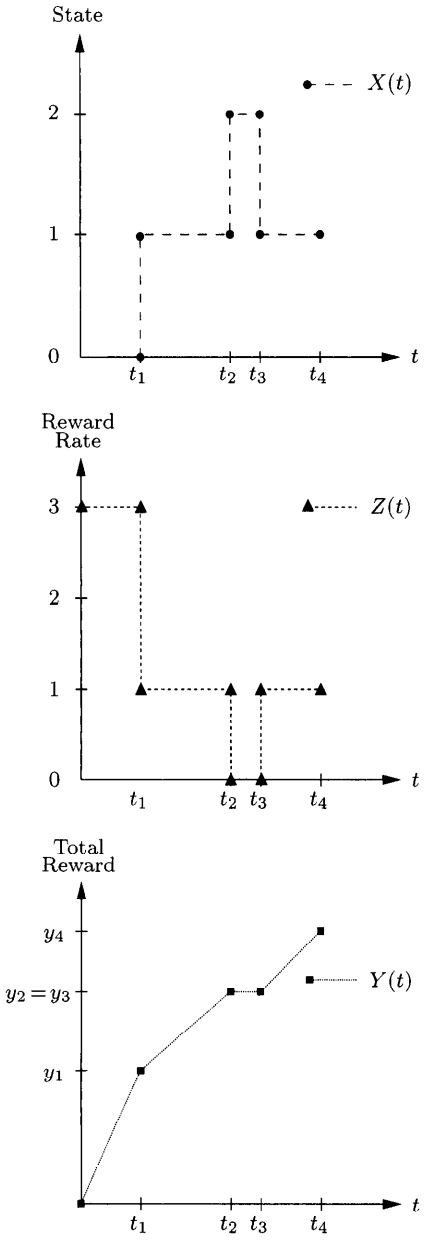
With MRMs, rewards can be assigned to states or to transitions between states of a CTMC. In the former case, these rewards are referred to as *reward rates* and in the latter as *impulse rewards*. In this text we consider state-based rewards only.

The reward rates are defined based on the system requirements, be it availability, reliability, or task oriented. Let the reward rate r_i be assigned to state $i \in S$. Then, a reward $r_i\tau_i$ is accrued during a sojourn of time τ_i in state i. Let $\{X(t), t \geq 0\}$ denote a homogeneous finite-state CTMC with state space S. Then, the random variable:

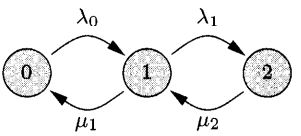
$$Z(t) = r_{X(t)} \tag{2.70}$$

refers to the *instantaneous reward rate of the MRM* at time t. Note the difference between reward rates r_i assigned to individual states i and the overall reward rate $Z(t)$ of the MRM characterizing the stochastic process as a whole. With the instantaneous reward rate of the CTMC defined as in Eq. (2.70), the *accumulated reward* $Y(t)$ in the finite time horizon $[0, t)$ is given by:

$$Y(t) = \int_0^t Z(\tau)d\tau = \int_0^t r_{X(\tau)}d\tau. \tag{2.71}$$



Markov Reward Model



$r_0 = 3.0 \qquad r_1 = 1.0 \qquad r_2 = 0.0$

$$Q = \begin{pmatrix} -\lambda_0 & \lambda_0 & 0 \\ \mu_1 & -(\mu_1 + \lambda_1) & \lambda_1 \\ 0 & \mu_2 & -\mu_2 \end{pmatrix}$$

$$y_1 = r_0 t_1 = 3t_1$$
$$y_2 = y_1 + r_1(t_2 - t_1) = y_1 + t_2 - t_1$$
$$y_3 = y_2 + r_2(t_3 - t_2) = y_2$$
$$y_4 = y_3 + r_1(t_4 - t_3) = y_3 + t_4 - t_3$$

Fig. 2.9 A three-state Markov reward model with sample paths of the $X(t)$, $Z(t)$, and $Y(t)$ processes.

For example, consider the sample paths of $X(t), Z(t)$, and $Y(t)$ processes in Fig. 2.9 adapted from [SmTr90]. A simple three-state MRM is presented, consisting of a CTMC with infinitesimal generator matrix $\mathbf{Q}$, and the reward rate vector $\mathbf{r} = (3, 1, 0)$ assigning reward rates to the states 0, 1, and 2, respectively. Assuming an initial probability vector $\boldsymbol{\pi}(0) = (1, 0, 0)$, the process is initiated in state 0, that is, $X(0) = 0$. Since the sojourn time of the first visit to state 0 is given by t_1, the reward $y_1 = 3t_1$ is accumulated during this period. After transition to state 1, additional reward $y_2 - y_1 = 1(t_2 - t_1)$ is earned, and so forth. While process $X(t)$ is assumed to follow the state transition pattern as shown in Fig. 2.9, the processes $Z(t)$ and $Y(t)$ necessarily show the behavior as indicated, because they are depending on $X(t)$ through the given reward vector $\mathbf{r}$. While $X(t)$ and $Z(t)$ are discrete valued and non-monotonic functions, $Y(t)$, in contrast, is a continuous-valued, monotonically non-decreasing function.

Based on the definitions of $X(t), Z(t)$, and $Y(t)$, which are non-independent random variables, various measures can be defined. The most general measure is referred to as the *performability* [Meye80]:

$$\Psi(y, t) = P[Y(t) \leq y], \tag{2.72}$$

where $\Psi(y, t)$ is the distribution of the accumulated reward over a finite time $[0, t)$. Unfortunately, the performability is difficult to compute for unrestricted models and reward structures. Smaller models can be analyzed via double Laplace transform [SmTr90], while references to other algorithms are summarized in Table 2.17. The same mathematical difficulties arise if the distribution $\Phi(y, t)$ of the *time-average accumulated reward* is to be computed:

$$\Phi(y, t) = P\left[\frac{1}{t}Y(t) \leq y\right]. \tag{2.73}$$

The problem is considerably simplified if the system requirements are limited to expectations and other moments of random variables rather than distribution functions of cumulative rewards. Alternatively, efficient solution algorithms are available if the rewards can be limited to a binary structure or if the model structure is acyclic.

The *expected instantaneous reward rate* can be computed from the solution of differential Eq. (2.53):

$$E[Z(t)] = \sum_{i \in S} r_i \pi_i(t). \tag{2.74}$$

If the underlying model is ergodic, the solution of linear Eqs. (2.58) and (2.59) can be used to calculate the *expected reward rate in the limit* as $t \to \infty$:

$$\lim_{t \to \infty} \left(\frac{1}{t}E[Y(t)]\right) = E[Z(\infty)] = E[Z] = \sum_{i \in S} r_i \pi_i. \tag{2.75}$$

To compute the first moment of the performability, or the *expected accumulated reward*, advantage can be taken of the solution of differential Eq. (2.65):

$$E[Y(t)] = \sum_{i \in S} r_i L_i(t). \tag{2.76}$$

For models with absorbing states, the limit as $t \to \infty$ of the expected accumulated reward exists and is called the *expected accumulated reward until absorption*. It follows from the solution of linear Eq. (2.68) that:

$$E[Y(\infty)] = \sum_{i \in N} r_i L_i(\infty). \tag{2.77}$$

Also, higher moments of the random variables can be derived. As an example, the *variance* of the instantaneous reward rate $Z(t)$ is obtained from:

$$\text{var}[Z(t)] = \sum_{i \in S} r_i^2 \pi_i(t) - \left(\sum_{i \in S} r_i \pi_i(t) \right)^2. \tag{2.78}$$

Probabilities of $Z(t)$ and Z can also be easily written [TMWH92]:

$$P[Z(t) = x] = \sum_{i \in S, r_i = x} \pi_i(t), \tag{2.79}$$

$$P[Z = x] = \sum_{i \in S, r_i = x} \pi_i. \tag{2.80}$$

Two different distribution functions can also be easily derived. With the transient state probabilities gained from the solution of Eq. (2.53), the probability that the instantaneous reward rate $Z(t)$ does not exceed a certain level x is given as:

$$P[Z(t) \leq x] = \sum_{i \in S, r_i \leq x,} \pi_i(t). \tag{2.81}$$

The distribution $P[Y(\infty) \leq y]$ of the accumulated reward until absorption can be derived by applying a substitution according to Beaudry [Beau78] and Ciardo et al. [CMST90]. The essential idea is to generate a (computational) model with a *binary* reward structure and infinitesimal generator matrix $\hat{\mathbf{Q}}$ from the given (high-level) CTMC with a general reward structure and infinitesimal generator matrix $\mathbf{Q}$. This transformation is achieved by applying a normalization in the time domain according to the reward rates so that the distribution of the time until absorption of the new CTMC can be used as a substitute measure:

$$P[Y(\infty) \leq y] = 1 - \sum_{i \in N} \hat{\pi}_i(y). \tag{2.82}$$

Returning to the example model in Fig. 2.7, we apply the MRM framework with respect to different system requirements to demonstrate how the model can be instantiated in several ways to yield meaningful computational models. The following case study [HMT91] serves as a guideline as to how to describe models in light of different system requirements, such as availability and reliability or task-oriented and system performance.

2.2.2.3 A Case Study We now illustrate the use of MRMs by means of the example in Fig. 2.7.

2.2.2.3.1 System Availability Availability measures are based on a *binary* reward structure. Assuming for the model in Fig. 2.7 that one processor is sufficient for the system to be "up" (set of states $U = \{2, 1\}$) and that otherwise it is considered as being down (set of states $D = \{RC, RB, 0\}$, where $S = U \cup D$), a reward rate 1 is attached to the states in U and a reward rate 0 to those in D. The resulting reward function r is summarized in Table 2.1.

The instantaneous availability is given by:

$$A(t) = E[Z(t)] = \sum_{i \in S} r_i \pi_i(t) = \sum_{i \in U} \pi_i(t) = \pi_1(t) + \pi_2(t).$$

Unavailability can be calculated with a reverse reward assignment to that for availability. The instantaneous unavailability, therefore, is given by:

$$UA(t) = E[Z(t)] = \sum_{i \in S} r_i \pi_i(t) = \sum_{i \in D} \pi_i(t) = \pi_{RC}(t) + \pi_{RB}(t) + \pi_0(t).$$

Steady-state availability, on the other hand, is given by:

$$A = E[Z] = \sum_{i \in S} r_i \pi_i = \sum_{i \in U} \pi_i = \pi_2 + \pi_1.$$

The interval availability provides a time average value:

$$\bar{A}(t) = \frac{1}{t} E[Y(t)] = \frac{1}{t} \int_0^t A(x)dx = \frac{1}{t} \sum_{i \in U} \int_0^t \pi_i(x)dx = \frac{1}{t}[L_2(t) + L_1(t)].$$

Note that a different classification of the model states would lead to other values of the availability measures. Instantaneous, interval, and steady-state availabilities are the fundamental measures to be applied in this context, but there are related measures that do not rely on the binary reward structure. Other measures related to availability can be considered. To compute the mean transient uptime $T^{(1)}(t)$ and the mean steady-state uptime $T^{(2)}(t)$ in a

Table 2.1 Reward assignment for computation of availability

State i	Reward Rate r_i
2	1
RC	0
RB	0
1	1
0	0

Table 2.2 Reward assignment for mean uptimes

State i	Reward Rate r_i
2	t
RC	0
RB	0
1	t
0	0

given time horizon $[0, t)$, reward rates 1 in Table 2.1 are replaced by reward rates t in Table 2.2:

$$T^{(1)}(t) = \frac{1}{t}E[Y(t)] = \frac{1}{t}t\sum_{i\in U} L_i(t) = \sum_{i\in U} L_i(t) = t\bar{A}(t),$$
$$T^{(2)}(t) = E[Z] = tA.$$

Very important measures relate to the *frequency* of certain events of interest. Any event represented in the model can be captured here. For example, we could be interested in the average number of repair calls made in a given period of time $[0, t)$. With assumed repair rate δ and the fact that repair calls are made from states 0 and 1 of the model of Fig. 2.7, the transient average number of repair calls $N^{(1)}(t)$ and steady-state average number of repair calls $N^{(2)}(t)$ can again be determined, with the reward function as defined in Table 2.3:

$$N^{(1)}(t) = \frac{1}{t}E[Y(t)] = \delta(L_0(t) + L_1(t)),$$
$$N^{(2)}(t) = E[Z] = t\delta(\pi_0 + \pi_1).$$

Table 2.3 Reward assignment for number of repair calls in $[0, t)$

State i	Reward Rate r_i
2	0
RC	0
RB	0
1	$t\delta$
0	$t\delta$

Table 2.4 Reward assignment for computation of mean number of outages exceeding certain state-dependent thresholds in finite time $[0, t)$ with the CTMC from Fig 2.7

State i	Reward Rate r_i
2	0
RC	$t\beta e^{-\beta\tau_{RC}}$
RB	$t\alpha e^{-\alpha\tau_{RB}}$
1	0
0	$t\delta e^{-\delta\tau_0}$

Sometimes, local tolerance levels $\tau_i, i \in S$ are considered such that certain events, such as outages, can be tolerated as long as they do not last longer than a specified threshold τ_i. In our context it would be natural to assume a tolerance with respect to duration of reconfiguration and possibly also with respect to reboot. Based on these assumptions, the reward function as shown in Table 2.4 is defined to compute the mean number of *severe* interruptions $I^{(1)}(t), I^{(2)}(t)$ in finite time $[0, t)$, i.e., interruptions lasting longer than certain thresholds for the transient and the steady-state based cases, respectively:

$$I^{(1)}(t) = \frac{1}{t}E[Y(t)] = \beta e^{-\beta\tau_{RC}} L_{RC}(t) + \alpha e^{-\alpha\tau_{RB}} L_{RB}(t) + \delta e^{-\delta\tau_0} L_0(t),$$
$$I^{(2)}(t) = E[Z] = t\beta e^{-\beta\tau_{RC}} \pi_{RC} + t\alpha e^{-\alpha\tau_{RB}} \pi_{RB} + t\delta e^{-\delta\tau_0} \pi_0.$$

The reward rates in Table 2.4 are specified based on the observation that event duration times are exponentially distributed in CTMCs, such as the one depicted in Fig. 2.7. Therefore, if the parameter were given by λ, the probability of event duration lasting longer than τ_i units of time would be $e^{-\lambda\tau_i}$. This probability is used to weight the average number of events $t\lambda$. But note that the reward rates are by *no* means limited to exponential distributions, any known distributions could be applied here. It is useful to specify events related to timeouts or events that have a (deterministic) deadline attached to them.

2.2.2.3.2 System Reliability Again, a *binary* reward function r is defined that assigns reward rates 1 to up states and reward rates 0 to (absorbing) down states, as given in Table 2.5. Recalling that reliability is the likelihood that an unwanted event has not yet occured since the beginning of the system operation, reliability can be expressed as:

$$R(t) = P[T > t] = P[Z(t) = 1] = 1 - P[Z(t) \leq 0] = E[Z(t)],$$

where the random variable T characterizes the time to the next occurrence of such an unwanted (failure) event. Referring to the scenarios depicted in Fig. 2.8, three different reliability functions can be identified:

$$R_1(t) = \pi_2(t),$$
$$R_2(t) = \pi_2(t) + \pi_{RC}(t) + \pi_{RB}(t) + \pi_1(t),$$
$$R_3(t) = \pi_2(t) + \pi_{RC}(t) + \pi_1(t).$$

Remember that $R_1(t), R_2(t)$, and $R_3(t)$ are computed on the basis of three different computational models. The function to be used depends on the application requirements.

With a known reliability function $R(t)$, the mean time to the occurrence of an unwanted (failure) event is given by:

$$E[T] = \text{MTTF} = \text{MTTA} = \int_0^\infty R(t)dt.$$

MTTF and MTTA are acronyms for *mean time to failure* and *mean time to absorption*, respectively. With the formula of Eq. (2.69), the MTTF (MTTA) can be efficiently computed.

Table 2.5 Reward assignment for computation of reliability with variant-2 CTMC from Fig 2.8

State i	Reward Rate r_i
2	1
RC	1
RB	1
1	1
0	0

Table 2.6 Reward assignment for predicting the number of catastrophic incidents with variant-2 CTMC from Fig 2.8

State i	Reward Rate r_i
2	0
RC	0
RB	0
1	$t\gamma$
0	0

With the reliability $R(t) = E[Z(t)]$ given, the unreliability follows as the complement:

$$UR(t) = 1 - E[Z(t)] = 1 - R(t) = 1 - P[T > t] = P[T \leq t].$$

It is an important observation that the unreliability is given as a *distribution function* of a time variable. Note that we have gained this result simply by computing $E[Z(t)]$. The trick was to modify the computational model in the right way such that absorbing states were introduced and the appropriate reward structure was chosen, as the one depicted in Table 2.5. Rather general time distribution functions can be effectively computed in this way. This method allows the calculation of *percentiles* and other powerful performance measures like response-time distribution functions, which are not necessarily of exponential type [MMKT94]. It is also worth mentioning that the unreliability could be calculated based on a reward assignment complementing the one in Table 2.5. In this case, we would get $UR(t) = E[Z(t)] = \pi_0(t)$.

Related to reliability measures, we would often be interested in knowing the expected number of catastrophic events $C(t)$ to occur in a given time interval $[0, t)$. To this end:

$$C(t) = \frac{1}{t}E[Y(t)] = \gamma L_1(t)$$

needs to be calculated for some interesting initial up state, $i \in U$ with the reward assignment, as given in Table 2.6.

2.2.2.3.3 System and Task Performance In the preceding two subsections, where availability and reliability measures were considered, binary reward functions were of predominant importance. However, other reward functions have already been used to model availability- and reliability-oriented measures

such as frequencies of outages or other related incidents of interest. A more general view is presented in this section.

First, we look at a particular task with some assumed task execution time x. Under the given circumstances, we would like to know the probability of successful completion of this task. Two different aspects come into the picture here: the dynamics of the system and the susceptibility of the task (or its user) to these dynamic changes in the configuration. In the case of the example in Fig 2.7, just outages and their impact on the task completion are considered. But this situation can be generalized to different degrees of the task's susceptibility to loss of data or other disturbing events as well.

In the case of a user requiring uninterrupted service for some time x, the probabilities that *no* interruption occurs in the respective states are assigned as reward rates. We allow task requirement to be state dependent as well so that in state 2 the task requires x_2 time units to complete and likewise for state 1. The probabilities can be easily concluded from Fig. 2.7. In state 2, for example, the probability $P[O > x_2]$ that the system will be operational for more than x_2 units of time is given by $e^{-2\gamma x_2}$. Therefore, the task interruption probability is:

$$P[O \leq x_2] = 1 - P[O > x_2] = 1 - e^{-2\gamma x_2},$$

conditioned on the assumption the system is running in state 2 and on the hypothesis that the first failure causes a task's interruption. These probabilities can be assigned as reward rates as shown in Table 2.7, so that the interruption probability $IP^{(1)}(x_1, x_2)$ of a task of length x_1, x_2, respectively is computed with $E[Z]$, when assuming that probability of a user finding the system initially being up is given by the steady-state probabilities:

$$IP^{(1)}(x_1, x_2) = E[Z] = (1 - e^{-2\gamma x_2})\pi_2 + (1 - e^{-\gamma x_1})\pi_1.$$

Table 2.7 Reward assignment for computing a task's interruption probability with duration x for the CTMC from Fig 2.7

State i	Reward Rate r_i
2	$1 - e^{-2\gamma x_2}$
RC	0
RB	0
1	$1 - e^{-\gamma x_1}$
0	0

Table 2.8 Reward assignment for computing a task's interruption probability in case of uncovered error or loss of required processor for the CTMC from Fig 2.7

State i	Reward Rate $r(i)$
2	$1 - e^{-(\gamma(1-c)+\gamma)x_2}$
RC	0
RB	0
1	$1 - e^{-\gamma x_1}$
0	0

Another scenario would be that a running task is constrained to using a single processing element, even if two were available. In this case, a task would

only be susceptible to a failure of its own required processor or if an uncovered failure occurs that also has severe impacts on the currently running one. With the reward assignment as shown in Table 2.8 the desired interruption probability $IP^{(2)}$ is obtained:

$$IP^{(2)}(x_1, x_2) = E[Z] = (1 - e^{-(\gamma(1-c)+\gamma)x_2})\pi_2 + (1 - e^{-\gamma x_1})\pi_1.$$

Recall also the reward function as defined in Table 2.4. Based on these reward rates, it is also possible to compute the mean number of *severe* interruptions in a finite time duration. These measures can also be applied as success measures for running a task with finite execution time x and with certain interruption tolerance patterns attached to it. In particular, continuous media exhibit such a kind of susceptibility profile. Interruptions or losses can be tolerated up to a certain threshold. Beyond that threshold, a severe degradation in quality is perceived by users.

Referring back to the example in Fig.2.7, it could be of interest to take into account different levels of performance that the modeled system may exhibit in various states. One simple method is to use the processing power (proportional to the number of processing elements actually working). This would lead to the reward function depicted in Table 2.9. With this reward assignment, the expected available computing capacity in steady state $E[Z]$, the expected instantaneous capacity at time t, $E[Z(t)]$, or the cumulative capacity deliverable in finite time $[0, t)$, $E[Y(t)]$, or its time average $\frac{1}{t}E[Y(t)]$ could be computed. These availability measures are known as capacity-oriented availability. But this reward assignment is too coarse to capture performance effects accurately. First, it is overly optimistic with respect to the actual performance and, second, effects due to outages tend to be masked by the coarse reward structure and tend to become insignificant as a consequence. From this point of view, either the reward function or the model structure are ill-posed with respect to the system requirements.

Table 2.9 Reward assignment for computation of capacity oriented availability measures for the CTMC from Fig 2.7

State i	Reward Rate r_i
2	2
RC	0
RB	0
1	1
0	0

Table 2.10 Reward assignment for computing the total loss probability for the CTMC from Fig 2.7

State i	Reward Rate $r(i)$
2	l_2
RC	1
RB	1
1	l_1
0	1

More sophisticated approaches either take into account the user's requirements related to the tasks to be processed or incorporate more information about the investigated system into the model. A hierarchical method is pos-

sible, where every working configuration is modeled separately with some (queueing) models so that more interesting parameters can be derived. Trivedi et al. [TMWH92], for example, use a finite queueing system that is subject to losses due to buffer overflow and to response time deadline violation [TSIH90], while de Meer et al. [MTD94] apply infinite queueing systems with limits imposed on the system's response time for processing the tasks, so that losses occur if a deadline is missed. In both cases, closed-form solutions are available to calculate the loss probabilities l_i for all working configurations or up states $i \in U$ [GrHa85]. Of course, knowledge of the processing power of the working components and of the traffic characteristics are needed to parameterize the underlying performance models. Many of the details are given in later chapters when appropriate.

Table 2.11 Reward assignment for computing the normalized throughput for the CTMC from Fig 2.7

State i	Reward Rate $r(i)$
2	$1 - l_2$
RC	0
RB	0
1	$1 - l_1$
0	0

Table 2.12 Reward assignment for computing of the expected number of tasks lost in finite time for the CTMC from Fig 2.7

State i	Reward Rate r_i
2	$l_2\lambda$
RC	λ
RB	λ
1	$l_1\lambda$
0	λ

The values l_i are used to characterize the percentage loss of tasks arriving at the system in state $i \in U$. Consequently, tasks arriving in down states are lost with a probability of 1.0, that is, $l_i = 1, i \in D$. Furthermore, it is assumed that (quasi) steady state in the performance model is maintained in all configuration states, such as those depicted in Fig. 2.7. In other words, tasks arriving at the system in a certain state are assumed to leave the system in the same state. This assumption is admissible because the events related to traffic effects usually occur in orders of magnitude faster than events related to changes in configurations (due to failure-repair events). The resulting reward assignment is summarized in Table 2.10. Of course, a complementary reward structure as in Table 2.11 could also be chosen, so that the normalized throughput would be calculated as has been done by Meyer [Meye80].

The expected total loss probability, TLP, in steady state and the transient expected total loss probability $TLP(t)$ are then given by:

$$\begin{aligned} TLP(t) &= E[Z(t)] \\ &= \sum_{i \in U} l_i \pi_i(t) + \sum_{i \in D} 1 \pi_i(t) \\ &= l_2 \pi_2(t) + l_1 \pi_1(t) + \pi_{RC}(t) + \pi_{RB}(t) + \pi_0(t), \end{aligned}$$

$$\begin{aligned} TLP &= E[Z] \\ &= \sum_{i \in U} l_i \pi_i + \sum_{i \in D} 1 \pi_i \\ &= l_2 \pi_2 + l_1 \pi_1 + \pi_{RC} + \pi_{RB} + \pi_0. \end{aligned}$$

To compute the expected number of tasks lost in finite time $[0, t)$, two choices are available: the interval-based measure $TL^{(1)}(t)$ or the steady-state-based measure $TL^{(2)}(t)$ can be applied. In the first case, the reward rates in Table 2.10 are multiplied by the tasks' arrival rate λ to yield Table 2.12 and to compute $E[Y(t)]$. In the second case, $E[Z]$ is calculated on the basis of a modified reward assignment. All reward rates from Table 2.12 need to be multiplied by the length of the investigated time horizon $[0, t)$ to get Table 2.13. Of course, the same reward assignment could be used for the interval-based case and $\frac{1}{t} E[Y(t)]$ could be evaluated instead:

$$\begin{aligned} TL^{(1)}(t) &= E[Y(t)] \\ &= \int_0^t \left(\sum_{i \in U} l_i \lambda \pi_i(\tau) + \sum_{i \in D} 1 \lambda \pi_i(\tau) \right) d\tau \\ &= \sum_{i \in U} \lambda l_i L_i(t) + \sum_{i \in D} \lambda L_i(t) \\ &= \lambda \left(l_2 L_2(t) + l_1 L_1(t) + L_{RC}(t) + L_{RB}(t) + L_0(t) \right), \end{aligned}$$

$$\begin{aligned} TL^{(2)}(t) &= E[Z] \\ &= t\lambda \left(\sum_{i \in U} l_i \pi_i + \sum_{i \in D} 1 \pi_i \right) \\ &= t\lambda \left(l_2 \pi_2 + l_1 \pi_1 + \pi_{RC} + \pi_{RB} + \pi_0 \right). \end{aligned}$$

Table 2.13 Reward assignment for computing the expected total number of tasks lost in finite time for the CTMC from Fig 2.7

State i	Reward Rate r_i
2	$tl_2\lambda$
RC	$t\lambda$
RB	$t\lambda$
1	$tl_1\lambda$
0	$t\lambda$

Table 2.14 Reward assignment for throughput computation with the CTMC of Fig 2.10

State i	Reward Rate r_i
3	μ
2	μ
1	μ
0	0

Response time distributions and percentiles of more complicated systems can rarely be computed by simple elementary queueing systems, and closed-

form solutions are rarely available. Muppala et al. have shown how CTMCs with absorbing states (or their equivalent SRNs) can be exploited to compute these measures numerically in rather general cases [MuTr91, MMKT94]. Once the probabilities that the response time exceeds a certain threshold have been derived with this method, the reward assignments can be made corresponding to Tables 2.10, 2.12, or 2.13 and the appropriate measures can be derived.

The use of reward rates is not restricted to reliability, availability, and performability models. This concept can also be used in pure (failure-free) performance models to conveniently describe performance measures of interest. In many computer performance studies, expected throughput, mean response time, or utilization are the most important measures. These measures can easily be specified by means of an appropriate reward function. Throughput characterization, for example, can be achieved by assigning the state transition rate corresponding to departure from a queue (service completion) as reward rates to the state where the transition originates. Mean response time of queueing systems can be represented indirectly and computed in two steps. First, by assigning the number of customers present in a state as the reward rate to that state and to compute the measures as required. Second, it can be proven that there exists a linear correspondence between mean response time and mean number of customers present so that Little's theorem [Litt61, King90] can be used for the computation of the mean response times from the mean number of customers. Finally, it is worth mentioning that the utilization is also based on a binary reward assignment. If a particular resource is occupied in a given state, reward rate 1 is assigned, otherwise reward rate 0 indicates the idleness of the resources.

To illustrate these measures, reward definitions for a simple CTMC are considered. Imagine customers arriving at a system with exponentially distributed interarrival times with mean $1/\lambda$. In the system they compete for service from a single server station. Since service is exclusively received by each customer, if more than one customer is in the system at the same time, the others have to wait in line until their turn comes. Service times are independent exponentially distributed with mean $1/\mu$. To keep the example simple, we limit the maximum number of customers in the system to three. The system is described by the MRM shown in Fig. 2.10.

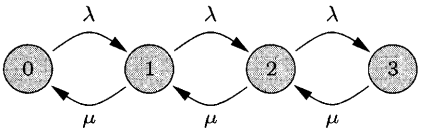


Fig. 2.10 A simple CTMC with arrivals and services.

Every state in $S = \{3, 2, 1, 0\}$ of the MRM represents the number of customers in the system. A state transition occurs if a customer's service is completed by the service station (arc annotated with μ) or if a new customer arrives (arc annotated with λ). Tables 2.14, 2.15, and 2.16 summarize the

Table 2.15 Reward assignment for computing the mean number of customers with the CTMC of Fig 2.10

State i	Reward Rate r_i
3	3
2	2
1	1
0	0

Table 2.16 Reward assignment for computing utilization measures with the CTMC of Fig 2.10

State i	Reward Rate r_i
3	1
2	1
1	1
0	0

reward assignments for throughput, mean number of customers, and utilization measure computations, respectively. The formulae for the computation of throughput measures $\lambda^{(1)}(t)$, $\lambda^{(2)}(t)$, $\lambda^{(3)}$; the mean number of customers $\overline{K^{(1)}}(t)$, $\overline{K^{(2)}}(t)$, $\overline{K^{(3)}}$; mean response time measures $\overline{T^{(1)}}(t)$, $\overline{T^{(2)}}(t)$, $\overline{T^{(3)}}$; and utilization measures $\rho^{(1)}(t)$, $\rho^{(2)}(t)$, $\rho^{(3)}$ are also given. Note that response time measures $\overline{T^{(i)}}$ can be calculated with the help of Little's theorem [Litt61]:

$$\overline{T^{(i)}} = \frac{1}{\lambda}\overline{K^{(i)}}. \tag{2.83}$$

$$\begin{aligned}
\lambda^{(1)}(t) &= E[Y(t)] = \int_0^t \left(\sum_i r_i \pi_i(\tau)\right) d\tau = \sum_i r_i L_i(t) \\
&= \mu\left(L_3(t) + L_2(t) + L_1(t)\right), \\
\lambda^{(2)}(t) &= E[Z(t)] = \sum_i r_i \pi_i(t) = \mu\left(\pi_3(t) + \pi_2(t) + \pi_1(t)\right), \\
\lambda^{(3)} &= E[Z] = \sum_i r_i \pi_i = \mu\left(\pi_3 + \pi_2 + \pi_1\right). \\
\overline{K^{(1)}}(t) &= E[Z(t)] = \sum_i r_i \pi_i(t) = 3\pi_3(t) + 2\pi_2(t) + 1\pi_1(t), \\
\overline{T^{(1)}}(t) &= \frac{1}{\lambda}\overline{K^{(1)}}(t) = \frac{1}{\lambda}\left(3\pi_3(t) + 2\pi_2(t) + 1\pi_1(t)\right), \\
\overline{K^{(2)}}(t) &= \frac{1}{t}E[Y(t)] = \frac{1}{t}\int_0^t \left(\sum_i r_i \pi_i\right) d\tau = \frac{1}{t}\sum_i r_i L_i(t) \\
&= \frac{1}{t}\left(3L_3(t) + 2L_2(t) + 1L_1(t)\right),
\end{aligned}$$

$$\overline{T^{(2)}}(t) = \frac{1}{\lambda}\overline{K^{(2)}}(t) = \frac{1}{\lambda t}\left(3L_3(t) + 2L_2(t) + 1L_1(t)\right),$$

$$\overline{K^{(3)}} = E[Z] = \sum_i r_i\pi_i = 3\pi_3 + 2\pi_2 + 1\pi_1,$$

$$\overline{T^{(3)}} = \frac{1}{\lambda}\overline{K^{(3)}} = \frac{1}{\lambda}\left(3\pi_3 + 2\pi_2 + 1\pi_1\right).$$

$$\rho^{(1)}(t) = E[Z(t)] = \sum_i r_i\pi_i(t) = \pi_3(t) + \pi_2(t) + \pi_1(t),$$

$$\rho^{(2)}(t) = \frac{1}{t}E[Y(t)] = \frac{1}{t}\int_0^t \left(\sum_i r_i\pi_i(\tau)\right) d\tau = \frac{1}{t}\sum_i r_i L_i(t)$$

$$= \frac{1}{t}\left(L_3(t) + L_2(t) + L_1(t)\right),$$

$$\rho^{(3)} = E[Z] = \sum_i r_i\pi_i = \pi_3 + \pi_2 + \pi_1.$$

More MRM examples with all these types of reward assignments can be found in [MTBH93], and MuTr92, and in [TMWH92] where multiprocessor architectures with different interconnection techniques are studied by defining corresponding reward functions. As another example, queues with breakdown are investigated.

Tables 2.17, 2.18, 2.19, and 2.20 summarize the different reward assignments discussed in this section.

2.2.3 Generation Methods

We have already pointed out the importance of model generation. There are many reasons for the separation of high-level model description and lower-level computational model. For one thing, it will allow the modeler to focus more on the system being modeled rather than on low-level modeling details. Recall that CTMCs and similar other state-space based models tend to become very large when real-world problems are studied. Therefore, it is attractive to be able to specify such large models in a compact way and to avoid the error-prone and tedious creation of models manually. The compact description is based on the identification of regularities or repetitive structures in the system model. Besides reducing the size of the description, such abstractions provide visual and conceptual clarity.

It is worth repeating that regularities must be present in order to take advantage of automatic model generation techniques. If repetitive structures cannot be identified, the "higher-level" representation would hardly be smaller

Table 2.17 Overview of important MRM measures

Measure	Formula	Literature
$E[Z]$	$\sum_{i \in S} r_i \pi_i$	[GaKe79], [Kuba86], [MuTr91], [MTBH93], [TSIH90]
$E[Z(t)]$	$\sum_{i \in S} r_i \pi_i(t)$	[BRT88], [DaBh85], [HMT91], [MuTr91], [NaGa88]
$E[Y(t)]$	$\sum_{i \in S} r_i L_i(t)$	[BRT88], [NaGa88], [MTD94], [MTBH93]
$E[Y(\infty)]$	$\sum_{i \in S} r_i L_i(\infty)$	[TMWH92]
$P[Z(t) = x]$	$\sum_{r_i = x, i \in S} \pi_i(t)$	[Husl81], [LeWi89], [Wu82]
$P[Z(t) \leq x]$	$\sum_{r_i \leq x, i \in S} \pi_i(t)$	[SoGa89]
$P[Y(\infty) \leq y]$	$1 - \sum_{i \in N} \hat{\pi}_i(y)$	[Beau78], [CMST90]
$P[Y(t) \leq y]$	$(sI + uR - Q)\tilde{\Psi}^*(u,s) = e$ (for small models)	[STR88], [GoTa87], [CiGr87], [SoGa89], [DoIy87], [Meye82]

and may even be more cumbersome. Furthermore, since software tools, hardware resources, human labor, and time are necessary to install and run such an automatic generation facility, its application is recommended only beyond a certain model size. Finally, the structure of the high-level description language generally makes it more suited for a specific domain of applications. As an example, effects of resource sharing and queueing are naturally represented by means of queueing systems, while synchronization can be easily specified with some Petri net variant.

Many high-level description languages exist, each with its own merits and limitations. A comprehensive review of specification techniques for MRMs can be found in [HaTr93]. In the remainder of this chapter, we discuss two types of stochastic Petri nets: generalized stochastic Petri nets (GSPNs) and stochastic reward nets (SRNs). We also discuss algorithms to convert a GSPN into a CTMC and an SRN into an MRM.

2.2.3.1 Petri Nets Petri nets (PNs) were originally introduced by C.A. Petri in 1962 [Petr62]. A PN is a bipartite directed graph, consisting of two types of nodes, namely *places*, P, and *transition*, T. Arcs between the nodes fall in two categories: *input arcs* lead from an *input place* to a transition and *output arcs* connect a transition to an *output place*. Arcs always have to lead from one type of nodes to the complementary one. A finite number of *tokens* can be distributed among the places and every place may contain an arbitrary (natural) number of tokens. A *marking* $m \in \mathcal{M}$ is defined as a possible assignment of tokens to all places in the PN. Markings are sometimes referred to as *states* of the PN. If P denotes the set of places, then a marking m is a multiset, $m \in \mathcal{M} \subset \mathbb{N}^{|P|}$, describing the number of tokens in each place.

Table 2.18 Use of reward rates to compute different measures – part I

Requirement	Reward Assignments	Measure
Availability [HMT91], [MTD94], [MCT94], [DaBh85]	$r_i = \begin{cases} 1 & \text{if } i \in U \\ 0 & \text{else} \end{cases}$ [Table 2.1]	$E[Z]$(steady state) $E[Z(t)] = P[Z(t) = 1]$ (instantaneous) $\frac{1}{t}E[Y(t)]$(interval) $E[Y(\infty)]^*$, $P[Y(\infty) \leq y]^*$ $P[\frac{1}{t}Y(t) \leq y]$
Unavailability [HMT91], [MTD94], [MCT94]	$r_i = \begin{cases} 1 & \text{if } i \in D \\ 0 & \text{else} \end{cases}$	$E[Z]$(steady state) $E[Z(t)] = P[Z(t) = 1]$ (instantaneous) $\frac{1}{t}E[Y(t)]$(interval) $P[\frac{1}{t}Y(t) < y]$
Mean uptime in $[0, t)$	$r_i = \begin{cases} t & \text{if } i \in U \\ 0 & \text{else} \end{cases}$ [Table 2.2]	$E[Z]$ (approx.) $\frac{1}{t}E[Y(t)]$
Mean uptime in $[0, t)$	$r_i = \begin{cases} 1 & \text{if } i \in U \\ 0 & \text{else} \end{cases}$ [Table 2.1]	$E[Y(t)]$ $E[Y(\infty)]^*$, $P[Y(\infty) \leq y]^*$
Approx. frequency of repair calls in $[0, t)$ with mean duration $\frac{1}{\delta}$ [HMT91]	$r_i = \begin{cases} t\delta & \text{if } i \in S_R \\ 0 & \text{else} \end{cases}$ (S_R: states where repair is needed) [Table 2.3]	$E[Z]$
Frequency of repair calls in $[0, t)$ with mean duration $\frac{1}{\delta}$ [HMT91]	$r_i = \begin{cases} \delta & \text{if } i \in S_R \\ 0 & \text{else} \end{cases}$ (S_R: states where repair is needed) [Table 2.3]	$E[Y(t)]$
Mean percentage of severe interruptions [HMT91]	$r_i = \begin{cases} P[T > \tau_i] & \text{if } i \in U \\ 0 & \text{else} \end{cases}$ ($\tau_i \geq 0$: threshold for state i, T: duration of outage) [Table 2.4]	$E[Y(t)]$

*The measure is valid for models with absorbing states.

N: non-absorbing states; A: absorbing states; U: up states; D: down states.

Table 2.19 Use of reward rates to compute different measures – part II

Requirement	Reward Assignments	Measure
Reliability [HMT91], [MCT94], [DaBh85]	$r_i = \begin{cases} 1 & \text{if } i \in N \\ 0 & \text{else} \end{cases}$ [Table 2.5]	$E[Z(t)]$, $P[Z(t) = 1]$
Unreliability [STP96]	$r_i = \begin{cases} 1 & \text{if } i \in A \\ 0 & \text{else} \end{cases}$	$E[Z(t)]$, $P[Z(t) = 1]$
System MTTF [HMT91]	$r_i = \begin{cases} 1 & \text{if } i \in N \\ 0 & \text{else} \end{cases}$ [Table 2.5]	$E[Y(\infty)]^*$
Expected number of catastrophic events in $[0, t)$ [HMT91]	$r_i = \begin{cases} \lambda & \text{if } i \in S_C \\ 0 & \text{else} \end{cases}$ (S_C: states where catastrophe may occur, λ: rate of catastrophe) [Table 2.5]	$E[Y(t)]$
Task interruption probability [HMT91], [TSIH90]	$r_i = \begin{cases} P[O_i \leq x_i] & \text{if } i \in U \\ 0 & \text{else} \end{cases}$ (x_i: task execution time in state i, O_i: sojourn time in state i) [Table 2.7, 2.8]	$E[Z]$
Capacity oriented availability [HMT91], [MTD94], [CiGr87]	$r_i = \begin{cases} cap(i) & \text{if } i \in U \\ 0 & \text{else} \end{cases}$ ($cap(i)$: capacity of state i) [Table 2.9]	$E[Z]$, $E[Z(t)]$, $E[Y(t)]$, $\frac{1}{t}E[Y(t)]$ $P[Z(t) = x]$, $P[Z(t) \leq x]$, $P[Y(t) \leq y]$
Total loss probability [MuTr91], [TMWH92], [TSIH90]	$r_i = \begin{cases} 1 & \text{if } i \in D \\ l_i & \text{else} \end{cases}$ (l_i: percentage loss in state i) [Table 2.10]	$E[Z]$, $E[Z(t)]$, $E[Y(t)]$, $\frac{1}{t}E[Y(t)]$, $P[Z(t) = x]$, $P[Z(t) \leq x]$, $P[Y(t) \leq y]$
Normalized throughput [Meye80], [Meye82], [GaKe79], [TMWH92], [STP96]	$r_i = \begin{cases} 1 - l_i & \text{if } i \in U \\ 0 & \text{else} \end{cases}$ [Table 2.11]	$E[Z]$, $E[Z(t)]$, $E[Y(t)]$, $\frac{1}{t}E[Y(t)]$, $P[Z(t) = x]$, $P[Z(t) \leq x]$, $P[Y(t) \leq y]$

*The measure is valid for models with absorbing states.

Table 2.20 Use of reward rates to compute different measures – part III

Requirement	Reward Assignments	Measure
Approx. total loss [MTD94], [MeSe97]	$r_i = \begin{cases} t\lambda l_i & \text{if } i \in U \\ t\lambda & \text{else} \end{cases}$ [Table 2.13]	$E[Z]$
Loss [HMT91]	$r_i = \begin{cases} \delta l_i & \text{if } i \in U \\ \delta & \text{else} \end{cases}$	$E[Z], E[Z(t)], E[Y(t)]$, $\frac{1}{t}E[Y(t)]$, $P[Z(t) = x]$, $P[Z(t) \leq x]$, $P[Y(t) \leq y]$
Mean number of customers [MTBH93]	$r_i = c(i)$ ($c(i)$: number of customers in state i)	$E[Z]$, $E[Z(t)]$, $P[Z(t) = x]$
Expected throughput [MeSe97], [Doly87], [MTBH93]	$r_i = \begin{cases} \mu_i & \text{if } i \in S_T \\ 0 & \text{else} \end{cases}$ (μ_i: service rate while in state i)	$E[Z], E[Z(t)], E[Y(t)]$, $\frac{1}{t}E[Y(t)], E[Y(\infty)]^*$, $P[Z(t) = x]$, $P[Z(t) \leq x]$, $P[Y(\infty) \leq y]^*$, $P[Y(t) \leq y]$
Utilization [Husl81], [GoTa87], [MTBH93]	$r_i = \begin{cases} 0 & \text{if resource is idle} \\ 1 & \text{else} \end{cases}$	$E[Z(t)]$, $\frac{1}{t}E[Y(t)]$, $E[Z], P[\frac{1}{t}Y(t) \leq y]$

*The measure is valid for models with absorbing states

To refer to the number of tokens at a particular place $P_i, 1 \leq i \leq |P|$, in a marking m, the notation $\#(P_i, m)$ is used.

A transition is *enabled* if all of its input places contain at least one token. Note that a transition having no input arc is always enabled. An enable transition can *fire* by removing from each input place one token and by adding to each output place one token. The firing of a transition may transform a PN from one marking into another. With respect to a given *initial* marking, the *reachability set*, $\mathcal{RS}$, is defined as the set of all markings reachable through any possible firing sequences of transitions, starting from the initial marking. The $\mathcal{RS}$ could be infinite if no further restrictions are imposed. Even PNs appearing as being relatively simple can give rise to large or infinite $\mathcal{RS}$s. If more than one transition is simultaneously enabled but cannot fire in parallel, a *conflict* between transitions arises that needs to be resolved through some selection policy, for example, based on a priority scheme or according to a given pmf. Markings in which no transition is enabled are called *absorbing* markings.

For a graphical presentation, places are usually depicted as circles and transitions as bars. A simple PN with an initial marking:

$$m_0 = (2, 0, 0, 0)\,,$$

and subsequent marking:

$$m_1 = (1, 1, 1, 0)\,,$$

reachable through firing of transition t_1, is shown in Fig. 2.11.

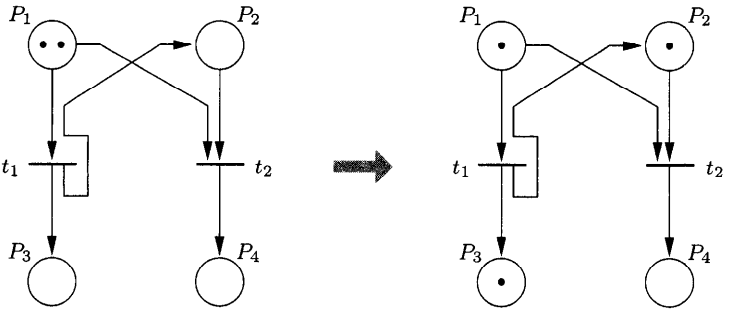


Fig. 2.11 A simple PN before and after the firing of transition t_1.

Transition t_1 is enabled because its single input place P_1 contains two tokens. Since P_2 is empty, transition t_2 is disabled. When firing takes place, one token is removed from input place P_1 and one token is deposited in both output places P_2 and P_3. In the new marking, both transitions t_1 and t_2 are enabled. The apparent conflict between t_1 and t_2 must be solved to decide which of the alternative (absorbing) markings will be reached:

$$m_2 = (0, 2, 2, 0) \quad \text{or}$$
$$m_3 = (0, 0, 1, 1)\,.$$

The reachability set in this example is given by $\{m_0, m_1, m_2, m_3\}$, where the markings are given as defined earlier.

Definition 2.16 A PN is given by a 5-tuple $PN = \{P, T, D^-, D^+, m_0\}$ with:

- A finite set of places $P = \{P_1, P_2, \ldots, P_{|P|}\}$, where each place contains a non-negative number of tokens.
- A finite set of transitions $T = \{t_1, \ldots, t_{|T|}\}$, such that $T \cap P = \emptyset$.
- The set of input arcs $D^- \in \{0,1\}^{|P \times T|}$ and the set of output arcs $D^+ \in \{0,1\}^{|P \times T|}$. If $D^-(P_i, t_k) = 1$, then there is an input arc from place P_i to transition t_k, and if $D^+(P_j, t_l) = 1$, then there is an output arc from transition t_l to place P_j.
- The initial marking $m_0 \in \mathcal{M}$.

2.2.3.2 Generalized Stochastic Petri Nets GSPNs generalize PNs in such a way that each transition has a *firing time* assigned to it, which may be exponentially distributed or constant zero. Transitions with exponentially distributed firing times are called *timed* transitions, while the others are referred to as *immediate* transitions. The new type of transitions require enabling and firing rules to be adapted accordingly.

The markings $\mathcal{M} = \mathcal{V} \cup \mathcal{T}$ in the reachability set $\mathcal{RS}$ of a GSPN are partitioned into two sets, the *vanishing* markings $\mathcal{V}$ and the *tangible* markings $\mathcal{T}$. Vanishing markings comprise those in which *at least one* immediate transition is enabled. If no immediate transition is enabled, that is, only timed transitions or no transitions are enabled, a tangible marking results. Vanishing markings are not resided in for any finite non-zero time and firings are performed instantaneously in this case. Therefore, immediate transitions always have priority over timed transitions to fire. If several immediate transitions compete for firing, a specified pmf is used to break the tie. If timed transitions compete, a *race model* is applied so that the transition whose firing time elapses first is the next one to fire.

From a given GSPN, an *extended reachability graph* ($\mathcal{ERG}$) is generated containing the markings of the reachability set as nodes and some stochastic information attached to the arcs, thereby relating the markings to each other. Absorbing loops of vanishing markings are a priori excluded from the $\mathcal{ERG}$, because a stochastic discontinuity would result otherwise. The GSPNs we are interested in are bounded, i.e., the underlying reachability set is finite. Marsan, Balbo, and Conte proved that exactly one CTMC corresponds to a given GSPN under condition that only a finite number of transitions can fire in finite time with non-zero probability [ABC84].

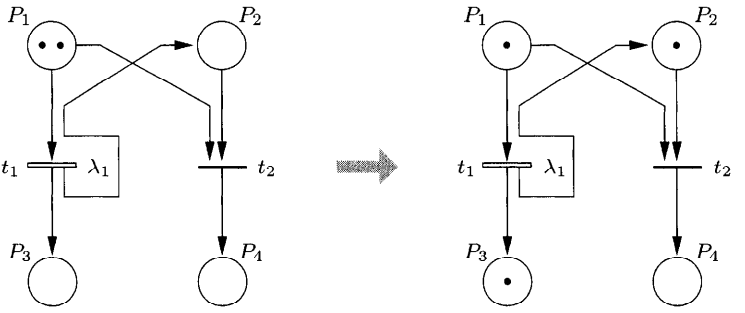


Fig. 2.12 A simple GSPN before and after the firing of transition t_1.

Thick bars are used to represent timed transitions graphically, while thin bars are reserved for immediate transitions. An example of a GSPN is shown in Fig. 2.12. In contrast to the PN from Fig. 2.11, an exponentially distributed firing time with rate λ_1 has been assigned to transitions t_1. While in the PN of the right part in Fig. 2.11 both transitions t_1 and t_2 were enabled, only

immediate transition t_2 is enabled in GSPN of the right part in Fig. 2.12, because immediate transitions have priority over timed transitions.

2.2.3.3 Stochastic Reward Nets Many proposals have been launched to provide extensions to GSPNs (PNs). Some of the most prominent ones are revisited in this section: arc multiplicity, inhibitor arcs, priorities, guards, marking-dependent arc multiplicity, marking-dependent firing rates, and reward rates defined at the net level. With these extensions we obtain the formalism of stochastic reward nets. We note that the first three of these extensions are already present in the GSPN formalism.

Arc Multiplicity. Frequently more than one token needs to be removed from a place or deposited into a place, which can be easily represented with arc multiplicities. It could be awkward, otherwise, to draw an arc for each token to be moved. Arc multiplicity is a syntactical extension that does not increase the principal modeling power but makes the use of GSPNs more convenient. Arcs with multiplicity are represented by a small number attached to the arc or by a small line cutting through the arc. By default, a multiplicity of 1 is assumed. An example of arcs with multiplicities is depicted in Fig. 2.13.

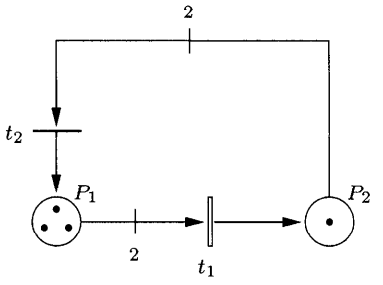


Fig. 2.13 Simple example of multiple arcs.

Since there are initially three tokens in place P_1, transition t_1 is enabled, and after elapsing of the firing time, two tokens are removed from place P_1 (multiplicity 2) and one token is deposited into place P_2. No other transition is enabled in the initial marking. Since in the new marking there are two tokens in P_2 and one token in P_1, only t_2 is enabled and fires immediately, thereby removing two tokens from P_2 (multiplicity 2) and adding one token to P_1. After some exponentially distributed firing time has elapsed, transition t_1 fires for the last time, because an absorbing marking will then be reached. Finally, there will be zero tokens in P_1 and one token in P_2.

Inhibitor Arcs. An inhibitor arc from a place to a transition *disables* the transition in any marking where the number of tokens in the place is equal to or greater than the multiplicity of the inhibitor arc. An inhibitor arc exhibits a complementary semantic to the one of a corresponding ordinary arc

(with the same multiplicity). If a transition were enabled under the conditions imposed by an ordinary arc it would never be enabled under the conditions imposed by its counterpart, an inhibitor arc, and vice versa. Graphically, an inhibitor arc is indicated by a small circle instead of an arrowhead. The use of an inhibitor arc is demonstrated in Fig. 2.14.

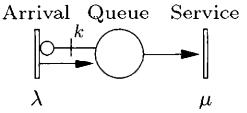


Fig. 2.14 A GSPN with an inhibitor arc having a multiplicity assigned.

The figure entails a queue with finite capacity, which customers enter with a mean interarrival time of $1/\lambda$ as long as the queue is not full, that is, fewer than k customers are already in the system. Arrivals to a full system may not enter and are considered as lost. The arrival transition is disabled in this case. The effective arrival rate, therefore, turns out to be less than λ. Note that we have made use of these kind of scenarios in earlier sections when specifying reward assignments capturing throughput losses or related performance requirements. Customers leave the system after receiving service with exponentially distributed length of mean duration $1/\mu$.

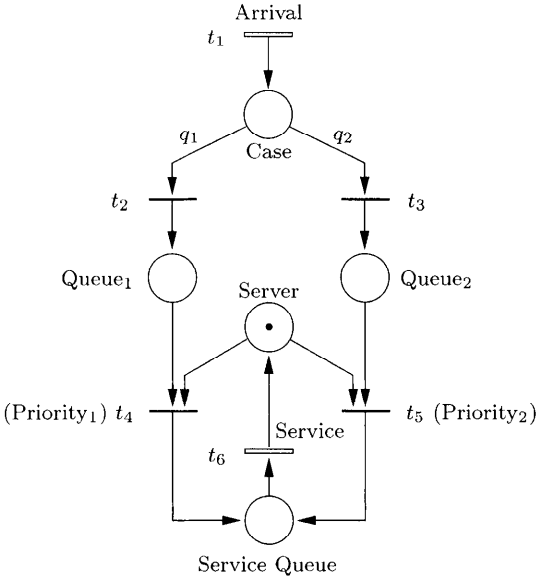


Fig. 2.15 A GSPN with priorities.

Priorities. Although inhibitor arcs can be used to specify priority relations, it is easier if priority assignments are explicitly introduced in the pa-

radigm. Priorities are specified by integer numbers assigned to transitions. A transition is enabled only if no other transition with a higher priority is enabled. In Fig. 2.15 a simple example of a GSPN with priorities is shown. An idle server is assigned with different priority to each type of waiting customers. Arriving customers belong to priority class i with probability $q_i, \sum_i q_i = 1$. Recall that immediate transitions must always have priority over timed tranisitons.

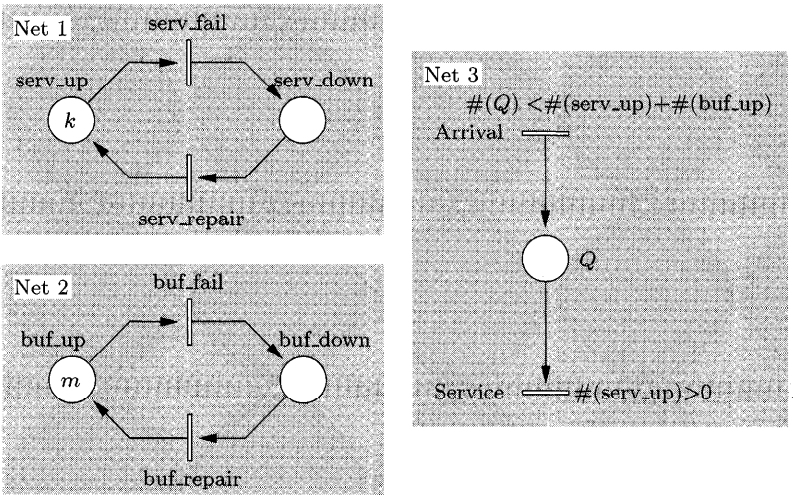


Fig. 2.16 An SRN with guards.

Guards. Guards are general predicates that determine when transitions are to be enabled. This feature provides a powerful means to simplify the graphical representation and to make GSPNs easier to understand, for example, by identifying modular substructures. With these extensions, we have moved out from GSPN class to SRNs [CBC+92]. An example for an SRN with guards is given in Fig. 2.16. A system with two types of resources being subject to failure and repair is represented. The scenario modeled captures the situation of a finite capacity queueing system with finite number of servers (net 1) and additional buffer space (net 2) usable if the active servers are temporarily occupied. The total number of customers in the system is limited by the total of failure-free buffer spots plus the failure-free server units. Hence, arriving customers, which enter with a mean interarrival time $1/\lambda$ (net 3), can only be admitted if fewer customers are already present than given by the current capacity. This condition is described by the guard attached to the "arrival" transition. The guard characterizing the enabling conditions of the "service" transition explicates the fact that service can only be delivered if at least one server is failure free (besides the apparent condition of the presence of at least one customer).

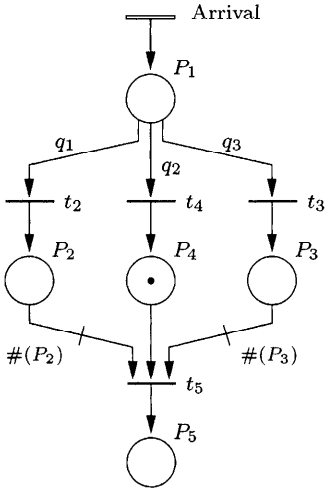


Fig. 2.17 Place flushing modeled with marking-dependent arc multiplicity.

Marking-Dependent Arc Multiplicity. This feature can be applied when the number of tokens to be transferred depends on the current marking. A useful application is given when places need to be "flushed" at the occurrence of a certain event, such as the appearance of a token at a particular place. Such a case is shown in Fig. 2.17. Customers arrive from outside at place P_1 and are immediately forwarded and deposited at places P_2, P_3, or P_4, with probabilities q_1, q_2, and q_3, respectively. Whenever there is at least one token at place P_4, transition t_5 is enabled, regardless of the number of tokens in the other input places P_2 and P_3. When t_5 fires, one token is removed from place P_4, *all* tokens are removed from place P_2 and P_3, and a single token appears in place P_5.

Marking-Dependent Firing Rates. A common extension of GSPNs is to allow marking-dependent firing rates, where a firing rate can be specified as a function of the number of tokens in any place of the Petri net. Restricted variants do exist, where the firing rate of a transition may depend only on the number of tokens in the input places. As an example, the firing rate of transition service in net 3 of Fig. 2.16 is a function of the number of servers currently up, that is, the number of tokens in place serv_up.

Reward Rate Specification. Traditional output measures obtained from a GSPN are the throughput of a transition and the mean number of tokens in a place. Very often it is useful to have more general information such as:

- The probability that a place P_i is empty while the transition t_j is enabled.

- The probability that two different transitions t_1 and t_2 are simultaneously enabled.

Very general reward rates can be specified on an SRN so as to allow the user to compute such custom measures. Examples of the use of reward rates can be found in [CFMT94, MCT94, MMKT94, MaTr93, MuTr91, MuTr92, WLTV96].

2.2.3.4 GSPN/SRN Analysis In this section we introduce the techniques of automated generation of stochastic processes underlying a GSPN/SRN. But it is worthwhile remembering that GSPNs can also be evaluated with discrete-event simulation techniques, a non-state-space based method, and that each technique has its merits.

In general, the analysis of a GSPN/SRN can be decomposed into four subtasks:

1. Generation of the $\mathcal{ERG}$, thereby defining the underlying stochastic process. A semi-Markov process (SMP) results, with zero or exponentially distributed sojourn times. In the case of an SRN, a reward rate for each tangible marking is also generated in this step.

2. Transformation of the SMP into a CTMC by elimination of the vanishing markings with zero sojourn times and the corresponding state transitions.

3. Steady-state, transient, or cumulative transient analyses of the CTMC with methods presented in Chapters 3, 4, and 5.

4. Computation of measures. In the the case of a GSPN, standard measures such as the average number of tokens in each place and the throughput of each timed transition are computed. For the SRN, the specification of reward rates at the net level enables the computation of very general custom measures.

In the $\mathcal{ERG}$, which reflects the properties of the underlying stochastic process, arcs representing the firing of timed transitions are labeled with the corresponding rates, and arcs representing the firing of immediate transitions are marked with probabilities. The $\mathcal{ERG}$ can be transformed into a reduced reachability graph ($\mathcal{RG}$) by eliminating the vanishing markings and the corresponding transitions. Finally, a CTMC will result. In the case of an SRN, a reward vector is also generated and, hence, an MRM is produced.

2.2.3.4.1 $\mathcal{ERG}$ Generation An $\mathcal{ERG}$ is a directed graph with nodes corresponding to the markings (states) of a GSPN/SRN and weighted arcs representing the probabilities or rates with which marking changes occur. The basic $\mathcal{ERG}$ generation algorithm is summarized in Fig. 2.18.

Starting with an initial marking m_0, subsequent markings are generated by systematically following all possible firing patterns. If a new marking is

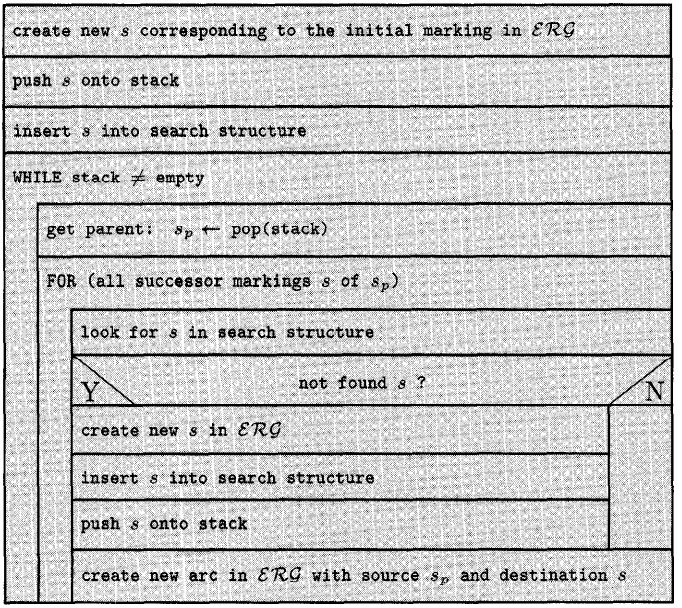


Fig. 2.18 Basic $\mathcal{ERG}$ generation algorithm.

generated, the arcs labeled with the corresponding rates and probabilities are added to the $\mathcal{ERG}$. The algorithm terminates if all possible markings have been generated. Three main data structures are used in this algorithm: a stack (S), the $\mathcal{ERG}$, and a search structure (D).

- **Stack (S):** Newly generated markings are buffered in a stack until they are processed.
- **$\mathcal{ERG}$:** This data structure is used to represent the generated extended reachability graph.
- **Search structure (D):** This data structure contains all earlier generated markings so that a current marking can be checked as to whether it is new or already included in the set D. For efficiency reasons, memory requirements and access time to the data structure should be of concern in an implementation. If the data structure is stored in hash tables, fast access results but memory requirements could be excessive. In contrast, tree structures are more flexible and can be stored more compactly, but access time could be intolerably high for unbalanced trees. Balanced trees such as AVL trees and B-trees are used for this reason [AKH97].

2.2.3.4.2 Elimination of Vanishing Markings The elimination of vanishing markings is an important step to be accomplished for generation of a CTMC from a given GSPN/SRN. Two different techniques can be distinguished,

"elimination on the fly" and "post-elimination" [CMT91]. To give an example, consider the GSPN in Fig. 2.19*a*.

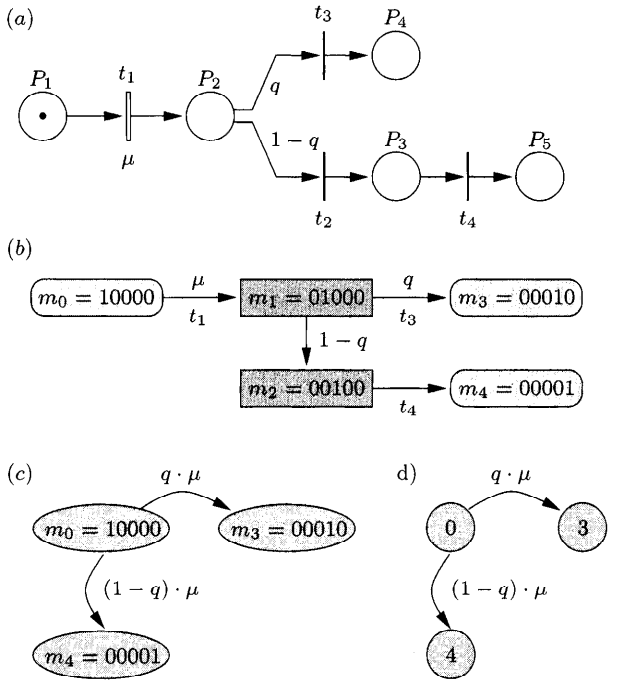


Fig. 2.19 The elimination of vanishing markings demonstrated by: (*a*) a GSPN, (*b*) the underlying $\mathcal{ERG}$, (*c*) the resulting $\mathcal{RG}$, and (*d*) the corresponding CTMC.

The underlying $\mathcal{ERG}$ in Fig. 2.19*b* contains five markings. Tangible markings are represented by oval boxes, while rectangular boxes are reserved for vanishing markings. The elimination of the vanishing markings $(0,1,0,0,0)$ and $(0,0,1,0,0)$ leads to an $\mathcal{RG}$, shown in Fig. 2.19*c*. The $\mathcal{RG}$ is a CTMC (redrawn in Fig 2.19*d*). The elimination of vanishing markings can either be integrated into the generation of the $\mathcal{ERG}$ (elimination on the fly) or performed afterwards (post-elimination). The merits of both methods are discussed in the subsequent subsections.

2.2.3.4.3 Elimination on the Fly Consider the snapshot in Fig. 2.19*b*. Generation of vanishing markings such as $(0,1,0,0,0)$ is avoided by splitting the arc and its attached rate, which would be leading to this vanishing marking, and redirecting the resulting new arcs to subsequent markings. If at least one of these subsequent markings is also of vanishing type, the procedure must be iterated accordingly. Such a situation is indicated in Fig. 2.19*b*. The splitting is accomplished by weighing the rates in relation to the firing probabilities of the immediate transitions (in our example, t_2 and t_3). As a result, only tan-

gible markings and the arcs with adjusted rates need to be stored. Vanishing markings and the associated firing probabilities are discarded immediately. The principle of rate splitting and elimination of vanishing markings is illustrated in Fig. 2.20.

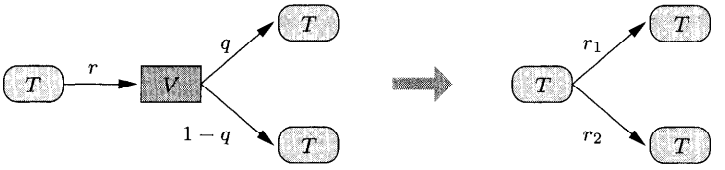


Fig. 2.20 On-the-fly elimination of vanishing markings.

Elimination on the fly is efficient with respect to memory requirements, because vanishing markings need not be stored at all. But these savings in space have to be traded with additional cost in time. The same vanishing markings may be hit several times through the $\mathcal{ERG}$-generation process. The elimination step would be repeated although it had been perfectly executed before. This repetition could amount to a significant wast of execution time.

2.2.3.4.4 Post-Elimination In post-elimination, the complete $\mathcal{ERG}$ is stored during generation. With this technique, it is easier to recognize and resolve (transient) loops consisting of vanishing markings only [CMT90]. Furthermore, no work is duplicated during $\mathcal{ERG}$ generation, and, as an additional advantage, the complete information included in the $\mathcal{ERG}$ is kept for further use. Once the $\mathcal{ERG}$ has been generated, it is transformed into a CTMC by simply applying the same *rate splitting method* as has been described in Section 2.2.3.4.3.

The success of the post-elimination method largely depends on the use of efficient matrix algorithms. Let:

$$\mathbf{P}^{\mathcal{V}} = \left[\mathbf{P}^{\mathcal{VV}} \mid \mathbf{P}^{\mathcal{VT}}\right]$$

denote a matrix, which is split into transition probabilities between vanishing markings only ($\mathbf{P}^{\mathcal{VV}}$) and between vanishing markings and tangible markings ($\mathbf{P}^{\mathcal{VT}}$), where $\mathbf{P}^{\mathcal{V}}$ is of dimension $|\mathcal{V}| \times |\mathcal{M}|$ and the set of markings is partitioned such that $\mathcal{M} = \mathcal{V} \cup \mathcal{T}$. Furthermore, let:

$$\mathbf{U}^{\mathcal{T}} = \left[\mathbf{U}^{\mathcal{TV}} \mid \mathbf{U}^{\mathcal{TT}}\right]$$

denote a matrix, which is split into transition rates from tangible markings to vanishing markings ($\mathbf{U}^{\mathcal{TV}}$) and between tangible markings only ($\mathbf{U}^{\mathcal{TT}}$), where $\mathbf{U}^{\mathcal{T}}$ is of dimension $|\mathcal{T}| \times |\mathcal{M}|$. Precisely the same information as contained in the $\mathcal{ERG}$ is provided by $\mathbf{U}^{\mathcal{T}}$ together with $\mathbf{P}^{\mathcal{V}}$.

With representation in matrix form, it is easy to complete the set of transition rates by formally applying the rate-splitting method as indicated earlier.

The complete set of transition rates, gained after the rate-splitting method has been applied, is summarized in the rate matrix $\mathbf{U}$, which is of dimension $|\mathcal{T}| \times |\mathcal{T}|$ [CMT91]:

$$\mathbf{U} = \mathbf{U}^{\mathcal{TT}} + \mathbf{U}^{\mathcal{TV}} \left(\mathbf{I} - \mathbf{P}^{\mathcal{VV}}\right)^{-1} \mathbf{P}^{\mathcal{VT}}. \tag{2.84}$$

Rate matrix $\mathbf{U} = [u_{ij}]$, which contains all initially present rates between tangible markings plus the derived rates between tangible markings, obtained by applying the rate-splitting method, needs to be completed to derive the infinitesimal generator matrix $\mathbf{Q} = [q_{ij}]$, the entries of which are given by:

$$q_{ij} = \begin{cases} u_{ij} & \text{if } i \neq j, \\ -\sum_{k \in \mathcal{T}, k \neq i} u_{ik} & \text{if } i = j, \end{cases} \tag{2.85}$$

where $\mathcal{T}$ denotes the set of tangible markings. Note that $\mathbf{U}$ may contain non-zero entries on its diagonal, which can be simply ignored as redundant information when creating the infinitesimal generator matrix $\mathbf{Q}$ as shown in Eq. (2.85).

Once the generator matrix has been derived with the described method, numerical solution methods can be applied to yield performance measures as required. Finally, it may be of interest to realize that instead of eliminating the vanishing markings, they can be preserved and the underlying semi-Markov process can be solved numerically [CMT90, MTD94].

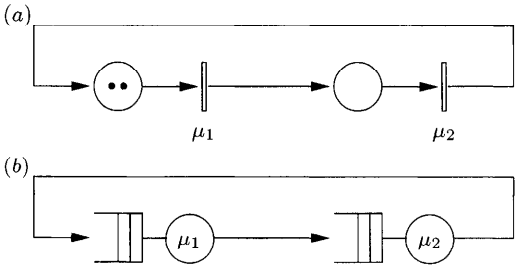


Fig. 2.21 Simple queueing network.

Example 2.13 A simple example is given to demonstrate the generation of an $\mathcal{ERG}$ by applying the algorithm from Fig. 2.18. Consider the GSPN in Fig. 2.21*a* that models a simple network of queues. An equivalent queueing network representation is shown in Fig. 2.21*b* (for a detailed introduction to queueing networks see Chapter 6). The network contains $K = 2$ customers, which are assumed to be initially in node 1. Let (i, j) denote a state of the network, where i represents the number of customers at node 1 and j those at node 2.

```
Algorithm generate_ERG()

initially: S = {}, D = {}, R = {}

create new marking s in ERG: R = {(2,0)}
push s onto the stack: S = {(2,0)}
insert s into the search structure: D = {(2,0)}

WHILE (S != {})
   get parent marking: s_p = pop(S) = {(2,0)},
   R = {(2,0)} S = {}, D = {(2,0)}
   FOR (all successor markings s of s_p)
      s = succ(2,0) = (1,1)
      (not found s in D) = true
         create new marking in ERG: R = {(2,0)(1,1)}
         insert marking into the search structure: D = {(2,0), (1,1)}
         push s onto stack: S = {(1,1)}
      create new arc in ERG: R = {(2,0) --> (1,1)}
   get parent marking: s_p = pop(S) = {(1,1)},
   R = {(2,0) --> (1,1)} S = {}, D = {(2,0), (1,1)}
   FOR (all successor markings s of s_p)
      s = succ(1,1) = (0,2)
      (not found s in D) = true
         create new marking in ERG: R = {(2,0) --> (1,1)(0,2)}
         insert marking into the search structure: D = {(2,0), (1,1), (0,2)}
         push s onto stack: S = {(0,2)}
      create new arc in ERG: R = {(2,0) --> (1,1) --> (0,2)}
      s = succ(1,1) = (2,0)
      (not found s in D) = false
      create new arc in ERG: R = {(2,0) <--> (1,1) --> (0,2)}
   get parent marking: s_p = pop(S) = {(0,2)}
   R = {(2,0) <--> (1,1) --> (0,2)}, S = {}, D = {(2,0), (1,1), (0,2)}
   FOR (all successor markings s of s_p)
     s = succ(0,2) = (1,1)
      (not found s in D) = true
      create new arc in ERG: R = {(2,0) <--> (1,1) <--> (0,2)}

   S == {} ==> the generation algorithm stops
```

The CTMC can be directly derived from the resulting $\mathcal{ERG}$ since there are no vanishing markings in this $\mathcal{ERG}$.

Example 2.14 With the example GSPN from Fig. 2.22*a*, the intermediate steps to create the rate matrix $\mathbf{U}$ and the infinitesimal generator matrix $\mathbf{Q}$ are demonstrated. Post-elimination is assumed so that the complete $\mathcal{ERG}$ is available, as indicated in Fig. 2.22*b*.

The four matrices $\mathbf{P}^{\mathcal{VV}}$, $\mathbf{P}^{\mathcal{VT}}$, $\mathbf{U}^{\mathcal{TV}}$, and $\mathbf{U}^{\mathcal{TT}}$ are given by:

$$\mathbf{P}^{\mathcal{VV}} = \begin{pmatrix} 0 & q_2 \\ 0 & 0 \end{pmatrix}, \qquad \mathbf{P}^{\mathcal{VT}} = \begin{pmatrix} 0 & q_1 & 0 & 0 \\ 0 & 0 & q_4 & q_3 \end{pmatrix},$$

$$\mathbf{U}^{\mathcal{TV}} = \begin{pmatrix} \mu_1 & \mu_2 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \qquad \mathbf{U}^{\mathcal{TT}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu_3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

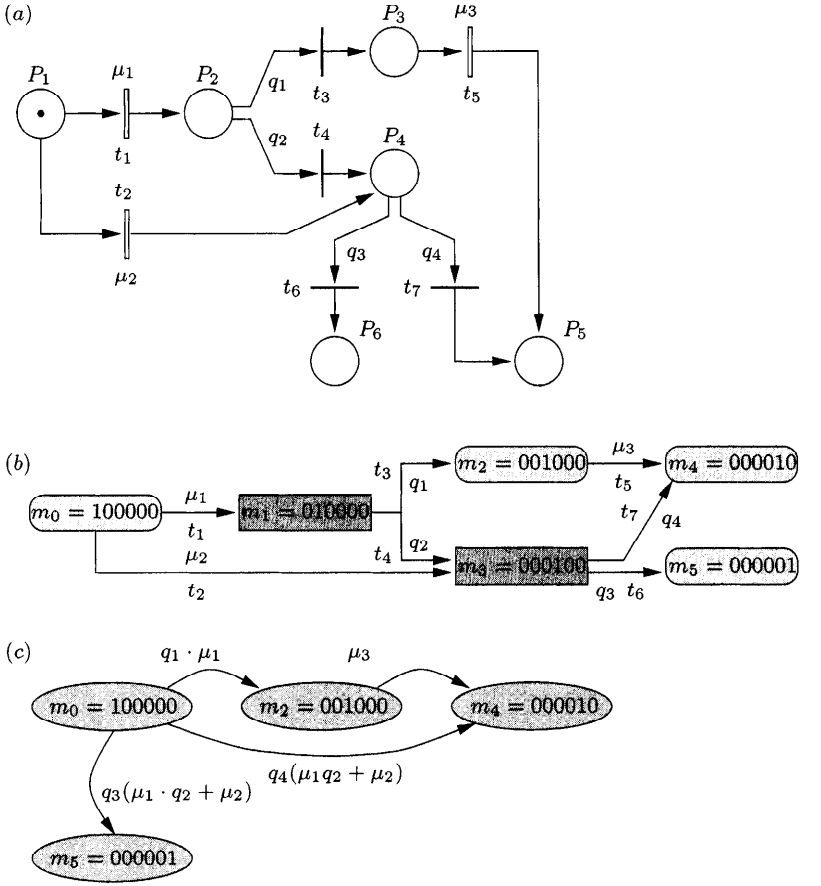


Fig. 2.22 (*a*) Simple GSPN, (*b*) the underlying ERG, and (*c*) the corresponding CTMC.

Then we get:

$$(\mathbf{I}-\mathbf{P}^{\mathcal{VV}})^{-1}=\begin{pmatrix}1 & q_2\\ 0 & 1\end{pmatrix},\qquad \mathbf{U}^{\mathcal{TV}}(\mathbf{I}-\mathbf{P}^{\mathcal{VV}})^{-1}=\begin{pmatrix}\mu_1 & q_2\mu_1+\mu_2\\ 0 & 0\\ 0 & 0\\ 0 & 0\end{pmatrix},$$

$$\mathbf{U}^{\mathcal{TV}}(\mathbf{I}-\mathbf{P}^{\mathcal{VV}})^{-1}\mathbf{P}^{\mathcal{VT}}=\begin{pmatrix}0 & q_1\mu_1 & q_4(q_2\mu_1+\mu_2) & q_3(q_2\mu_1+\mu_2)\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{pmatrix},$$

$$\mathbf{U} = \begin{pmatrix} 0 & q_1\mu_1 & q_4(q_2\mu_1+\mu_2) & q_3(q_2\mu_1+\mu_2) \\ 0 & 0 & 0 & \mu_3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\mathbf{Q} = \begin{pmatrix} -(q_1\mu_1+q_4(q_2\mu_1+\mu_2) & q_1\mu_1 & q_4(q_2\mu_1+\mu_2) & q_3(q_2\mu_1+\mu_2) \\ \quad +q_3(q_2\mu_1+\mu_2)) & & & \\ 0 & -\mu_3 & 0 & \mu_3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

With Eq. (2.85), the entries of the generator matrix $\mathbf{Q}$ can be determined from $\mathbf{U}$.

2.2.3.5 A Larger Example To demonstrate the automated generation of CTMCs from high-level GSPN description, an example of a polling system adapted from [IbTr90] is now presented. (For an introduction to polling systems see [Taka93].)

Consider a two-station single-buffer polling system as modeled by the GSPN shown in Fig. 2.23. The places in the GSPN are interpreted as follows. A token in place P_1 represents the case that station 1 is idle and a token in P_2 correspondingly indicates the idleness of station 2. Tokens in places P_5 (P_6) indicate that the server is serving station 1 (station 2). Tokens appearing in Places P_3 (P_4) indicate that station 1 (station 2) has generated a message. If a token appears in place P_7, the server is polling station 2, while a token in place P_8 indicates polling of station 1. In the situation illustrated in Fig. 2.23, both stations are idle and the server has finished polling station 1 (the server is ready to serve station 1). We assume that the time until station i generates a message is exponentially distributed with mean $1/\lambda_i$. Service and polling times at station i are assumed to be exponentially distributed with mean $1/\mu_i$ and $1/\gamma_i$, respectively ($i = 1, 2$). As soon as station 1 (station 2) generates a message, the timed transition t_1 (t_2) fires.

Consider initial marking m_0 shown in Fig. 2.23, where P_5 contains a token and P_3 contains no token. In this situation the immediate transition s_1 is enabled. It fires immediately and a token is deposited in place P_7. This represents the condition that the server, when arriving at station 1 and finding no message there, immediately proceeds to poll station 2. In the case that a token is found in place P_3, the timed transition t_3 is enabled and its firing causes a token to be deposited in place P_1 and another token to be deposited in place P_7. This represents the condition that station 1 returns to the idle state while the server proceeds to poll station 2.

Let l_i denote the number of tokens in place P_i and $m_j = (l_1, \ldots, l_8)$ denote the jth marking of the GSPN. The $\mathcal{ERG}$, obtained from the initial marking m_0, is shown in Fig. 2.24. Labels attached to directed arcs represents those transitions whose firing generates the successor markings. If a previ-

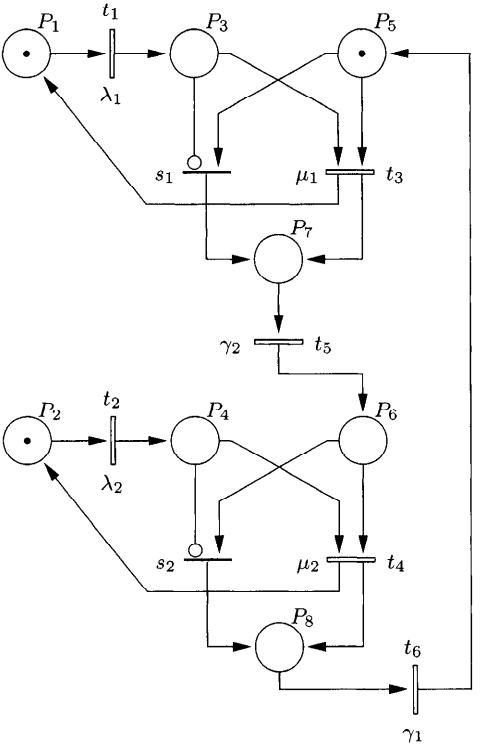


Fig. 2.23 GSPN Model for the two-station single-buffer polling system.

ously found marking is obtained again (like m_3), further generations from such a marking are not required. All sixteen possible and unique markings are included in Fig. 2.24. To reduce the clutter, several markings appear more than once in the figure. The $\mathcal{ERG}$ consists of four vanishing markings $\{m_0, m_4, m_6, m_{13}\}$, represented in the graph by rectangles and tangible markings represented by ovals. We can eliminate the vanishing markings by applying the algorithm presented in Section 2.2.3.4.2.

From Fig. 2.24 we can see that all the tangible markings containing a token in place P_1 indicate the condition that station 1 is idle. The set of markings in which station 1 is idle is $S^{(1)} = \{m_1, m_3, m_7, m_{10}, m_{15}\}$. The set of markings $S^{(2)} = \{m_1, m_2, m_7, m_9, m_{12}\}$ gives the condition that station 2 is idle.

In Fig. 2.25, the finite and irreducible CTMC corresponding to the GSPN of Fig. 2.23 is shown, where each state j corresponds to the tangible marking m_j of the $\mathcal{ERG}$ in Fig. 2.24. With the ergodic CTMC in Fig. 2.25, the steady-state probability vector $\boldsymbol{\pi}$ can be derived according to Eq. (2.58). Letting π_k denote the steady-state probability of the CTMC being in state k and letting $p^{(i)}$ denote the steady-state probability that station $i, i = 1, 2$ is idle,

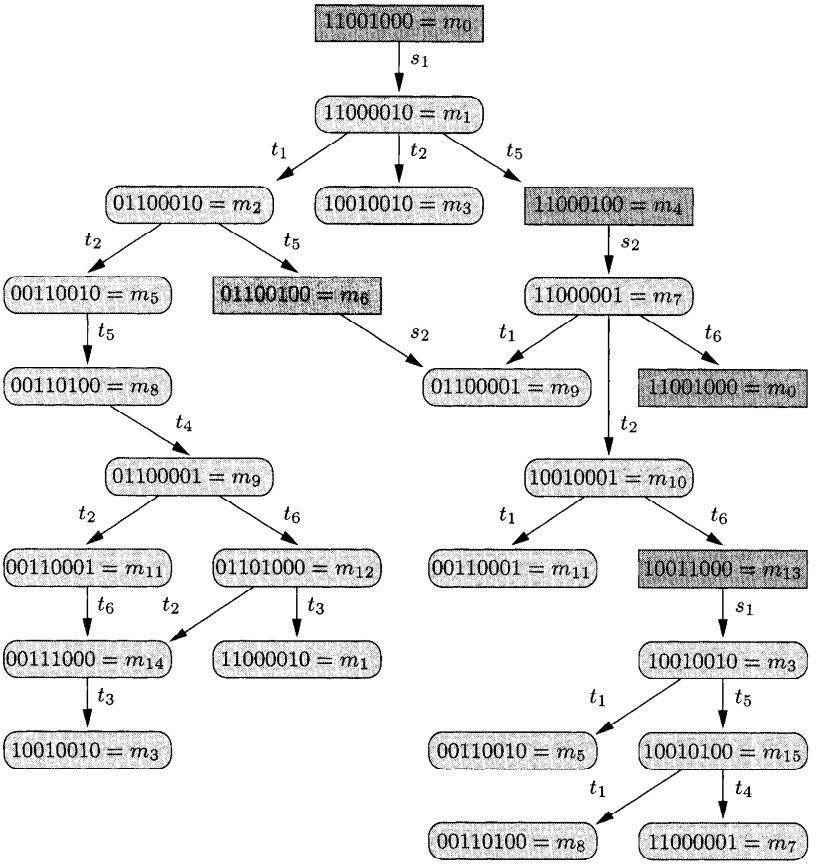


Fig. 2.24 Extended reachability graph for the GSPN Model in Fig. 2.23.

we conclude:

$$\begin{aligned} p^{(1)} &= \pi_1 + \pi_3 + \pi_7 + \pi_{10} + \pi_{15}, \\ p^{(2)} &= \pi_1 + \pi_2 + \pi_7 + \pi_9 + \pi_{12}. \end{aligned} \tag{2.86}$$

As can be seen in Fig. 2.23, the model is structurally symmetric relative to each station and therefore it can easily be extended to more than two stations.

With Eq. (2.86), the mean number of customers $E[K^{(i)}] = \overline{K^{(i)}}$ at single-buffer station i is determined as:

$$\overline{K^{(i)}} = 1 - p^{(i)}.$$

Also with Eq. (2.86), the effective arrival rate at single-buffer station i follows as:

$$\lambda_e^{(i)} = p^{(i)}\lambda.$$

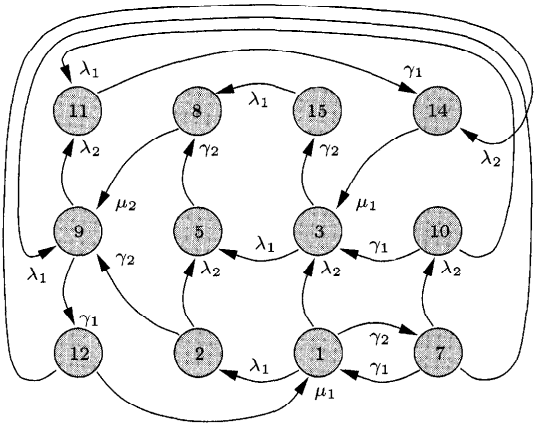


Fig. 2.25 CTMC for the GSPN in Fig. 2.23.

From Little's theorem [Litt61], formulae can be derived for the computation of mean response times of simple queueing models such as the polling system we are investigating in this section. The mean response time $E[T^{(i)}] = \overline{T^{(i)}}$ at station i can be computed as:

$$\overline{T^{(i)}} = \frac{\overline{K^{(i)}}}{\lambda_e^{(i)}}. \tag{2.87}$$

As an extension of the two-station model in Fig. 2.23, let us consider a symmetric polling system with $n = 3, 5, 7, 10$ stations, with $\mu_i = \mu = 1$, $1/\gamma = 1/\gamma_i = 0.005$, and $\lambda_i = \lambda$ for all stations $1 \leq i \leq n$. The offered load is defined by $\rho = \sum_{j=1}^{n} \lambda/\mu = n\lambda/\mu$.

For a numerical computation of steady-state probabilities of CTMCs such as the one depicted in Fig. 2.25, algorithms introduced in Sections 3.3 and 3.4 can be used. In Table 2.21, the mean response times $\overline{T^{(i)}} = \overline{T}$ of single buffer schemes are given for different number of stations. For many other versions of polling system models, see [IbTr90].

Problem 2.1 With Definition 2.64, prove differential Eq. (2.65) by integration of Eq. (2.53) on both sides.

Problem 2.2 With Definition 2.66, prove differential Eq. (2.67).

Problem 2.3 Prove linear Eq. (2.68) by observing $\frac{d}{dt}\mathbf{L}_N(\infty) = \mathbf{0}_N$.

Problem 2.4 Classify the measure mean time to absorption (MTTA) as being of transient, steady-state, or stationary type. Refer back to Eqs. (2.65) and (2.68) and give reasons for your decision.

Problem 2.5 Does $lim_{t\to\infty} E[Z(t)]$ exist if the underlying model has at least one absorbing state? If so, what would this measure mean?

Table 2.21 Mean response times $\overline{T}$ at a station

ρ	3 Stations	5 Stations	7 Stations	10 Stations
0.1	1.07702	1.09916	1.11229	1.12651
0.2	1.14326	1.18925	1.21501	1.24026
0.3	1.20813	1.28456	1.32837	1.37051
0.4	1.27122	1.38418	1.45216	1.51879
0.5	1.33222	1.48713	1.58582	1.68618
0.6	1.39096	1.59237	1.72836	1.87310
0.7	1.44732	1.69890	1.87850	2.07914
0.8	1.50126	1.80573	2.03464	2.30293
0.9	1.55279	1.91201	2.19507	2.54220

Problem 2.6 Specify the reward assignments to model variants one and three in Fig. 2.8 for reliability computations.

Problem 2.7 Compare reliability (unreliability) $R(t)$ $(UR(t))$ functions from Section 2.2.2.3.2 with the computation formula of the distribution of the accumulated reward until absorption, $P[Y(\infty) \leq y]$ in Eq. (2.82) and comment on it.

3

Steady-State Solutions of Markov Chains

In this chapter, we restrict ourselves to the computation of the steady-state probability vector[1] of *ergodic* Markov chains. Most of the literature on solution techniques of Markov chains assumes ergodicity of the underlying model. A comprehensive source on algorithms for steady-state solution techniques is the book by Stewart [Stew94].

From Eq. (2.15) and Eq. (2.58), we have $\boldsymbol{\nu} = \boldsymbol{\nu}\mathbf{P}$ and $\mathbf{0} = \boldsymbol{\pi}\mathbf{Q}$, respectively, as points of departure for the study of steady-state solution techniques. Eq. (2.15) can be transformed so that:

$$\mathbf{0} = \boldsymbol{\nu}(\mathbf{P} - \mathbf{I}). \tag{3.1}$$

Therefore, both for CTMC and DTMC, a linear system of the form:

$$\mathbf{0} = \mathbf{x}\mathbf{A} \tag{3.2}$$

needs to be solved. Due to its type of entries representing the parameters of a Markov chain, matrix $\mathbf{A}$ is singular and it can be shown that $\mathbf{A}$ is of rank $n-1$ for any Markov chain of size $|S| = n$. It follows immediately that the resulting set of equations is not linearly independent and that one of the equations is redundant. To yield a unique, positive solution, we must impose a normalization condition on the solution $\mathbf{x}$ of equation $\mathbf{0} = \mathbf{x}\mathbf{A}$. One way to approach the solution of Eq. (3.2) is to directly incorporate the normalization condition

$$\mathbf{x}\mathbf{1} = 1 \tag{3.3}$$

[1]For the sake of convenience we sometimes use the term 'steady-state analysis' as a shorthand notation.

into the Eq. (3.2). This can be regarded as substituting one of the columns (say, the last column) of matrix $\mathbf{A}$ by the unit vector $\mathbf{1} = [1, 1, \ldots, 1]^T$. With a slight abuse of notation, we denote the new matrix also by $\mathbf{A}$. The resulting linear system of non-homogeneous equations is:

$$\mathbf{b} = \mathbf{x}\mathbf{A}, \quad \mathbf{b} = [0, 0, \ldots, 0, 1]. \tag{3.4}$$

An alternative to solving Eq. (3.2) is to separately consider normalization Eq. (3.3) as an additional step in numerical computations. We demonstrate both ways when example studies are presented. It is worthwhile pointing out that for any given ergodic CTMC, a DTMC can be constructed that yields an identical steady-state probability vector as the CTMC, and vice versa. Given the generator matrix $\mathbf{Q} = [q_{ij}]$ of a CTMC, we can define:

$$\mathbf{P} = \mathbf{Q}/q + \mathbf{I}, \tag{3.5}$$

where q is chosen such that $q > \max_{i,j \in S} |q_{ij}|$. Setting $q = \max_{i,j \in S} |q_{ij}|$ should be avoided in order to assure *aperiodicity* of the resulting DTMC [GLT87]. The resulting matrix $\mathbf{P}$ can be used to determine the steady-state probability vector $\boldsymbol{\pi} = \boldsymbol{\nu}$, by solving $\boldsymbol{\nu} = \boldsymbol{\nu}\mathbf{P}$ and $\boldsymbol{\nu}\mathbf{1} = 1$. This method, used to reduce a CTMC to a DTMC, is called *randomization* or sometimes *uniformization* in the literature. If, on the other hand, a transition probability matrix $\mathbf{P}$ of an ergodic DTMC were given, a generator matrix $\mathbf{Q}$ of a CTMC can be defined by:

$$\mathbf{Q} = \mathbf{P} - \mathbf{I}. \tag{3.6}$$

By solving $\mathbf{0} = \boldsymbol{\pi}\mathbf{Q}$ under the condition $\boldsymbol{\pi}\mathbf{1} = 1$, the desired steady-state probability vector $\boldsymbol{\pi} = \boldsymbol{\nu}$ can be obtained.

To determine the steady-state probabilities of finite Markov chains, three different approaches for the solution of a linear system of the form $\mathbf{0} = \mathbf{x}\mathbf{A}$ are commonly used: *direct* or *iterative numerical* methods and techniques that yield *closed-form* results. Both types of numerical methods have merits of their own. Whereas direct methods yield exact results,[2] iterative methods are generally more efficient, both in time and space. Disadvantages of iterative methods are that for some of these methods no guarantee of convergence can be given in general and that determination of suitable error bounds for termination of the iterations is not always easy. Since iterative methods are considerably more efficient in solving Markov chains, they are commonly used for larger models. For smaller models with fewer than a few thousand states, direct methods are reliable and accurate. Though closed-form results are highly desirable, they can be obtained for only a small class of models that have some structure in their matrix.

[2] Modulo round-off errors resulting from finite precision arithmetic.

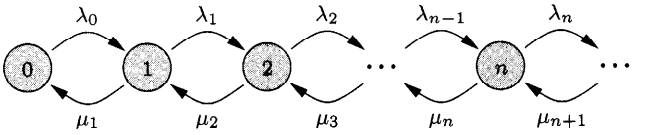


Fig. 3.1 Birth-death process.

Problem 3.1 Show that $\mathbf{P} - \mathbf{I}$ has the properties of a CTMC generator matrix.

Problem 3.2 Show that $\mathbf{Q}/q + \mathbf{I}$ has the properties of a stochastic matrix.

Problem 3.3 Define a CTMC and its generator matrix $\mathbf{Q}$ so that the corresponding DTMC would be periodic if randomization were applied with $q = \max_{i,j \in S} |q_{ij}|$ in Eq. (3.5).

3.1 SYMBOLIC SOLUTION: BIRTH-DEATH PROCESS

Birth-death processes are Markov chains where transitions are allowed only between neighboring states. We treat the continuous time case here, but analogous results for the discrete-time case are easily obtained.

A one-dimensional birth-death process is shown in Fig. 3.1 and its generator matrix is shown as Eq. (3.7):

$$\mathbf{Q} = \begin{pmatrix} -\lambda_0 & \lambda_0 & 0 & 0 & \cdots \\ \mu_1 & -(\lambda_1 + \mu_1) & \lambda_1 & 0 & \cdots \\ 0 & \mu_2 & -(\lambda_2 + \mu_2) & \lambda_2 & \cdots \\ 0 & 0 & \mu_3 & -(\lambda_3 + \mu_3) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{3.7}$$

The transition rates $\lambda_k, k \geq 0$ are state dependent *birth rates* and $\mu_l, l \geq 1$, are referred to as state dependent *death rates*. Assuming ergodicity, the steady-state probabilities of CTMCs of the form depicted in Fig. 3.1 can be uniquely determined from the solution of Eq. (2.58):

$$0 = -\pi_0 \lambda_0 + \pi_1 \mu_1, \tag{3.8}$$

$$0 = -\pi_k(\lambda_k + \mu_k) + \pi_{k-1}\lambda_{k-1} + \pi_{k+1}\mu_{k+1}, \quad k \geq 1. \tag{3.9}$$

Solving Eq. (3.8) for π_1, and then using this result for substitution with $k = 1$ in Eq. (3.9), and solving it for π_2 yields:

$$\pi_1 = \frac{\lambda_0}{\mu_1}\pi_0, \qquad \pi_2 = \frac{\lambda_0 \lambda_1}{\mu_1 \mu_2}\pi_0. \tag{3.10}$$

Eq. (3.10) together with Eq. (3.9) suggest a general solution of the following form:

$$\pi_k = \pi_0 \cdot \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}, \quad k \geq 1. \tag{3.11}$$

Indeed, Eq. (3.11) provides the unique solution of a one-dimensional birth-death process. Since it is not difficult to prove this hypothesis by induction, we leave this as an exercise to the reader. From the law of total probability, $\sum_i \pi_i = 1$, we get for the probability π_0 of the CTMC being in State 0:

$$\pi_0 = \frac{1}{1 + \sum_{k=1}^{\infty} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}} = \frac{1}{\sum_{k=0}^{\infty} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}}. \tag{3.12}$$

The condition for convergence of the series in the denominator of Eq. (3.12), which is also the condition for the ergodicity of the birth-death CTMC, is:

$$\exists k_0, \quad \forall k > k_0 : \quad \frac{\lambda_k}{\mu_k} < 1. \tag{3.13}$$

Eq. (3.11) and Eq. (3.12) are used extensively in Chapter 6 to determine the probabilities π_k for many different queueing systems. These probabilities are then used to calculate performance measures such as mean queue length, or mean waiting time for these queueing systems. We deal with multidimensional birth-death processes in Chapter 7.

Problem 3.4 Consider a discrete-time birth-death process with birth probability b_i, the death probability d_i, and no state change probability $1 - b_i - d_i$ in state i. Derive expressions for the steady-state probabilities and conditions for convergence [Triv82].

3.2 HESSENBERG MATRIX: NON-MARKOVIAN QUEUES

Section 3.1 shows that an infinite state CTMC (or DTMC) with a tridiagonal matrix structure can be solved to obtain a closed-form result. In this section we consider two other infinite state DTMCs. However, the structure is more complex so as to preclude the solution by "inspection" that we adopted in the previous section. Here we use the method of generating functions (or z-transform) to obtain a solution. The problems we tackle originate from non-Markovian queueing systems where the underlying stochastic process is Markov regenerative [Kulk96]. One popular method for the steady-state analysis of Markov regenerative processes is to apply *embedding technique* so as to produce an *embedded* DTMC from the given Markov regenerative process. An alternative approach is to use the *method of supplementary variables* [Hend72]. We follow the embedded DTMC approach.

We are interested in the analysis of a queueing system, where customers arrive according to a Poisson process, so that successive interarrival times are independent, exponentially distributed random variables with parameter λ. The customers experience service at a single server with the only restriction on the service time distribution being that its first two moments are finite. Order of service is first-come-first-served and there is no restriction on the size of the waiting room. We shall see later in Chapter 6 that this is the M/G/1 queueing system. Characterizing the system in such a way that the whole history is summarized in a state, we need to specify the number of customers in the system plus the elapsed service time received by the current customer in service. This description results in a continuous state stochastic process that is difficult to analyze.

But it is possible to identify time instants where the elapsed time is always known so they need not be explicitly represented. A prominent set of these time instants is given by the departure instants, i.e., when a customer has just completed receiving service and before the turn of the next customer has come. In this case, elapsed time is always zero. As a result, a state description given by the number of customers is sufficient. Furthermore, because the service time distribution is known and arrivals are Poissonian, the state transition probabilities can be easily computed. It is not difficult to prove that the stochastic process defined in the indicated way constitutes a DTMC. This DTMC is referred to as *embedded* into the more general continuous state stochastic process.

Conditions can be identified under which the embedded DTMC is ergodic, i.e., a unique steady-state pmf does exist. In Section 3.2.1 we show how the steady-state probability vector of this DTMC can be computed under given constraints. Fortunately, it can be proven that the steady-state pmf of the embedded DTMC is the same as the limiting probability vector of the original non-Markovian stochastic process we started with. The proof of this fact relies on the so-called PASTA theorem, stating that "Poisson arrivals see time averages" [Wolf82]. The more difficult analysis of a stochastic process of non-Markovian type can thus be reduced to the analysis of a related DTMC, yielding the same steady-state probability vector as of the original process.

3.2.1 Non-Exponential Service Times

The service times are given by independent, identically distributed (i.i.d.) random variables and they are independent of the arrival process. Also, the first moment $E[S] = \overline{S}$ and the second moment $E[S^2] = \overline{S^2}$ of the service time S must be finite. Upon completion of service, the customers leave the system.

To define the embedded DTMC $X = \{X_n; n = 0, 1 \ldots\}$, the state space is chosen as the number of customers in the system $\{0, 1, \ldots\}$. As time instants where the DTMC is defined, we select the departure epochs, that is, the points

in time when service is just completed and the corresponding customer leaves the system.

Consider the number of customers $X = \{X_n; n = 0, 1, \ldots\}$ left behind by a departing customer, labeled n. Then the state evolution until the next epoch $n + 1$, where the next customer, labeled $n + 1$, completes service, is probabilistically governed by the Poisson arrivals.

Let the random variable Y describe the number of arrivals during a service epoch. The following one-step state transitions are then possible:

$$X_{n+1} = \begin{cases} X_n + Y - 1, & \text{if } X_n > 0, \\ Y, & \text{if } X_n = 0. \end{cases} \tag{3.14}$$

Let $a_k = P[Y = k]$ denote the probability of k arrivals in a service period with given distribution $B(t)$. If we fix the service time at t, then Y is Poisson distributed with parameter λt. To obtain the unconditional probability a_k, we uncondition using the service time distribution:

$$a_k = \int_0^\infty e^{-\lambda t} \frac{(\lambda t)^k}{k!} dB(t). \tag{3.15}$$

Now the transition probabilities are given as:

$$P[X_{n+1} = j \mid X_n = i] = \begin{cases} a_{j-i+1}, & i > 0, \ j \geq i - 1, \\ a_j, & i = 0, \ j \geq 0. \end{cases} \tag{3.16}$$

The transition probability matrix $\mathbf{P}$ of the embedded stochastic process, which is a Hessenberg matrix, is then given by:

$$\mathbf{P} = \begin{pmatrix} a_0 & a_1 & a_2 & \ldots \\ a_0 & a_1 & a_2 & \ldots \\ 0 & a_0 & a_1 & \ldots \\ 0 & 0 & a_0 & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{3.17}$$

$\mathbf{P}$ can be effectively created, because the arrivals are Poisson and the service time distribution $B(t)$ is known. With $\mathbf{P}$ given as defined in Eq. (3.17), it is not difficult to prove that a DTMC has been defined, i.e., the Markov property from Eq. (2.2) holds. The reader should also verify that this DTMC is aperiodic, and irreducible. It is intuitively clear that State 0 is positive recurrent, if the server can keep up with arrivals, i.e., the mean service time $\overline{S}$ is smaller than than mean interarrival time $1/\lambda$. Equivalently, the mean number of arrivals $E[N] = \overline{N}$ during mean service period $\overline{S}$ should be less than one [Triv82]:

$$\overline{N} = \lambda \overline{S} < 1. \tag{3.18}$$

We know from Section 2.1.2.1 that the states of an irreducible DTMC are all of the same type. Therefore, positive recurrence of state 0 implies positive recurrence of the DTMC constructed in the indicated way. A formal proof of the embedded DTMC being positive recurrent if and only if Relation (3.18) holds has been given, for example, by Cinlar [Cinl75]. We also know from Section 2.1.2.1 that irreducible, aperiodic, and positive recurrent DTMCs are ergodic. Therefore, the embedded DTMC is ergodic if Relation (3.18) holds. Assuming the DTMC to be ergodic, the corresponding infinite set of global balance equations is given as:

$$\begin{aligned} \nu_0 &= \nu_0 a_0 + \nu_1 a_0, \\ \nu_k &= \nu_0 a_k + \sum_{i=1}^{k+1} \nu_i a_{k-i+1}, \quad k \geq 1. \end{aligned} \tag{3.19}$$

To compute the steady-state probability vector $\boldsymbol{\nu}$ of the DTMC, we use the method of generating functions wherein we also need to use the LST of the service time random variable. Let the service times distribution be denoted by $B(t)$ and its LST $B^{\tilde{}}(s)$ be given by:

$$B^{\tilde{}}(s) = \int_0^\infty e^{-st} dB(t). \tag{3.20}$$

Defining generating functions of the state probabilities:

$$G(z) = \sum_{k=0}^{\infty} \nu_k z^k, \tag{3.21}$$

and of the $\{a_j\}$ sequence:

$$G_A(z) = \sum_{j=0}^{\infty} a_j z^j, \tag{3.22}$$

from Eq. (3.19), we have:

$$G(z) = \sum_{k=0}^{\infty} \nu_k z^k = \nu_0 \sum_{k=0}^{\infty} a_k z^k + \sum_{k=0}^{\infty} \sum_{i=1}^{k+1} \nu_i a_{k-i+1} z^k,$$

or:

$$\begin{aligned} G(z) &= \nu_0 G_A(z) + \sum_{i=1}^{\infty} \sum_{k=i-1}^{\infty} \nu_i a_{k-i+1} z^k \\ &= \nu_0 G_A(z) + \sum_{i=1}^{\infty} \sum_{j=0}^{\infty} \nu_i a_j z^{j+i-1} \end{aligned}$$

$$= \nu_0 G_A(z) + \frac{1}{z}\left(\sum_{i=1}^{\infty} \nu_i z^i\right)\left(\sum_{j=0}^{\infty} a_j z^j\right)$$

$$= \nu_0 G_A(z) + \frac{1}{z}(G(z) - \nu_0) G_A(z),$$

or:

$$G(z)\frac{z - G_A(z)}{z} = \nu_0 \frac{(zG_A(z) - G_A(z))}{z},$$

or:

$$G(z) = \nu_0 \frac{zG_A(z) - G_A(z)}{z - G_A(z)}. \tag{3.23}$$

Now, because the arrival process is Poisson, we can obtain the generating function of the $\{a_j\}$ sequence by first conditioning on a fixed service time and then unconditioning with the service time distribution:

$$\begin{aligned} G_A(z) &= \sum_{j=0}^{\infty} a_j z^j = \sum_{j=0}^{\infty} \int_0^{\infty} P[Y = j \mid service = t] dB(t) z^j \\ &= \sum_{j=0}^{\infty} \int_0^{\infty} e^{-\lambda t} \frac{(\lambda t)^j}{j!} dB(t) z^j \\ &= \int_0^{\infty} \left(\sum_{j=0}^{\infty} \frac{(\lambda t z)^j}{j!} e^{-\lambda t}\right) dB(t) \\ &= \int_0^{\infty} e^{-\lambda t + \lambda t z} dB(t) \\ &= \int_0^{\infty} e^{-\lambda t(1-z)} dB(t) \\ &= B^{\sim}(\lambda(1 - z)). \end{aligned} \tag{3.24}$$

Thus, the generating function for the $\{a_j\}$ sequence is given by the LST of the service time distribution at $\lambda(1 - z)$. Hence, we get an expression for the generating function of the DTMC steady-state probabilities:

$$\begin{aligned} C(z) &= \nu_0 \frac{zB^{\sim}(\lambda(1 - z)) - B^{\sim}(\lambda(1 - z))}{z - B\tilde{3}x(\lambda(1 - z))} \\ &= \nu_0 \frac{B^{\sim}(\lambda(1 - z))(z - 1)}{z - B^{\sim}(\lambda(1 - z))}. \end{aligned} \tag{3.25}$$

The generating function in Eq. (3.25) allows the computation of the infinite steady-state probability vector $\boldsymbol{\nu}$ of the DTMC embedded into a continuous-time stochastic process at the departure epochs. These epochs are given by the time instants when customers have just completed their generally distributed service period. The steady-state probabilities are obtained by repeated differentiations of the probability generating function $G(z)$, evaluated at $z = 1$:

$$\nu_i = \frac{1}{i!}\frac{d^i}{dz^i}G(z)\bigg|_{z=1}. \tag{3.26}$$

If we set $z = 1$ on the right hand side of Eq. (3.25) and since $G_A(1) = \sum_{j=0}^{\infty} a_j = 1$, we have $0/0$. Differentiating the numerator and the denominator, as per L'Hospital's rule, we obtain:

$$\begin{aligned} G(1) &= 1 \\ &= \nu_0 \frac{\tilde{B}(\lambda(1-z)) + z(-1)\lambda\tilde{B}'(\lambda(1-z)) + \lambda\tilde{B}'(\lambda(1-z))}{1 + \lambda\tilde{B}'(\lambda(1-z))} \\ &= \nu_0 \frac{\tilde{B}(\lambda(1-z)) + \lambda\tilde{B}'(\lambda(1-z))[1-z]}{1 + \lambda\tilde{B}'(\lambda(1-z))} \\ &= \nu_0 \frac{\tilde{B}(0)}{1 + \lambda\tilde{B}'(0)}. \end{aligned}$$

From Eq. (3.24) it can be easily shown $\tilde{B}(0) = 1$ and $-\tilde{B}'(0) = \overline{S}$, so that we have:

$$1 = \nu_0 \frac{1}{1 - \lambda\overline{S}},$$

or:

$$\nu_0 = 1 - \lambda\overline{S}. \tag{3.27}$$

Taking the derivative of $G(z)$ and setting $z = 1$, we get the expected number of customers in the system, $E[X] = \overline{X}$, in steady state:

$$\overline{X} = \lambda\overline{S} + \frac{\lambda^2\overline{S^2}}{2(1 - \lambda\overline{S})}. \tag{3.28}$$

Equation (3.28) is known as the *Pollaczek-Khintchine* formula. Remember that this formula has been derived by restricting observation to departure epochs. It is remarkable that this formula holds at random observation points in steady state, if the arrival process is Poisson. The proof is based on the PASTA theorem by Wolff, stating that "Poisson arrivals see time averages" [Wolf82].

We refrain from presenting numerical examples at this point, but refer the reader to Chapter 6 where many examples are given related to the stochastic process introduced in this section.

Problem 3.5 Specialize Eq. (3.28) above to:

(a) Exponential service time distribution with parameter μ

(b) Deterministic service time with value $1/\mu$

(c) Erlang service time distribution with mean service time $1/\mu$ and k phases

In the notation of Chapter 6, these three cases correspond to M/M/1, M/D/1, and M/E$_k$/1 queueing systems, respectively.

Problem 3.6 Given a mean interarrival time of 1 second and a mean service time of 2 seconds, compute the steady-state probabilities for the three queueing systems M/M/1, M/D/1, and M/E$_k$/1 to be idle. Compare the results for the three systems and comment.

Problem 3.7 Show that the embedded discrete-time stochastic process $X = \{X_n; n = 0, 1, \ldots\}$ defined at the departure time instants (with transition probability matrix $\mathbf{P}$ given by Eq. (3.17)), forms a DTMC, i.e., it satisfies the Markov property Eq. (2.2).

Problem 3.8 Give a graphical representation of the DTMC defined by transition probability matrix $\mathbf{P}$ in Eq. (3.17).

Problem 3.9 Show that the embedded DTMC $X = \{X_n; n = 0, 1, \ldots\}$ defined in this section is aperiodic and irreducible.

Problem 3.10 Consider a single-server queueing system with independent, exponentially distributed service times. Furthermore, assume an arrival process with independent, identically distributed interarrival times; a general service time distribution is allowed. Service times and interarrival times are also independent. In the notation of Chapter 6, this is the GI/M/1 queueing system.

1. Select a suitable state space and identify appropriate time instants where a DTMC X^* should be embedded into this non-Markovian continuous state process.

2. Define a DTMC $X^* = \{X_n^*; n = 0, 1, \ldots\}$ to be embedded into the non-Markovian continuous state process. In particular, define a DTMC X^* by specifying state transitions by taking Eq. (3.14) as a model. Specify the transition probabilities of the DTMC X^* and its transition probability matrix $\mathbf{P}^*$.

3.2.2 Server with Vacations

Server vacations can be modeled as an extension of the approach presented in Section 3.2.1. In particular, the impact of vacations on the investigated

performance parameters is of additive nature and can be derived with decomposition techniques. Non-Markovian queues with server vacations have proven to be a very useful class of models. They have been applied in a great variety of contexts [Dosh90], some of which are:

- Analysis of *server breakdowns*, which may occur randomly and preempt a customer (if any) in service. Since breakdown (vacation) has priority over customer service, it is interesting to find out how the overall service capacity is affected by such breakdowns. Such insights can be provided through the analysis of queueing systems with vacations.

- Investigation of *maintenance* strategies of computer, communication, or manufacturing systems. In contrast to breakdowns, which occur randomly, maintenance is usually scheduled at certain fixed intervals in order to optimize system dependability.

- Application of *polling* systems or *cyclic server queues*. Different types of polling systems that have been used include systems with *exhaustive service*, *limited service*, *gated service*, or some combinations thereof.

3.2.2.1 Polling Systems Because polling systems are often counted as one of the most important applications of queueing systems with server vacations, some remarks are in order here. While closed-form expressions are derived later in this section, numerical methods for the analysis of polling systems on the basis of GSPNs is covered in Section 2.2.3.

The term *polling* comes from the *polling* data link control scheme in which a central computer interrogates each terminal on a multidrop communication line to find out whether it has data to transmit. The addressed terminal transmits data, and the computer examines the next terminal. Here, the server represents the computer, and a queue corresponds to a terminal.

Basic polling models have recently been applied to analyze the performance of a variety of systems. In the late 1950s, a polling model with a single buffer for each queue was first used in an analysis of a problem in the British cotton industry involving a patrolling machine repairman [MMW57]. In the 1960s, polling models with two queues were investigated for the analysis of vehicle-actuated traffic signal control [Newe69, NeOs69]. There were also some early studies from the viewpoint of queueing theory, apparently independent of traffic analysis [AMM65]. In the 1970s, with the advent of computer communication networks, extensive research was carried out on a polling scheme for data transfer from terminals on multidrop lines to a central computer. Since the early 1980s, the same model has been revived by [Bux81] and others for token passing schemes (e.g., token ring and token bus) in local area networks (LANs). In the current investigation of asynchronous transfer mode (ATM) for broadband ISDN (**i**ntegrated **s**ervices **d**ata **n**etwork), cyclic scheduling is often proposed. Polling models have been applied for scheduling moving arms in secondary storage devices [CoHo86] and for resource arbitration and load

sharing in multiprocessor computers. A great number of applications exist in manufacturing systems, in transportation including moving passengers on circular and on back-and-forth routes, internal mail delivery, and shipyard loading, to mention a few. A major reason for the ubiquity of these applications is that the cyclic allocation of the server (resource) is natural and fair (since no station has to wait arbitrarily long) in many fields of engineering.

The main aim of analyzing polling models is to find the *message waiting time*, defined as the time from the arrival of a randomly chosen message to the beginning of its service. The mean waiting time plus the mean service time is the mean *message response time*, which is the *single most important performance measure* in most computer communication systems. Another interesting characteristic is the *polling cycle time*, which is the time between the server's visit to the same queue in successive cycles. Many variants and related models exist and have been studied. Due to their importance, the following list includes some polling systems of interest:

- Single-service polling systems, in which the server serves only one message and continues to poll the next queue.
- Exhaustive-service polling systems, in which the server serves all the messages at a queue until it is empty before polling the next queue.
- Gated-service polling systems, in which the server serves only those messages that are in the queue at the polling instant before moving to the next queue. In particular, message requests arriving after the server starts serving the queue will wait at the queue until the next time the server visits this queue.
- Mixed exhaustive- and single-service polling systems.
- Symmetric and asymmetric limited-service polling systems, in which the server serves at most $l(i)$ customers in in each service cycle at station i, with $l(i) = l$ for all stations i in the symmetric case.

3.2.2.2 Analysis As in Section 3.2.1, we embed a DTMC into a more general continuous-time stochastic process. Besides non-exponential service times S with its distribution given by $B(t)$ and its LST given by $B^*(s)$, we consider *vacations* of duration V with distribution function $C(t)$ and LST $C^*(s)$, and *rest* periods of the server of duration R, with distribution $D(t)$ and LST $D^*(s)$. Rest periods and vacations are both given by i.i.d. random variables. Finally, the arrivals are again assumed to be Poisson. Our treatment here is based on that given in [King90].

If the queue is inspected and a customer is found to be present in inspection in epoch n, that is $X_n > 0$, the customer is served according to its required service time, followed by a rest period of the server (see Fig. 3.2). Thus, after each service completion the server takes a rest before returning for inspection of the queue. If the queue is found to be empty on inspection, the server takes a

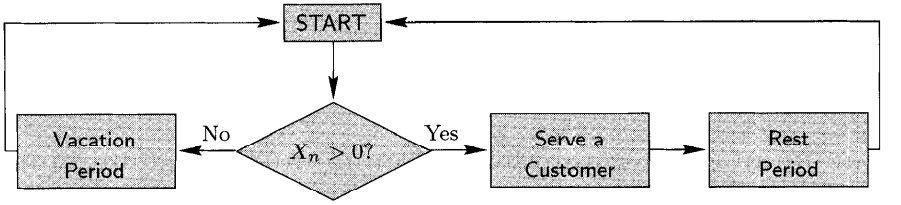


Fig. 3.2 Phases of an M/G/1 queue with vacations.

vacation before it returns for another inspection. We embed a DTMC into this process at the inspection time instants. With E denoting the arrivals during service period including following rest period and F denoting the arrivals during a vacation, Eq. (3.14) describing possible state transitions from state X_n at embedding epoch n to state X_{n+1} at embedding epoch $n+1$ can be restated:

$$X_{n+1} = \begin{cases} X_n + E - 1, & \text{if } X_n > 0, \\ F, & \text{if } X_n = 0. \end{cases} \tag{3.29}$$

With $e_k = P[E = k]$ and $f_l = P[F = l]$ denoting the probabilities of k or l arrivals during the respective time periods, the transition probabilities of the embedded DTMC can be specified in analogy to Eq. (3.16):

$$P[X_{n+1} = j \mid X_n = i] = \begin{cases} e_{j-i+1}, & i > 0, \ j \geq i - 1 \geq 0, \\ f_j, & i = 0, \ j \geq 0. \end{cases} \tag{3.30}$$

The resulting transition probability matrix of the embedded DMTC is given by:

$$\mathbf{P} = \begin{pmatrix} f_0 & f_1 & f_2 & \cdots \\ e_0 & e_1 & e_2 & \cdots \\ 0 & e_0 & e_1 & \cdots \\ 0 & 0 & e_0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{3.31}$$

Note that this transition probability matrix is also in Hessenberg form like the M/G/1 case.

Let $\mathbf{p}$ denote the state probability vector of the embedded DTMC at inspection instants. Then its generating function is given by:

$$G(z) = \sum_{k=0}^{\infty} p_k z^k = p_0 \sum_{k=0}^{\infty} f_k z^k + \sum_{k=0}^{\infty} \sum_{i=1}^{k+1} p_i e_{k-i+1} z^k \tag{3.32}$$

$$= p_0 G_F(z) + \sum_{i=1}^{\infty} \sum_{k=i-1}^{\infty} p_i e_{k-i+1} z^k \tag{3.33}$$

$$= p_0 \frac{z G_F(z) - G_E(z)}{z - G_E(z)}. \tag{3.34}$$

In analogy to Eq. (3.24), the generating functions $G_E(z)$ and $G_F(z)$ can be derived:

$$G_E(z) = \sum_{j=0}^{\infty} e_j z^j = \sum_{j=0}^{\infty} \int_0^{\infty} P[E = j \mid service = t] d[B(t) + D(t)] z^j$$

$$= \sum_{j=0}^{\infty} \int_0^{\infty} e^{-\lambda t} \frac{(\lambda t)^j}{j!} d[B(t) + D(t)] z^j$$

$$= \int_0^{\infty} e^{-\lambda t(1-z)} d[B(t) + D(t)]$$

$$= \tilde{B}(\lambda(1-z)) \tilde{D}(\lambda(1-z)), \tag{3.35}$$

$$G_F(z) = \tilde{C}(\lambda(1-z)). \tag{3.36}$$

With generating functions $G(z)$, $G_E(z)$, and $G_F(z)$ defined, and the transition probability matrix as given in Eq. (3.31), an expression for the probability generation function at inspection time instants can be derived for the server with vacation:

$$G(z) = p_0 \frac{z\tilde{C}(\lambda(1-z)) - \tilde{B}(\lambda(1-z))\tilde{D}(\lambda(1-z))}{z - \tilde{B}(\lambda(1-z))\tilde{D}(\lambda(1-z))}. \tag{3.37}$$

Remembering $G(1) = 1$ and by evaluating Eq. (3.37) at $z = 1$, again by differentiating the denominator and the numerator, we have:

$$p_0 = \frac{1 - \lambda(\overline{S} + \overline{R})}{1 - \lambda(\overline{S} + \overline{R}) + \lambda\overline{V}}. \tag{3.38}$$

So far, we have derived an expression for the probabilities p_i at *inspection* time instants. But to arrive at the steady-state probabilities, we infer the state probabilities ν_n at *departure* instants. Then an inspection at a time instant immediately preceding the departure must find at least one customer present, $i > 0$, with probability p_i, because a departure occurred. With

$a_k = P[Y = k]$ denoting again the probability of k arrivals in a service period, the probabilities ν_n, p_i, and a_k can be related as follows:

$$\nu_n = \sum_{i=1}^{n+1} \frac{p_i a_{n+1-i}}{1 - p_0}.$$

Then, the generating function of the steady-state queue length

$$X(z) = \sum_{j=0}^{\infty} \nu_j z^j$$

is given by:

$$\begin{aligned} X(z) &= \sum_{n=0}^{\infty} z^n \sum_{i=1}^{n+1} \frac{p_i}{1-p_0} a_{n+1-i} = \sum_{i=1}^{\infty} \sum_{n=i-1}^{\infty} z^n \frac{p_i}{1-p_0} a_{n+1-i} \\ &= \frac{1}{1-p_0} \sum_{i=1}^{\infty} \sum_{j=0}^{\infty} z^{i+j-1} p_i a_j = \frac{1}{z(1-p_0)} \sum_{i=1}^{\infty} p_i z^i \sum_{j=0}^{\infty} a_j z^j \\ &= \frac{1}{z(1-p_0)} (G(z) - p_0) G_A(z). \end{aligned} \tag{3.39}$$

Recalling $G_A(z) = B^*(\lambda(1-z))$ and the formula for $G(z)$, we have:

$$\begin{aligned} G(z) &= p_0 \left[\frac{zC^*(\lambda(1-z)) - B^*(\lambda(1-z))D^*(\lambda(1-z))}{z - B^*(\lambda(1-z))D^*(\lambda(1-z))} \right. \\ &\quad \left. - \frac{z - B^*(\lambda(1-z))D^*(\lambda(1-z))}{z - B^*(\lambda(1-z))D^*(\lambda(1-z))} \right] \\ &= p_0 \left[\frac{zC^*(\lambda(1-z)) - z}{z - B^*(\lambda(1-z))D^*(\lambda(1-z))} \right], \end{aligned} \tag{3.40}$$

and:

$$\begin{aligned} X(z) &= \frac{p_0}{z(1-p_0)} \frac{z[C^*(\lambda(1-z)) - 1]B^*(\lambda(1-z))}{z - B^*(\lambda(1-z))D^*(\lambda(1-z))} \\ &= \frac{1 - \lambda E[S+R]}{\lambda E[V]} \frac{[C^*(\lambda(1-z)) - 1]B^*(\lambda(1-z))}{z - B^*(\lambda(1-z))D^*(\lambda(1-z))}. \end{aligned} \tag{3.41}$$

Taking the derivative and setting $z = 1$, we have:

$$\overline{X} = \lambda \overline{S} + \frac{\lambda^2 \overline{(S+R)^2}}{2(1 - \lambda \overline{S+R})} + \frac{\lambda \overline{V^2}}{2\overline{V}}. \tag{3.42}$$

If there is no rest period, that is, $R = 0$, and if the server takes no vacation, that is, $V = 0$, then Eq. (3.42) reduces to the Pollaczek-Khintchine formula

of Eq. (3.28). Furthermore, the average number of arrivals during a vacation, given by the term:

$$\frac{\lambda \overline{V^2}}{2\overline{V}},$$

is simply added to the average number of customers that would be in the system without vacation. With respect to accumulating arrivals, the rest period can simply be considered as an extension of the service time.

Problem 3.11 Give an interpretation of servers with vacation in terms of polling systems as discussed in Section 3.2.2.1. Give this interpretation for all types of polling systems enumerated in Section 3.2.2.1.

Problem 3.12 How can servers with breakdown and maintenance strategies be modeled by servers with vacation as sketched in Fig. 3.2? What are the differences with polling systems?

Problem 3.13 Give the ergodicity condition for an M/G/1 queue with vacation defined according to Fig. 3.2.

Problem 3.14 Derive the steady-state probabilities ν_0 and ν_1 of an M/G/1 queue with vacation. Assume that the following parameters are given: Rest period is constant at 1 second, vacation is a constant 2 seconds, arrival rate $\lambda = 1$, and service time distribution is Erlang k with $k = 2$ and mean $1/\mu = 0.2$ seconds. Check for ergodicity first.

Problem 3.15 Use the same assumptions as specified in Problem 3.14 and, if steady state exists, compute the mean number of customers in system for the following cases:

1. M/G/1 queue with vacation according to Fig. 3.2.
2. Parameters same as in Problem 3.14 above but with rest period being constant at 0 seconds.
3. Parameters same as in Problem 3.15 part 2 above with vacation being also constant at 0 seconds.

3.3 NUMERICAL SOLUTION: DIRECT METHODS

The closed-form solution methods explored in Sections 3.1 and 3.2 exploited special structures of the Markov chain (or, equivalently, of its parameter matrix). For Markov chains with a more general structure, we need to resort to numerical methods. There are two broad classes of numerical methods to solve the linear systems of equations that we are interested in: direct methods and iterative methods. Direct methods operate and modify the parameter

matrix. They use a fixed amount of computation time independent of the parameter values and there is no issue of convergence. But they are subject to fill-in of matrix entries, that is, original zero entries can become non-zeros. This makes the use of sparse storage difficult. Direct methods are also subject the accumulation of round-off errors.

There are many direct methods for the solution of a system of linear equations. Some of these are restricted to certain regular structures of the parameter matrix that are of less importance for Markov chains, since these structures generally cannot be assumed in the case of a Markov chain. Among the techniques most commonly applied are the well known Gaussian elimination (GE) algorithm and, a variant thereof, Grassmann's algorithm. The original version of the algorithm, which was published by Grassmann, Taksar, and Heyman, is usually referred to as the GTH algorithm [GTH85] and is based on a renewal argument. We introduce a newer variant by Kumar, Grassmann, and Billington [KGB87] where interpretation gives rise to a simple relation to the GE algorithm. The GE algorithm suffers sometimes from numerical difficulties created by subtractions of nearly equal numbers. It is precisely this property that is avoided by the GTH algorithm and its variant through reformulations relying on regenerative properties of Markov chains. Cancellation errors are nicely circumvented in this way.

3.3.1 Gaussian Elimination

As the point of departure for a discussion of Gaussian elimination, we refer back to Eq. (3.4). The idea of the algorithm is to transform the system of Eq. (3.43), which corresponds in matrix notation to Eq. (3.4), into an equivalent one by applying elementary operations on the parameter matrix that preserve the rank of the matrix:

$$\begin{aligned} a_{0,0}x_0 + a_{1,0}x_1 + \ldots + a_{n-1,0}x_{n-1} &= b_0, \\ a_{0,1}x_0 + a_{1,1}x_1 + \ldots + a_{n-1,1}x_{n-1} &= b_1, \\ &\vdots \\ a_{0,n-1}x_0 + a_{1,n-1}x_1 + \ldots + a_{n-1,n-1}x_{n-1} &= b_{n-1}. \end{aligned} \tag{3.43}$$

As a result, an equivalent system of linear equations specified by Eq. (3.44) with a triangular matrix structure is derived, from which the desired solution $\mathbf{x}$, which is identical to the solution of the original system given by Eq. (3.43), can be obtained:

$$\begin{aligned} a_{0,0}^{(n-1)}x_0 &= b_0^{(n-1)}, \\ a_{0,1}^{(n-2)}x_0 + a_{1,1}^{(n-2)}x_1 &= b_1^{(n-2)}, \\ &\vdots \\ a_{0,n-1}^{(0)}x_0 + a_{1,n-1}^{(0)}x_1 + \ldots + a_{n-1,n-1}^{(0)}x_{n-1} &= b_{n-1}^{(0)}. \end{aligned} \tag{3.44}$$

If the system of linear equations has been transformed into a triangular structure, as indicated in Eq. (3.44), the final results can be obtained by means of a straightforward substitution process. Solving the first equation for x_0, substituting the result in the second equation and solving it for x_1, and so on, finally leads to the calculation of x_{n-1}. Hence, the x_i are recursively computed according to Eq. (3.45):

$$\begin{aligned} x_0 &= \frac{b_0^{(n-1)}}{a_{0,0}^{(n-1)}}, \\ x_j &= \frac{b_j^{(n-j)}}{a_{j,j}^{(n-j)}} - \sum_{k=0}^{j-1} \frac{a_{k,j}^{(n-j)}}{a_{j,j}^{(n-j)}} x_k, \quad j = 1, 2, \ldots, n-1. \end{aligned} \tag{3.45}$$

To arrive at Eq. (3.44), an elimination procedure first needs to be performed on the original system of Eq. (3.43). Informally, the algorithm can be described as follows: first the nth equation of Eq. (3.43) is solved for x_{n-1}, and then x_{n-1} is eliminated from all other $n-1$ equations. Next, the $(n-1)$th equation is used to solve for x_{n-2}, and, again, x_{n-2} is eliminated from the remaining $n-2$ equations, and so forth. Finally, Eq. (3.44) results, where $a_{i,j}^{(k)}$ denotes the coefficient of x_i in the $(j+1)$th equation, obtained after the kth elimination step.[3] For the sake of completeness it is pointed out that $a_{i,j}^{(0)} = a_{i,j}$.

More formally, for the kth elimination step, i.e, the elimination of x_{n-k} from equations $j, j = n-k, n-k-1, \ldots, 1$, the $(n-k+1)$th equation is to be multiplied on both sides by:

$$-\frac{a_{n-k,j-1}^{(k-1)}}{a_{n-k,n-k}^{(k-1)}}, \tag{3.46}$$

and the result is added to both sides of the jth equation. The computation of the coefficients shown in the system of Eq. (3.47) and Eq. (3.48) for the kth elimination step is:

$$a_{ij}^{(k)} = \begin{cases} 0, & \begin{array}{l} j = n-k-1, n-k-2, \ldots, 0, \\ i = n-1, n-2, \ldots, n-k, \end{array} \\ a_{ij}^{(k-1)} - a_{i,n-k}^{(k-1)} \dfrac{a_{n-k,j}^{(k-1)}}{a_{n-k,n-k}^{(k-1)}}, & \text{otherwise}, \end{cases} \tag{3.47}$$

$$b_j^{(k)} = b_j^{(k-1)} - b_{n-k}^{(k-1)} \frac{a_{n-k,j}^{(k-1)}}{a_{n-k,n-k}^{(k-1)}}, \quad j = n-k-1, n-k, \ldots, 0. \tag{3.48}$$

[3]Note that in the system of Eq. (3.44) relation $k = n - j \geq 0$ holds.

In matrix notation we begin with the system of equations:

$$(x_0, x_1, \ldots, x_{n-1}) \begin{pmatrix} a_{0,0} & a_{0,1} & \ldots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \ldots & a_{1,n-1} \\ \vdots & \vdots & & \vdots \\ a_{n-1,0} & a_{n-1,1} & \ldots & a_{n-1,n-1} \end{pmatrix} = (b_0, b_1, \ldots, b_{n-1}) .$$

After the elimination procedure, a modified system of equations results, which is equivalent to the original one. The resulting parameter matrix is in upper triangular form, where the parameters of the matrix are defined according to Eq. (3.47) and the vector representing the right-hand side of the equations according to Eq. (3.48):

$$\begin{aligned} &(x_0, x_1, \ldots, x_{n-1}) \begin{pmatrix} a_{0,0}^{(n-1)} & a_{0,1}^{(n-2)} & a_{0,2}^{(n-3)} & \ldots & a_{0,n-1} \\ 0 & a_{1,1}^{(n-2)} & a_{1,2}^{(n-3)} & \ldots & a_{1,n-1} \\ 0 & 0 & a_{2,2}^{(n-3)} & \ldots & a_{2,n-1} \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & \ldots & 0 & a_{n-1,n-1} \end{pmatrix} \\ &= (x_0, x_1, \ldots, x_{n-1}) \begin{pmatrix} u_{0,0} & u_{0,1} & u_{0,2} & \ldots & u_{0,n-1} \\ 0 & u_{1,1} & u_{1,2} & \ldots & u_{1,n-1} \\ 0 & 0 & u_{2,2} & \ldots & u_{2,n-1} \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & \ldots & 0 & u_{n-1,n-1} \end{pmatrix} \\ &= \mathbf{xU} \\ &= \left(b_0^{(n-1)}, b_1^{(n-2)}, \ldots, b_{n-1}\right) . \end{aligned} \tag{3.49}$$

The Gaussian elimination procedure takes advantage of elementary matrix operations that preserve the rank of the matrix. Such elementary operations correspond to interchanging of equations, multiplication of equations by a real-valued constant, and addition of a multiple of an equation to another equation. In matrix terms, the essential part of Gaussian elimination is provided by the factorization of the parameter matrix $\mathbf{A}$ into the components of an *upper triangular matrix* $\mathbf{U}$ and a *lower triangular matrix* $\mathbf{L}$. The elements of matrix $\mathbf{U}$ resulted from the elimination procedure while the entries of $\mathbf{L}$ are the terms from Eq. (3.46) by which the columns of the original matrix $\mathbf{A}$ were multiplied

during the elimination process:

$$\mathbf{A} = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix}$$

$$= \begin{pmatrix} a_{0,0}^{(n-1)} & a_{0,1}^{(n-2)} & a_{0,2}^{(n-3)} & \cdots & a_{0,n-1} \\ 0 & a_{1,1}^{(n-2)} & a_{1,2}^{(n-3)} & \cdots & a_{1,n-1} \\ 0 & 0 & a_{2,2}^{(n-3)} & \cdots & a_{3n} \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & \cdots & 0 & a_{n-1,n-1} \end{pmatrix} \begin{pmatrix} 1 & 0 & \cdots & & 0 \\ \frac{a_{1,0}^{(n-2)}}{a_{1,1}^{(n-2)}} & 1 & 0 & \cdots & 0 \\ \vdots & & 1 & 0 & \vdots \\ \frac{a_{n-2,0}^{(1)}}{a_{n-2,n-2}^{(1)}} & \cdots & \frac{a_{n-2,n-3}^{(1)}}{a_{n-2,n-2}^{(1)}} & 1 & 0 \\ \frac{a_{n-1,0}}{a_{n-1,n-1}} & & \cdots & \frac{a_{n-1,n-2}}{a_{n-1,n-1}} & 1 \end{pmatrix}$$

$$= \mathbf{UL}. \tag{3.50}$$

As a result of the factorization of the parameter matrix $\mathbf{A}$, the computation of the result vector $\mathbf{x}$ can split into two simpler steps:

$$\mathbf{b} = \mathbf{xA} = \mathbf{xUL} = \mathbf{yL}.$$

The solutions of both equations, first of:

$$\mathbf{yL} = \mathbf{b} \tag{3.51}$$

for the vector of unknowns $\mathbf{y}$ and then finally of:

$$\mathbf{xU} = \mathbf{y} \tag{3.52}$$

$$= \left(b_0^{(n-1)}, b_1^{(n-2)}, \ldots, b_{n-1}\right) \tag{3.53}$$

for the vector of unknowns $\mathbf{x}$ is required.

Note that the intermediate result $\mathbf{y}$ from Eq. (3.51), being necessary to compute the final results in Eq. (3.52), is identical to Eq. (3.49) and can readily be calculated with the formulae presented in Eq. (3.48). Since only the coefficients of matrix $\mathbf{U}$ are used in this computation, it is not necessary to compute and to represent explicitly the lower triangular matrix $\mathbf{L}$. It is finally worth mentioning that pivoting is not necessary due to the structure of the underlying generator matrix, which is weakly diagonal dominant, since $|q_{i,i}| \geq q_{i,j}, \forall i, j$. This property is inherited by the parameter matrices.

Now the *Gaussian elimination algorithm* can be summarized as follows:

STEP 1 Construct the parameter matrix $\mathbf{A}$ and the right-side vector $\mathbf{b}$ according to Eq. (3.4) as discussed in Chapter 3.

STEP 2 Carry out elimination steps or, equivalently, apply the standard algorithm to split the parameter matrix $\mathbf{A}$ into upper triangular matrix $\mathbf{U}$ and lower triangular matrix $\mathbf{L}$ such that Eq. (3.50) holds. Note that the parameters of $\mathbf{U}$ can be computed with the recursive formulae in Eq. (3.47) and the computation of $\mathbf{L}$ can be deliberately avoided.

STEP 3 Compute the intermediate results $\mathbf{y}$ according to Eq. (3.51) or, equivalently, compute the intermediate results with the result from Eq. (3.53) according to Eq. (3.48).

STEP 4 Perform the substitution to yield the final result $\mathbf{x}$ according to Eq. (3.52) by recursively applying the formulae shown in Eq. (3.45).

Example 3.1 Consider the CTMC depicted in Fig. 3.3. This simple finite birth-death process is ergodic for any finite λ and μ so that their unique steady-state probabilities can be computed. Since closed-form formulae have

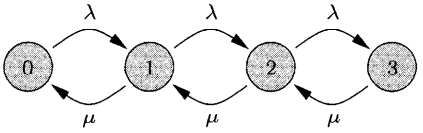


Fig. 3.3 A simple finite birth-death process.

been derived for this case, we can easily compute the steady-state probabilities π_i as summarized in Table 3.1 with $\lambda = 1$ and $\mu = 2$ and with:

$$\pi_0 = \frac{1}{\sum_{i=0}^{3} \left(\frac{\lambda}{\mu}\right)^i}, \quad \pi_k = \pi_0 \left(\frac{\lambda}{\mu}\right)^k, \quad k = 1, 2, 3.$$

Table 3.1 Steady-state probabilities computed using closed-form expressions.

π_0	π_1	π_2	π_3
$\frac{8}{15}$	$\frac{4}{15}$	$\frac{2}{15}$	$\frac{1}{15}$

Alternatively, the state probabilities can be computed applying the Gaussian elimination method introduced in this section. The results from Table 3.1 can then be used to verify the correctness of the results.

STEP 1 First, the generator matrix $\mathbf{Q}$ is derived from Fig. 3.3:

$$\mathbf{Q} = \begin{pmatrix} -1 & 1 & 0 & 0 \\ 2 & -3 & 1 & 0 \\ 0 & 2 & -3 & 1 \\ 0 & 0 & 2 & -2 \end{pmatrix}.$$

In order to include the normalization condition and to derive the parameter matrix $\mathbf{A}$ of the linear system to be solved, the last column of $\mathbf{Q}$ is replaced by the unit vector:

$$\mathbf{A} = \begin{pmatrix} -1 & 1 & 0 & 1 \\ 2 & -3 & 1 & 1 \\ 0 & 2 & -3 & 1 \\ 0 & 0 & 2 & 1 \end{pmatrix}.$$

The resulting system of linear equations is fully specified according to Eq. (3.4), when vector $\mathbf{b} = (0, 0, 0, 1)$ is given.

STEP 2 With Eq. (3.47) and Eq. (3.48) in mind, matrix $\mathbf{A}$ is transformed into upper-triangular matrix $\mathbf{U}$ via intermediate steps $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)} = \mathbf{U}$. In parallel, vector $\mathbf{b}$ is transformed via $\mathbf{b}^{(1)}$ and $\mathbf{b}^{(2)}$ into $\mathbf{b}^{(3)}$, resulting in the following sequence of matrices:

$$\mathbf{A}^{(1)} = \begin{pmatrix} -1 & 1 & -2 & 1 \\ 2 & -3 & -1 & 1 \\ 0 & 2 & -5 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{b}^{(1)} = (0, 0, -2, 1),$$

$$\mathbf{A}^{(2)} = \begin{pmatrix} -1 & \frac{1}{5} & -2 & 1 \\ 2 & -3\frac{2}{5} & -1 & 1 \\ 0 & 0 & -5 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{b}^{(2)} = \left(0, -\frac{4}{5}, -2, 1\right),$$

$$\mathbf{A}^{(3)} = \begin{pmatrix} -\frac{15}{17} & \frac{1}{5} & -2 & 1 \\ 0 & -3\frac{2}{5} & -1 & 1 \\ 0 & 0 & -5 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{b}^{(3)} = \left(-\frac{8}{17}, -\frac{4}{5}, -2, 1\right).$$

STEP 3 For the sake of completeness we also present the lower triangular matrix $\mathbf{L}$ containing the factors from Eq. (3.46) used in the elimination steps:

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{10}{17} & 1 & 0 & 0 \\ 0 & -\frac{2}{5} & 1 & 0 \\ 0 & 0 & 2 & 1 \end{pmatrix}.$$

It can be easily verified that:

$$\mathbf{y} = \mathbf{b}^{(3)} = (-\frac{8}{17}, -\frac{4}{5}, -2, 1)$$

is the intermediate solution vector $\mathbf{y}$ of Eq. (3.51) with given lower triangular matrix $\mathbf{L}$.

STEP 4 The last step is the substitution according to the recursive Eq. (3.45) to yield the final results $\mathbf{x}$ as a solution of Eqs. (3.52) and (3.53):

$$x_0 = \frac{b_0^{(3)}}{a_{0,0}^{(3)}} = \frac{y_0}{u_{0,0}} = \frac{-\frac{8}{17}}{-\frac{15}{17}} = \frac{8}{15},$$

$$\begin{aligned} x_1 &= \frac{b_1^{(2)}}{a_{1,1}^{(2)}} - \frac{a_{0,1}^{(2)}}{a_{1,1}^{(2)}} x_0 = \frac{y_1}{u_{1,1}} - \frac{u_{0,1}}{u_{1,1}} x_0 \\ &= \frac{-\frac{4}{5}}{-\frac{17}{5}} - \frac{\frac{1}{5}}{-\frac{17}{5}} \frac{8}{15} = \frac{4}{15}, \end{aligned}$$

$$\begin{aligned} x_2 &= \frac{b_2^{(1)}}{a_{2,2}^{(1)}} - \frac{a_{0,2}^{(1)}}{a_{2,2}^{(1)}} x_0 - \frac{a_{1,2}^{(1)}}{a_{2,2}^{(1)}} x_1 = \frac{y_2}{u_{2,2}} - \frac{u_{0,2}}{u_{2,2}} x_0 - \frac{u_{1,2}}{u_{2,2}} x_1 \\ &= \frac{-2}{-5} - \frac{-2}{-5} \frac{8}{15} - \frac{-1}{-5} \frac{4}{15} = \frac{2}{15}, \end{aligned}$$

$$\begin{aligned} x_3 &= \frac{b_3}{a_{3,3}} - \frac{a_{0,3}}{a_{3,3}} x_0 - \frac{a_{1,3}}{a_{3,3}} x_1 - \frac{a_{2,3}}{a_{3,3}} x_2 \\ &= \frac{y_3}{u_{3,3}} - \frac{u_{0,3}}{u_{3,3}} x_0 - \frac{u_{1,3}}{u_{3,3}} x_1 - \frac{u_{2,3}}{u_{3,3}} x_2 \\ &= 1 - x_0 - x_1 - x_2 = 1 - \frac{8+4+2}{15} = \frac{1}{15}. \end{aligned}$$

The computational complexity of the Gaussian elimination algorithm can be characterized by $O(n^3/3)$ multiplications or divisions and a storage requirement of $O(n^2)$, where n is the number of states, and hence the number of equations.

Note that cancellation and rounding errors possibly induced by Gaussian elimination can adversely affect the results. This difficulty is specially relevant true if small parameters have to be dealt with, as is often the case with the analysis of large Markov chains, where relatively small state probabilities may result. For some analyses though, such as in dependability evaluation studies, we may be particularly interested in the probabilities of states that are relatively rarely entered, e.g., system down states or unsafe states. In this case, the accuracy of the smaller numbers can be of predominant interest.

3.3.2 The Grassmann Algorithm

Grassmann's algorithm constitutes a numerically stable variant of the Gaussian elimination procedure. The algorithm completely avoids subtractions

and it is therefore less sensitive to rounding and cancellation errors caused by the subtraction of nearly equal numbers. Grassmann's algorithm was originally introduced for the analysis of ergodic, *discrete-time* Markov chains $X = \{X_n; n = 0, 1, \ldots\}$ and was based on arguments from the theory of regenerative processes [GTH85]. A modification of the GTH algorithm has been suggested by Marsan, Meo, and de Souza e Silva [MSA96]. We follow the variant presented by Kumar et al., which allows a straightforward interpretation in terms of continuous-time Markov chains [KGB87].

We know from Eq. (2.41) that the following relation holds:

$$-q_{i,i} = \sum_{j,j\neq i} q_{i,j}. \tag{3.54}$$

Furthermore, Eq. (2.57) can be suitably rearranged:

$$-\pi_i q_{i,i} = \sum_{j=0,j\neq i}^{n-1} \pi_j q_{j,i}. \tag{3.55}$$

Letting $i = n-1$ and dividing Eq. (3.55) on both sides by $q_{n-1,n-1}$ yields:

$$-\pi_{n-1} = \sum_{j=0}^{n-2} \pi_j \frac{q_{j,n-1}}{q_{n-1,n-1}}.$$

This result can be used to eliminate π_{n-1} on the right side of Eq. (3.55):

$$\begin{aligned} -\pi_i q_{i,i} &= \sum_{j=0,j\neq i}^{n-2} \pi_j q_{j,i} - \sum_{j=0}^{n-2} \pi_j \frac{q_{j,n-1}q_{n-1,i}}{q_{n-1,n-1}} \\ &= \sum_{j=0,j\neq i}^{n-2} \pi_j \left(q_{j,i} - \frac{q_{j,n-1}q_{n-1,i}}{q_{n-1,n-1}} \right) - \pi_i \frac{q_{i,n-1}q_{n-1,i}}{q_{n-1,n-1}}. \end{aligned} \tag{3.56}$$

Adding the last term of Eq. (3.56) on both sides of that equation results in an equation that can be interpreted similarly as Eq. (3.55):

$$-\pi_i \left(q_{i,i} - \frac{q_{i,n-1}q_{n-1,i}}{q_{n-1,n-1}} \right) = \sum_{j=0,j\neq i}^{n-2} \pi_j \left(q_{j,i} - \frac{q_{j,n-1}q_{n-1,i}}{q_{n-1,n-1}} \right), \quad 0 \leq i \leq n-2. \tag{3.57}$$

With:

$$\overline{q}_{j,i} = q_{j,i} - \frac{q_{j,n-1}q_{n-1,i}}{q_{n-1,n-1}} = q_{j,i} + \frac{q_{j,n-1}q_{n-1,i}}{\sum_{l=0}^{n-2} q_{n-1,l}},$$

the transition rates of a new Markov chain, having one state less than the original one, are defined. Note that this elimination step, i.e., the computation

of $\overline{q}_{j,i}$, is achieved merely by *adding non-negative* quantities to originally non-negative values $q_{j,i}, j \neq i$. Only the diagonal elements $q_{i,i}$ and $\overline{q}_{i,i}$ are negative. It should be noted that the computation of the rates $\overline{q}_{i,i}$ on the diagonal of the reduced Markov chain can be completely avoided due to the property of Markov chains reflected in Eq. (3.54). In order to assure the $[\overline{q}_{j,i}]$ properly define a CTMC, it remains to be proven that the following equation:

$$\sum_{i=0}^{n-2} \overline{q}_{j,i} = 0,$$

holds for all j:

$$\sum_{i=0}^{n-2} \overline{q}_{j,i} = \sum_{i=0}^{n-2} q_{j,i} + \sum_{i=0}^{n-2} \frac{q_{j,n-1} q_{n-1,i}}{\sum_{l=0}^{n-2} q_{n-1,l}} = \sum_{i=0}^{n-2} q_{j,i} + q_{j,n-1} \sum_{i=0}^{n-2} \frac{q_{n-1,i}}{\sum_{l=0}^{n-2} q_{n-1,l}}$$

$$= \sum_{i=0}^{n-2} q_{j,i} + q_{j,n-1} = \sum_{i=0}^{n-1} q_{j,i} = 0.$$

The new transition rates $\overline{q}_{j,i}$ can be interpreted as being composed of rate $q_{j,i}$, describing direct transitions from state j to state i, and of rate $q_{j,n-1}$ describing transitions from state j via state $n-1$ to state i with conditional branching probability:

$$\frac{q_{n-1,i}}{\sum_{l=0}^{n-2} q_{n-1,l}}.$$

This interpretation implies that no time is spent in (the otherwise eliminated) state $n-1$. The elimination procedure is iteratively applied to the generator matrix with entries $q_{j,i}^{(k)}$ of stepwise reduced state spaces until an upper triangular matrix results, where $q_{j,i}^{(k)}$ denotes the matrix entries after having applied elimination step $k, 1 \leq k \leq n-1$. Finally, each element $q_{i,i}^{(n-1)}$ on the main diagonal is equal to -1.

As usual, the elimination is followed by a substitution process to express the relations of the state probabilities to each other. Since the normalization condition has not been included initially, it must still be applied to yield the final state probability vector. Grassmann's algorithm has been presented in terms of a CTMC generator matrix and, therefore, the parameter matrix must initially be properly defined:

STEP 1

$$\mathbf{A} = \begin{cases} \mathbf{Q}, & \text{for a CTMC} \\ \mathbf{P} - \mathbf{I}, & \text{for a DTMC} \end{cases}$$

STEP 2 For $l = n-1, n-2, \ldots, 1$ do:

$$a_{j,i}^{(n-l)} = \begin{cases} \dfrac{a_{j,i}^{(n-l-1)}}{\sum_{m=0}^{l-1} a_{l,m}^{(n-l-1)}}, & j < l, i = l, \\ a_{j,i}^{(n-l-1)} + \dfrac{a_{j,l}^{(n-l-1)} a_{l,i}^{(n-l-1)}}{\sum_{m=0}^{l-1} a_{l,m}^{(n-l-1)}}, & j \neq i, 1 \leq j, i \leq l-1, \\ -1, & j = i = l, \\ 0, & j = l, i < l.^4 \end{cases}$$

STEP 3 For $l = 1, 2, \ldots, n-1$ do:

$$x_l = \sum_{i=0}^{l-1} x_i a_{il}^{(n-l)}.$$

STEP 4 For $i = 0, 1, \ldots, n-1$ do:

$$\left.\begin{matrix} \pi_i \\ \nu_i \end{matrix}\right\} = \frac{x_i}{\sum_{j=0}^{n-1} x_j}.$$

In matrix notation, the parameter matrix $\mathbf{A}$ is decomposed into factors of an upper triangular matrix $\mathbf{U}$ and a lower triangular matrix $\mathbf{L}$ such that the following equations hold:

$$\mathbf{0} = \mathbf{xA} = \mathbf{xUL}.$$

Of course, any (non-trivial) solution of $\mathbf{0} = \mathbf{xU}$ is also a solution of the original equation $\mathbf{0} = \mathbf{xA}$. Therefore, there is no need to represent $\mathbf{L}$ explicitly.

Example 3.2 Consider the CTMC presented in Fig. 3.4. It represents a finite queueing system with capacity 3, where customers arrive according to independent, exponentially distributed interarrival times with rate λ as long as buffer space is available. Service is received from a single server in two consecutive phases, both phases having mean duration $1/(2\mu)$. The pair of integers (k, l) assigned to a state i represents the number of customers $k(i)$ in the system and the phase $l(i)$ of the customer currently in service. To construct the generator matrix $\mathbf{Q}$, the states are ordered as indicated in Table 3.2.

[4]Setting the elements on the diagonal equal to -1 and below the diagonal equal to 0 is only included here for the sake of completeness. These assignments can be skipped in an actual implementation for efficiency reasons.

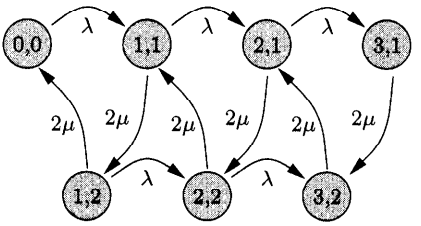


Fig. 3.4 A CTMC for a system with Erlang-2 service time distribution.

Table 3.2 A possible state ordering for Fig. 3.4

(k,l)	$(0,0)$	$(1,1)$	$(2,1)$	$(3,1)$	$(1,2)$	$(2,2)$	$(3,2)$
#	0	1	2	3	4	5	6

Grassmann's algorithm is applied with $\lambda = \mu = 1$:

STEP 1

$$\mathbf{A} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & 1 & 0 & 2 & 0 & 0 \\ 0 & 0 & -3 & 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 & 2 \\ 2 & 0 & 0 & 0 & -3 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 & -3 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 & -2 \end{pmatrix}.$$

STEP 2

$$\mathbf{A}^{(1)} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & 1 & 0 & 2 & 0 & 0 \\ 0 & 0 & -3 & 1 & 0 & 2 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & -3 & 1 & 0 \\ 0 & 2 & 1 & 0 & 0 & -3 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix},$$

$$\mathbf{A}^{(2)} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & 1 & 0 & 2 & 0 & 0 \\ 0 & \frac{4}{3} & -3 & 1 & 0 & \frac{2}{3} & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 1 \\ 2 & \frac{2}{3} & \frac{1}{3} & 0 & -3 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix},$$

$$\mathbf{A}^{(3)} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{4}{3} & -3 & \frac{11}{9} & 0 & \frac{2}{3} & 0 & 0 \\ 0 & \frac{4}{3} & -3 & 1 & 0 & \frac{2}{3} & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix},$$

$$\mathbf{A}^{(4)} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{4}{3} & -3 & \frac{11}{9} & 0 & \frac{2}{3} & 0 & 0 \\ 0 & \frac{4}{3} & -3 & \frac{1}{2} & 0 & \frac{2}{3} & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix},$$

$$\mathbf{A}^{(5)} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{4}{3} & -3 & \frac{11}{12} & 0 & \frac{2}{3} & 0 & 0 \\ 0 & 0 & -1 & \frac{1}{2} & 0 & \frac{2}{3} & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix},$$

$$\mathbf{A}^{(6)} = \begin{pmatrix} 0 & \frac{3}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & \frac{11}{12} & 0 & \frac{2}{3} & 0 & 0 \\ 0 & 0 & -1 & \frac{1}{2} & 0 & \frac{2}{3} & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}.$$

STEP 3 Since $\mathbf{A}^{(6)}$ is in upper triangular form the solution of $\mathbf{x}\mathbf{A}^{(6)} = \mathbf{0}$ can be easily obtained:

$$x_1 = \frac{3}{4}x_0 \quad x_2 = \frac{11}{12}x_1 \quad x_3 = \frac{1}{2}x_2$$
$$x_4 = \frac{2}{3}x_1 \quad x_5 = \frac{1}{3}x_4 + \frac{2}{3}x_2 \quad x_6 = \frac{1}{2}x_5 + x_3.$$

Because only six equations with seven unknowns were derived, one equation is deliberately added. For convenience, we let $x_0 = \frac{4}{3}$, thereby yielding the intermediate solution:

$$\mathbf{x} = \left(\frac{4}{3}, 1, \frac{11}{12}, \frac{11}{24}, \frac{2}{3}, \frac{5}{6}, \frac{21}{24}\right).$$

STEP 4 With:

$$\frac{1}{\sum_{i=0}^{n-1} x_i} = \frac{12}{73},$$

and with:

$$\pi_i = \frac{x_i}{\sum_{i-0}^{n-1} x_i},$$

the state probability vector $\boldsymbol{\pi}$ results:

$$\boldsymbol{\pi} = (0.2192, 0.1644, 0.1507, 0.0753, 0.1096, 0.1370, 0.1438).$$

Measures of interest could be the average number of customers in the system $E[N] = \overline{N}$:

$$\overline{N} = \sum_{i=0}^{n-1} k(i)\pi_i = \pi_1 + \pi_4 + 2(\pi_2 + \pi_5) + 3(\pi_3 + \pi_6) = 1.507,$$

the effective throughput:

$$\lambda_e = \lambda(\pi_0 + \pi_1 + \pi_2 + \pi_4 + \pi_5) = 0.7809,$$

or the mean response time $E[T] = \overline{T}$:

$$\overline{T} = \frac{\overline{N}}{\lambda_e} = 1.9298.$$

Comparing these results to the measures obtained from an analysis of the birth-death process for this queueing system assuming exponentially (rather than Erlang) distributed service times with same parameters $\lambda = \mu = 1$ shows that response time has decreased and the effective throughput has increased simply by reducing the variance of the service process:

$$\begin{aligned}
\pi_k^{(\mathrm{EXP})} &= \frac{1}{4} = 0.25, \quad k = 0, 1, 2, 3, \\
\overline{N}^{(\mathrm{EXP})} &= (1 + 2 + 3) \cdot \frac{1}{4} = \frac{3}{2} = 1.5, \\
\lambda_e^{(\mathrm{EXP})} &= 3 \cdot \frac{1}{4} = 0.75, \\
\overline{T}^{(\mathrm{EXP})} &= \frac{3}{2} \cdot \frac{4}{3} = 2.
\end{aligned}$$

The computational complexity of Grassmann's algorithm is, similar to Gaussian elimination, $O(n^3/3)$ multiplications or divisions, and the storage

requirement is $O(n^2)$. Sparse storage techniques can be applied under some circumstances, when regular matrix structures, like those given with diagonal matrices, are preserved during computation. Although cancellation errors are being avoided with Grassmann's algorithm, rounding errors can still occur, propagate, and accumulate during the computation. Therefore, applicability of the algorithm is also limited to medium size (around 500 states) Markov models.

3.4 NUMERICAL SOLUTION: ITERATIVE METHODS

The main advantage of iterative methods over direct methods is that they preserve the sparsity the parameter matrix. Efficient sparse storage schemes and efficient sparsity-preserving algorithms can thus be used. Further advantages of iterative methods are based on the property of successive convergence to the desired solution. A good initial estimate can speed up the computation considerably. The evaluation can be terminated if the iterates are sufficiently close to the exact value, i.e., a prespecified tolerance level is not exceeded. Finally, because the parameter matrix is not altered in the iteration process, iterative methods do not suffer from the accumulation of round-off errors. The main disadvantage of iterative methods is that convergence is not always guaranteed and depending on the method, the rate of convergence is highly sensitive to the values of entries in the parameter matrix.

3.4.1 Convergence of Iterative Methods

Convergence is a very important issue for iterative methods that must be dealt with consciously. There are some heuristics that can be applied for choosing appropriate techniques for decisions on convergence, but no general algorithm for the selection of such a technique exists. In what follows we mention a few of the many issues to be considered with respect to convergence. A complete picture is beyond the scope of this text.

A tolerance level ϵ must be specified to provide a measure of how close the current iteration vector $\mathbf{x}^{(k)}$ is to the desired solution vector $\mathbf{x}$. Because the desired solution vector is not known, an *estimate* of the error must be used to determine convergence. Some distance measures are commonly used to evaluate the current iteration vector $\mathbf{x}^{(k)}$ in relation to some earlier iteration vectors $\mathbf{x}^{(l)}, l < k$. If the current iteration vector is "close enough" to earlier ones with respect to ϵ, then this condition is taken as an *indicator* of convergence to the final result. If ϵ were too small, convergence could become very slow or not take place at all. If ϵ were too large, accuracy requirements could be violated or, worse, convergence could be wrongly assumed. Some appropriate norm functions have to be applied in order to compare different iteration vectors. Many such norm functions exist, all having a different impact on speed and pattern of convergence. Size and type of the parameter

matrix should be taken into consideration for the right choice of such a norm function. Concerning the right choice of ϵ and the norm function, it should be further noted that the components x_i of the solution vector can differ by many orders of magnitude from each other in their values. An appropriate definition must take these differences into account and relate them to the accuracy requirements derived from the modeling context.

3.4.2 Power Method

Referring to Eq. (2.15) immediately leads to a first, reliable, though sometimes slowly converging, iterative method for the computation of the steady-state probability vector of finite *ergodic* Markov chains. For the sake of completeness we may mention here that for the power method to converge the transition probability, matrix $\mathbf{P}$ only needs to be *aperiodic*; irreducibility is not necessary. The power method mimics the transient behavior of the underlying DTMC until some stationary, not necessarily steady-state, convergence is reached. Therefore, it can also be exploited as a method for computing the transient state probability vector $\boldsymbol{\nu}(n)$ of a DTMC.

Equation $\boldsymbol{\nu} = \boldsymbol{\nu}\mathbf{P}$ suggests starting with an initial guess of some probability vector $\boldsymbol{\nu}^{(0)}$ and repeatedly multiplying it by the transition probability matrix $\mathbf{P}$ until convergence to $\boldsymbol{\nu}$ is assured, with $\lim_{i\to\infty} \boldsymbol{\nu}^{(i)} = \boldsymbol{\nu}$. Due to the assumed ergodicity, or at least aperiodicity, of the underlying Markov chain, this procedure is guaranteed to converge to the desired fixed point of the unique steady-state probability vector. A single iteration step is as follows:

$$\boldsymbol{\nu}^{(i+1)} = \boldsymbol{\nu}^{(i)}\mathbf{P}, \quad i \geq 0. \tag{3.58}$$

Hence, the iteration vector at step i is related to the initial probability vector via the $\mathbf{P}^i$, the ith power of the transition probability matrix $\mathbf{P}$:

$$\boldsymbol{\nu}^{(i)} = \boldsymbol{\nu}^{(0)}\mathbf{P}^i, \quad i \geq 0. \tag{3.59}$$

The power method is justified by the theory of eigenvalues and eigenvectors. The vector $\boldsymbol{\nu}$ can be interpreted as the left eigenvector $\mathbf{y}_1$ corresponding to the unit eigenvalue $\lambda_1 = 1$, where λ_1 is the dominant eigenvalue, which is the largest of all n eigenvalues, $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$. Since the eigenvectors are mutually orthogonal, $\boldsymbol{\nu}^{(0)}$ can be expressed as a linear combination of all left eigenvectors of $\mathbf{P}$ so that the following relation holds:

$$\boldsymbol{\nu}^{(0)} = c_1\mathbf{y}_1 + c_2\mathbf{y}_2 + \cdots + c_n\mathbf{y}_n, \quad c_j \in \mathbb{R}, \ 1 \leq j \leq n. \tag{3.60}$$

Accordingly, after the first iteration step, the estimate $\boldsymbol{\nu}^{(1)} = \boldsymbol{\nu}^{(0)}\mathbf{P}$ can again be expressed in terms of the eigenvectors $\mathbf{y}_i$ and eigenvalues λ_i:

$$\boldsymbol{\nu}^{(1)} = c_1\lambda_1\mathbf{y}_1 + c_2\lambda_2\mathbf{y}_2 + \cdots + c_n\lambda_n\mathbf{y}_n, \quad c_j \in \mathbb{R}, \ 1 \leq j \leq n. \tag{3.61}$$

Repeatedly applying this procedure yields:

$$\boldsymbol{\nu}^{(i)} = c_1\lambda_1^i\mathbf{y}_1 + c_2\lambda_2^i\mathbf{y}_2 + \cdots + c_n\lambda_n^i\mathbf{y}_n, \quad c_j \in \mathbb{R}, \ 1 \leq j \leq n. \tag{3.62}$$

Since all eigenvalues $|\lambda_j| < 1$ for $j \geq 2$, the ith power of the eigenvalues λ_j, $j \geq 2$, approaches 0 in the limit, $\lim_{i\to\infty} |\lambda_j|^i = 0$. With respect to the unit eigenvalue $|\lambda_1| = 1$ we obtain $|\lambda_1|^i = 1$, for all $i \geq 1$. Therefore, Relation (3.63) holds and proves our assumption of the correct convergence of the power method:

$$\begin{aligned} \lim_{i\to\infty} \boldsymbol{\nu}^{(i)} &= \lim_{i\to\infty} \{c_1\lambda_1^i\mathbf{y}_1 + c_2\lambda_2^i\mathbf{y}_2 + \cdots + c_n\lambda_n^i\mathbf{y}_n\} \\ &= \lim_{i\to\infty} c_1\lambda_1^i\mathbf{y}_1 \\ &= c_1\mathbf{y}_1. \end{aligned} \tag{3.63}$$

Only a renormalization remains to be performed to yield the final result of the steady-state probability vector $\boldsymbol{\nu}$.

An immediate consequence of the nature of the power method is that speed of convergence depends on the relative sizes of the eigenvalues. The closer the non-dominant eigenvalues are to 1, the slower the convergence, as can be concluded from Eq. (3.63). The second largest eigenvalue, $|\lambda_2|$, is the dominating factor in this process, i.e., if it is close to 1, convergence will be slow.

The algorithm of the power method is outlined as follows:

STEP 1 Select q appropriately:

$$\mathbf{A} = \begin{cases} \mathbf{P}, \\ \mathbf{Q}/q + \mathbf{I}; \end{cases}$$
$$\boldsymbol{\nu}^{(0)} = \left(\nu_0^{(0)}, \nu_1^{(0)}, \ldots, \nu_{n-1}^{(0)}\right).$$

Select convergence criterion ϵ, and let $n = 0$. Define some vector norm function $f\left(\|\boldsymbol{\nu}^{(n)}, \boldsymbol{\nu}^{(l)}\|\right)$, $n \geq l$.
Set *convergence* = false.

STEP 2 Repeat until *convergence*:

STEP 2.1 $\boldsymbol{\nu}^{(n+1)} = \boldsymbol{\nu}^{(n)}\mathbf{A}$;

STEP 2.2 IF $f\left(\|\boldsymbol{\nu}^{(n+1)}, \boldsymbol{\nu}^{(l+1)}\|\right) < \epsilon$, $l \leq n$
THEN *convergence* = true;

STEP 2.3 $n = n + 1, l = l + 1$.

STEP 3 $\left.\begin{matrix} \boldsymbol{\pi} \\ \boldsymbol{\nu} \end{matrix}\right\} \approx \boldsymbol{\nu}^{(n)}$.

Example 3.3 Assume three customers circulating indefinitely between two service stations, alternately receiving service from both stations having exponentially distributed service times with parameters μ_1 and μ_2, respectively. Let (i, j) denote a possible state of the system, where $i + j = 3$ and i, j refer

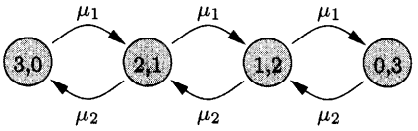


Fig. 3.5 A birth-death model of a two station cyclic queue with three customers.

to the number of customers at the first and the second stations, respectively. The CTMC shown in Fig. 3.5 models this two station cyclic queueing network.

With $\mu_1 = 1$ and $\mu_2 = 2$ the following generator matrix $\mathbf{Q}$ results:

$$\mathbf{Q} = \begin{pmatrix} -1 & 1 & 0 & 0 \\ 2 & -3 & 1 & 0 \\ 0 & 2 & -3 & 1 \\ 0 & 0 & 2 & -2 \end{pmatrix}.$$

Together with an arbitrarily given initial probability vector:

$$\begin{aligned} \pi(0) &= (\pi_{3,0}(0), \pi_{2,1}(0), \pi_{1,2}(0), \pi_{0,3}(0)) \\ &= (0.25, 0.25, 0.25, 0.25), \end{aligned}$$

and, given $q = 3.0000003$, a transition probability matrix:

$$\mathbf{P} = \begin{pmatrix} 0.6666667 & 0.3333333 & 0 & 0 \\ 0.6666666 & 0.0000001 & 0.3333333 & 0 \\ 0 & 0.6666666 & 0.0000001 & 0.3333333 \\ 0 & 0 & 0.6666666 & 0.3333334 \end{pmatrix},$$

the power method can be applied, recalling $\boldsymbol{\nu}^{(0)} = \pi(0)$.

Since we are dealing here with a small ergodic model, Gaussian elimination or Grassmann's algorithm could also be applied to yield the following exact steady-state probability vector:

$$\pi = (0.53\overline{3}, 0.26\overline{6}, 0.13\overline{3}, 0.6\overline{6}).$$

Using this result, the iteration vector can be compared to the exact probability vector π at intermediate steps for the purpose of demonstrating convergence as shown in Table 3.3. The results are gained by choosing $\epsilon = 10^{-7}$ and by applying the following test of convergence, presuming $n \geq 1$:

$$\frac{\|\boldsymbol{\nu}^{(n)} - \boldsymbol{\nu}^{n-1}\|_2}{\|\boldsymbol{\nu}^{n-1}\|_2} < \epsilon.$$

The number of iteration steps necessary to yield convergence to the exact results depends strongly on the uniformization factor q. Some examples demonstrating this dependency are summarized in Table 3.4.

Table 3.3 Intermediate steps and convergence of the power method

$\nu^{(0)}$	0.25	0.25	0.25	0.25
$\nu^{(1)}$	0.333333325	0.25	0.25	0.166666675
$\nu^{(2)}$	0.3888888778	0.2777777722	0.1944444556	0.1388888944
$\nu^{(3)}$	0.4444444278	0.2592592648	0.1851851880	0.1111111194
$\nu^{(4)}$	0.4691357926	0.2716049333	0.1604938370	0.09876543704
$\nu^{(5)}$	0.4938271481	0.2633744897	0.1563786029	0.08641975926
$\nu^{(10)}$	0.5276973339	0.2671002038	0.1357177959	0.06948466636
$\nu^{(20)}$	0.5332355964	0.2666741848	0.1333746836	0.06671553511
$\nu^{(30)}$	0.5333316384	0.2666667970	0.1333340504	0.06666751412
$\nu^{(38)}$	0.5333332672	0.2666666718	0.1333333613	0.06666669973

Table 3.4 Convergence of the power method as a function uniformization factor q

q	Iterations
3.000003	54
3.003	55
3.03	55
3.3	61
6	115
30	566

3.4.3 Jacobi's Method

The system of linear equations of interest to us is:

$$\mathbf{b} = \mathbf{xA}. \tag{3.64}$$

The normalization condition may or may not be incorporated in Eq. (3.64). The parameters of both DTMC and CTMC are given by the entries of the matrix $\mathbf{A} = [a_{ij}]$. The solution vector $\mathbf{x}$ will contain the unconditional state probabilities, or, if not yet normalized, a real-valued multiple thereof. If the normalization is incorporated, we have $\mathbf{b} = [0, 0, \ldots, 0, 1]$, and $\mathbf{b} = \mathbf{0}$ otherwise. Consider the jth equation from the system of Eqs. (3.64):

$$b_j = \sum_{i \in S} a_{ij} x_i. \tag{3.65}$$

Solving Eq. (3.65) for x_j immediately results in:

$$x_j = \frac{b_j - \sum\limits_{i, i \neq j} a_{ij} x_i}{a_{jj}}. \tag{3.66}$$

Any given approximate solution $\hat{\mathbf{x}} = [\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{n-1}]$ can be inserted for the variables $x_i, i \neq j$, on the right side of Eq. (3.66). From these intermediate

values, better estimates of the x_j on the left side of the equation may be obtained. Applying this procedure repeatedly and in parallel for all n equations results in our first iterative method. The values $x^{(k)}$ of the kth iteration step are computed from the values obtained from the $(k-1)$st step for each equation independently:

$$x_j^{(k)} = \frac{b_j - \sum_{i,i \neq j} a_{ij} x_i^{(k-1)}}{a_{jj}}, \quad \forall j \in S. \tag{3.67}$$

The iteration may be started with an arbitrary initial vector $\mathbf{x}^{(0)}$. The closer the initial vector is to the solution, the faster the algorithm will generally converge. Note that the equations can be evaluated in parallel, a fact that can be used as a means for computational speed-up. The method is therefore called the *method of simultaneous displacement* or, simply, the *Jacobi method.* Although the method is strikingly simple, it suffers from poor convergence and hence is rarely applied in its raw form.

Splitting the matrix $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$ into its constituents of the *diagonal* matrix $\mathbf{D}$, the *strictly lower-triangular* matrix $-\mathbf{L}$, and the *strictly upper-triangular* matrix $-\mathbf{U}$ provides a way to present the main computation step of the Jacobi method in matrix notation:

$$\mathbf{x}^{(k)} = \left(\mathbf{b} + \mathbf{x}^{(k-1)} (\mathbf{U} + \mathbf{L})\right) \mathbf{D}^{-1}. \tag{3.68}$$

Before the algorithm is presented, we may recall that we can deliberately incorporate the normalization condition into the parameter matrix $\mathbf{A}$. Note that repeated normalization could have an adverse effect on convergence if large models are investigated, since relatively small numbers could result. Therefore, we leave it open whether to repeat normalization in each iteration step or not.

STEP 1 Define parameter matrix $\mathbf{A}$ and $\mathbf{b}$ properly from generator matrix $\mathbf{Q}$ or transition probability matrix $\mathbf{P}$.

- Choose initial vector $\mathbf{x}^{(0)}$.
- Choose convergence criterion ϵ.
- Choose some norm function $f\left(\|\mathbf{x}^{(k)}, \mathbf{x}^{(l)}\|\right)$, $k \geq l$.
- Split parameter matrix $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$.
- *convergence* = false, and $k = l = 1$.

STEP 2 Repeat until *convergence*:

STEP 2.1 $\mathbf{x}^{(k)} = \left(\mathbf{b} + \mathbf{x}^{(k-1)} (\mathbf{U} + \mathbf{L})\right) \mathbf{D}^{-1}$;

STEP 2.2
- IF $f\left(\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-l)}\|\right) < \epsilon$
  - THEN *convergence* = true

– ELSE $k = k + 1$ and $l \in \{1, \ldots, k\}$.

STEP 3

$$\left.\begin{matrix}\boldsymbol{\pi} \\ \boldsymbol{\nu}\end{matrix}\right\} = \frac{\mathbf{x}^{(k)}}{\sum_{j=0}^{n-1} x_j^{(k)}}.$$

Example 3.4 Consider two customers circulating among three service stations according to the following pattern. When a customer has received service of mean duration $1/\mu_1$ at the first station, it then queues with probability p_{12} at station two for service of mean duration $1/\mu_2$, or with p_{13} at station three with a mean service duration of $1/\mu_3$. After completion of service at stations two or three, customers return with probability one back to station one.

This scenario is captured by a CTMC with the state diagram in Fig. 3.6, where the notation (k, l, m) indicates that there are k customers in station one, l customers in station two, and m customers in station three. We number the states of the CTMC of Fig. 3.6 as per Table 3.5.

Table 3.5 A possible state ordering according to Fig. 3.6

(k,l,m)	$(2,0,0)$	$(1,1,0)$	$(1,0,1)$	$(0,2,0)$	$(0,1,1)$	$(0,0,2)$
#	1	2	3	4	5	6

With $\mu_1 = 1$, $\mu_2 = 2$, $\mu_3 = 3$, $p_{12} = 0.4$, and $p_{13} = 0.6$, the generator matrix $\mathbf{Q}$ can be obtained:

$$\mathbf{Q} = \begin{pmatrix} -1 & 0.4 & 0.6 & 0 & 0 & 0 \\ 2 & -3 & 0 & 0.4 & 0.6 & 0 \\ 3 & 0 & -4 & 0 & 0.4 & 0.6 \\ 0 & 2 & 0 & -2 & 0 & 0 \\ 0 & 3 & 2 & 0 & -5 & 0 \\ 0 & 0 & 3 & 0 & 0 & -3 \end{pmatrix}.$$

To provide a controlled experiment with respect to the iteration process, we first compute the exact state probabilities in Table 3.6 with Grassmann's algorithm introduced in Section 3.3.2. With Table 3.6 we know the exact state probabilities $\mathbf{x} = \boldsymbol{\pi}$ and can use them for comparison with the iteration vector $\mathbf{x}^{(k)}$.

With the following constraints concerning equation vector $\mathbf{b}$, initial probability vector $\mathbf{x}^{(0)}$, normalization after each iteration step, convergence criterion

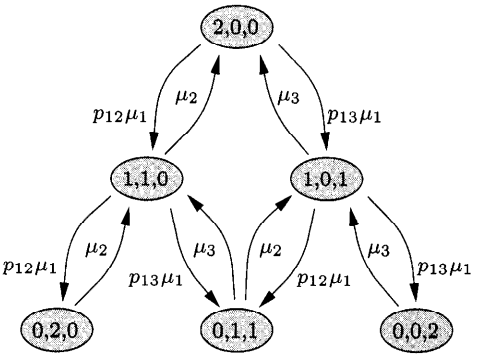


Fig. 3.6 A birth-death model of two customers traversing three stations.

Table 3.6 State probabilities for Fig. 3.6 computed using Grassmann's algorithm

π_0	0.6578947368
π_1	0.1315789474
π_2	0.1315789474
π_3	0.02631578947
π_4	0.02631578947
π_5	0.02631578947

Table 3.7 State probabilities for Fig. 3.6 computed using Jacobi's method

π_0	0.6578947398
π_1	0.1315789172
π_2	0.1315789478
π_3	0.02631578619
π_4	0.02631578846
π_5	0.02631582059

ϵ, and the norm for test of convergence, Jacobi's method is applied:

$$\mathbf{b} = (0,0,0,0,0,0)\,, \quad \mathbf{x}^{(0)} = \left(\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6},\frac{1}{6}\right),$$

$$\mathbf{x}^{(k)} = \frac{\mathbf{x}^{(k)}}{\|\mathbf{x}^{(k)}\|_1}, \quad \epsilon = 10^{-7}, \quad \frac{\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|_2}{\|\mathbf{x}^{(k-1)}\|_2} < \epsilon.$$

The results of using Jacobi algorithm, needing 324 iterations are given in Table 3.7.

The method of Jacobi is of less practical importance due to its slow pattern of convergence. But techniques have been derived to speed up its convergence, resulting in well-known algorithms such as Gauss-Seidel iteration or the successive over-relaxation (SOR) method.

3.4.4 Gauss-Seidel Method

To improve convergence, a given method often needs to be changed only slightly. Instead of deploying full parallelism by evaluating Eq. (3.67) for each state $j \in S$ independently, we can serialize the procedure and take advantage of the already updated new estimates in each step. Assuming the computations to be arranged in the order $0, 1, \ldots, n-1$, where $|S| = n$, it immediately follows,

that for the calculation of the estimates $x_j^{(k)}$, all j previously computed estimates $x_i^{(k)}$, $i < j$, can be used in the computation. Taking advantage of the more up-to-date information, we can significantly speed up the convergence. The resulting method is called the *Gauss-Seidel* iteration:

$$x_j^{(k)} = \frac{b_j - \left(\sum_{i=0}^{j-1} a_{ij} x_i^{(k)} + \sum_{i=j+1}^{n-1} a_{ij} x_i^{(k-1)} \right)}{a_{jj}}, \quad \forall j \in S. \tag{3.69}$$

Note that the order in which the estimates $x_j^{(k)}$ are calculated in each iteration step can have a decisive impact on the speed of convergence. In particular, it is generally a matter of fact that matrices of Markov chains are sparse. The interdependencies between the equations are therefore limited to a certain degree, and parallel evaluation might still be possible, even if the most up-to-date information is incorporated in each computation step. The equations can deliberately be arranged so that the interdependencies become more or less effective for the convergence process. Equivalently, the possible degree of parallelism can be maximized or minimized by a thoughtful choice of that order. Apparently, a trade-off exists between the pattern of convergence and possible speedup due to parallelism. In matrix notation, the Gauss-Seidel iteration step is written as:

$$\mathbf{x}^{(k)} = \left(\mathbf{b} + \mathbf{x}^{(k-1)} \mathbf{L} \right) (\mathbf{D} - \mathbf{U})^{-1}, \quad k \geq 1. \tag{3.70}$$

Equivalently, we could rewrite Eq. (3.70), thereby reflecting the Gauss-Seidel step from Eq. (3.69) more obviously:

$$\mathbf{x}^{(k)} = \left(\mathbf{b} + \mathbf{x}^{(k)} \mathbf{U} + \mathbf{x}^{(k-1)} \mathbf{L} \right) \mathbf{D}^{-1}, \quad k \geq 1. \tag{3.71}$$

3.4.5 The Method of Successive Over-Relaxation

Though Gauss-Seidel is widely applied, further variants of the iteration scheme do exist. A promising approach for an increased rate of convergence is to apply extrapolations via the so-called method of *successive over-relaxation*, SOR for short. The SOR method provides means to weaken or to enhance the impact of a Gauss-Seidel iteration step. A new estimate is calculated by weighting (using a relaxation parameter ω) a previous estimate with the newly computed Gauss-Seidel estimate. The iteration step is presented in matrix notation in Eq. (3.72), from which it can be concluded that the SOR method coincides with Gauss-Seidel if ω is set equal to *one*:

$$\mathbf{x}^{(k)} = \left(\omega \mathbf{b} + \mathbf{x}^{(k-1)} \left(\omega \mathbf{L} + (1 - \omega) \mathbf{D} \right) \right) (\mathbf{D} - \omega \mathbf{U})^{-1}, \quad k \geq 1. \tag{3.72}$$

Equation (3.72) can be easily verified. Remembering the splitting of matrix $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$ and introducing a scalar ω, we can immediately derive the

equation $\omega\mathbf{b} = \mathbf{x}\omega\,(\mathbf{D} - \mathbf{L} - \mathbf{U})$ from Eq. (3.64) as a starting point. Adding $\mathbf{xD}$ to both sides of the equation yields:

$$\omega\mathbf{b} + \mathbf{xD} = \mathbf{x}\omega\,(\mathbf{D} - \mathbf{L} - \mathbf{U}) + \mathbf{xD}. \tag{3.73}$$

By applying simple arithmetic manipulations we get:

$$\omega\mathbf{b} + \mathbf{x}\,(\omega\mathbf{L} + (1-\omega)\,\mathbf{D}) = \mathbf{x}\,(\mathbf{D} - \omega\mathbf{U})\,, \tag{3.74}$$

and, finally, by multiplying both sides of Eq. (3.74) with $(\mathbf{D} - \omega\mathbf{U})^{-1}$, we get Eq. (3.72). To complete the description of the SOR algorithm, we need to focus on the impact of the numerical properties of the iterative method on the convergence process. First of all, a sensible choice of the relaxation parameter ω is crucial for an accelerated convergence. Then we need to carefully specify appropriate criteria to test convergence in each iteration step. It is wise to choose these criteria as a function of the problem domain, i.e., in a model-specific manner. Finally, overflows and underflows have to be dealt with, especially if the investigated models contain states with very small probabilities associated with them. This situation commonly occurs if very large or so-called stiff models are investigated, such that some states have a very small probability mass.

3.4.5.1 The Relaxation Parameter The relaxation parameter ω was introduced to gain an increased speed of convergence. It was mentioned that the incorporation of the normalization condition $\mathbf{x1} = 1$ in each iteration step can have a negative impact on the rate of convergence. From this point of view, normalization should, for practical reasons, be postponed until convergence has been assured. Henceforth, we continue discussion of the iterative method on the basis of a homogeneous linear system $\mathbf{0} = \mathbf{xA}$ according to Eq. (3.64). With this variant in mind, the solution of Eq. (3.72) can be identified as the largest left eigenvector $\mathbf{y}_1$ of the matrix $(\omega\mathbf{L} + (1-\omega)\,\mathbf{D})\,(\mathbf{D} - \omega\mathbf{U})^{-1}$. For the corresponding eigenvalue, we have $|\lambda_1| = 1$. (Note that the normalized eigenvector equals the desired solution vector $\mathbf{x}$.) It is known from [StGo85] that the speed of convergence is a function of the second largest eigenvalue λ_2, with $|\lambda_1| \geq |\lambda_2|$, of the iteration matrix, which can be influenced by an accurate choice of ω. To guarantee convergence, all eigenvalues should be real valued, the second largest eigenvalue $|\lambda_2|$ should be *smaller than one* in magnitude because the smaller $|\lambda_2|$ is, the faster the convergence would be. Conditioned on the assumption that $|\lambda_2| < 1$, which is not always true, the following relation holds for the number of iteration steps i, the second largest eigenvalue $|\lambda_2|$, and the required accuracy ϵ:

$$i \geq \frac{\log(\epsilon)}{\log(|\lambda_2|)}. \tag{3.75}$$

If all the eigenvalues were known, an *optimal* ω_0 that minimizes $|\lambda_2|$ of the iteration matrix could be derived. But this computation is generally

more expensive than the gain due to the implied acceleration of the convergence. Therefore, heuristics are often applied and instead of an exact value ω_0, an approximation $\omega_0^{\approx}$ is derived. Sometimes advantage is taken of certain regularities of the parameter matrix for a simplified estimation $\omega_0^{\approx}$ of ω_0. Equivalently, the parameter matrix can sometimes be rearranged in a problem-dependent way so that a more favorable structure results.

For the SOR iteration method to converge, it has been shown in [HaYo81] that the relaxation parameter must obey the relation $0 < \omega < 2$. It is usually a promising approach to choose ω adaptively by successive reestimations of $\omega_0^{\approx}$. In particular, we follow [StGo85] for such a technique. With the subsequently introduced approach, ω should be chosen such that $\omega \leq \omega_0^{\approx}$, otherwise convergence is not guaranteed because of possible oscillation effects. Furthermore, we assume $\omega_0^{\approx} \geq 1$, i.e., underrelaxation is not considered.

The computation would be initiated with $\omega^{(1)} = 1$, which is fixed for some, say 10, steps of iterations. Using the intermediate estimate $|\lambda_2^{\approx(k)}|$ of the second largest eigenvalue, which is also called the subdominant eigenvalue, in the iteration process an intermediate estimate $\omega_0^{\approx(k)}$ of the optimal relaxation parameter can be derived by evaluating Eq. (3.76):

$$\omega_0^{\approx(k)} = \frac{2}{1 + \sqrt{1 - \left(|\lambda_2^{\approx(k)}|\right)^2}}. \tag{3.76}$$

The estimated subdominant eigenvalue $|\lambda_2^{\approx(k)}|$ is approximated as a function of the recent relaxation parameter $\omega^{(k-1)}$ and the relative change in values $\delta^{(k-1)}$ of the successive iterates $\mathbf{x}^{(k-3)}, \mathbf{x}^{(k-2)}, \mathbf{x}^{(k-1)}$ as shown in Eq. (3.77):

$$\left|\lambda_2^{\approx(k)}\right| = \frac{\delta^{(k-1)} + \omega^{(k-1)} - 1}{\omega^{(k-1)}\sqrt{\delta^{(k-1)}}}. \tag{3.77}$$

Finally, the relative change in values $\delta^{(k-1)}$ is calculated from the intermediate results as presented in Eq. (3.78):

$$\delta^{(k-1)} = \frac{\|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k-2)}\|_\infty}{\|\mathbf{x}^{(k-2)} - \mathbf{x}^{(k-3)}\|_\infty}. \tag{3.78}$$

Note that the maximum vector norm $\|\mathbf{x}\|_\infty$ is defined as $\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|$, where $\mathbf{x} \in \mathbb{R}^n$. We hasten to add that the preceding heuristic for the estimation of the relaxation parameter has to be applied carefully. The setting in which it had originally been introduced was restricted to a certain regular model structure. If it is applied otherwise, we must be aware of the heuristic nature of the approach. The choice of the relaxation parameter ω has a decisive impact on the speed of convergence. Due to the lack of a universally applicable formula for an accurate computation of the parameter, its nontrivial estimation is the most critical task for a successful application of the SOR method. In addition, we would like to point out that ω does not have to be reestimated in each iteration step k but, rather, periodically with some period of, say, 10 iteration steps.

3.4.5.2 The Test of Convergence To terminate an iterative computation process, appropriate criteria need to be provided by means of which it can be assured that the iteration process converges to the desired solution $\mathbf{x}$. Generally, some vector norm $||\mathbf{x} - \mathbf{x}^{(k)}||$ would be used to estimate the error that should be smaller than some threshold ϵ. Since $\mathbf{x}$ is not known, it must either be estimated as shown in [StGo85], or some different heuristics have to be applied. Intuitively, convergence is reached if successive iterates do not differ substantially. But what "substantial difference" means can only be answered in the context of the model under consideration. Convergence could be relatively slow, that is, the vectors $\mathbf{x}^{(k-1)}, \mathbf{x}^{(k)}$ would not be much apart, even though still being far from the final solution $\mathbf{x}$. Furthermore, if the underlying model is large, the elements $x_i^{(k)}$ of $\mathbf{x}^{(k)}$ could easily become smaller than ϵ and convergence could be mistakenly assumed.

To overcome these two problems, it is often a good choice to use the criterion from Eq. (3.79) of the relative change in values of the most recent iterate $\mathbf{x}^{(k)}$ and the vector obtained $m > 1$ steps earlier, $\mathbf{x}^{(k-m)}$, namely:

$$\frac{||\mathbf{x}^{(k)} - \mathbf{x}^{(k-m)}||_\infty}{||\mathbf{x}^{(k)}||_\infty} < \epsilon. \tag{3.79}$$

According to [StGo85], m should be chosen as a function of the number of necessary iterations, $5 < m < 50$. If SOR converges quickly, the much simpler and computationally more efficient criterion as given in Eq. (3.80) might be applicable:

$$||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}|| < \epsilon. \tag{3.80}$$

Commonly applied vector norms are:

$$||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i|, \tag{3.81}$$

$$||\mathbf{x}||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}. \tag{3.82}$$

3.4.5.3 Overflow and Underflow If the normalization condition is incorporated in the transition matrix, as discussed in the context of Eq. (3.64), overflow and underflow are less likely to occur, but convergence is much slower and computational overhead is higher. Therefore, some heuristics are applied to circumvent these numerical problems. To avoid underflow, elements $x_i^{(k)}$ shrinking below a lower threshold ϵ_l are simply set equal to zero. Conversely, if there exist elements $x_i^{(k)}$ that exceed an upper threshold ϵ_u, a normalization is adaptively performed and the values are scaled back to a more convenient range.

3.4.5.4 The Algorithm The algorithm presented in Fig. 3.7 is in a skeletal form. As we have discussed at length, it needs to be carefully adapted for the peculiarities of specific cases to which it is applied. Since for $\omega = 1$ SOR coincides with Gauss-Seidel, we do not separately present the Gauss-Seidel algorithm. Furthermore, for the sake of simplicity, we do not consider scaling and rescaling of the parameter matrix, which is a technique that is commonly performed in order to avoid divisions by diagonal elements a_{jj} (cf. Eq. (3.69)). It should be noted that a sum indexed from 0 to -1 is interpreted to be zero, $\sum_0^{-1} \cdots = 0$.

3.5 COMPARISON OF NUMERICAL SOLUTION METHODS

The solution algorithms discussed in earlier sections are herein compared by the means of the model introduced in Fig. 2.7 (Section 2.2.2). Patterns of convergence and numerical accuracy are compared. According to Eq. (2.58), the following system of steady-state equations results:

$$
\begin{aligned}
0 &= -2\gamma\pi_2 + \delta\pi_1, \\
0 &= -\beta\pi_{RC} + 2c\gamma\pi_2, \\
0 &= -\alpha\pi_{RB} + 2(1-c)\gamma\pi_2, \\
0 &= -(\gamma+\delta)\pi_1 + \alpha\pi_{RB} + \beta\pi_{RC} + \delta\pi_0, \\
0 &= -\delta\pi_0 + \gamma\pi_1.
\end{aligned}
$$

With the normalization condition:

$$1 = \pi_2 + \pi_{RC} + \pi_{RB} + \pi_1 + \pi_0,$$

and the parameters from Table 3.8, three computational experiments were performed using the Gaussian elimination algorithm. The results are summa-

Table 3.8 Experimental parameters for the model with two processor elements

Parameter	Experiment 1	Experiment 2	Experiment 3
$1/\gamma$	24 hr 54 sec	1 yr 12 hr	100 yr 262 hr 48 min
$1/\delta$	4 hr 10 min	4 hr 10 min	4 hr 10 min
$1/\alpha$	10 min	10 min	10 min
$1/\beta$	30 sec	30 sec	30 sec
c	0.99	0.99	0.99

rized in Table 3.9. From the data of the first experiment it is concluded that, for example, with probability $\pi_2 + \pi_1 \approx .71 + .25 = .96$, at least one processor is up in steady state. This quantity could provide an important measure for the availability of a modeled system. Other measures of interest can be similarly derived. By comparing the results of different experiments in Table 3.9

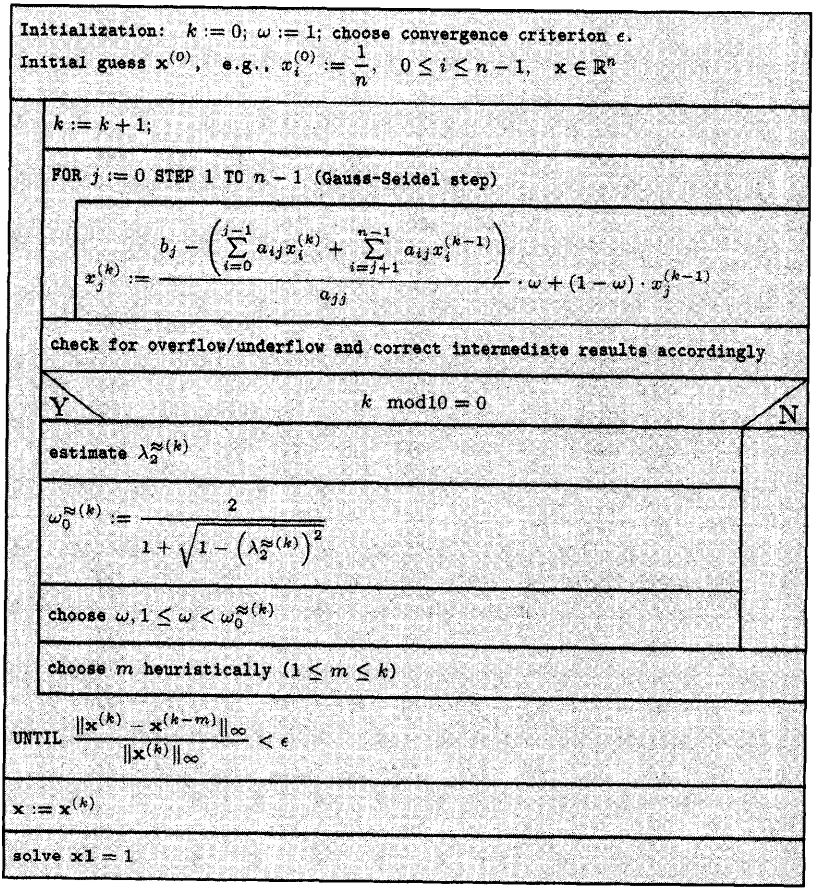


Fig. 3.7 The SOR iteration algorithm.

Table 3.9 Steady-state probabilities of the model with two processor elements

Probability	Experiment 1	Experiment 2	Experiment 3
π_2	0.7100758490525	0.9990481945467	0.9999904674119
π_{RB}	0.0000986153339	0.0000003796383	0.0000000038040
π_{RC}	0.0004881459029	0.0000018792097	0.0000000188296
π_1	0.2465383347910	0.0009490957848	0.0000095099093
π_0	0.0427990549197	0.0000004508205	0.0000000000452

it can be seen how the state probabilities depend on the MTTF, being varied from approximately 1 day to a little less than 100 years and 11 days. Although the probability mass is captured to a larger extent by π_2, the other probabilities are of no less interest to designers of dependable systems because the effectiveness of redundancy and recovery techniques is measured by the values of these small probabilities. Note, in the case of the third experiment in particular, that the numbers differ by many orders of magnitude. This property could impose challenges on the numerical algorithms used to analyze the underlying CTMC since an appropriate accuracy has to be assured. For instance, the results in Table 3.9 are exact values up to 13 digits. While such an accuracy is most likely not very significant for the probability π_2, it seems to be a minimum requirement for the computation of π_0, since the 10 leading digits of π_0 are zeroes in the third experiment. The more the values of the model parameters diverge from each other and the larger the investigated model is, the more this property tends to manifest itself. Note that for the analysis of larger models, iterative methods are usually preferred. The convergence criteria have to be chosen very carefully in cases where the resulting values differ by orders of magnitude. On the one hand, the convergence criteria should be stringent if the significant measures result in very small values. One the other hand, computational complexity increases substantially with a more stringent convergence criteria. Even worse, the probability of convergence might significantly decrease. More details are presented in the following sections.

3.5.1 Case Studies

In the following, we refer back to the model in Fig. 2.7 as a starting point. The model can easily be extended to the case of n processors, while the basic structure is preserved. Furthermore, the same parameters as in Table 3.8 are used in three corresponding experiments. The computations are performed with *Gauss elimination*, the *power* method, the *Gauss-Seidel* algorithm, and different variants of the *SOR* method. The intention is to study the accuracy and pattern of convergence of the iterative methods in a comparative manner. Concerning the SOR method, for example, no sufficiently efficient algorithm

is known for a computation of the optimum ω. With the help of these case studies some useful insights can be obtained.

There are many possibilities to specify convergence criteria, as has been discussed earlier. In our example, convergence is assumed if both a *maximum norm* variant:

$$\|\mathbf{x}\|_\infty = \max_i |x_i| = \max_i |x_i^{(n)} - x_i^{(n-1)}|, \tag{3.83}$$

of the difference between the most recent iterates $\mathbf{x}^{(n)}$ and $\mathbf{x}^{(n-1)}$ and the *square root norm*:

$$\|\mathbf{r}\|_2 = \sqrt{\sum_{i=0}^{n-1} (r_i)^2}, \tag{3.84}$$

of the *residues*:

$$r_i = \sum_{j=0}^{i-1} q_{j,i} x_j^{(n)} + \sum_{j=i}^{n-1} q_{j,i} x_j^{(n-1)} - b_i, \tag{3.85}$$

are less than the specified ϵ:

$$\|\mathbf{x}\|_\infty < \epsilon \wedge \|\mathbf{r}\|_2 < \epsilon.$$

Note that the pattern of convergence of iterative methods can depend on the ordering of the states as well, and thus on the iteration matrix. This dependence is particularly important for the SOR-type methods where in each iteration step the most up-to-date information is used in the computation of the components of the intermediate solution vector. Therefore, we provide in our comparative study the results of two different state ordering strategies. The resulting number of iterations are summarized in Table 3.10 for the cases of $\epsilon = 10^{-10}$, 10^{-15}, 10^{-16}, 10^{-17}, and 10^{-20} as some examples.

Comparing corresponding experiments, it can be seen from Table 3.10 how the number of iteration steps of the power method increases as the convergence criterion is tightened from $\epsilon = 10^{-10}$ to $\epsilon = 10^{-20}$. Furthermore, the power method tends to converge faster, the more the model parameters differ quantitatively from each other. In these examples, convergence is relatively slow in experiment one, medium in experiment two, and fast in the third experiment for fixed ϵ. A similar pattern of convergence using the Gauss-Seidel method and the SOR variants can be observed. The relative best results are achieved by SOR with $\omega = 0.8$. Gauss-Seidel and SOR ($\omega = 1.2$) do not converge if $\epsilon \leq 10^{-17}$.

Another view is provided in Fig. 3.8 where the number of iterations is depicted as a function of ω that is systematically varied, $0 < \omega < 2$, and the criterion of convergence $\epsilon = 10^{-15}$ is chosen. Entries on the graph indicate that convergence was observed within 20,000 iterations for a particular ω.

Table 3.10 Number of iterations required to analyze the model with ten processors

	$\epsilon = 1 \cdot e^{-10}$			$\epsilon = 1 \cdot e^{-15}$		
Methods	Exp. 1	Exp. 2	Exp. 3	Exp. 1	Exp. 2	Exp. 3
Power	32119	16803	16703	61013	24305	24083
Gauss-Seidel	653	23	17	*	25	17
SOR ($\omega = 1.2$)	*	49	–	*	55	–
SOR ($\omega = 0.8$)	239	41	41	367	47	41

	$\epsilon = 1 \cdot e^{-16}$			$\epsilon = 1 \cdot e^{-17}$		
Methods	Exp. 1	Exp. 2	Exp. 3	Exp. 1	Exp. 2	Exp. 3
Power	66547	26487	25309	70221	26883	26625
Gauss-Seidel	*	25	17	*	*	*
SOR ($\omega = 1.2$)	*	57	*	*	*	*
SOR ($\omega = 0.8$)	369	47	43	*	47	43

	$\epsilon = 1 \cdot e^{-20}$		
Methods	Exp. 1	Exp. 2	Exp. 3
Power	71573	30039	30703
Gauss-Seidel	*	*	*
SOR ($\omega = 1.2$)	*	*	*
SOR ($\omega = 0.8$)	*	*	47

Note: An asterisk (*) indicates no convergence in 20,000 iterations and a dash (–) indicates convergence with negative results.

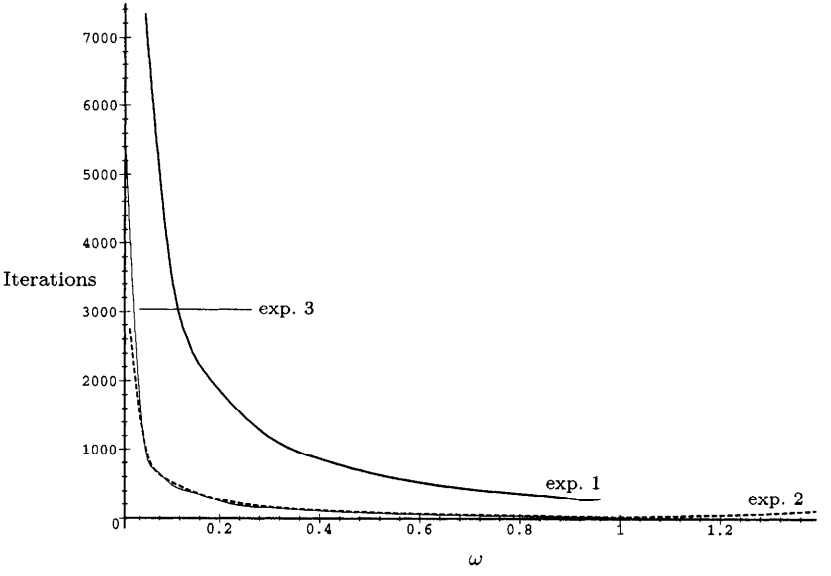


Fig. 3.8 The number of SOR iterations as a function of ω.

The results confirm that convergence is relatively slow in experiment one. Furthermore, in experiment one, SOR converges only if $\omega < 1$ while the number of iterations are quite sensitive to variations in ω. Fewer iteration steps are needed in experiments two and three. Relatively fewer iterations, 279, are observed in experiment one with $\omega = 0.95$, in experiment two, 25, with $\omega = 1.0$; and in experiment three, 21, with $\omega = 1.01$. It is interesting to note that in experiments one and three, even if ω is chosen only slightly larger than 0.95 and 1.01, respectively, no convergence results. Similar outcomes are observed for different criteria of convergence ϵ. Over-relaxation, that is $\omega > 1$, can only be successfully applied in experiment two.

Table 3.11 Accuracy (A) in the number of digits and number of iterations (I) needed by different SOR variants with $\epsilon = 10^{-15}$

	Exp. 2 (d)		Exp. 2 (r)		Exp. 3 (d)		Exp. 3 (r)	
ω	A	I	A	I	A	I	A	I
0.1	*	*	20	539	*	*	20	507
0.2	21	279	20	273	22	265	20	247
0.5	22	97	20	91	23	91	21	83
0.7	23	59	22	57	23	55	21	51
0.8	23	47	23	47	24	43	23	41
1.0	33	27	23	25	49	21	30	17
1.1	28	41	23	39	45	45	–	–
1.2	23	57	23	55	*	*	–	–
1.3	23	81	23	79	*	*	–	–

Note: An asterisk (*) indicates no convergence in 20,000 iterations and a dash (–) indicates convergence with negative results.

Using different norms affects pattern of convergence and accuracy of the numerical results even for the same criterion of convergence ϵ. In Fig. 3.9 the number of iterations is depicted as a function of ω in experiment three, comparing the residue norm (r) to the difference norm (d) with $\epsilon = 10^{-15}$. It can be seen that applying the difference norm consistently results in slightly more iterations for a given ω. A similar pattern was observed in all the experiments. There are ranges of ω where the application of the difference norm *does not* lead to convergence while the application of the residue norm *does* lead to convergence, and vice versa.

The accuracy in terms of the number of correct digits of the numerical values resulting from the application of different norms in the SOR algorithm is indicated in Table 3.11. The numbers are compared to the results obtained using Gauss elimination. It can be seen that the difference norm, both in experiments two and three, consistently results in the same or higher accuracy as the residue norm for a given ω. A trade-off is observed between the number of necessary iterations and the achieved accuracy. Both factors have to be taken into account when designing calculations.

In experiment four, the data from Table 3.12 are applied in a series of computations. The number of iterations versus ω is depicted in Fig. 3.10. The sensitivity of the convergence behavior to ω is impressive in this study. While the minimum number of iterations are observed with $\omega \approx 0.8$, the number of iterations grows quickly as ω approaches one.

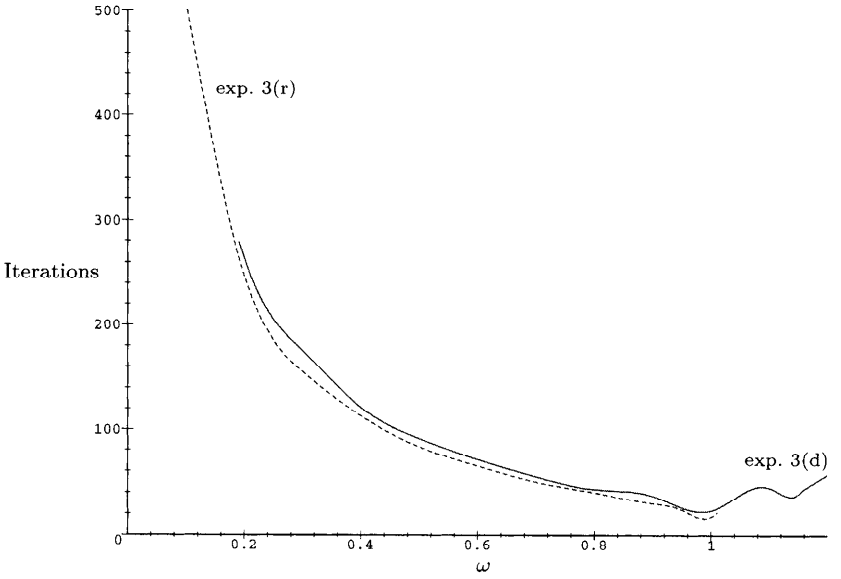


Fig. 3.9 Comparison of difference (d) and residue (r) norm in experiment three.

Table 3.12 Experimental parameters for the model with ten processors

Parameter	Experiment 4
$1/\gamma$	9512 yr 8243 hr 24 min
$1/\delta$	50 min
$1/\alpha$	5 min
$1/\beta$	30 sec
c	0.99
ϵ	10^{-20}
norm	residue

We do not claim to be complete in the discussion of factors that have an impact on the speed of convergence of numerical techniques. We only presented *some* issues of practical relevance that have to be taken into account while considering convergence. The results presented are conditioned on the fact that other convergence dominating issues such as state ordering were not

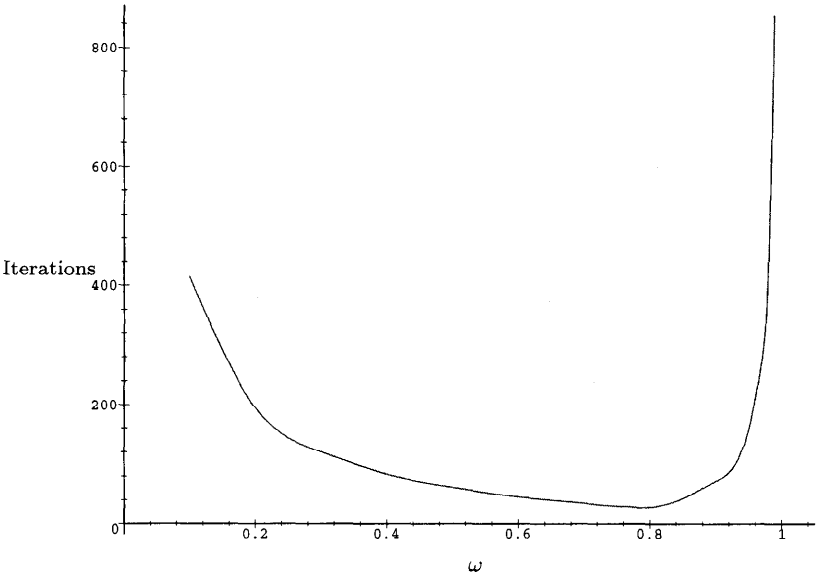


Fig. 3.10 Number of iterations applying the square root norm $\|\mathbf{r}\|_2 < 10^{-20}$ of the residue in experiment four.

taken into account. The methods could have been tuned further, of course, resulting in a different pattern of convergence.

4

Steady-State Aggregation/Disaggregation Methods

In this chapter we consider two main approximation methods: Courtois's decomposition method and Takahashi's iterative aggregation/disaggregation method.

4.1 COURTOIS'S APPROXIMATE METHOD

In this section we introduce an efficient method for the steady-state analysis of Markov chains. Whereas direct and iterative techniques can be used for the *exact* analysis of Markov chains as previously discussed, the method of Courtois [Cour75, Cour77] is mainly applied to *approximate* computations $\nu^{\approx}$ of the desired state probability vector ν. Courtois's approach is based on decomposability properties of the models under consideration. Initially, substructures are identified that can separately be analyzed. Then, an aggregation procedure is performed that uses independently computed subresults as constituent parts for composing the final results. The applicability of the method needs to be verified in each case. If the Markov chain has tightly coupled subsets of states, where the states within each subset are tightly coupled to each other and weakly coupled to states outside the subset, it provides a strong intuitive indication of the applicability of the approach. Such a subset of states might then be aggregated to form a *macro state* as a basis for further analysis. The macro state probabilities, together with the conditional micro state probabilities from within the subsets, can be composed to yield the micro state probabilities of the initial model. Details of the approach are clarified through the following example.

4.1.1 Decomposition

Since the method of Courtois is usually expressed in terms of a DTMC, whereas we are emphasizing the use of a CTMC in our discussion of methodologies, we would like to take advantage of this example and bridge the gap by choosing a CTMC as a starting point for our analysis. With the following model in mind, we can explain the CTMC depicted in Fig. 4.1. We assume a system in which two customers are circulating among three stations according to some stochastic regularities. Each arbitrary pattern of distribution of the customers among the stations is represented by a state. In general, if there are N such stations over which K customers are arbitrarily distributed, then from simple combinatorial reasoning we know that $\binom{N+K-1}{N-1} = \binom{N+K-1}{K}$ such combinations exist. Hence, in our example with $N = 3$ and $K = 2$ we have $\binom{4}{2} = 6$ states.

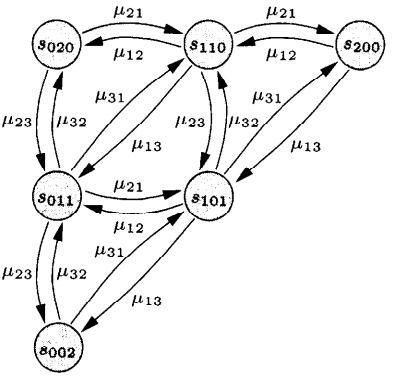


Fig. 4.1 CTMC subject to decomposition.

In state 020, for example, two customers are in station two, while stations one and three are both empty. After a time period of exponentially distributed length, a customer travels from station two to station one or station three. The transition behavior is governed by the transition rates μ_{21} or μ_{23} to states 110 or 011, respectively. The transition behavior between the other states can be explained similarly.

The analysis of such a simple model could be easily carried out by using one of the standard direct or iterative methods. Indeed, we use an exact method to validate the accuracy of the decomposition/aggregation approach for our example. We use the simple example to illustrate Courtois's method. To this end, the model needs to be explored further as to whether it is *nearly completely decomposable*, that is, whether we can find state subsets that represent tightly coupled structures.

The application may suggest a state set partitioning along the lines of the customers' circulation pattern among the visited stations. This would be a promising approach if the customers are preferably staying within the

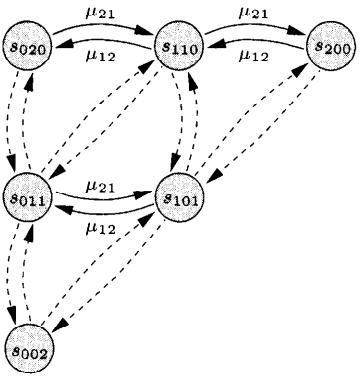


Fig. 4.2 Decomposition of the CTMC with regard to station three.

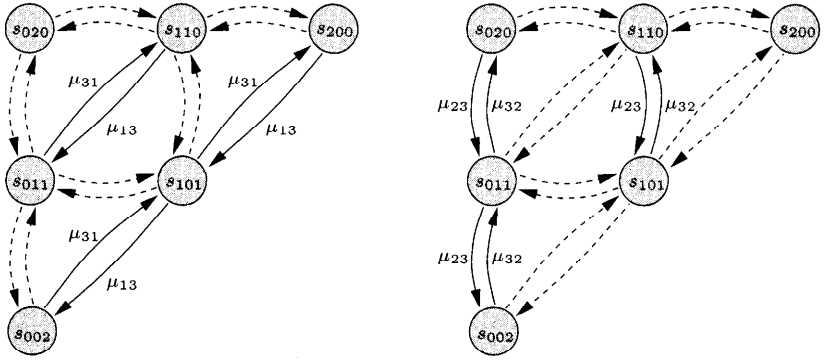


Fig. 4.3 Decompositions of the CTMC with regard to stations two and one.

bounds of a subset of two stations and only relatively rarely transfer to the third station, i.e., the most isolated one. All such possibilities are depicted in Fig. 4.2 and 4.3. In Fig. 4.2, station three is assumed to be the most isolated one, i.e., the one with the least interactions with the others. Solid arcs emphasize the tightly coupled states, whereas dotted arcs are used to represent the "loose coupling." When customers are in the first station, they are much more likely to transit to the second station, then return to the first station, before visiting the third station. Hence, we would come up with three subsets $\{020, 110, 200\}$, $\{011, 101\}$, and $\{002\}$ in each of which the number of customers in the third station is fixed at 0, 1, and 2, respectively. Alternatively, in Fig. 4.3, we have shown the scenario where stations two and three are isolated.

Now we proceed to discuss Courtois's method on the basis of the set of parameters in Table 4.1. It suggests a decomposition according to Fig. 4.2. Clearly, the parameter values indicate strong interactions between stations one and two, whereas the third station seems to interact somewhat less with

Table 4.1 Transition rates

$\mu_{12} = 4.50$	$\mu_{21} = 2.40$	$\mu_{31} = 0.20$
$\mu_{13} = 0.30$	$\mu_{23} = 0.15$	$\mu_{32} = 0.20$

the others. This level of interaction should be reflected at the CTMC level representation. The corresponding infinitesimal generator matrix $\mathbf{Q}$ is given as:

$$\mathbf{Q} = \begin{pmatrix} -4.80 & 4.50 & 0 & 0.30 & 0 & 0 \\ 2.40 & -7.35 & 4.50 & 0.15 & 0.30 & 0 \\ 0 & 2.40 & -2.55 & 0 & 0.15 & 0 \\ 0.20 & 0.20 & 0 & -5.20 & 4.50 & 0.30 \\ 0 & 0.20 & 0.20 & 2.40 & -2.95 & 0.15 \\ 0 & 0 & 0 & 0.20 & 0.20 & -0.40 \end{pmatrix},$$

and is depicted symbolically in Table 4.2.

Table 4.2 Generator matrix $\mathbf{Q}$ of the CTMC structured for decomposition

	200	110	020	101	011	002
200	$-\sum$	μ_{12}	0	μ_{13}	0	0
110	μ_{21}	$-\sum$	μ_{12}	μ_{23}	μ_{13}	0
020	0	μ_{21}	$-\sum$	0	μ_{23}	0
101	μ_{31}	μ_{32}	0	$-\sum$	μ_{12}	μ_{13}
011	0	μ_{31}	μ_{32}	μ_{21}	$-\sum$	μ_{23}
002	0	0	0	μ_{31}	μ_{32}	$-\sum$

It is known from Eq. (3.5) that we can transform any CTMC to a DTMC by defining $\mathbf{P} = \mathbf{Q}/q + \mathbf{I}$, with $q > \max_{i,j \in S} |q_{ij}|$. Next we solve $\boldsymbol{\nu} = \boldsymbol{\nu}\mathbf{P}$ instead of $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$ and we can assert that $\boldsymbol{\nu} = \boldsymbol{\pi}$. Of course, we have to fulfill the normalization condition $\boldsymbol{\nu}\mathbf{1} = 1$. For our example, the transformation results in the transition probability matrix as shown in Table 4.3, where q is appropriately chosen.

Given the condition $q > \max_{i,j} |q_{ij}| = 7.35$, we conveniently fix $q = 10$. Substituting the parameter values from Table 4.1 in Table 4.3 yields the tran-

Table 4.3 Transition-probability matrix $\mathbf{P}$ of the DTMC structured for decomposition

	200	110	020	101	011	002
200	$1-\frac{\sum}{q}$	$\frac{\mu_{12}}{q}$	0	$\frac{\mu_{13}}{q}$	0	0
110	$\frac{\mu_{21}}{q}$	$1-\frac{\sum}{q}$	$\frac{\mu_{12}}{q}$	$\frac{\mu_{23}}{q}$	$\frac{\mu_{13}}{q}$	0
020	0	$\frac{\mu_{21}}{q}$	$1-\frac{\sum}{q}$	0	$\frac{\mu_{23}}{q}$	0
101	$\frac{\mu_{31}}{q}$	$\frac{\mu_{32}}{q}$	0	$1-\frac{\sum}{q}$	$\frac{\mu_{12}}{q}$	$\frac{\mu_{13}}{q}$
s011	0	$\frac{\mu_{31}}{q}$	$\frac{\mu_{32}}{q}$	$\frac{\mu_{21}}{q}$	$1-\frac{\sum}{q}$	$\frac{\mu_{23}}{q}$
002	0	0	0	$\frac{\mu_{31}}{q}$	$\frac{\mu_{32}}{q}$	$1-\frac{\sum}{q}$

sition probability matrix $\mathbf{P}$:

$$\mathbf{P} = \begin{pmatrix} 0.52 & 0.45 & 0 & 0.03 & 0 & 0 \\ 0.24 & 0.265 & 0.45 & 0.015 & 0.03 & 0 \\ 0 & 0.24 & 0.745 & 0 & 0.015 & 0 \\ 0.02 & 0.02 & 0 & 0.48 & 0.45 & 0.03 \\ 0 & 0.02 & 0.02 & 0.24 & 0.705 & 0.015 \\ 0 & 0 & 0 & 0.02 & 0.02 & 0.96 \end{pmatrix}. \tag{4.1}$$

As indicated, $\mathbf{P}$ is partitioned into $|M \times M|$ number of submatrices $\mathbf{P}_{IJ}$, with $M = 3$ and $0 \leq I, J \leq 2$ in our example. The submatrix with macro row index I and column index J is denoted by $\mathbf{P}_{IJ}$, while the elements of each submatrix can be addressed via double subscription $p_{IJ_{ij}}$. The M diagonal submatrices $\mathbf{P}_{II}$ represent for $I = 0, 1, 2$ the interesting cases of zero, one, and two customers, respectively, staying in the third station. For example, the second diagonal submatrix $\mathbf{P}_{11}$ is:

$$\mathbf{P}_{11} = \begin{pmatrix} 0.48 & 0.45 \\ 0.24 & 0.705 \end{pmatrix}. \tag{4.2}$$

The elements of the submatrix can be addressed by the schema just introduced: $p_{11_{10}} = 0.24$. Since the indices $I, 0 \leq I \leq M-1$ are used to uniquely refer to subsets as elements of a partition of the state space S, we conveniently denote the corresponding subset of states by S_I; for example, $S_1 = \{101, 011\}$.

Of course, each diagonal submatrix could possibly be further partitioned into substructures according to this schema. The depth to which such multilayer decomposition/aggregation technique is carried out depends on the number of stations N and on the degree of coupling between the stations. We further elaborate on criteria for the decomposability in the following.

A convenient way to structure the transition probability matrix in our example is to number the states using radix K notation:

$$\sum_{i=1}^{N} k_i K^i, \quad \text{where} \quad \sum_{i=1}^{N} k_i = K.$$

where k_i denotes the number of customers in station i and N is the index of the least coupled station, i.e., the one to be isolated. For instance, in our example a value of 4 would be assigned to state 200 and a value of 12 to state 011. Note that the states in Table 4.2 and 4.3 have been numbered according to this rule.

The transition probability matrix in block form is given as:

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_{00} & \mathbf{P}_{01} & \cdots & \mathbf{P}_{0(M-1)} \\ \mathbf{P}_{10} & \mathbf{P}_{11} & \cdots & \mathbf{P}_{1(M-1)} \\ \vdots & & \vdots & \\ \mathbf{P}_{I0} & \cdots & \mathbf{P}_{IJ} \cdots & \mathbf{P}_{I(M-1)} \\ \vdots & & \vdots & \\ \mathbf{P}_{(M-2)0} & & \cdots & \mathbf{P}_{(M-2)(M-1)} \\ \mathbf{P}_{(M-1)0} & & \cdots & \mathbf{P}_{(M-1)(M-1)} \end{pmatrix}. \tag{4.3}$$

Generally, the partitioning of matrix $\mathbf{P}$ in submatrices $\mathbf{P}_{IJ}$ is done as a function of the number of customers in the least coupled station N. Given that K customers are in the system, there will be $M = K + 1$ diagonal submatrices. Each diagonal submatrix $\mathbf{P}_{II}, 0 \leq I \leq K$, is then considered to be a basic building block for defining transition probability matrices $\mathbf{P}^*_{II}$ of a reduced system of $N-1$ stations and $K-I$ customers. For example, $\mathbf{P}^*_{11}$ would be used to describe a system with $K-1$ customers circulating among $N-1$ stations, while one customer stays in station N all the time. The transitions from station N to all other stations would be temporarily ignored.[1] In terms of our example, we would only consider transitions between states 101 and 011.

To derive stochastic submatrices, the transition probability matrix $\mathbf{P}$ is decomposed into two matrices $\mathbf{A}$ and $\mathbf{B}$ that add up to the original one. Matrix $\mathbf{A}$ comprises the diagonal submatrices $\mathbf{P}_{II}$, and matrix $\mathbf{B}$ the complementary off-diagonal submatrices $\mathbf{P}_{IJ}, I \neq J$:

$$\mathbf{P} = \mathbf{A} + \mathbf{B}. \tag{4.4}$$

In our example, Eq. (4.4) has the following form:

[1] Note that $\mathbf{P}_{11}$ is not a stochastic matrix; it has to be modified to obtain a stochastic matrix $\mathbf{P}^*_{11}$.

$$\mathbf{P} = \underbrace{\begin{pmatrix} 0.52 & 0.45 & 0 & & & \\ 0.24 & 0.265 & 0.45 & & \mathbf{0} & \\ 0 & 0.24 & 0.745 & & & \\ & & & 0.48 & 0.45 & \\ & \mathbf{0} & & 0.24 & 0.705 & \\ & & & & & 0.96 \end{pmatrix}}_{\mathbf{A}} + \underbrace{\begin{pmatrix} & & & 0.03 & 0 & 0 \\ & \mathbf{0} & & 0.015 & 0.03 & 0 \\ & & & 0 & 0.015 & 0 \\ 0.02 & 0.02 & 0 & & & 0.03 \\ 0 & 0.02 & 0.02 & & \mathbf{0} & 0.015 \\ 0 & 0 & 0 & 0.02 & 0.02 & 0 \end{pmatrix}}_{\mathbf{B}}.$$

Since we wish to transform each diagonal submatrix $\mathbf{P}_{II}$ into a stochastic matrix $\mathbf{P}^*_{II}$, we need to define a matrix $\mathbf{X}$, such that Eq. (4.5) holds and matrices $\mathbf{P}^*, \mathbf{P}^*_{00}, \ldots \mathbf{P}^*_{(M-1)(M-1)}$ are all stochastic:

$$\mathbf{P}^* = (\mathbf{A} + \mathbf{X}) = \begin{pmatrix} \mathbf{P}^*_{00} & & & \\ & \ddots & \mathbf{0} & \\ & & \mathbf{P}^*_{II} & \\ & \mathbf{0} & & \ddots \\ & & & \mathbf{P}^*_{(M-1)(M-1)} \end{pmatrix}. \tag{4.5}$$

There are multiple ways to define matrix $\mathbf{X}$. Usually, accuracy and computational complexity of the method depend on the way $\mathbf{X}$ is defined. It is most important, though, to ensure that the matrices $\mathbf{P}^*_{II}, 0 \leq I \leq M-1$, are all ergodic, i.e., they are aperiodic and irreducible. Incorporating matrix $\mathbf{X}$ in matrix $\mathbf{P}$, we get:

$$\begin{aligned} \mathbf{P} &= (\mathbf{A} + \mathbf{X}) + (\mathbf{B} - \mathbf{X}) \\ &= \mathbf{P}^* + \mathbf{C}. \end{aligned} \tag{4.6}$$

In our example, we suggest the use of following matrix $\mathbf{X}$ in order to create a stochastic matrix $\mathbf{P}^*$:

$$\mathbf{X} = \begin{pmatrix} 0 & 0 & 0.03 & 0 & 0 & 0 \\ 0 & 0.045 & 0 & 0 & 0 & 0 \\ 0.015 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.07 & 0 \\ 0 & 0 & 0 & 0.055 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.04 \end{pmatrix}.$$

This gives us:

$$\mathbf{P} = \underbrace{\begin{pmatrix} 0.52 & 0.45 & 0.03 & & & \\ 0.24 & 0.31 & 0.45 & & \mathbf{0} & \\ 0.015 & 0.24 & 0.745 & & & \\ & & & 0.48 & 0.52 & \\ & \mathbf{0} & & 0.295 & 0.705 & \\ & & & & & 1 \end{pmatrix}}_{\mathbf{P}^*} + \underbrace{\begin{pmatrix} 0 & 0 & -0.03 & 0.03 & 0 & 0 \\ 0 & -0.045 & 0 & 0.015 & 0.03 & 0 \\ -0.015 & 0 & 0 & 0 & 0.015 & 0 \\ 0.02 & 0.02 & 0 & 0 & -0.07 & 0.03 \\ 0 & 0.02 & 0.02 & -0.055 & 0 & 0.015 \\ 0 & 0 & 0 & 0.02 & 0.02 & -0.04 \end{pmatrix}}_{\mathbf{C}}.$$

4.1.2 Applicability

The applicability of Courtois's method needs to be checked in each case. In general, partitioning of the state space and subsequent aggregation of the resulting subsets into macro states can be exactly performed if the DTMC under consideration has a lumpable transition probability matrix. In such a case, the application of Courtois's method will not be an approximation. A transition probability matrix $\mathbf{P} = [p_{ij}]$ is *lumpable* with respect to a partition of S in subsets S_I, $0 \leq I \leq M-1$, if for each submatrix $\mathbf{P}_{IJ}, \forall I, J, 0 \leq I \neq J \leq M-1$ real-valued numbers $0 \leq r_{IJ} \leq 1$ exist such that Eq. (4.7) holds [KeSn78]:

$$\sum_{j \in S_J} p_{IJ_{ij}} = r_{IJ}, \qquad \forall i \in S_I. \tag{4.7}$$

Note that the diagonal submatrices $\mathbf{P}_{II}$ need not be completely decoupled from the rest of the system, but rather the matrix has to exhibit regularities imposed by the lumpability condition in order to allow an exact aggregation. In fact, if the $\mathbf{P}_{II}$ are completely decoupled from the rest of the system, i.e., the $r_{IJ} = 0$, $I \neq J$, then this can be regarded as a special case of lumpability of $\mathbf{P}$. More details on and an application example of state lumping techniques are given in Section 4.2, particularly in Section 4.2.2.

From our example in Eq. (4.1), $\mathbf{P}$ is not lumpable with respect to the chosen partition. Hence Courtois's method will be an approximation. A measure of accuracy can be derived according to Courtois from Eqs. (4.4) and (4.6). The degree ϵ of coupling between macro states can be computed from matrix $\mathbf{B} = [b_{ij}]$ in Eq. (4.4). If ϵ is sufficiently small it can be shown that the error induced by Courtois's method is bounded by $O(\epsilon)$.

Let ϵ be defined as follows:

$$\epsilon = \max_{i \in S} \left\{ \sum_{j \in S} b_{ij} \right\}. \tag{4.8}$$

Eq. (4.6) can be rewritten as:

$$\mathbf{P} = \mathbf{P}^* + \epsilon \tilde{\mathbf{C}}. \tag{4.9}$$

In our example, ϵ can be computed to be:

$$\epsilon = \max \left\{ \begin{array}{c} 0.03 \\ 0.015 + 0.03 \\ 0.015 \\ 0.02 + 0.02 + 0.03 \\ 0.02 + 0.02 + 0.015 \\ 0.02 + 0.02 \end{array} \right\} = 0.07.$$

Eq. (4.9) thus results in:

$$\mathbf{P} = \underbrace{\begin{pmatrix} 0.52 & 0.45 & 0.03 & & & \\ 0.24 & 0.31 & 0.45 & & \mathbf{0} & \\ 0.015 & 0.24 & 0.745 & & & \\ & & & 0.48 & 0.52 & \\ & \mathbf{0} & & 0.295 & 0.705 & \\ & & & & & 1 \end{pmatrix}}_{\mathbf{P}^*} + 0.07 \underbrace{\begin{pmatrix} 0 & 0 & -0.429 & 0.429 & 0 & 0 \\ 0 & -0.643 & 0 & 0.214 & 0.429 & 0 \\ -0.214 & 0 & 0 & 0 & 0.214 & 0 \\ 0.286 & 0.286 & 0 & 0 & -1 & 0.429 \\ 0 & 0.286 & 0.286 & -0.786 & 0 & 0.214 \\ 0 & 0 & 0 & 0.286 & 0.286 & -0.572 \end{pmatrix}}_{\tilde{\mathbf{C}}}.$$

To prove that $\mathbf{P}$ is nearly completely decomposable, it is sufficient to show that Relation (4.10) holds between ϵ and the maximum of the second largest eigenvalues $\lambda_I^*(2)$ of $\mathbf{P}_{II}^*$ for all $I, 0 \leq I \leq M-1$ [Cour77]:

$$\epsilon < \frac{1 - \max_I |\lambda_I^*(2)|}{2}. \tag{4.10}$$

For each $\mathbf{P}_{II}^*$, the eigenvalues $\lambda_I^*(k)$ can be arranged according to their decreasing absolute values:

$$|\lambda_I^*(1)| > |\lambda_I^*(2)| > \cdots > |\lambda_I^*(n(I))|. \tag{4.11}$$

Since $\mathbf{P}_{II}^*$, $0 \leq I \leq 2$, are all stochastic, we can immediately conclude $|\lambda_I^*(1)| = 1$ for all I. In our example, the eigenvalues of the three diagonal submatrices:

$$\mathbf{P}_{00}^* = \begin{pmatrix} 0.52 & 0.45 & 0.03 \\ 0.24 & 0.31 & 0.45 \\ 0.015 & 0.24 & 0.745 \end{pmatrix}, \quad \mathbf{P}_{11}^* = \begin{pmatrix} 0.48 & 0.52 \\ 0.295 & 0.705 \end{pmatrix}, \quad \mathbf{P}_{22}^* = (1)$$

need to be computed. The eigenvalues of $\mathbf{P}_{00}^*$ are the roots of the following equation:

$$\begin{aligned} \det(\mathbf{P}_{00}^* - \lambda_0^* \mathbf{I}) &= \begin{vmatrix} (0.52 - \lambda_1^*) & 0.45 & 0.03 \\ 0.24 & (0.31 - \lambda_1^*) & 0.45 \\ 0.015 & 0.24 & (0.745 - \lambda_1^*) \end{vmatrix} \\ &= (\lambda_1^* - 1)(\lambda_1^* - 0.595)(\lambda_1^* + 0.02) \\ &= 0, \end{aligned}$$

where $\mathbf{I}$ denotes the identity matrix. The resulting eigenvalues of $\mathbf{P}_{00}^*$ are arranged according to their absolute value as:

$$|\lambda_0^*(1)| = 1, \quad |\lambda_0^*(2)| = 0.595, \quad |\lambda_0^*(3)| = 0.02.$$

The eigenvalues of $\mathbf{P}_{11}^*$ are determined by the solution of:

$$\begin{aligned} \det(\mathbf{P}_{11}^* - \lambda_1^* \mathbf{I}) &= \begin{vmatrix} (0.48 - \lambda_1^*) & 0.52 \\ 0.295 & (0.705 - \lambda_1^*) \end{vmatrix} \\ &= (\lambda_1^* - 1)(\lambda_1^* - 0.185) \\ &= 0, \end{aligned}$$

which, in turn, gives:

$$|\lambda_1^*(1)| = 1 \quad \text{and} \quad |\lambda_1^*(2)| = 0.185.$$

The eigenvalue of $\mathbf{P}_{22}^*$ immediately evaluates to:

$$|\lambda_2^*(1)| = 1.$$

With the second largest eigenvalues computed, we can examine whether decomposability Condition (4.10) holds:

$$\max_I |\lambda_I^*(2)| = \max \left\{ \begin{matrix} 0.595 \\ 0.185 \end{matrix} \right\} = 0.595$$

$$0.07 = \epsilon < \frac{1 - \max_I |\lambda_I^*(2)|}{2}$$

$$0.07 = \epsilon < \frac{1 - 0.595}{2} = 0.2025.$$

Because $\epsilon < 0.2025$, Condition (4.10) holds and the transition probability matrix $\mathbf{P}$ is nearly completely decomposable with respect to the chosen decomposition strategy depicted in Fig. 4.2.

4.1.3 Analysis of the Substructures

As a first step toward the computation of the approximate state probability vector $\boldsymbol{\nu}^{\approx} \approx \boldsymbol{\nu}$, we analyze each submatrix $\mathbf{P}_{II}^*$ separately and compute the conditional state probability vector $\boldsymbol{\nu}_I^*$, $0 \leq I \leq M-1$:

$$\boldsymbol{\nu}_I^*(\mathbf{P}_{II}^* - \mathbf{I}) = \mathbf{0}, \qquad \boldsymbol{\nu}_I^* \mathbf{1} = 1. \tag{4.12}$$

Thus $\boldsymbol{\nu}_I^*$ is the left eigenvector of $\mathbf{P}_{II}^*$ corresponding to the eigenvalue $\lambda_I^*(1) = 1$. Substituting the parameters from $\mathbf{P}_{00}^*$ of our example, the solution of

$$(\nu_{00}^*, \nu_{01}^*, \nu_{02}^*) \cdot \begin{pmatrix} (0.52-1) & 0.45 & 0.03 \\ 0.24 & (0.31-1) & 0.45 \\ 0.015 & 0.24 & (0.745-1) \end{pmatrix} = \mathbf{0}, \quad \boldsymbol{\nu}_0^* \mathbf{1} = 1$$

yields the conditional steady-state probabilities of the micro states aggregated to the corresponding macro state 0:

$$\nu_{00}^* = 0.164, \quad \nu_{01}^* = 0.295, \quad \nu_{02}^* = 0.54.$$

Similarly, $\mathbf{P}_{11}^*$ is used in

$$(\nu_{10}^*, \nu_{11}^*) \cdot \begin{pmatrix} (0.48-1) & 0.52 \\ 0.295 & (0.705-1) \end{pmatrix} = \mathbf{0}, \quad \boldsymbol{\nu}_1^* \mathbf{1} = 1$$

to obtain the conditional state probabilities of the micro states aggregated to the corresponding macro state 1:

$$\nu^*_{10} = 0.362, \quad \nu^*_{11} = 0.638.$$

Finally:

$$\nu^*_2 = \nu^*_{20} = 1.$$

Nearly completely decomposable systems can be characterized with respect to "long-term" and "short-term" behavior:

- From a "short-term" perspective, systems described by their probability transition matrix $\mathbf{P}$ can be decomposed into M independent subsystems, each of whose dynamics is governed by a stochastic process approximately described by the matrices $\mathbf{P}^*_{II}, 0 \leq I \leq M-1$. As an outcome of the analyses of M independent subsystems the conditional micro-state probability vector $\boldsymbol{\nu}^*_I$ results.

- In the "long run," the impact of the interdependencies between the subsystems cannot be neglected. The interdependencies between subsystems, or macro states, I and J, for all I, J, are described by the transition probabilities Γ_{IJ} that can be – approximately – derived from the transition probability matrix $\mathbf{P}$ and the conditional state probability vector $\boldsymbol{\nu}^*_I$. Solving the transition matrix $\boldsymbol{\Gamma} = [\Gamma_{IJ}]$ for the macro states yields the macro-state probability vector $\boldsymbol{\gamma}$. The macro-state probability vector $\boldsymbol{\gamma}$ can, in turn, be used for unconditioning of $\boldsymbol{\nu}^*_I, 0 \leq I \leq M-1$, to yield the final result, the approximate state probability vector $\boldsymbol{\nu}^{\approx} \approx \boldsymbol{\nu}$.

4.1.4 Aggregation and Unconditioning

Having obtained the steady-state probability vector for each subset, we are now ready for the next step in Courtois's method. The transition probability matrix over the macro states, $\boldsymbol{\Gamma} = [\Gamma_{IJ}]$ is approximately computed as:

$$\Gamma_{IJ} = \sum_{i \in S_I} \left(\nu^*_{Ii} \sum_{j \in S_J} p_{IJij} \right). \tag{4.13}$$

Comparing Eq. (4.13) with the lumpability Condition (4.7), it is clear that the Γ_{IJ} can be exactly determined if the model under consideration is lumpable, that is:

$$\Gamma_{IJ} = r_{IJ}, \tag{4.14}$$

holds, independent of $\boldsymbol{\nu}^*_I$.

In our example, the macro-state transition probabilities Γ_{IJ} are derived according to Eq. (4.13):

$$\begin{aligned}
\Gamma_{00} &= \nu^*_{00}\cdot(p_{0000}+p_{0001}+p_{0002})+\nu^*_{01}\cdot(p_{0010}+p_{0011}+p_{0012})\\
&\quad+\nu^*_{02}\cdot(p_{0020}+p_{0021}+p_{0022})=0.9737,\\
\Gamma_{01} &= \nu^*_{00}\cdot(p_{0100}+p_{0101})+\nu^*_{01}\cdot(p_{0110}+p_{0111})\\
&\quad+\nu^*_{02}\cdot(p_{0120}+p_{0121})-0.0263,\\
\Gamma_{02} &= \nu^*_{00}\cdot(p_{0200})+\nu^*_{01}\cdot(p_{0210})+\nu^*_{13}\cdot(p_{0220})=1-\Gamma_{00}-\Gamma_{01}=0,\\
\Gamma_{10} &= \nu^*_{10}\cdot(p_{1000}+p_{1001}+p_{1002})+\nu^*_{11}\cdot(p_{1010}+p_{1011}+p_{1012})=0.04,\\
\Gamma_{11} &= \nu^*_{10}\cdot(p_{1100}+p_{1101})+\nu^*_{11}\cdot(p_{1110}+p_{1111})=0.9396,\\
\Gamma_{12} &= 1-\Gamma_{10}-\Gamma_{11}=0.0204,\\
\Gamma_{20} &= \nu^*_{20}\cdot(p_{2000}+p_{2001}+p_{2002})=0,\\
\Gamma_{21} &= \nu^*_{20}\cdot(p_{2100}+p_{2101})=0.04,\\
\Gamma_{33} &= 1-\Gamma_{20}-\Gamma_{21}=0.96.
\end{aligned}$$

Accordingly, the macro-state transition probability matrix $\mathbf{\Gamma}$ is:

$$\mathbf{\Gamma}=\begin{pmatrix}0.9737 & 0.0263 & 0\\ 0.04 & 0.9396 & 0.0204\\ 0 & 0.04 & 0.96\end{pmatrix}.$$

The next step is to compute the macro steady-state probability vector $\boldsymbol{\gamma}$ using:

$$\boldsymbol{\gamma}\mathbf{\Gamma}=\boldsymbol{\gamma},\qquad \boldsymbol{\gamma}\mathbf{1}=1. \tag{4.15}$$

In our example, Eq. (4.15) results in:

$$(\gamma_0,\gamma_1,\gamma_2)\begin{pmatrix}(0.9737-1) & 0.0263 & 0\\ 0.04 & (0.9396-1) & 0.0204\\ 0 & 0.04 & (0.96-1)\end{pmatrix}=\mathbf{0},\quad \boldsymbol{\gamma}\mathbf{1}=1,$$

from which the macro steady-state probabilities are calculated as:

$$\gamma_0=0.502,\quad \gamma_1=0.330,\quad \gamma_2=0.168.$$

The final step is to obtain the approximate steady-state probabilities for the original model. For convenience, the elements of the state probability vector $\boldsymbol{\nu}^{\approx}\approx\boldsymbol{\nu}$ are partitioned along the lines of the decomposition strategy. Hence, we refer to the elements of $\boldsymbol{\nu}^{\approx}=[\nu^{\approx}_{Ii}]$ in compliance with the usual notation in this context. The unconditioning in Eq. (4.16) of the $\boldsymbol{\nu}^*_I$ is to be performed for all macro states I:

$$\nu^{\approx}_{Ii}=\gamma_I\nu^*_{Ii},\qquad 0\le I\le M-1 \text{ and } \forall i\in S_I \tag{4.16}$$

Table 4.4 State probabilities computed using Courtois's method and exact ones

State	Probability	Formula	Approx.	Exact	% Error
200	$\nu^{\approx}_{0\,0}$	$\gamma_0\nu^*_{0\,0}$	0.0828	0.0787	+ 5.2
110	$\nu^{\approx}_{0\,1}$	$\gamma_0\nu^*_{0\,1}$	0.1480	0.1478	+ 0.1
020	$\nu^{\approx}_{0\,2}$	$\gamma_0\nu^*_{0\,2}$	0.2710	0.2777	- 2.4
101	$\nu^{\approx}_{1\,0}$	$\gamma_1\nu^*_{1\,0}$	0.1194	0.1144	+ 4.4
011	$\nu^{\approx}_{1\,1}$	$\gamma_1\nu^*_{1\,1}$	0.2108	0.2150	- 2.0
002	$\nu^{\approx}_{2}$	$\gamma_2\nu^*_{2}$	0.1680	0.1664	+ 0.1

The use of Eq. (4.16) in our example gives the results summarized in Table 4.4. For the sake of completeness, the exact state probabilities are also shown for comparison.

For models with large state spaces, Courtois's method can be very efficient if the underlying model is nearly completely decomposable. If the transition probability matrix obeys certain regularity conditions the results can be exact. The error induced by the method can, in principle, be bounded [CoSe84]. But since the computational complexity of this operation is considerable, a formal error bounding is often omitted. The efficiency of Courtois's method is due to the fact that instead of solving one linear system of equations of the size of the state space S, several much smaller linear systems are solved independently, one system for each subset S_I of the partitioned state space S, and one for the aggregated chain.

We conclude this section by summarizing the entire algorithm. For the sake of simplicity, only one level of decomposition is considered. Of course, the method can be iteratively applied on each diagonal submatrix $\mathbf{P}^*_{II}$.

4.1.5 The Algorithm

STEP 1 Create the state space and organize it appropriately according to a pattern of decomposition.

STEP 2 Build the transition probability matrix $\mathbf{P}$ (by use of randomization technique $\mathbf{P} = \mathbf{Q}/q + \mathbf{I}$ if the starting point is a CTMC), and partition $\mathbf{P}$ into $M \times M$ number of submatrices $\mathbf{P}_{IJ}, 0 \leq I, J, \leq M-1$, appropriately.

STEP 3 Verify the nearly complete decomposability of $\mathbf{P}$ according to Relation (4.10) with the chosen value of ϵ.

STEP 4 Decompose $\mathbf{P}$ such that $\mathbf{P} = \mathbf{P}^* + \epsilon\tilde{\mathbf{C}}$ according to Eq. (4.9). Matrix $\mathbf{P}^*$ contains only stochastic diagonal submatrices $\mathbf{P}^*_{II}$, and ϵ is a measure of the accuracy of Courtois's method. It is defined as the maximum sum of the entries of the non-diagonal submatrices $\mathbf{P}_{IJ}$, $I \neq J$, of $\mathbf{P}$.

STEP 5 For each I, $0 \leq I \leq M-1$, solve equation $\boldsymbol{\nu}^*_I\mathbf{P}^*_{II} = \boldsymbol{\nu}^*_I$ with $\boldsymbol{\nu}^*_I\mathbf{1} = 1$ to obtain the conditional state probability vectors $\boldsymbol{\nu}^*_I$.

STEP 6 Compute the coupling between the decomposed macro states:

STEP 6.1 Generate the transition probability matrix $\mathbf{\Gamma} = [\Gamma_{IJ}]$ according to Eq. (4.13).

STEP 6.2 Solve Eq. (4.15) to obtain the macro steady-state probability vector $\boldsymbol{\gamma}$.

STEP 7 Compute the approximate steady-state probability vector $\boldsymbol{\nu}^{\approx}$ of the micro states by unconditioning of the conditional state probability vectors $\boldsymbol{\nu}_I^*, 0 \leq I \leq M-1$, according to Eq. (4.16).

STEP 8 From $\boldsymbol{\nu}^{\approx}$ compute steady-state performance and dependability measures along the lines of a specified reward structure.

Problem 4.1 Verify that the transition probability matrix from Eq. (4.1) is not lumpable according to the chosen partition.

Problem 4.2 Recall Eq. (4.7) and modify the parameters in Table 4.1 of the example in Fig. 4.1 such that the resulting model is lumpable and nearly completely decomposable. Derive the generator matrix according to Table 4.2 and the transition probability matrix according to Table 4.3 for the resulting model.

Problem 4.3 Discuss the relationship between lumpability and nearly complete decomposability. Under what condition does Courtois's method yield the exact state probabilities if the model is both lumpable and nearly completely decomposable? As a hint, compare Eq. (4.13) with lumpability Condition (4.7) and relate it to Eq. (4.16).

Problem 4.4 Modify Courtois's method such that it becomes directly applicable for an analysis of CTMCs. Apply the new algorithm directly, that is, without using uniformization, to the original model in Fig. 4.1 and solve for the steady-state probability vector.

4.2 TAKAHASHI'S ITERATIVE METHOD

Although closely related to and under some circumstances even coinciding with Courtois' approach, Takahashi's method [Taka75] differs substantially both with respect to the methodology used and the applicability conditions. While Courtois's method is non-iterative and is applied for the approximate computation of the steady-state probability vector $\boldsymbol{\nu}^{\approx}$ for a given ergodic DTMC (or $\boldsymbol{\pi}^{\approx}$ in the case of a CTMC), Takahashi's iterative method allows a computation of the *exact* state probability vector. (Note that numerical errors are still encountered even in such "exact" methods. The method is exact in that there are no modeling approximations.) To allow a straightforward comparison of the two methods, we prefer a continued discussion in terms of

DTMCs. Recall that any given ergodic CTMC can easily be transformed into an ergodic DTMC.

Much like Courtois's method, Takahashi's approach is to partition the state space S into M disjoint subsets of states $S_I \subset S, 0 \leq I \leq M-1$ such that each subset S_I is *aggregated* into a *macro* state I. The criteria, however, used to cluster states differ in the two approaches. The calculation of transition probabilities Γ_{IJ} among the macro states I and J is performed on the basis of conditional micro-state probability vector $\boldsymbol{\nu}_I^*$ and the originally given transition probabilities p_{ij}, or p_{IJij} according to Eq. (4.13). With Courtois's method, the conditional probability vectors $\boldsymbol{\nu}_I^*$, given partition element I, can be separately computed for each subset of micro states $S_I, 0 \leq I \leq M-1$, if the original model is nearly completely decomposable. By contrast, in Takahashi's method, the partitioning of the state space is performed on the basis of *approximate lumpability*.[2]

Note that if the state space is approximately lumpable with respect to a partition, it does not necessarily imply that the subsets are nearly decoupled as needed in Courtois's approach. As an immediate consequence, the conditional micro-state probabilities cannot be calculated independently for each subset of states. Instead, the complementary subset of states and the interactions with this complement is taken into account when approximating conditional micro-state probabilities of a partition element. The whole complementary set is aggregated into a single state representing all external interactions from the states within the particular subset. Since the micro-state probabilities as well as the macro-state probabilities are not known in the beginning, initial estimates are needed. The macro- and micro-state probabilities are *iteratively* calculated with Takahashi's method. Aggregation and disaggregation phases are repeated alternately until some convergence criterion is satisfied. By adding an extra computational step, Schweitzer [Schw84] shows that geometric convergence can be guaranteed. Usually, the extra computation takes the form of a power or a Gauss-Seidel iteration step.

4.2.1 The Fundamental Equations

Our discussion of Takahashi's method is based on that given by Schweitzer [Schw84]. Further details can also be found in [Stew94]. Let a given state space S be partitioned into M subsets:

$$S = \bigcup_{I=0}^{M-1} S_I \quad \text{and} \quad S_I \cap S_J = \emptyset, \quad \forall I \neq J, \quad 0 \leq I, J \leq M-1. \tag{4.17}$$

[2]While lumpability has been introduced in Eq. (4.7), approximate lumpability is defined later in Section 4.2.2.

Formally, the macro-state probability vector γ could be calculated from the micro-state probability vector $\boldsymbol{\nu}$ for each component I of the vector:

$$\gamma_I = \sum_{i \in S_I} \nu_i, \quad 0 \leq I \leq M-1. \tag{4.18}$$

Since $\boldsymbol{\nu}$ is not known in advance, Eq. (4.18) cannot be directly used. The macro-state probabilities can be derived, however, if the macro-state transition probability matrix $\boldsymbol{\Gamma} = [\Gamma_{IJ}]$ were known via the solution of Eq. (4.19):

$$\boldsymbol{\gamma} = \boldsymbol{\gamma}\boldsymbol{\Gamma}, \quad \boldsymbol{\gamma}\mathbf{1} = 1. \tag{4.19}$$

The implied aggregation step is based on iterative approximations of the probabilities ν^*_{Ii} of states $i \in S_I$ given an estimate $\boldsymbol{\nu}^{\approx}$ of the micro-state probability vector:

$$\nu^*_{Ii} \approx \frac{\nu_i^{\approx}}{\sum_{k \in S_I} \nu_k^{\approx}}, \forall i \in S_I. \tag{4.20}$$

Therefore, from Eq. (4.20), and Eq. (4.13), the macro-state transition probabilities Γ_{IJ} can be approximated as:

$$\begin{aligned} \Gamma_{IJ} \approx \sum_{i \in S_I} \sum_{j \in S_J} \frac{\nu_i^{\approx} p_{ij}}{\sum_{k \in S_I} \nu_k^{\approx}} &= \sum_{i \in S_I} \frac{\nu_i^{\approx}}{\sum_{k \in S_I} \nu_k^{\approx}} \sum_{j \in S_J} p_{ij} \\ &\approx \sum_{i \in S_I} \left(\nu^*_{Ii} \sum_{j \in S_J} p_{IJij} \right). \end{aligned} \tag{4.21}$$

For the disaggregation step, the probabilities Γ_{Ii} of transitions from macro state I to micro state i are pairwise needed for all $0 \leq I \leq M-1$ and for all $i \in S$. The transition probabilities Γ_{Ii} can be derived from Eq. (4.21) and are given by Eq. (4.22):

$$\Gamma_{Ij} = \sum_{i \in S_I} \frac{\nu_i^{\approx} p_{ij}}{\sum_{k \in S_I} \nu_k^{\approx}}. \tag{4.22}$$

To complete the disaggregation step, the micro-state probabilities ν_i are expressed in terms of macro-state probabilities γ_I and transition probabilities Γ_{Ii} for each subset of states S_I separately. To accomplish this task, $|S_I|$ linear equations need to be solved for each subset S_I:

$$\nu_i = \sum_{j \in S_I} \nu_j p_{ji} + \sum_{K=0, K \neq I}^{M-1} \gamma_K \Gamma_{Ki}, \quad \forall i \in S_I. \tag{4.23}$$

The second term in Eq. (4.23) represents the aggregation of the complementary subset of macro states into a single external "super" state and the interactions in terms of transition probabilities with this state.

Macro-state probability vector Eq. (4.19), together with transition probability aggregation steps as given in Eq. (4.21), constitute one major part of Takahashi's method, the *aggregation* step. Micro-state probability vector Eq. (4.23), together with transition probability disaggregations, as indicated in Eq. (4.22), form the other major part of Takahashi's method, the *disaggregation* step.

4.2.2 Applicability

Given partition:

$$S = \bigcup_{J=0}^{M-1} S_J,$$

a subset S_I is *lumpable* if and only if real-valued constants r_{Ij} exist such that condition (4.24) holds $\forall j \in S - S_I$:

$$p_{ij} = r_{Ij} \quad \forall i \in S_I. \tag{4.24}$$

Condition (4.24) is necessary and sufficient for the lumpability of S_I with respect to $S - S_I$. The lumpability condition may be restated by referring to Eq. (4.7) if pairwise lumpability is required such that real-valued constants r_{IJ} exist and Condition (4.25) holds $\forall I, J, 0 \leq I \neq J \leq M-1$:

$$\sum_{j \in S_J} p_{ij} = r_{IJ} = \frac{1}{|S_I|} \sum_{i \in S_I} \sum_{j \in S_J} p_{ij}, \quad \forall i \in S_I. \tag{4.25}$$

If the state space can be partitioned such that the lumpability condition holds for all pairs of subsets $S_I, S_J \subset S$, then Takahashi's algorithm terminates in one iteration with the exact state probability vector $\boldsymbol{\nu}$. Unfortunately, lumpability implies very strict structural limitations on the underlying Markov model. More often, however, Markov chains exhibit *approximate lumpability* such that there exists a sufficiently small lumpability error ϵ:

$$\epsilon = \max_{0 \leq I \leq M-1} \sum_{J=0, J \neq I}^{M-1} \sum_{i \in S_I} \left| \sum_{j \in S_J} p_{ij} - \frac{1}{|S_I|} \sum_{i \in S_I} \sum_{j \in S_J} p_{ij} \right|. \tag{4.26}$$

Lumpability error ϵ is a measure for the speed of convergence of Takahashi's method. The problem remains, however, to find a good partitioning among the states of a given DMTC. Some hints concerning convergence and state space partitioning can be found in Stewart's book [Stew94]. The underlying Markov models of some so-called product-form queueing networks, which are discussed in Chapter 7, exhibit the exact lumpability property. Furthermore, the lumpability equations can be exploited theoretically to investigate accuracy of approximate product-form queueing network algorithms.

In general, the efficiency of Takahashi's method can benefit from an exploitable structure of the underlying transition probability matrix. This fact, however, imposes the burden on the user of finding a good clustering strategy, and no generally applicable rule is known for a good clustering strategy. Finally, it is worth mentioning that approximate lumpability is different from weak lumpability, which can be applied for an aggregation-disaggregation based transient analysis of CTMCs. A comprehensive study on lumpability and weak lumpability is presented by Nicola [Nico90].

4.2.3 The Algorithm

Since convergence of Takahashi's method is a non-trivial problem, some care is needed here. We follow an approach that is often applied to enforce geometric convergence, namely, to incorporate an intermediate power, Gauss-Seidel, or SOR step between successive aggregations and disaggregations. If the decrease in residual error is not sufficient from iteration to iteration, such a step is suggested to be included in the computation. Usually, the corresponding condition is stated in terms of a real-valued lower improvement bound $c, 0 < c < 1$ on the error ratio from iteration step $n-1$ to iteration step n:

$$\frac{\text{residualerror}\left(\boldsymbol{\nu}^{(n)}\right)}{\text{residualerror}\left(\boldsymbol{\nu}^{(n-1)}\right)} \leq c. \tag{4.27}$$

The complete algorithm is presented as Fig. 4.4.

4.2.4 Application

We demonstrate Takahashi's method with the same example used in Section 4.1 to illustrate Courtois's method to allow an easy comparison of the numerical behavior of the two algorithms. Recall that we considered a network in which two customers were circulating among three stations according to some stochastic regularities. The resulting state transition diagram of the set S of six states was depicted in Fig. 4.1. Again, we will apply the decomposition strategy indicated in Fig. 4.2 and partition S into three disjoint subsets: $S_0 = \{020, 110, 200\}$, $S_1 = \{011, 101\}$, and $S_2 = \{002\}$. Arranging the states in the same order as defined in Table 4.3 and using the same numbers in our example results in the following transition probability matrix:

$$\mathbf{P} = \begin{pmatrix} 0.52 & 0.45 & 0 & 0.03 & 0 & 0 \\ 0.24 & 0.265 & 0.45 & 0.015 & 0.03 & 0 \\ 0 & 0.24 & 0.745 & 0 & 0.015 & 0 \\ 0.02 & 0.02 & 0 & 0.48 & 0.45 & 0.03 \\ 0 & 0.02 & 0.02 & 0.24 & 0.705 & 0.015 \\ 0 & 0 & 0 & 0.02 & 0.02 & 0.96 \end{pmatrix}. \tag{4.28}$$

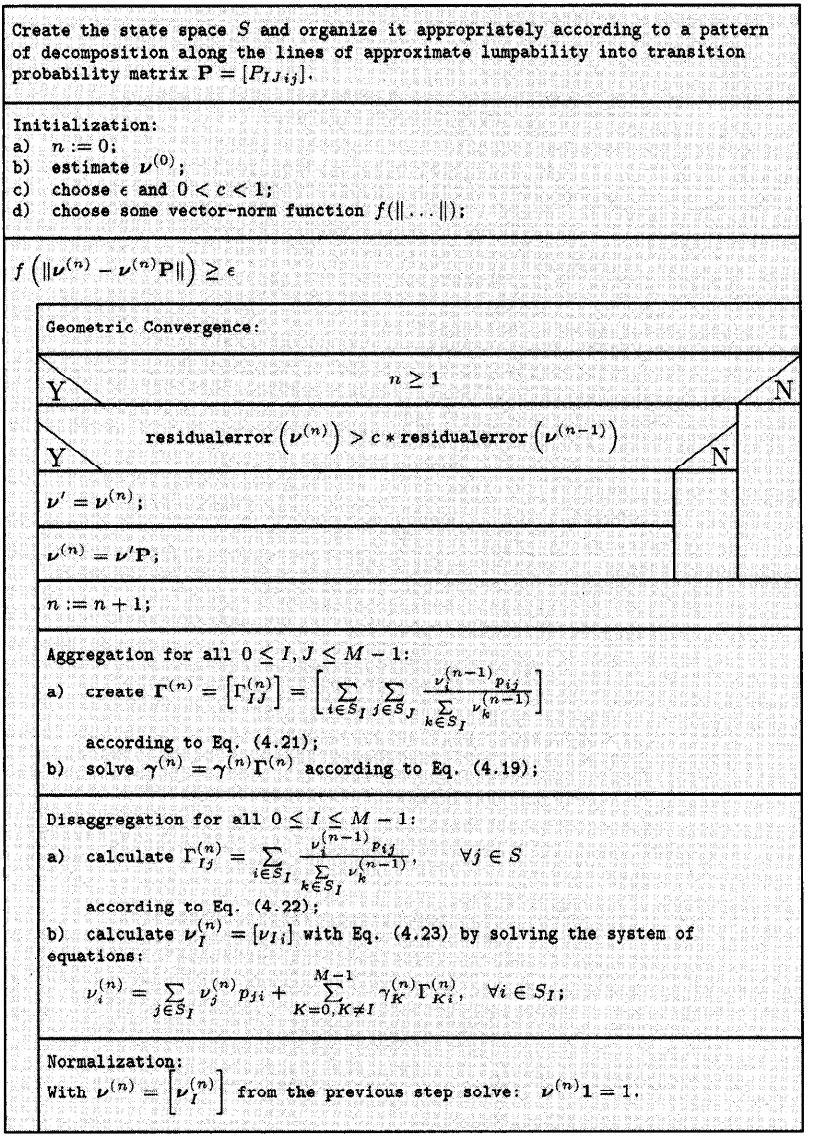


Fig. 4.4 Takahashi's algorithm.

It has been shown that the transition probability matrix $\mathbf{P}$ in Eq. (4.28) is nearly completely decomposable so that Courtois's method could be applied. We now investigate the lumpability error induced for the same partition with respect to approximate lumpability as defined in Eq. (4.26). To calculate the lumpability error, we first let $I = 0, J = 1$ and denote with ϵ_{IJ} the inner sum:

$$\begin{aligned}
\epsilon_{01} &= \sum_{i \in S_0} \left| \sum_{j \in S_1} p_{ij} - \frac{1}{|S_0|} \sum_{i \in S_0} \sum_{j \in S_1} p_{ij} \right| \\
&= \left| 0.03 - \frac{0.03 + 0.015 + 0.03 + 0.015}{3} \right| \\
&\quad + \left| 0.015 + 0.03 - \frac{0.03 + 0.015 + 0.03 + 0.015}{3} \right| \\
&\quad + \left| 0.015 - \frac{0.03 + 0.015 + 0.03 + 0.015}{3} \right| \\
&= |0.03 - 0.03| + |0.045 - 0.03| + |0.015 - 0.03| \\
&= 0 + 0.015 + 0.015 = 0.03.
\end{aligned}$$

Similarly, we get:

$$\epsilon_{02} = 0.$$

Therefore, by letting $I = 0, J \neq I$, the total lumpability error is given by:

$$\epsilon_0 = \epsilon_{01} + \epsilon_{02} = 0.03 + 0 = 0.03.$$

For the other elements of the partition, we get:

$$\epsilon_1 = 0.015, \qquad \epsilon_2 = 0.$$

With:

$$\epsilon = \max_{0 \leq I \leq M-1} \epsilon_I = \max\{0.03, 0.015, 0\} = 0.03,$$

measures can be derived for the speed of convergence if Takahashi's method is applied with respect to the given partition of $S = \bigcup_{I=0}^{2} S_I$.

4.2.4.1 Aggregation With the initial probability vector:

$$\boldsymbol{\nu}^{(0)} = \boldsymbol{\nu}(0) = \left(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6} \right),$$

the first aggregation step, i.e., the computation of

$$\boldsymbol{\Gamma}^{(1)} = \left[\Gamma_{IJ}^{(1)} \right]$$

can be accomplished according to Eq. (4.21). As an example, the calculation of $\Gamma_{00}^{(1)}$ is shown:

$$\begin{aligned}\Gamma_{00}^{(1)} &= \frac{1/6}{1/2} \cdot \sum_{i \in S_0} \sum_{j \in S_0} p_{ij} \\ &= \frac{1}{3} \cdot (0.52 + 0.45 + 0.24 + 0.265 + 0.45 + 0.24 + 0.745) \\ &= \frac{1}{3} \cdot 2.91 = 0.97.\end{aligned}$$

Thus, in the first aggregation step, we get the following macro-state transition probability matrix:

$$\mathbf{\Gamma}^{(1)} = \begin{pmatrix} 0.97 & 0.03 & 0 \\ 0.04 & 0.9375 & 0.0225 \\ 0 & 0.04 & 0.96 \end{pmatrix}. \tag{4.29}$$

Then the macro-state probability vector $\boldsymbol{\gamma}^{(1)}$ is given by the solution of the following system of linear equations:

$$\left(\gamma_0^{(1)}, \gamma_1^{(1)}, \gamma_2^{(1)}\right) \begin{pmatrix} -0.03 & 0.03 & 0 \\ 0.04 & -0.0625 & 0.0225 \\ 0 & 0.04 & -0.04 \end{pmatrix} = 0, \quad \boldsymbol{\gamma}^{(1)}\mathbf{1} = 1. \tag{4.30}$$

From Eq. (4.30) we obtain the macro-state probabilities up to the fourth decimal digit:

$$\boldsymbol{\gamma}^{(1)} = (0.4604, 0.3453, 0.1943). \tag{4.31}$$

4.2.4.2 Disaggregation In the first part of the disaggregation step, transition probabilities $\Gamma_{Ij}^{(1)}$, are computed:

$$\Gamma_{10}^{(1)} = \frac{1}{2}0.02 + \frac{1}{2}0 = 0.01, \quad \Gamma_{20}^{(1)} = 0.$$

From there we compute:

$$\begin{aligned}\nu_0^{(1)} &= \sum_{j=0,1,2} \nu_j^{(1)} p_{j0} + \gamma_1^{(1)}\Gamma_{10}^{(1)} + \gamma_2^{(1)}\Gamma_{20}^{(1)} \\ &= (\nu_0^{(1)} \cdot 0.52 + \nu_1^{(1)} \cdot 0.24 + \nu_2^{(1)} \cdot 0) + (0.3453 \cdot 0.01 + 0.1943 \cdot 0) \\ &= (\nu_0^{(1)} \cdot 0.52 + \nu_1^{(1)} \cdot 0.24 + \nu_2^{(1)} \cdot 0) + 0.003453.\end{aligned}$$

The remaining transition probabilities from macro states to micro states are summarized as follows:

$$\begin{aligned}
\Gamma_{11}^{(1)} &= 0.02, & \Gamma_{21}^{(1)} &= 0,\\
\Gamma_{12}^{(1)} &= 0.01, & \Gamma_{22}^{(1)} &= 0,\\
\Gamma_{03}^{(1)} &= 0.015, & \Gamma_{04}^{(1)} &= 0.015,\\
\Gamma_{24}^{(1)} &= 0.02, & \Gamma_{05}^{(1)} &= 0,\\
\Gamma_{15}^{(1)} &= 0.0225.
\end{aligned}$$

To complete disaggregation in the first iteration step, the following three sets of equations, for $\{\nu_0^{(1)}, \nu_1^{(1)}, \nu_2^{(1)}\}$, $\{\nu_3^{(1)}, \nu_4^{(1)}\}$, and $\{\nu_5^{(1)}\}$ need to be solved separately for a calculation of micro-state probabilities:

$$\begin{aligned}
\nu_0^{(1)} &= (\nu_0^{(1)} \cdot 0.52 + \nu_1^{(1)} \cdot 0.24) + 0.003453,\\
\nu_1^{(1)} &= (\nu_0^{(1)} \cdot 0.45 + \nu_1^{(1)} \cdot 0.265 + \nu_2^{(1)} \cdot 0.24) + 0.006906,\\
\nu_2^{(1)} &= (\nu_0^{(1)} \cdot 0 + \nu_1^{(1)} \cdot 0.45 + \nu_2^{(1)} \cdot 0.745) + 0.003453,\\
\nu_3^{(1)} &= (\nu_3^{(1)} \cdot 0.48 + \nu_4^{(1)} \cdot 0.24) + 0.010792,\\
\nu_4^{(1)} &= (\nu_3^{(1)} \cdot 0.45 + \nu_4^{(1)} \cdot 0.705) + 0.010792,\\
\nu_5^{(1)} &= (\nu_5^{(1)} \cdot 0.96) + 0.00776925.
\end{aligned}$$

With the solutions of these systems of equations and the normalization condition:

$$\boldsymbol{\nu}^{(1)}\mathbf{1} = 1,$$

the probability vector $\boldsymbol{\nu}^{(1)}$ at the end of the first iteration with a precision up to the fourth digit is:

$$\boldsymbol{\nu}^{(1)} = (0.0812, 0.1486, 0.2753, 0.1221, 0.1864, 0.1864)\,. \tag{4.32}$$

In Table 4.5, the micro-state probabilities obtained after one iteration of Takahashi's method are compared to the results gained if Courtois's method is applied. The percentage error in the results obtained with Courtois's approach are also included in the table. It can be seen that even after the first iteration step, the results are already relatively close to the exact values. After four iterations, the results gained with Takahashi's method resemble the exact values up to the fourth decimal digit.

4.2.5 Final Remarks

Takahashi's method was presented in terms of DMTCs. However this choice implies no limitation since it is a well-known fact that ergodic DTMCs and

Table 4.5 State probabilities: Takahashi's, Courtois's and exact methods

State	$\boldsymbol{\nu}^{(0)}$	$\boldsymbol{\nu}^{(1)}$	$\boldsymbol{\nu}^{(2)}$	$\boldsymbol{\nu}^{(3)}$	$\boldsymbol{\nu}^{(4)}$	Exact	% Error$^{(1)}$	% Error$^{(4)}$
200	$0.16\overline{6}$	0.0785	0.0787	0.0787	0.0787	0.0787	- 0.3	0.0
110	$0.16\overline{6}$	0.1436	0.1476	0.1478	0.1478	0.1478	- 2.8	0.0
020	$0.16\overline{6}$	0.2660	0.2770	0.2777	0.2777	0.2777	- 4.2	0.0
101	$0.16\overline{6}$	0.1179	0.1146	0.1144	0.1144	0.1144	+ 3.1	0.0
011	$0.16\overline{6}$	0.2138	0.2150	0.2150	0.2150	0.2150	- 0.6	0.0
002	$0.16\overline{6}$	0.1801	0.1672	0.1665	0.1664	0.1664	+ 8.2	0.0

State	Courtois	Exact	% Error
200	0.0828	0.0787	+ 5.2
110	0.1480	0.1478	+ 0.1
020	0.2710	0.2777	- 2.4
101	0.1194	0.1144	+ 4.4
011	0.2108	0.2150	- 2.0
002	0.1680	0.1664	+ 0.1

CTMCs are equivalent with respect to their steady-state probability computations, when applying the transformations given in Eq. (3.6) and Eq. (3.5). But Takahashi's method can be more conveniently applied directly on the generator matrix $\mathbf{Q} = [q_{ij}]$ of a CTMC. As usual, we refer to the steady-state probability vector of the given micro-state CTMC through $\boldsymbol{\pi}$ or $\boldsymbol{\pi}^{\approx}$, respectively. In the continuous-time case, the resulting equations correspond directly to their discrete-time counterparts. Instead of using Eq. (4.19), the macro-state probability vector $\boldsymbol{\sigma}$ is now calculated based on infinitesimal generator matrix $\boldsymbol{\Sigma}$ of the continuous-time macro state process:

$$\mathbf{0} = \boldsymbol{\sigma}\boldsymbol{\Sigma}, \quad \boldsymbol{\sigma}\mathbf{1} = 1. \tag{4.33}$$

The matrix entries Σ_{IJ} of $\boldsymbol{\Sigma}$ are calculated with an interpretation corresponding to the one applied to Eq. (4.21):

$$\Sigma_{IJ} = \sum_{i \in S_I} \sum_{j \in S_J} \frac{\pi_i^{\approx} q_{ij}}{\sum_{k \in S_I} \pi_k^{\approx}}, \quad I \neq J. \tag{4.34}$$

The disaggregation steps are analogous to Eq. (4.22) and Eq. (4.23) $\forall I, 0 \leq I \leq M-1$:

$$\Sigma_{Ij} = \sum_{i \in S_I} \frac{\pi_i^{\approx} q_{ij}}{\sum_{k \in S_I} \pi_k^{\approx}}, \quad j \in S_J, J \neq I, \tag{4.35}$$

$$\pi_i \sum_{j \in S, j \neq i} q_{ij} = \sum_{j \in S_I, j \neq i} \pi_j q_{ji} + \sum_{J=0, J \neq I}^{M-1} \sigma_J \Sigma_{Ji}, \quad \forall i \in S_I. \tag{4.36}$$

Problem 4.5 Compare Courtois's method to Takahashi's method for numerical accuracy on the example studies performed in Sections 4.1 and 4.2.4. In particular, explain the observed numerical differences in the corresponding macro-state transition probability matrices $\mathbf{\Gamma}$, in the corresponding macro-state probability vectors $\boldsymbol{\gamma}$, and in the corresponding micro-state probability vectors $\boldsymbol{\nu}$.

Problem 4.6 Apply decomposition with regard to station two, as indicated in Fig. 4.3, to the DTMC underlying the discussions in Sections 4.1 and 4.2.4. Build up and organize a transition probability matrix that properly reflects the chosen strategy of decomposition. Calculate the approximate lumpability error ϵ according to Eq. (4.26), compare it to the results obtained in Section 4.2.4, and interpret the differences.

Problem 4.7 Apply Takahashi's algorithm on the basis of the decomposition strategy with regard to station two, as discussed in the previous problem. Compare the resulting macro-state transition probability matrices, the macro-state probability vectors, and the micro-state probability vectors obtained in the iteration step to those obtained under the original decomposition strategy in Section 4.2.4. Comment on and interpret the results.

Problem 4.8 Use the modified algorithm of Takahashi and apply it for a numerical analysis of the CTMC specified by its generator matrix $\mathbf{Q}$ in Table 4.2 and the parameters in Table 4.1. Apply the same partition as indicated in Table 4.2.

5

Transient Solution of Markov Chains

Transient solution is more meaningful than steady-state solution when the system under investigation needs to be evaluated with respect to its short-term behavior. Using steady-state measures instead of transient measures could lead to substantial errors in this case. Furthermore, applying transient analysis is the only choice if non-ergodic models are investigated. Transient analysis of Markov chains has been attracting increasing attention and is of particular importance in dependability modeling.

Unlike steady-state analysis, CTMCs and DTMCs have to be treated differently while performing transient analysis. Surprisingly, not many algorithms exist for the transient analysis of DTMCs. Therefore, we primarily focus on methods for computing the transient state probability vector $\boldsymbol{\pi}(t)$ for CTMCs as defined in Eq. (2.53). Furthermore, additional attention is given to the computation of quantities related to transient probabilities such as cumulative measures.

Recall from Eq. (2.53) that for the computation of transient state probability vector $\boldsymbol{\pi}(t)$, the following linear differential equation has to be solved, given infinitesimal generator matrix $\mathbf{Q}$ and initial probability vector $\boldsymbol{\pi}(0)$:

$$\frac{d\boldsymbol{\pi}(t)}{dt} = \boldsymbol{\pi}(t)\mathbf{Q}, \quad \boldsymbol{\pi}(0) = (\pi_0(0), \pi_1(0), \ldots)\,. \tag{5.1}$$

Measures that can be immediately derived from transient state probabilities are often referred to as *instantaneous* measures. However, sometimes measures based on cumulative accomplishments during a given period of time

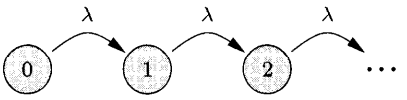


Fig. 5.1 A pure birth process.

$[0, t)$ could be more relevant. Let:

$$\mathbf{L}(t) = \int_0^t \boldsymbol{\pi}(u)du$$

denote the vector of the total expected times spent in the states of the CTMC during the indicated period of time. By integrating Eq. (2.53) on both sides, we obtain a new differential equation for $\mathbf{L}(t)$:

$$\frac{d\mathbf{L}(t)}{dt} = \mathbf{L}(t)\mathbf{Q} + \boldsymbol{\pi}(0), \quad \mathbf{L}(0) = \mathbf{0}. \tag{5.2}$$

Cumulative measures can be directly computed from the transient solution of Eq. (5.2).

5.1 TRANSIENT ANALYSIS USING EXACT METHODS

We introduce transient analysis with a simple example of a pure birth process.

5.1.1 A Pure Birth Process

As for birth-death processes in steady-state case, we derive a transient closed-form solution in this special case. Consider the infinite state CTMC depicted in Fig. 5.1 representing a pure birth process with constant birth rate λ. The number of births, $N(t)$, at time t is defined to be the state of the system. The only transitions possible are from state k to state $k+1$ with rate λ. Note that this is a non-irreducible Markov chain for any finite value of λ, so the steady-state solution does not exist.

From Fig. 5.1, the infinitesimal generator matrix $\mathbf{Q}$ is given by:

$$\mathbf{Q} = \begin{pmatrix} -\lambda & \lambda & 0 & 0 & \dots \\ 0 & -\lambda & \lambda & 0 & \dots \\ 0 & 0 & -\lambda & \lambda & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

Due to the special structure of the generator matrix, it is possible to obtain a closed-form transient solution of this process. With generator matrix $\mathbf{Q}$ and

Eq. (2.53), we derive a system of linear differential equations for our example:

$$\frac{d}{dt}\pi_0(t) = -\lambda\pi_0(t), \tag{5.3}$$

$$\frac{d}{dt}\pi_k(t) = -\lambda\pi_k(t) + \lambda\pi_{k-1}(t), \quad k \geq 1. \tag{5.4}$$

With the initial state probabilities:

$$\pi_k(0) = \begin{cases} 1 & k = 0, \\ 0 & k \geq 1, \end{cases} \tag{5.5}$$

the solution of the differential equations can be obtained. From differentiation and integration theory the unique solution of Eq. (5.3) is:

$$\pi_0(t) = e^{-\lambda t}. \tag{5.6}$$

By letting $k = 1$ in Eq. (5.4) and subsequently substituting the solution for $\pi_0(t)$ into the differential equation, we get:

$$\begin{aligned} \frac{d}{dt}\pi_1(t) &= -\lambda\pi_1(t) + \lambda\pi_0(t) \\ &= -\lambda\pi_1(t) + \lambda e^{-\lambda t}. \end{aligned}$$

Applying again elementary differentiation and integration rules to this differential equation results in another simple unique solution:

$$\pi_1(t) = \lambda t e^{-\lambda t}. \tag{5.7}$$

If this process is repeated, we get a closed-form solution for each transient state probability $\pi_k(t)$:

$$\pi_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}, \quad k \geq 0. \tag{5.8}$$

The proof of Eq. (5.8) is straightforward using the principle of induction and we leave it as an exercise to the reader. We recognize Eq. (5.8) as the *Poisson pmf*. Poisson probabilities are plotted as functions of t in Fig. 5.2 for several values of λ. Thus, the random variable $N(t)$ at time t is Poisson distributed with parameter λt, while the stochastic process $\{N(t)|t \geq 0\}$ is the Poisson process with rate λ. So the transient analysis of a very simple birth process provided as a by-product one of the most important results of Markov chain and queueing theory, i.e., the stochastic process under consideration is the Poisson process. One important measure of the Poisson process is its mean value function $m(t)$, defined as the expected number of births in the

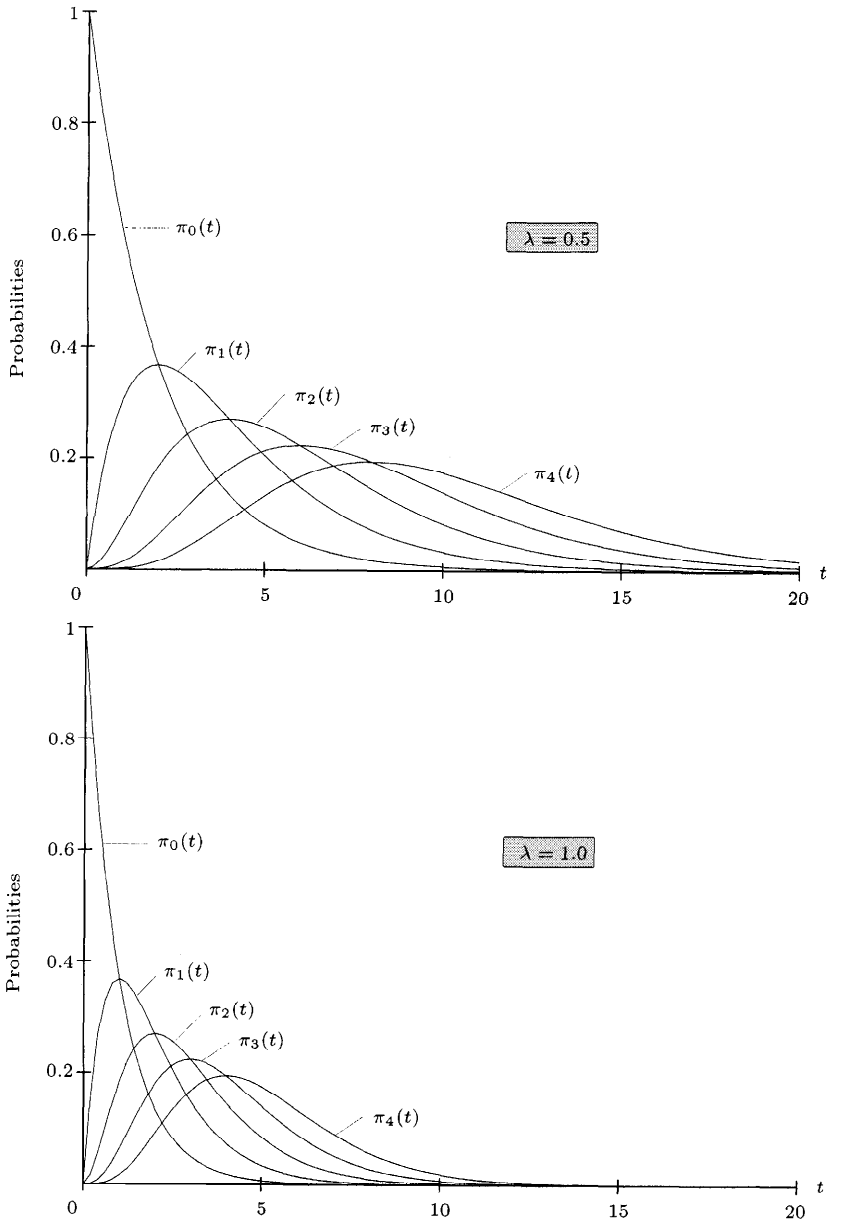


Fig. 5.2 Selected Poisson probabilities with parameters $\lambda = 0.5$ and 1.0.

interval $[0, t)$. This measure is computed as:

$$\begin{aligned} m(t) &= \sum_{i=0}^{\infty} i\pi_i(t) = \sum_{i=0}^{\infty} i\frac{(\lambda t)^i}{i!}e^{-\lambda t} \\ &= e^{-\lambda t}\sum_{i=1}^{\infty}\frac{(\lambda t)^{i-1}}{(i-1)!}(\lambda t) = (\lambda t)e^{-\lambda t}\sum_{i=0}^{\infty}\frac{(\lambda t)^i}{i!} \\ &= (\lambda t)e^{-\lambda t}e^{\lambda t} = \lambda t. \end{aligned} \tag{5.9}$$

Eq. (5.8) together with Eq. (5.9) characterize the Poisson pmf and its mean λt.

Problem 5.1 Using the principle of mathematical induction, show that the Poisson pmf as given in Eq. (5.8) is the solution to Eq. (5.4).

Problem 5.2 Derive the solution of Eq. (5.2) of the pure birth process of Fig. 5.1, assuming $\pi_0(0) = 1$ and $\pi_k(0) = 0, \quad \forall k \geq 1$.

5.1.2 A Two-State CTMC

While in the previous section transient analysis based on closed-form expressions was possible due to very simple and regular model structure, in this section we carry out closed-form transient analysis of a CTMC with two states. Our treatment closely follows that in [STP96]. Fig. 5.3 shows a homogeneous, continuous time Markov chain model for this system with state space $S = \{0, 1\}$. A typical application of this model is to a system subject to failure and repair.

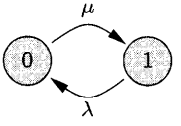


Fig. 5.3 A simple two-state CTMC.

The generator matrix of the CTMC of Fig. 5.3 is:

$$\mathbf{Q} = \begin{bmatrix} -\mu & \mu \\ \lambda & -\lambda \end{bmatrix}.$$

From the system of Eqs. (2.51), we get:

$$\begin{aligned} \frac{d}{dt}\pi_0(t) &= -\mu\pi_0(t) + \lambda\pi_1(t), \\ \frac{d}{dt}\pi_1(t) &= \mu\pi_0(t) - \lambda\pi_1(t). \end{aligned} \tag{5.10}$$

Applying the law of total probability, we conclude:

$$\pi_0(t) = 1 - \pi_1(t),$$

and rewrite Eq. (5.10) as:

$$\frac{d}{dt}\pi_1(t) = \mu(1 - \pi_1(t)) - \lambda\pi_1(t)$$

or:

$$\frac{d}{dt}\pi_1(t) + (\mu + \lambda)\pi_1(t) = \mu. \tag{5.11}$$

A standard method of analysis for linear differential Eq. (5.11) is to use the method of integrating factors [RaBe74]. Both sides of the (rearranged) equation are multiplied by the integrating factor:

$$e^{\int(\mu+\lambda)dt} = e^{(\mu+\lambda)t},$$

to get:

$$\mu e^{(\mu+\lambda)t} = e^{(\mu+\lambda)t}\frac{d}{dt}\pi_1(t) + (\mu + \lambda)e^{(\mu+\lambda)t}\pi_1(t) \tag{5.12}$$

$$= \frac{d}{dt}(e^{(\mu+\lambda)t}\pi_1(t)). \tag{5.13}$$

By inspection of Eq. (5.12) it can be seen that the sum expression on the right side of the equation equals the derivative of the product of the subterms so that Eq. (5.13) results. Integrating Eq. (5.13) on both sides gives us as an intermediate step:

$$\frac{\mu}{\mu + \lambda}e^{(\mu+\lambda)t} + c = e^{(\mu+\lambda)t}\pi_1(t). \tag{5.14}$$

Multiplying Eq. (5.14) with:

$$e^{-(\mu+\lambda)t}$$

provides the desired result:

$$\pi_1(t) = \frac{\mu}{\mu + \lambda} + ce^{-(\mu+\lambda)t}. \tag{5.15}$$

Integration constant c reflects the dependency of the transient state probabilities on the initial probability vector $\boldsymbol{\pi}(0)$. Assuming, for example:

$$\boldsymbol{\pi}^{(1)}(0) = (0, 1),$$

results in:

$$1 = \pi_1^{(1)}(0) = \frac{\mu}{\mu + \lambda} + ce^{-(\mu+\lambda)0}$$

$$= \frac{\mu}{\mu + \lambda} + c.$$

Thus, unconditioning with an initial pmf $\boldsymbol{\pi}^{(1)}(0)$ yields the corresponding integration constant:

$$c = 1 - \frac{\mu}{\mu+\lambda} = \frac{\lambda}{\mu+\lambda}.$$

The final expressions of the transient state probabilities in this case results in:

$$\pi_1^{(1)}(t) = \frac{\mu}{\mu+\lambda} + \frac{\lambda}{\mu+\lambda}\left(e^{-(\mu+\lambda)t}\right), \tag{5.16}$$

$$\begin{aligned}\pi_0^{(1)}(t) &= 1 - \pi_1^{(1)}(t) \\ &= \frac{\mu+\lambda}{\mu+\lambda} - \frac{\mu+\lambda\left(e^{-(\mu+\lambda)t}\right)}{\mu+\lambda} \\ &= \frac{\lambda - \lambda\left(e^{-(\mu+\lambda)t}\right)}{\mu+\lambda} \\ &= \frac{\lambda}{\mu+\lambda} - \frac{\lambda}{\mu+\lambda}\left(e^{-(\mu+\lambda)t}\right).\end{aligned} \tag{5.17}$$

With a closed-form expression of the transient state probabilities given, performance measure can be easily derived.

Problem 5.3 First, derive expressions for the steady-state probabilities of the CTMC shown in Fig. 5.3 by referring to Eq. (2.58). Then, take the limits as $t \to \infty$ of the transient pmf $\boldsymbol{\pi}^{(1)}(t)$ and compare the results to the steady-state case $\boldsymbol{\pi}$.

Problem 5.4 Derive the transient state probabilities of the CMTC shown in Fig. 5.3 assuming initial pmf $\boldsymbol{\pi}^{(2)}(0) = (0.5, 0.5)$. Take the limit as $t \to \infty$ of the resulting transient pmf $\boldsymbol{\pi}^{(2)}(t)$ and compare the results to those gained with $\boldsymbol{\pi}^{(1)}(t)$ when taking the limit.

Problem 5.5 Assume the system modeled by the CTMC in Fig. 5.3 to be "up" in state 1 and to be "down" in state 0.

(a) Define the reward assignments for a computation of availability measures based on Table 2.1.

(b) Derive formulas for the instantaneous availability $A(t)$, the interval unavailability $\overline{UA}(t)$, and the steady-state availability A for this model, based both on $\boldsymbol{\pi}^{(1)}(t)$ and $\boldsymbol{\pi}^{(2)}(t)$. Refer to Section 2.2.2.3.1 for details on how these availability measures are defined.

(c) Let $\mu = 1$, $\lambda = 0.2$ and compute all measures that have been specified in this problem. Evaluate the transient measures at time instants $t \in \{1, 2, 3, 5, 10\}$.

5.1.3 Solution Using Laplace Transforms

It is well known that *Laplace transforms* can be used for the solution of linear differential equations such as those in Eqs. (5.1) or (5.2). Unfortunately, the applicability of this method is limited to problem domains with small state spaces or to those that imply a regular matrix structure. These restrictions are due to the difficulties immanent in computations of roots of a polynomial and in the computation of the inverse Laplace transform. We will not go into detail with respect to the applicability of the Laplace transform method for the computation of transient state probabilities but refer the reader to [RaTr95, STP96, Triv82], where further information is given.

With the given initial probability vector $\boldsymbol{\pi}(0)$, the Laplace transform of Eq. (2.53) yields the following equation in terms of transform variable s:

$$s\boldsymbol{\pi}(s) - \boldsymbol{\pi}(0) = \boldsymbol{\pi}(s)\mathbf{Q}. \tag{5.18}$$

Eq. (5.18) can be solved for the transformed probability vector $\boldsymbol{\pi}(s)$:

$$\boldsymbol{\pi}(s) = \boldsymbol{\pi}(0)(s\mathbf{I} - \mathbf{Q})^{-1}. \tag{5.19}$$

Note that the closed-form solution obtainable from Eq. (5.19) is completely symbolic with respect to both the system parameters and the time variable t. However, the actual use of this expression requires the computation of all the roots of the characteristic polynomial (equivalently, all the eigenvalues of $\mathbf{Q}$) and subsequent symbolic inversion of Laplace transform [Triv82]. Thus, fully symbolic solution is restricted to very small CTMCs or a CTMC with a highly structured matrix. Seminumerical solution, where all model parameters are required to be in numerical form while the time variable t is symbolic, is possible for models of moderate sizes (about 500 states) [RaTr95, STP96]. For larger models, numerical solution is the only viable approach.

5.1.4 Numerical Solution Using Uniformization

We first introduce uniformization method for the computation of instantaneous state probabilities. Next we discuss a method to deal with stiff Markov chains. Finally, we consider the computation of cumulative state probabilities.

5.1.4.1 The Instantaneous Case Transient uniformization, originally also referred to as Jensen's method [Jens53] for a numerical solution of Eq. (2.53), is commonly considered as the method of choice under most circumstances. Recently, the method has been adapted by Muppala et al. [MMT94, MuTr92] in such a way that the important sub-class of so-called *stiff* Markov chains can be analyzed more efficiently and accurately. Further effort has been devoted to extend the original algorithm so that *cumulative* measures can be efficiently computed [MMT96, ReTr89, SoGa89].

Transient uniformization takes advantage of Eq. (3.5) to embed a DTMC into a given CTMC as we have seen earlier. Hence, the transient state prob-

ability vector $\boldsymbol{\pi}(t)$ of the CTMC is expressed in terms of a power series of the one-step transition probability matrix $\mathbf{P} = \mathbf{Q}/q + \mathbf{I}$ of the embedded DTMC. Since the transient state probability vector is computed at time t, an unconditioning needs to be performed with respect to the number of jumps the DTMC makes during the interval $[0, t)$. As a result of the transformation, state transitions occur from all states with a *uniform* rate q.

The one-step transition probabilities of the DTMC are defined as follows:

$$p_{ji} = \begin{cases} \dfrac{q_{ji}}{q}, & j \neq i, \\ 1 - \displaystyle\sum_{k \neq j} \frac{q_{jk}}{q}, & j = i. \end{cases} \tag{5.20}$$

The number of transitions of the DTMC during the interval $[0, t)$ is governed by a Poisson random variable with parameter qt. Thus, the desired CTMC probability vector at time t can be written in terms of the DTMC state probability vector $\boldsymbol{\nu}(i)$ at step i:

$$\boldsymbol{\pi}(t) = \sum_{i=0}^{\infty} \boldsymbol{\nu}(i) e^{-qt} \frac{(qt)^i}{i!}, \quad \boldsymbol{\nu}(0) = \boldsymbol{\pi}(0). \tag{5.21}$$

The DTMC state probability vector can be iteratively computed:

$$\boldsymbol{\nu}(i) = \boldsymbol{\nu}(i-1)\mathbf{P}. \tag{5.22}$$

Recall that in order to avoid periodicity of the embedded DTMC, parameter q has to be chosen so that $q > \max_{i,j} |q_{i,j}|$. The infinite series in Eq. (5.21) needs to be truncated for the purposes of numerical computation. Fortunately, a right truncation point r can be determined with a numerically bounded error ϵ_r for an approximation $\tilde{\boldsymbol{\pi}}(t)$ with respect to some vector norm, such as the maximum norm:

$$\begin{aligned} \|\boldsymbol{\pi}(t) - \tilde{\boldsymbol{\pi}}(t)\|_\infty &= \left\| \sum_{i=0}^{\infty} \boldsymbol{\nu}(i) e^{-qt} \frac{(qt)^i}{i!} - \sum_{i=0}^{r} \boldsymbol{\nu}(i) e^{-qt} \frac{(qt)^i}{i!} \right\|_\infty \\ &= \left\| \sum_{i=r+1}^{\infty} \boldsymbol{\nu}(i) e^{-qt} \frac{(qt)^i}{i!} \right\|_\infty \\ &\leq \sum_{i=r+1}^{\infty} e^{-qt} \frac{(qt)^i}{i!} \\ &= 1 - \sum_{i=0}^{r} e^{-qt} \frac{(qt)^i}{i!} \\ &\leq \epsilon_r. \end{aligned} \tag{5.23}$$

For large values of qt, r tends to increase and the lower terms contribute increasingly less to the sum. To avoid numerical inaccuracies stemming from

computations with small numbers, a left truncation point l can be introduced. To this end an overall error tolerance $\epsilon = \epsilon_l + \epsilon_r$ is partitioned to cover both left and right truncation errors. Applying again a vector norm, a left truncation point l can be determined similarly to the right truncation point r:

$$\sum_{i=0}^{l-1} e^{-qt} \frac{(qt)^i}{i!} \leq \epsilon_l. \tag{5.24}$$

With appropriately defined truncation points l and r as in Eqs. (5.23) and (5.24), the original uniformization Eq. (5.21) can be approximated for an implementation:

$$\boldsymbol{\pi}(t) \approx \sum_{i=l}^{r} \boldsymbol{\nu}(i) e^{-qt} \frac{(qt)^i}{i!}, \quad \boldsymbol{\nu}(0) = \boldsymbol{\pi}(0). \tag{5.25}$$

In particular, $O(\sqrt{qt})$ terms are needed between the left and the right truncation points and the transient DTMC state probability vector $\boldsymbol{\nu}(l)$ must be computed at the left truncation point l. The latter operation according to Eq. (5.22) requires $O(qt)$ computation time. Thus the overall complexity of uniformization is $O(\eta qt)$ [ReTr88]. (Here η denotes the number of non-zero entries in $\mathbf{Q}$.) To avoid underflow, the method of Fox and Glynn can be applied in the computation of l and r [FoGl88]. Matrix squaring can be exploited for a reduction of computational complexity [ReTr88, AbMa93]. But because matrix fill-ins might result, this approach is often limited to models of medium size (around 500 states).

5.1.4.2 Stiffness Tolerant Uniformization As noted, uniformization is plagued by the stiffness index qt [ReTr88]. Muppala and Trivedi observed that the most time consuming part of uniformization is the iteration to compute $\boldsymbol{\nu}(i)$ in Eq. (5.22). But this iteration is the main step in the power method for computing the steady-state probabilities of a DTMC as discussed in Chapter 2. Now if and when the values of $\boldsymbol{\nu}(i)$ converge to a stationary value $\tilde{\boldsymbol{\nu}}$, the iteration Eq. (5.22) can be terminated resulting in considerable savings in computation time [MuTr92]. We have also seen in Chapter 2 that speed of convergence to a stationary probability vector is governed by the second largest eigenvalue of the DTMC, but not by qt. However, since the probability vectors $\boldsymbol{\nu}(i)$ are computed iteratively according to the power method, convergence still remains to be effectively determined. Because an a priori determination of time of convergence is not feasible, three cases have to be differentiated for a computation of the transient state probability vector $\boldsymbol{\pi}(t)$ [MuTr92, MMT94]:

1. Convergence occurs beyond the right truncation point. In this case, computation of a stationary probability vector is not effective and the transient state probability vector $\boldsymbol{\pi}(t)$ is calculated according to Eq. (5.25) without modification.

2. Convergence occurs between left and right truncation points at time c, $l < c \leq r$. Here advantage can be taken of the fact that $\boldsymbol{\nu}(i) = \boldsymbol{\nu}(c) = \tilde{\boldsymbol{\nu}}$, $\forall i > c$. By letting the right truncation point $r \to \infty$, we can rewrite Eq. (5.25) as:

$$\begin{aligned}\boldsymbol{\pi}(t) &\approx \sum_{i=l}^{\infty} \boldsymbol{\nu}(i) e^{-qt} \frac{(qt)^i}{i!} \\ &= \sum_{i=l}^{c} \boldsymbol{\nu}(i) e^{-qt} \frac{(qt)^i}{i!} + \boldsymbol{\nu}(c) \sum_{i=c+1}^{\infty} e^{-qt} \frac{(qt)^i}{i!} \\ &= \sum_{i=l}^{c} \boldsymbol{\nu}(i) e^{-qt} \frac{(qt)^i}{i!} + \boldsymbol{\nu}(c)(1 - \sum_{i=0}^{c} e^{-qt} \frac{(qt)^i}{i!}). \end{aligned} \tag{5.26}$$

3. Convergence occurs before the left truncation point, that is, $c < l$. In this case, we get:

$$\boldsymbol{\pi}(t) = \boldsymbol{\nu}(c) = \tilde{\boldsymbol{\nu}}. \tag{5.27}$$

Recall that stationary pmfs do not necessarily exhibit the properties of steady-state probability vectors in our terminology. In particular, the CTMCs need not be ergodic for the approach of stiffness tolerant uniformization to work. The only required property is that the DTMC must be aperiodic for a stationary pmf to exist (see Section 2.1.2 for more details). Issues of steady-state or stationary pmf detection by means of applying different norms are discussed in Chapters 2 and 4. More details can be found in [CBC+92, StGo85, Stew94]. Error bounds and complexity estimations of the stiffness tolerant uniformization algorithm discussed in this section are given in [MMT94].

5.1.4.3 The Cumulative Case Uniformization can also be extended to the cumulative case in Eq. (5.2) [ReTr89]. By integration of Eq. (5.21) with respect to time parameter t, we get:

$$\mathbf{L}(t) = \frac{1}{q} \sum_{i=0}^{\infty} \boldsymbol{\nu}(i) \sum_{j=i+1}^{\infty} e^{-qt} \frac{(qt)^j}{j!} = \frac{1}{q} \sum_{i=0}^{\infty} \boldsymbol{\nu}(i) \left(1 - \sum_{j=0}^{i} e^{-qt} \frac{(qt)^j}{j!}\right). \tag{5.28}$$

The infinite series needs to be truncated again, so that Eq. (5.28) reduces to:

$$\mathbf{L}(t) \approx \frac{1}{q} \sum_{i=0}^{r} \boldsymbol{\nu}(i) \left(1 - \sum_{j=0}^{i} e^{-qt} \frac{(qt)^j}{j!}\right). \tag{5.29}$$

Again, techniques for detecting stationarity of pmfs can be applied for making uniformization less susceptible to stiffness and more efficient [MMT96]. Two cases need to be differentiated with respect to the relation of convergence time step c to truncation point r:

1. Convergence occurs beyond the truncation point, that is, $c > r$. As a consequence, Eq. (5.29) remains unaffected and must be evaluated as is.

2. If convergence occurs before truncation point, Eq. (5.29) is adjusted to:

$$\begin{aligned}
\mathbf{L}(t) &= \frac{1}{q}\sum_{i=0}^{\infty}\boldsymbol{\nu}(i)\mathbf{L}'(t) \\
&= \frac{1}{q}\sum_{i=0}^{c}\boldsymbol{\nu}(i)\mathbf{L}'(t) + \frac{1}{q}\boldsymbol{\nu}(c)\sum_{i=c+1}^{\infty}\mathbf{L}'(t) \\
&= \frac{1}{q}\sum_{i=0}^{c}\boldsymbol{\nu}(i)\mathbf{L}'(t) + \frac{1}{q}\boldsymbol{\nu}(c)\left(\sum_{i=0}^{\infty}\mathbf{L}'(t) - \sum_{i=0}^{c}\mathbf{L}'(t)\right) \\
&= \frac{1}{q}\sum_{i=0}^{c}\boldsymbol{\nu}(i)\mathbf{L}'(t) + \frac{1}{q}\boldsymbol{\nu}(c)\left(qt - \sum_{i=0}^{c}\mathbf{L}'(t)\right) \\
&= \frac{1}{q}\sum_{i=0}^{c}\boldsymbol{\nu}(i)\left(1 - \sum_{j=0}^{i}e^{-qt}\frac{(qt)^j}{j!}\right) \\
&\quad + \frac{1}{q}\boldsymbol{\nu}(c)\left(qt - \sum_{i=0}^{c}\left(1 - \sum_{j=0}^{i}e^{-qt}\frac{(qt)^j}{j!}\right)\right),
\end{aligned} \tag{5.30}$$

with

$$\mathbf{L}'(t) = \sum_{j=i+1}^{\infty}e^{-qt}\frac{(qt)^j}{j!} = \left(1 - \sum_{j=0}^{i}e^{-qt}\frac{(qt)^j}{j!}\right).$$

In case truncation needs to be performed, a time-dependent error-bound estimate $\epsilon^{(r)}(t)$ related to Eq. (5.29) can be given as a function of the truncation point r. Assuming an error tolerance allowance, the corresponding number of required terms r can be determined:

$$\begin{aligned}
\epsilon^{(r)}(t) &\leq \frac{1}{q}\sum_{i=r+1}^{\infty}\sum_{j=i+1}^{\infty}e^{-qt}\frac{(qt)^j}{j!} \\
&\leq \frac{1}{q}\sum_{i=r+1}^{\infty}(i-(r+1))\,e^{-qt}\frac{(qt)^i}{i!} \\
&\leq \frac{1}{q}\sum_{i=r+1}^{\infty}ie^{-qt}\frac{(qt)^i}{i!} - \frac{1}{q}\sum_{i=r+1}^{\infty}(r+1)\,e^{-qt}\frac{(qt)^i}{i!} \\
&\leq t\sum_{i=r}^{\infty}e^{-qt}\frac{(qt)^i}{i!} - \frac{r+1}{q}\sum_{i=r+1}^{\infty}e^{-qt}\frac{(qt)^i}{i!}.
\end{aligned} \tag{5.31}$$

5.1.5 Other Numerical Methods

We briefly discuss numerical methods based on ordinary differential equation approach followed by methods exploiting weak lumpability.

5.1.5.1 Ordinary Differential Equations Standard techniques for the solution of ordinary differential equations (ODE) can be utilized for the numerical solution of the Kolmogorov differential equations of a CTMC. Such ODE solution methods discretize the solution interval into a finite number of time intervals $\{t_1, t_2, \ldots, t_i, \ldots, t_n\}$. The difference between successive time points, called the step size h, can vary from step to step. There are two basic types of ODE solution methods: *explicit* and *implicit*.

In an explicit method, the solution $\boldsymbol{\pi}(t_i)$ is approximated based on values $\boldsymbol{\pi}(t_j)$ for $j < i$. The computational complexity of explicit methods such as Runge-Kutta is $O(\eta q t)$ [ReTr88]. Although explicit ODE-based methods generally provide good results for non-stiff models, they are inadequate if stiff models need to be studied. Note that stiff CTMCs are commonly encountered in dependability modeling.

In an implicit method, $\boldsymbol{\pi}(t_i)$ is approximated based on values $\boldsymbol{\pi}(t_j)$ for $j \leq i$. Examples of implicit methods are TR-BDF2 [ReTr88] and implicit Runge-Kutta [MMT94]. At each time step a linear system solution is required in an implicit method. The increased overhead is compensated for by better stability properties and lower computational complexity on stiff models [MMT94, ReTr88].

For non-stiff models, Uniformization is the method of choice, while for moderately stiff models, uniformization with steady-state detection is recommended [MMT94]. For extremely stiff models, TR-BDF2 works well if the accuracy required is low (eight decimal digits). For high accuracy on extremely stiff models, implicit Runge-Kutta is recommended [MMT94].

5.1.5.2 Weak Lumpability An alternative method for transient analysis based on weak lumpability has been introduced by Nicola [Nico90]. Nicola's method is the transient counterpart of Takahashi's steady-state method, which was introduced in Section 4.2. Recall that state lumping is an exact approach for an analysis of a CMTC with reduced state space and, therefore, with reduced computational requirements. State lumping can be a very efficient method for a computation of transient state probabilities, if the lumpability conditions apply. Note that state lumping can also be orthogonally combined with other computational methods of choice, such as stiff uniformization, to yield an overall highly accurate and efficient method.

For the sake of completeness, we present the basic definitions of weak lumpability without going into further detail here. Given conditional proba-

bility vector $\boldsymbol{\nu}_J^*$ and an initial probability vector $\boldsymbol{\nu}(0)$, with:

$$\nu_{Jj}^* = \frac{\nu_j(0)}{\sum\limits_{k \in S_J} \nu_k(0)}, \tag{5.32}$$

then a subset $S_J \subset S$ is *weakly lumpable* with respect to $\boldsymbol{\nu}(0)$ if for all $i \in S - S_J$ real-valued constants $0 \leq r_{iJ}, \leq 1$ exist such that Conditions (5.33) and (5.34) hold:

$$\frac{p_{ij}}{\nu_{Jj}^*} = r_{iJ}, \quad \forall j \in S_J, \tag{5.33}$$

$$\sum_{k \in S_J} p_{kj} \frac{\nu_{Jk}^*}{\nu_{Jj}^*} = r_{JJ}, \quad \forall j \in S_J. \tag{5.34}$$

A CTMC is weakly lumpable for a given initial pmf, while (strong) lumpability, as it was introduced in Section 4.2, implied a lumpability of a CTMC irrespective of the initial pmf [Abde82, KeSn78]. Nicola also extended (strong and weak) lumpability of CTMCs to (strong and weak) lumpability of MRMs. In particular, he identified measure-dependent conditions for lumping an MRM (see Section 2.2.2 for details on relevant measures). Lumping of an MRM requires conditions on lumping of the underlying CTMC plus further conditions on the reward rates attached to the states.

5.2 AGGREGATION OF STIFF MARKOV CHAINS

Bobbio and Trivedi [BoTr86] have introduced an approximate method for the computation of transient state probabilities $\boldsymbol{\pi}^{\approx}(t)$ of finite CTMCs that can be regarded as an extension of Courtois's method, which was presented in Section 4.1. Since transient analysis does not require ergodicity of the investigated models, the aggregation and disaggregation have to be applied differently with respect to different subsets of states. Therefore, in contrast to Courtois's method, aggregation/disaggregation also needs to be performed for the *initial state probability vector*, and for *transient states*. The aggregation/disaggregation of *recurrent* subsets of states is performed analogous to Courtois's method. It is worth emphasizing that the method introduced in this section not only provides a technique for an efficient, approximate computation of transient state probabilities of possibly large CTMCs, but is also often applicable where other transient numerical techniques fail. This application is especially important in cases where the transition rates of the infinitesimal generator matrix differ by several orders of magnitude.

5.2.1 Outline and Basic Definitions

Bobbio and Trivedi motivate their approach by observing that transient analysis is often performed for models that exhibit the so-called *stiffness* property. These models usually include fast states and slow states such that the transition rates differ by several orders of magnitude. A state is called *fast* if at least one rate leaving that state is classified to be high. If all its outgoing rates are classified to be low, the state is called *slow*. Subsets of fast recurrent states are aggregated to macro states with a similar argument as had been used by Courtois when taking advantage of near complete decomposability. Any slow state is by definition a macro state. Subsets of transient fast states are eliminated by adjusting the transition rates out of the resulting macro states.

It is convenient to characterize a transition rate by comparison to the ratio $1/T$, where the transient computation is to be performed for the period of time $[0, T)$. Rates are high if they are larger than some threshold τ, with $\tau >> 1/T$. Low rates, on the other hand, are of the order of magnitude of $1/T$. The actual computation of transient state probabilities is only performed for the slow or aggregated macro states. It is assumed that the fast states within an aggregate reach steady state at a time scale much shorter than that describing the dynamics among the macro states.

A subset $S_I \subset S$ of fast states is aggregated into a macro state I if and only if the following conditions are satisfied:

1. Between any pairs $i, j \in S_I$ of states within the subset a directed path from i to j, and vice versa, exists such that the path is defined along fast transitions only.

2. No other state outside of S_I can be reached from within S_I via a transition with a fast rate.

An immediate consequence of this definition is that transitions from the states of a fast recurrent subset S_I to any state in a disjoint subset of states $S_J, J \neq I$, have only low rates attached. These subsets of states can therefore be regarded as separable from the remaining states in the sense of Courtois's near complete decomposability property. In terms of matrix notation, the infinitesimal generator matrix $\mathbf{Q}$ can be reorganized in a nearly block diagonally dominant form. The transition rates inside each diagonal block dominate the transition rates between blocks.

Assume that the state space S has been partitioned into $F + 1$ subsets $S_I, 0 \leq I \leq F$. The set of slow states initially present in the model is contained in S_0. $S_1, S_2, \ldots, S_{F-1}$ are the subsets of fast recurrent states, and S_F is the possibly empty set of fast transient states. The cardinality of any set of states $S_J, 0 \leq J \leq F$, is denoted by $n_J = |S_J|$. Without loss of generality, the generator matrix $\mathbf{Q}$ can be restructured in a block form reflecting the

previously described state partition:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} & \cdots & \mathbf{A}_{0F} \\ \mathbf{A}_{10} & \mathbf{Q}_1 & \cdots & \mathbf{A}_{1F} \\ \vdots & & \ddots & \vdots \\ \mathbf{A}_{(F-1)0} & \cdots & \mathbf{Q}_{F-1} & \mathbf{A}_{(F-1)F} \\ \mathbf{A}_{F0} & \cdots & \mathbf{A}_{F(F-1)} & \mathbf{A}_{FF} \end{pmatrix}. \tag{5.35}$$

Let $\mathbf{Q}_I = \mathbf{Q}_{II}, 1 \leq I \leq F-1$, denote submatrices containing transition rates between grouped states within fast recurrent subset S_I only. Submatrices $\mathbf{A}_{IJ}, I \neq J$, contain entries of transitions rates from subset S_I to S_J. Finally, $\mathbf{A}_{00}$ and $\mathbf{A}_{FF}$ collect intra slow states and intra fast transient states transition rates, respectively.

Matrices $\mathbf{Q}_I, 1 \leq I \leq F-1$, contain at least one fast entry in each row. Furthermore, there must be at least one fast entry in one of the matrices $\mathbf{A}_{FI}, 0 \leq I \leq F-1$ if $S_F \neq \emptyset$. Matrix $\mathbf{A}_{FF}$ may contain zero or more fast entries. Finally, all other matrices, i.e., $\mathbf{A}_{00}, \mathbf{A}_{IJ}, I \neq J, 0 \leq I \leq F-1, 0 \leq J \leq F$, contain only slow entries. By definition, only slow transitions are possible among these subsets S_I and S_J. An approximation to $\boldsymbol{\pi}(t)$ is derived, where $\boldsymbol{\pi}(t)$ is the solution to the differential Eq. (5.36):

$$\frac{d}{dt}\boldsymbol{\pi}(t) = \boldsymbol{\pi}(t)\mathbf{Q}, \quad \boldsymbol{\pi}(0) = \left(\pi_{0_0}(0), \pi_{0_1}(0), \ldots, \pi_{F_{n_F-1}}(0)\right). \tag{5.36}$$

The reorganized matrix $\mathbf{Q}$ forms the basis to create the macro-state generator matrix $\boldsymbol{\Sigma}$. Three steps have to be carried out for this purpose: first, aggregation of the fast recurrent subsets into macro states and the corresponding adaptation of the transition rates among macro states and remaining fast transient states, resulting in intermediate generator matrix $\tilde{\boldsymbol{\Sigma}}$. In a second step, the fast transient states are eliminated and the transition rates between the remaining slow states are adjusted, yielding the final generator matrix $\boldsymbol{\Sigma}$. Finally, the initial state probability vector $\boldsymbol{\pi}(0)$ is condensed into $\boldsymbol{\sigma}(0)$ as per the aggregation pattern. Transient solution of differential Eq. (5.37) describing the long-term interactions between macro states is carried out:

$$\begin{aligned} &\frac{d}{dt}\boldsymbol{\sigma}(t) = \boldsymbol{\sigma}(t)\boldsymbol{\Sigma}, \\ &\boldsymbol{\sigma}(0) = \left(\sigma_{0_0}(0), \ldots, \sigma_{0_{n_0-1}}(0), \sigma_1(0), \ldots, \sigma_{F-1}(0)\right). \end{aligned} \tag{5.37}$$

Once the macro state probability vector $\boldsymbol{\sigma}(t)$ has been computed, disaggregations can be performed to yield an approximation $\boldsymbol{\pi}^{\approx}(t)$ of the complete probability vector $\boldsymbol{\pi}(t)$.

5.2.2 Aggregation of Fast Recurrent Subsets

Each subset of fast recurrent states is analyzed in isolation from the rest of the system by cutting off all slow transitions leading out of the aggregate. Collecting all states from such a subset S_I and arranging them together with the

corresponding entries of the originally given infinitesimal generator matrix $\mathbf{Q}$ into a submatrix $\mathbf{Q}_I$ gives rise to the possibility of computing the conditional steady-state probability vectors $\boldsymbol{\pi}_I^*, 1 \leq I \leq F-1$. Since $\mathbf{Q}_I$ does not, in general, satisfy the properties of an infinitesimal generator matrix, it needs to be modified to matrix $\mathbf{Q}_I^*$:

$$\mathbf{Q}_I^* = \mathbf{Q}_I + \mathbf{D}_I. \tag{5.38}$$

Matrix $\mathbf{D}_I$ is a diagonal matrix whose entries are the sum of all cut-off transition rates for the corresponding state. Note that these rates are, by definition, orders of magnitude smaller than the entries on the diagonal of $\mathbf{Q}_I$. The inverse of this quantity is used as a measure of coupling between the subset S_I and the rest of the system.

Since $\mathbf{Q}_I^*$ is the infinitesimal generator matrix of an ergodic CMTC, the solution of:

$$\boldsymbol{\pi}_I^* \mathbf{Q}_I^* = \mathbf{0}, \quad \boldsymbol{\pi}_I^* \mathbf{1} = 1 \tag{5.39}$$

yields the desired conditional steady-state probability vector $\boldsymbol{\pi}_I^*$ for the subset of fast states S_I. For each such subset S_I, a macro state I is defined. The set of aggregated states $\{I, 1 \leq I \leq F-1\}$, constructed in this way, together with the initially given slow states S_0 form the set of macro states $M = S_0 \cup \{I, 1 \leq I \leq F-1\}$ for which a transient analysis is performed in order to account for long-term effects among the set of all macro states.

To create the intermediate generator matrix $\tilde{\mathbf{\Sigma}}$ of size $(|M| + n_F)$ x $(|M| + n_F)$, unconditionings and aggregations of the transition rates have to be performed. The dimensions $[n_I \text{ x } n_J]$ of submatrices $\tilde{\mathbf{\Sigma}}_{IJ}$ are added to the corresponding equations in what follows. Let us define matrices of appropriate size:

$$\mathbf{E} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix}, \quad [n_0 \text{ x } n_I],$$

containing 1s only as entries. The following cases are distinguished:

- Transitions remain unchanged among slow states in S_0:

$$\tilde{\mathbf{\Sigma}}_{00} = \mathbf{A}_{00}, \quad [n_0 \text{ x } n_0]. \tag{5.40}$$

- Transitions involving fast transient states from S_F:
  - Transitions between slow states and fast transient states, and vice versa, remain unchanged:

$$\tilde{\mathbf{\Sigma}}_{F0} = \mathbf{A}_{F0}, \quad [n_F \text{ x } n_0], \tag{5.41}$$

$$\tilde{\mathbf{\Sigma}}_{0F} = \mathbf{A}_{0F}, \quad [n_0 \text{ x } n_F]. \tag{5.42}$$

- Transitions from aggregated macro states $I, 1 \leq I \leq F-1$, to fast transient states in S_F need to be weighted and added with the help of the conditional micro-state probability vector $\boldsymbol{\pi}_I^*$:

$$\tilde{\boldsymbol{\Sigma}}_{IF} = \boldsymbol{\pi}_I^* \mathbf{A}_{IF}, \quad [1 \text{ x } n_F]. \tag{5.43}$$

- Transition rates from fast transient states to aggregated macro states need to be aggregated into a single transition rate for all $I, 1 \leq I \leq F-1$:

$$\tilde{\boldsymbol{\Sigma}}_{FI} = \mathbf{A}_{FI}\mathbf{1}, \quad [n_F \text{ x } 1]. \tag{5.44}$$

- Transitions from aggregated macro states $I, 1 \leq I \leq F-1$, to slow states in S_0 need to be weighted and added with the help of the conditional micro state probability vector $\boldsymbol{\pi}_I^*$:

$$\tilde{\boldsymbol{\Sigma}}_{I0} = \boldsymbol{\pi}_I^* \mathbf{A}_{I0}, \quad [1 \text{ x } n_0]. \tag{5.45}$$

- Transitions from slow states in S_0 to aggregated macro states $I, 1 \leq I \leq F-1$ need to be aggregated into a single transition rate originating from each slow state in S_0:

$$\tilde{\boldsymbol{\Sigma}}_{0I} = \mathbf{A}_{0I}\mathbf{E}, \quad [n_0 \text{ x } 1]. \tag{5.46}$$

- Transition rates from aggregated macro state I to aggregated macro state J, $I \neq J$ and $1 \leq I, J \leq F-1$ have to be weighted and added with the help of the conditional micro-state probability vector $\boldsymbol{\pi}_I^*$ for each $j \in S_J$ separately. The resulting values are aggregated to yield the transition rates between macro states I and J:

$$\tilde{\Sigma}_{IJ} = \boldsymbol{\pi}_I^* \mathbf{A}_{IJ}\mathbf{1}, \quad [1 \text{ x } 1]. \tag{5.47}$$

- To complete the infinitesimal generator matrix $\tilde{\boldsymbol{\Sigma}}$, the diagonal elements need to be computed for all $I, 1 \leq I \leq F-1$, as usual:

$$\begin{aligned}
\tilde{\Sigma}_{II} &= -\sum_{J, I \neq J} \tilde{\Sigma}_{IJ}, \quad [1 \text{ x } 1] \\
&= -\left(\tilde{\boldsymbol{\Sigma}}_{I0}\mathbf{1} + \sum_{J, J \geq 1, J \neq I} \tilde{\Sigma}_{IJ} + \tilde{\boldsymbol{\Sigma}}_{IF}\mathbf{1} \right) \\
&= -\left(\boldsymbol{\pi}_I^* \mathbf{A}_{I0}\mathbf{1} + \sum_{J, J \geq 1, J \neq I} \boldsymbol{\pi}_I^* \mathbf{A}_{IJ}\mathbf{1} + \boldsymbol{\pi}_I^* \mathbf{A}_{IF}\mathbf{1} \right).
\end{aligned} \tag{5.48}$$

Collecting the results from Eq. (5.40) through Eq. (5.48) yields the intermediate infinitesimal generator matrix $\tilde{\boldsymbol{\Sigma}}$. The matrix $\tilde{\boldsymbol{\Sigma}}$ is structured in such

a way that it reflects the partitioning into macro states and fast transient states:

$$\tilde{\mathbf{\Sigma}} = \begin{pmatrix} \tilde{\mathbf{\Sigma}}_{MM} & \tilde{\mathbf{\Sigma}}_{MF} \\ \tilde{\mathbf{\Sigma}}_{FM} & \tilde{\mathbf{\Sigma}}_{FF} \end{pmatrix}. \tag{5.49}$$

If no fast transient states are included, then $\mathbf{\Sigma} = \tilde{\mathbf{\Sigma}}$ already constitutes the final generator matrix. In any case, a transient analysis can be carried out based on a model with reduced state space. Although the intermediate model may still be stiff because fast and slow states are still simultaneously included, the computational savings can be considerable due to smaller state space. But more aggregation is possible, as is highlighted in the following section. With $\tilde{\mathbf{\Sigma}}$ given, a differential equation can be derived for the computation of the transient state probability vector $\tilde{\boldsymbol{\sigma}}(t)$, conditioned on initial probability vector $\tilde{\boldsymbol{\sigma}}(0)$:

$$\frac{d}{dt}\tilde{\boldsymbol{\sigma}}(t) = \tilde{\boldsymbol{\sigma}}(t)\tilde{\mathbf{\Sigma}}, \tag{5.50}$$

$$\tilde{\boldsymbol{\sigma}}(0) = \left(\tilde{\sigma}_{0_0}(0), \ldots, \tilde{\sigma}_{0_{n_0-1}}(0), \tilde{\sigma}_1(0), \ldots, \tilde{\sigma}_{F-1}(0), \tilde{\sigma}_{F_0}(0), \ldots, \tilde{\sigma}_{F_{n_F-1}}(0)\right).$$

5.2.3 Aggregation of Fast Transient Subsets

Recall that a fast transient subset of states is connected to states in other subsets by at least one fast transition. Thus if such a set is not nearly completely decomposable from the rest of the system, it is tightly coupled to at least one other non-empty subset of states.

Now consider an intermediate state space consisting of a non-empty fast transient subset of states $S_F \subset S$, and the set of macro states $M = S_0 \cup \{I, 1 \leq I \leq F-1\}$ resulting from aggregation of fast recurrent subsets and collecting them together with the slow states. Furthermore, assume the intermediate transition rates among the states in $M \cup S_F$ to be already calculated as shown in the previous section.

From Eq. (5.50) and Eq. (5.49), the following two equations can be derived:

$$\frac{d}{dt}\tilde{\boldsymbol{\sigma}}_M(t) = \tilde{\boldsymbol{\sigma}}_M(t)\tilde{\mathbf{\Sigma}}_{MM} + \tilde{\boldsymbol{\sigma}}_F(t)\tilde{\mathbf{\Sigma}}_{FM}, \tag{5.51}$$

$$\frac{d}{dt}\tilde{\boldsymbol{\sigma}}_F(t) = \tilde{\boldsymbol{\sigma}}_M(t)\tilde{\mathbf{\Sigma}}_{MF} + \tilde{\boldsymbol{\sigma}}_F(t)\tilde{\mathbf{\Sigma}}_{FF}. \tag{5.52}$$

It is reasonable to assume that for the fast transient states, the derivative on the left-hand side of Eq. (5.52) approaches zero (i.e., $d/dt\tilde{\boldsymbol{\sigma}}_F(t) = 0$), with respect to the time scale of slow states. Hence from Eq. (5.52) we obtain an approximation of $\tilde{\sigma}_F$:

$$\tilde{\boldsymbol{\sigma}}_F^{\approx} = -\tilde{\boldsymbol{\sigma}}_M^{\approx}\tilde{\mathbf{\Sigma}}_{MF}\tilde{\mathbf{\Sigma}}_{FF}^{-1}. \tag{5.53}$$

Using the result from Eq. (5.53) in Eq. (5.51) provides us with the following approximation:

$$\frac{d}{dt}\tilde{\boldsymbol{\sigma}}_M^{\approx}(t) = \tilde{\boldsymbol{\sigma}}_M^{\approx}(t)\tilde{\boldsymbol{\Sigma}}_{MM} - \tilde{\boldsymbol{\sigma}}_M^{\approx}(t)\tilde{\boldsymbol{\Sigma}}_{MF}\tilde{\boldsymbol{\Sigma}}_{FF}^{-1}\tilde{\boldsymbol{\Sigma}}_{FM} \tag{5.54}$$

$$= \tilde{\boldsymbol{\sigma}}_M^{\approx}(t)\left(\tilde{\boldsymbol{\Sigma}}_{MM} - \tilde{\boldsymbol{\Sigma}}_{MF}\tilde{\boldsymbol{\Sigma}}_{FF}^{-1}\tilde{\boldsymbol{\Sigma}}_{FM}\right) \tag{5.55}$$

$$= \tilde{\boldsymbol{\sigma}}_M^{\approx}(t)\tilde{\boldsymbol{\Sigma}}_{MM}^{\approx}. \tag{5.56}$$

Hence, $\boldsymbol{\Sigma} \approx \tilde{\boldsymbol{\Sigma}}_{MM}^{\approx}$ is taken as the infinitesimal generator matrix of the final aggregated Markov chain. Bobbio and Trivedi [BoTr86] have shown that:

$$\mathbf{P}_{FM} = -\tilde{\boldsymbol{\Sigma}}_{FF}^{-1}\tilde{\boldsymbol{\Sigma}}_{FM} \tag{5.57}$$

is the *asymptotic exit probability* matrix from the subset of fast transient states S_F to macro states M. The matrix entries of $\mathbf{P}_{FM} = [p_{iI}(0,\infty)], i \in S_F, I \in M$ are the conditional transition probabilities that the stochastic process, once initiated in state i at time $t = 0$ will ultimately exit S_F and hit macro state $I \in M$ as time $t \to \infty$. These probabilities are used to adjust the transition rates among macro states during elimination of fast transient states. The interpretation is that the fast transient states form a probabilistic switch in the limit, and the time spent in these sets of states can be neglected in the long run. Furthermore, related to this assumption, the exit stationary probability will be reached in a period of time much smaller than the time scale characterizing the dynamics among the macro states. The error induced by this approximation is inversely proportional to the stiffness ratio.

For the sake of completeness, it is worth mentioning that the entries in matrix $\tilde{\boldsymbol{\Sigma}}_{MF}$ from Eq. (5.55) denote exit rates, leading from macro states in M via the probabilistic switch, represented by matrix $\mathbf{P}_{FM}$ Eq. (5.57), back to macro states. Hence, the rate matrix $\tilde{\boldsymbol{\Sigma}}_{MF}\mathbf{P}_{FM}$ is simply added to $\tilde{\boldsymbol{\Sigma}}_{MM}$ in Eq. (5.55) to obtain the rate adjustment.

5.2.4 Aggregation of Initial State Probabilities

Since the transient analysis is performed only on the set of macro states M, the initial probability vector $\boldsymbol{\pi}(0)$ must be adjusted accordingly. This adjustment is achieved in two steps. First, the probability mass $\boldsymbol{\pi}_I(0)$ assigned to micro states $i \in S_I$, which are condensed into a single macro state I, is accumulated for all $I, 1 \leq I \leq F-1$:

$$\tilde{\sigma}_I(0) = \boldsymbol{\pi}_I(0)\mathbf{1}. \tag{5.58}$$

Second, if $S_F \neq \emptyset$, then the probability mass $\boldsymbol{\pi}_F(0)$ assigned to the fast transient states S_F is spread probabilistically across the macro states M according to rule implied by the probabilistic switch $\mathbf{P}_{FM}$ given in Eq. (5.57):

$$\tilde{\boldsymbol{\sigma}}_M^{\approx}(0) = \tilde{\boldsymbol{\sigma}}(0) + \boldsymbol{\pi}_F(0)\mathbf{P}_{FM}. \tag{5.59}$$

Depending on S_F, the initial probability vector $\boldsymbol{\sigma}(0)$ is given by $\tilde{\boldsymbol{\sigma}}_M^{\approx}(0)$ according to Eq. (5.58) or by $\tilde{\boldsymbol{\sigma}}(0)$ according to Eq. (5.59).

The aggregation part has now completely been described. With this technique, the transient analysis of large stiff Markov chains can be reduced to the analysis of smaller non-stiff chains. Transient analysis is accomplished only for the macro states in the model, be it either the initially present slow states or the macro states resulting from an aggregation of fast recurrent subsets of states. The accuracy of the method is inversely proportional to the stiffness ratio of the initial model.

With the assumption that steady state is reached much quicker for the fast states than for the slow states, further approximations become possible. Applying certain disaggregation steps, approximations of the transient fast state probabilities can also be derived, thus resulting in a complete state probability vector.

5.2.5 Disaggregations

5.2.5.1 Fast Transient States With the approximate macro state probability vector $\tilde{\boldsymbol{\sigma}}_M^{\approx}(t)$, obtained as the solution of Eq. (5.56), the approximate fast transient state probability vector $\tilde{\boldsymbol{\sigma}}_F^{\approx}(t)$ can be derived easily with a transient interpretation of Eq. (5.53):

$$\tilde{\boldsymbol{\sigma}}_F^{\approx}(t) = -\tilde{\boldsymbol{\sigma}}_M^{\approx}(t)\tilde{\boldsymbol{\Sigma}}_{MF}\tilde{\boldsymbol{\Sigma}}_{FF}^{-1}. \tag{5.60}$$

With:

$$c = \tilde{\boldsymbol{\sigma}}_M^{\approx}(t)\mathbf{1} + \tilde{\boldsymbol{\sigma}}_F^{\approx}(t)\mathbf{1}, \tag{5.61}$$

normalization can be taken into account for computation of a probability vector so that the first disaggregation step is accomplished, leading to the intermediate transient state probability vector $\tilde{\boldsymbol{\sigma}}^{\approx(c)}(t)$ as an approximate solution of Eq. (5.50):

$$\tilde{\boldsymbol{\sigma}}(t) \approx \tilde{\boldsymbol{\sigma}}^{\approx(c)}(t) = \frac{1}{c}\left(\tilde{\boldsymbol{\sigma}}_M^{\approx}(t), \tilde{\boldsymbol{\sigma}}_F^{\approx}(t)\right). \tag{5.62}$$

5.2.5.2 Fast Recurrent States In a second disaggregation step, the transient micro-state probability vector $\boldsymbol{\pi}^{\approx}(t)$ is also calculated as an approximation of $\boldsymbol{\pi}(t)$, which is formally defined as the solution of Eq. (5.36), incorporating the original infinitesimal generator matrix $\mathbf{Q}$ from (5.35). For a computation of $\boldsymbol{\pi}^{\approx}(t) = (\boldsymbol{\pi}_I^{\approx}(t))$, we need to refer to the conditional micro-state probability vectors $\boldsymbol{\pi}_I^* = (\pi_{I_0}^*, \ldots, \pi_{I_{n_I-1}}^*), 1 \leq I \leq F-1$, from Eq. (5.39). The transient macro-state probabilities $\sigma_I(t), 1 \leq I \leq F-1$, are used to uncondition the conditional steady-state probability vectors $\boldsymbol{\pi}_I^*, 1 \leq I \leq F-1$:

$$\boldsymbol{\pi}_I(t) \approx \boldsymbol{\pi}_I^{\approx}(t) = \left(\sigma_I(t)\boldsymbol{\pi}_I^*\right). \tag{5.63}$$

If $S_F \neq \emptyset$ unconditioning has to be performed on the basis of $\tilde{\boldsymbol{\sigma}}(t) \approx \tilde{\boldsymbol{\sigma}}^{\approx(c)}(t)$ according to Eq. (5.62):

$$\boldsymbol{\pi}_I(t) \approx \boldsymbol{\pi}_I^{\approx}(t) = \left(\tilde{\sigma}_I^{\approx(c)}(t)\boldsymbol{\pi}_I^*\right). \tag{5.64}$$

Collecting all the intermediate results from Eq. (5.62) and Eq. (5.64) yields the final approximate transient probability vector in its most general form:

$$\begin{aligned}\boldsymbol{\pi}(t) \approx \boldsymbol{\pi}^{\approx}(t) = \Big(&\tilde{\sigma}_{0_0}^{\approx(c)}(t), \ldots, \tilde{\sigma}_{0_{n_0-1}}^{\approx(c)}(t),\\ &\tilde{\sigma}_1^{\approx(c)}(t)\pi_{1_0}^*, \ldots, \tilde{\sigma}_{F-1}^{\approx(c)}(t)\pi_{F-1_{n_{F-1}-1}}^*,\\ &\tilde{\sigma}_{F_0}^{\approx(c)}(t), \ldots, \tilde{\sigma}_{F_{n_F-1}}^{\approx(c)}(t)\Big)\\ = \Big(&\tilde{\boldsymbol{\sigma}}_0^{\approx(c)}(t), \boldsymbol{\pi}_1^{\approx}(t), \ldots, \boldsymbol{\pi}_{F-1}^{\approx}(t), \tilde{\boldsymbol{\sigma}}_F^{\approx(c)}(t)\Big).\end{aligned} \tag{5.65}$$

If $S_F = \emptyset$ then Eq. (5.65) simplifies to the following:

$$\begin{aligned}\boldsymbol{\pi}(t) \approx \boldsymbol{\pi}^{\approx}(t) = \Big(&\tilde{\sigma}_{0_0}(t), \ldots, \tilde{\sigma}_{0_{n_0-1}}(t),\\ &\tilde{\sigma}_1(t)\pi_{1_0}^*, \ldots, \tilde{\sigma}_{F-1}(t)\pi_{F-1_{n_{F-1}-1}}^*\Big)\\ = \big(&\tilde{\boldsymbol{\sigma}}_0(t), \boldsymbol{\pi}_1^{\approx}(t), \ldots, \boldsymbol{\pi}_{F-1}^{\approx}(t)\big).\end{aligned} \tag{5.66}$$

5.2.6 The Algorithm

STEP 1 Initialization:

- Specify $\tau >> 1/T$, the transition rate threshold, as a function of the time horizon T for which the transient analysis is to be carried out.
- Partition the state space S into slow states S_0 and fast states $S - S_0$ with respect to τ.
- Partition the fast states $S - S_0$ further into fast recurrent subsets $S_I, 1 \leq I \leq F-1$ and possibly one subset of fast transient states S_F.
- Arrange the infinitesimal generator matrix $\mathbf{Q}$ according to Eq. (5.35).

STEP 2 Steady-state analysis of fast recurrent subsets:

- From submatrices $\mathbf{Q}_I$ (Eq. (5.35)), obtain infinitesimal generator matrices $\mathbf{Q}_I^*$ according to Eq. (5.38).
- Compute the conditional micro state probability vectors $\boldsymbol{\pi}_I^*$ according to Eq. (5.39).

STEP 3 Construction of the intermediate generator matrix $\tilde{\boldsymbol{\Sigma}}$ as in Eq. (5.49):

- Use $\mathbf{Q}$ from Eq. (5.35) and $\boldsymbol{\pi}_I^*, 1 \leq I \leq F-1$ from Eq. (5.39) and apply operations starting from Eq. (5.40) through Eq. (5.48).

STEP 4 Aggregation of fast transient states:

- IF $S_F \neq \emptyset$
  - THEN perform aggregation of fast transient states and construct $\tilde{\mathbf{\Sigma}}^{\approx}_{MM}$:
    - Compute the probabilistic switch $\mathbf{P}_{FM}$ according to Eq. (5.57) and $\tilde{\mathbf{\Sigma}}^{\approx}_{MM}$ according to Eq. (5.55) and Eq. (5.56) from $\tilde{\mathbf{\Sigma}}$.
    - $\mathbf{\Sigma} = \tilde{\mathbf{\Sigma}}^{\approx}_{MM}$.
  - ELSE $\mathbf{\Sigma} = \tilde{\mathbf{\Sigma}}$.

STEP 5 Aggregation of the initial state probability vector $\boldsymbol{\sigma}(0)$:

- Accumulate initial probabilities $\pi_{I_i}(0)$ of micro states $i \in S_I$ into $\tilde{\sigma}_I(0), 1 \leq I \leq F-1$ according to Eq. (5.58).
- IF $S_F \neq \emptyset$
  - THEN calculate approximate initial macro-state probability vector $\boldsymbol{\sigma}(0) = \tilde{\boldsymbol{\sigma}}^{\approx}_M(0)$ according to Eq. (5.59).
  - ELSE $\boldsymbol{\sigma}(0) = \tilde{\boldsymbol{\sigma}}(0)$ according to Eq. (5.58).

STEP 6 Computation of the transient macro-state probability vector $\boldsymbol{\sigma}(t)$:

- Solve Eq. (5.37).

STEP 7 Disaggregations:

- IF $S_F \neq \emptyset$
  - THEN
    - Compute intermediate transient state probability vector $\tilde{\boldsymbol{\sigma}}(t)$ according to Eq. (5.62).
    - Compute the approximate micro-state probability subvectors $\boldsymbol{\pi}^{\approx}_I(t)$ by unconditioning of $\boldsymbol{\pi}^*_I$ according to Eq. (5.64) for all $1 \leq I \leq F-1$.
  - ELSE compute $\boldsymbol{\pi}^{\approx}_I(t)$ by unconditioning of $\boldsymbol{\pi}^*_I$ according to Eq. (5.63) for all $1 \leq I \leq F-1$.

STEP 8 Final result:

- Compose the approximate transient probability vector $\boldsymbol{\pi}(t) \approx \boldsymbol{\pi}^{\approx}(t)$ according to Eq. (5.65) if $S_F \neq \emptyset$ or Eq. (5.66) if $S_F = \emptyset$.

5.2.7 An Example: Server Breakdown and Repair

Assume a queueing system with a single server station, where a maximum of m customers can wait for service. Customers arrive at the station with arrival rate λ. Service time is exponentially distributed with mean $1/\mu$. Furthermore, the single server is subject to failure and repair, both times to failure and repair times being exponentially distributed with parameters γ and δ, respectively. It is reasonable to assume that λ and μ differ by orders of magnitude relative to γ and δ. Usually, repair of a failed unit takes much longer than traffic-related events in computer systems. This condition is even more relevant for failure events that are relatively infrequent. While traffic-related events typically take place in the order of micro seconds, repair durations are in the order of minutes, hours, or days, and failure events in the order of months, years, or multiple thereof. Thus, the transition rates λ and μ can be classified as being fast, and γ and δ as slow. Note that we are interested in computing performance measures related to traffic events so that for the rate threshold, the relation $\tau < \delta < \gamma$ generally holds and the application of the aggregation/disaggregation approach is admissible.

The described scenario is captured by a CTMC with a state space $S = \{(l,k), 0 \leq l \leq m, k \in \{0,1\}\}$, where l denotes the number of customers in the queue and k the number of non-failed servers, as depicted in Fig. 5.4*a*. The state space S is partitioned according to the classification scheme applied to the rates, into a set of slow states S_0, a set of fast recurrent states S_1, and a set of fast transient states $S_F = S_2$:

$$
\begin{aligned}
S_0 &= \{(m,0)\},\\
S_1 &= \{(0,1),(1,1),\cdots,(m,1)\},\\
S_2 &= \{(0,0),(1,0),\cdots,(m-1,0)\}.
\end{aligned}
$$

For the case of $m = 2$, the following state ordering is used in compliance with the partitioning:

$$
\begin{array}{cccccc}
0 & 1_0 & 1_1 & 1_2 & 2_0 & 2_1 \\
(2,0) & (0,1) & (1,1) & (2,1) & (0,0) & (1,0)
\end{array}.
$$

To complete initialization, the infinitesimal generator matrix $\mathbf{Q}$, $m = 2$, is restructured accordingly to reflect the partitioning:

$$
\mathbf{Q} = \begin{pmatrix}
-\delta & 0 & 0 & \delta & 0 & 0 \\
0 & -(\lambda+\gamma) & \lambda & 0 & \gamma & 0 \\
0 & \mu & -(\lambda+\gamma+\mu) & \lambda & 0 & \gamma \\
\gamma & 0 & \mu & -(\mu+\gamma) & 0 & 0 \\
0 & \delta & 0 & 0 & -(\lambda+\delta) & \lambda \\
\lambda & 0 & \delta & 0 & 0 & -(\lambda+\delta)
\end{pmatrix}.
$$

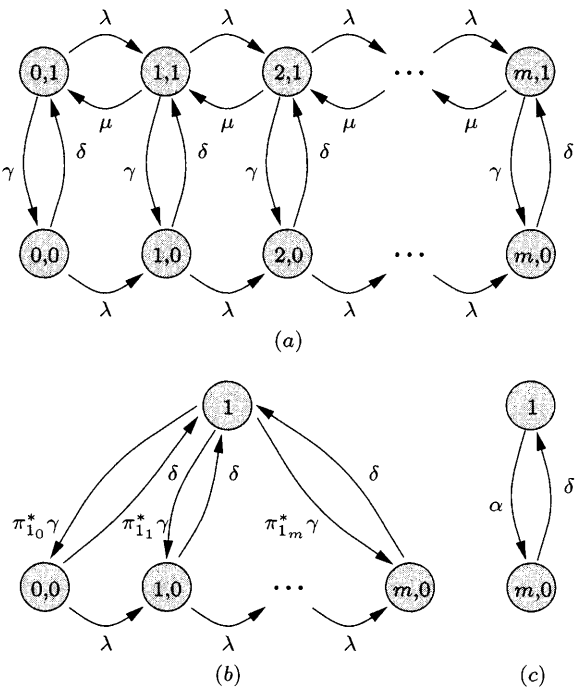


Fig. 5.4 (*a*) CTMC model of a finite capacity single server queueing system subject to server breakdown and repair. (*b*) The model of Fig. 5.4*a* after the aggregation of the fast recurrent subset into macro state 1. (*c*) Final aggregated macro-state chain for the model of Fig. 5.4*a* after elimination of the fast transient subset.

Next, the steady-state analysis of the fast recurrent subset S_1 has to be performed, and the infinitesimal generator matrix $\mathbf{Q}_1^*$ must be created:

$$\mathbf{Q}_1^* = \mathbf{Q}_1 + \mathbf{D}_1$$

$$= \begin{pmatrix} -(\lambda+\gamma) & \lambda & 0 \\ \mu & -(\lambda+\gamma+\mu) & \lambda \\ 0 & \mu & -(\mu+\gamma) \end{pmatrix} + \begin{pmatrix} \gamma & 0 & 0 \\ 0 & \gamma & 0 \\ 0 & 0 & \gamma \end{pmatrix}$$

$$= \begin{pmatrix} -\lambda & \lambda & 0 \\ \mu & -(\lambda+\mu) & \lambda \\ 0 & \mu & -\mu \end{pmatrix}.$$

Closed-form solutions of the steady-state probabilities are well known from literature (see, e.g., Gross and Harris [GrHa85]) for the case of the finite

birth-death process as described by $\mathbf{Q}_1^*$:

$$\pi_{1_i}^* = \left(\frac{\lambda}{\mu}\right)^i \frac{1-\frac{\lambda}{\mu}}{1-\left(\frac{\lambda}{\mu}\right)^3}, \quad i = 0, 1, \ldots, 2.$$

To aggregate the set of fast recurrent states S_1 into macro state 1, $\boldsymbol{\pi}_1^*$ is used to derive the intermediate infinitesimal generator matrix $\tilde{\boldsymbol{\Sigma}}$ for the reduced state space $\{(2,0), 1, (0,0), (1,0)\}$. By applying Eq. (5.40) through Eq. (5.48):

$$\tilde{\boldsymbol{\Sigma}} = \begin{pmatrix} -\delta & \delta & 0 & 0 \\ \pi_{1_2}^*\gamma & -\gamma & \pi_{1_0}^*\gamma & \pi_{1_1}^*\gamma \\ 0 & \delta & -(\lambda+\delta) & \lambda \\ \lambda & \delta & 0 & -(\lambda+\delta) \end{pmatrix}.$$

Matrix $\tilde{\boldsymbol{\Sigma}}$ represents the CTMC as depicted in Fig. 5.4*b* for the case of $m = 2$. For an aggregation of the fast transient states, first $\tilde{\boldsymbol{\Sigma}}_{FF}^{-1}$ and the probabilistic switch $\mathbf{P}_{FM}$ are calculated from $\tilde{\boldsymbol{\Sigma}}$ according to Eq. (5.57):

$$\tilde{\boldsymbol{\Sigma}}_{FF}^{-1} = \begin{pmatrix} -\frac{1}{\lambda+\delta} & -\frac{\lambda}{(\lambda+\delta)^2} \\ 0 & -\frac{1}{\lambda+\delta} \end{pmatrix}, \quad \mathbf{P}_{FM} = \begin{pmatrix} \frac{\lambda\delta}{(\lambda+\delta)^2} & \frac{\delta(\delta+2\lambda)}{(\lambda+\delta)^2} \\ \frac{\lambda}{\lambda+\delta} & \frac{\delta}{\lambda+\delta} \end{pmatrix}.$$

With $\tilde{\boldsymbol{\Sigma}}$ and $\mathbf{P}_{FM}$ given, $\boldsymbol{\Sigma} = \tilde{\boldsymbol{\Sigma}}_{MM}^{\approx}$ can be easily derived using some algebraic manipulation according to Eq. (5.55) and Eq. (5.56):

$$\boldsymbol{\Sigma} = \begin{pmatrix} -\delta & \delta \\ \gamma \sum_{i=0}^{2} \pi_{1_i}^* \left(\frac{\lambda}{\lambda+\delta}\right)^{2-i} & -\gamma \sum_{i=0}^{2} \pi_{1_i}^* \left(\frac{\lambda}{\lambda+\delta}\right)^{2-i} \end{pmatrix},$$

where $\lambda/(\lambda+\delta)$ denotes the probability that a new customer arrives while repair is being performed, and $\gamma\left(\lambda/(\lambda+\delta)\right)^{2-i}$ is the probability that a failure occurs when there are i customers present and that $m-i$ customers arrive during repair.

An initial probability vector $\boldsymbol{\pi}(0) = (0, 1, 0, \ldots, 0)$ is assumed, i.e., $\pi_{1_0} = 1$. This condition implies $\boldsymbol{\sigma}(0) = (0, 1)$ in Eq. (5.58) and Eq. (5.59). From Eqs. (5.16) and (5.17) we know symbolic expressions for the transient state probabilities of a two-state CTMC, given an initial pmf $\boldsymbol{\pi}(0)$. We can therefore apply this result by substituting the rates from $\boldsymbol{\Sigma}$ into these expressions. By letting:

$$\alpha = \gamma \sum_{i=0}^{2} \pi_{1_i}^* \left(\frac{\lambda}{\lambda+\delta}\right)^{2-i},$$

we get the following macro-state probabilities:

$$\sigma_0(t) = 1 - \sigma_1(t),$$
$$\sigma_1(t) = \frac{\delta}{\delta+\alpha} + \frac{\alpha}{\delta+\alpha}\left(e^{-(\delta+\alpha)t}\right).$$

Finally, disaggregations must be performed. Given $\tilde{\boldsymbol{\sigma}}\tilde{\approx}_M(t) = \boldsymbol{\sigma}(t)$, Eq. (5.60) needs to be evaluated for a computation of the fast transient probability vector:

$$\begin{aligned}\tilde{\boldsymbol{\sigma}}\tilde{\approx}_F(t) &= -\tilde{\boldsymbol{\sigma}}\tilde{\approx}_M(t)\tilde{\boldsymbol{\Sigma}}_{MF}\tilde{\boldsymbol{\Sigma}}_{FF}^{-1} \\ &= -\left(1-\sigma_1(t), \frac{\delta}{\delta+\alpha} + \frac{\alpha\cdot e^{-(\delta+\alpha)t}}{\delta+\alpha}\right)\cdot\begin{pmatrix} 0 & 0 \\ -\frac{\pi^*_{1_0}\gamma}{\lambda+\delta} & -\gamma\left(\frac{\pi^*_{1_0}\lambda}{(\lambda+\delta)^2} + \frac{\pi^*_{1_1}}{\lambda+\delta}\right)\end{pmatrix} \\ &= \left(\sigma_1(t)\frac{\pi^*_{1_0}\gamma}{\lambda+\delta}, \sigma_1(t)\frac{\gamma}{\lambda+\delta}\left(\frac{\pi^*_{1_0}\lambda}{\lambda+\delta} + \pi^*_{1_1}\right)\right).\end{aligned}$$

Next, normalization is accomplished with help of c according to Eq. (5.61):

$$c = 1 + \sigma_1(t)\frac{\gamma}{\lambda+\delta}\left(\pi^*_{1_0}\left(1+\frac{\lambda}{\lambda+\delta}\right) + \pi^*_{1_1}\right).$$

The intermediate transient state probability vector $\tilde{\boldsymbol{\sigma}}(t)$ from Eq. (5.62) follows immediately:

$$\tilde{\boldsymbol{\sigma}}(t) = \frac{1}{c}\left(1-\sigma_1(t), \sigma_1(t), \sigma_1(t)\frac{\pi^*_{1_0}\gamma}{\lambda+\delta}, \sigma_1(t)\frac{\gamma}{\lambda+\delta}\left(\frac{\pi^*_{1_0}\lambda}{\lambda+\delta} + \pi^*_{1_1}\right)\right).$$

The final step consists of unconditioning the micro-state probability vector $\boldsymbol{\pi}^*_1$:

$$\boldsymbol{\pi}\tilde{\approx}_1(t) = \frac{\sigma_1(t)}{c}\left(\pi^*_{1_0}, \pi^*_{1_1}, \pi^*_{1_2}\right).$$

Collecting all the results provides an approximation $\boldsymbol{\pi}^{\approx}(t)$ to the transient probability vector, which is represented as the transpose:

$$\boldsymbol{\pi}(t) \approx \boldsymbol{\pi}^{\approx}(t) = \begin{pmatrix} \frac{1}{c}(1-\sigma_1(t)) \\ \frac{\sigma_1(t)}{c}\pi^*_{1_0} \\ \frac{\sigma_1(t)}{c}\pi^*_{1_1} \\ \frac{\sigma_1(t)}{c}\pi^*_{1_2} \\ \frac{1}{c}\sigma_1(t)\frac{\pi^*_{1_0}\gamma}{\lambda+\delta} \\ \frac{1}{c}\sigma_1(t)\frac{\gamma}{\lambda+\delta}\left(\frac{\pi^*_{1_0}\lambda}{\lambda+\delta} + \pi^*_{1_1}\right) \end{pmatrix}^T.$$

From the transient state probabilities, all the performance measures of interest can be calculated.

Table 5.1 Parameter set for Table 5.2.

δ	γ	λ	μ
0.01	0.0001	0.5	1

The exact transient state probability vector $\boldsymbol{\pi}(t)$ is compared to the approximate state probability vector $\boldsymbol{\pi}^{\approx}(t)$ at time instants $t = 10, 100$ and 1000 in Table 5.2, with the parameter set from Table 5.1. (Note that the initial probability vectors are chosen as has been specified earlier.) The results of the approximate method are generally very close to the exact values for most states. Usually, the accuracy depends on the length of the investigated time horizon for a transient analysis. Short-term results are particularly dependent on the initial probability vector so that some more significant differences can arise as in the case of state $(2,0)$ in our example, where the absolute state probabilities are rather small, and hence the percentage difference is high. Furthermore, whether the error is positive or negative may change over time.

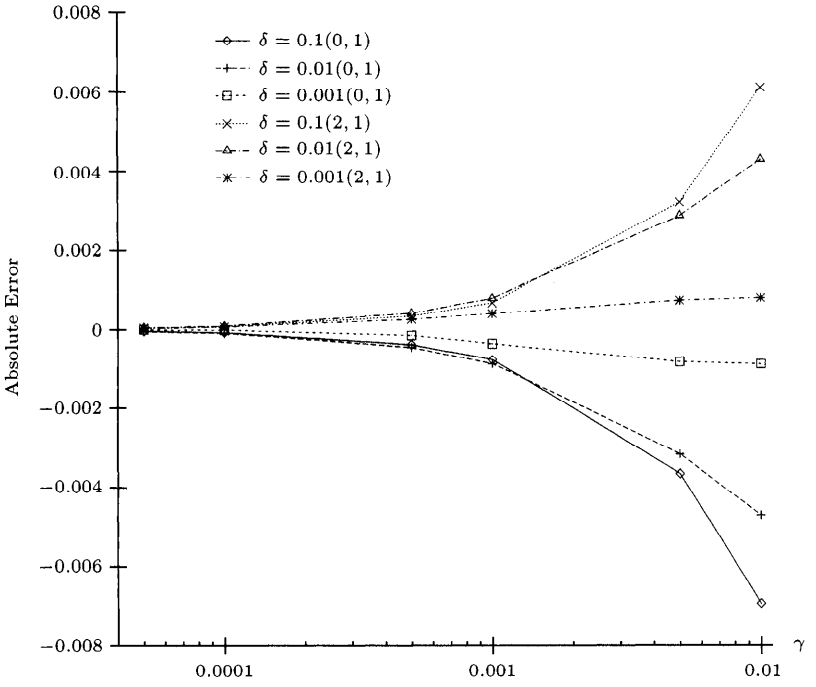


Fig. 5.5 Errors of the approximate method as a function γ.

Table 5.2 State probabilities as a function of time: exact vs. approximate

		Exact $\boldsymbol{\pi}(t)$			
No.	State	$t = 0$	$t = 10$	$t = 100$	$t = 1000$
0	(2,0)	0	0.00067567	0.00601869	0.00962544
1_0	(0,1)	1	0.57099292	0.56777268	0.56567586
1_1	(1,1)	0	0.28540995	0.28392083	0.28289339
1_2	(2,1)	0	0.14264597	0.14201127	0.14152880
2_0	(0,0)	0	0.00011195	0.00011134	0.00011092
2_1	(1,0)	0	0.00016354	0.00016484	0.00016421

		Approx $\boldsymbol{\pi}^{\approx}(t)$			
No.	State	$t = 0$	$t = 10$	$t = 100$	$t = 1000$
0	(2,0)	0	0.00092448	0.00611894	0.00962543
1_0	(0,1)	0.571429	0.57074200	0.56777400	0.56577100
1_1	(1,1)	0.285714	0.28537100	0.28388700	0.28288600
1_2	(2,1)	0.142857	0.14268500	0.14194400	0.14144300
2_0	(0,0)	0	0.00011191	0.00011133	0.00011094
2_1	(1,0)	0	0.00016567	0.00016481	0.00016423

		Absolute error			Error %		
No.	State	$t = 10$	$t = 100$	$t = 1000$	$t = 10$	$t = 100$	$t = 1000$
0	(2,0)	-2.488e-04	-1.003e-04	1e-08	-36.8242	-1.66564	0.00010
1_0	(0,1)	2.509e-04	-1.32e-06	-9.514e-05	0.04394	-0.00023	-0.01682
1_1	(1,1)	3.895e-05	3.383e-05	7.39e-06	0.01365	0.01192	0.00261
1_2	(2,1)	-3.903e-05	6.727e-05	8.58e-05	-0.02736	0.04737	0.06062
2_0	(0,0)	4e-08	1e-08	-2e-08	0.03573	0.00898	-0.01803
2_1	(1,0)	-2.13e-06	3e-08	-2e-08	-1.30243	0.01820	-0.01218

The impact of different degrees of stiffness on the accuracy of the results are indicated in Table 5.3 and in Fig. 5.5. In particular, the empirical results depicted in Fig. 5.5 support the assumption of the approximation being better for stiffer models. Almost no error is observed if $\gamma \leq 10^{-4}$, while the error substantially increases for the less stiff models as γ gets larger. The results depend on the three different parameter sets given in Table 5.4. A closer look at Fig. 5.5 reveals a non-monotonic ordering of the degree of accuracy. Let $\gamma = 0.01$, then the worst result is obtained for the largest δ, that is, $\delta = 0.1$ in state $(2, 1)$, as anticipated. But this behavior does not occur for $\gamma < 0.001$, where the approximate probability is closer to the exact value for $\delta = 0.1$ than for $\delta = 0.01$. Similar patterns can be observed in other cases, as well.

Problem 5.6 Verify the construction of the intermediate infinitesimal generator matrix $\tilde{\boldsymbol{\Sigma}}$ in the example study by applying Eq. (5.40) through Eq. (5.48) for each submatrix of $\mathbf{Q}$ in the indicated way.

Table 5.3 State probabilities as a function of stiffness: exact vs. approximate

		Exact $\pi(1000)$		
No.	State	(1)	(2)	(3)
0	(2,0)	0.06038036	0.08838416	0.07090560
1_0	(0,1)	0.53671266	0.51861447	0.51254489
1_1	(1,1)	0.26839084	0.25981567	0.26054365
1_2	(2,1)	0.13424710	0.13066102	0.13600236
2_0	(0,0)	0.00010714	0.00101689	0.00854241
2_1	(1,0)	0.00016050	0.00150639	0.01146107

		Approx $\pi^{\approx}(1000)$		
No.	State	(1)	(2)	(3)
0	(2,0)	0.06046620	0.08838110	0.07070710
1_0	(0,1)	0.53672400	0.51948100	0.51948100
1_1	(1,1)	0.26836200	0.25974100	0.25974000
1_2	(2,1)	0.13418100	0.12987000	0.12987000
2_0	(0,0)	0.00010713	0.00101859	0.00865801
2_1	(1,0)	0.00016048	0.00150791	0.01154400

		Absolute Error			Error %		
No.	State	(1)	(2)	(3)	(1)	(2)	(3)
0	(2,0)	-8.584e-05	3.06e-06	1.985e-04	-0.14217	0.00346	0.27995
1_0	(0,1)	-1.134e-05	-8.665e-04	-6.936e-03	-0.00211	-0.16709	-1.35327
1_1	(1,1)	2.884e-05	7.467e-05	8.037e-04	0.01075	0.02874	0.30845
1_2	(2,1)	6.61e-05	7.91e-04	6.132e-03	0.04924	0.60540	4.50901
2_0	(0,0)	1e-08	-1.7e-06	-1.156e-04	0.00933	-0.16718	-1.35325
2_1	(1,0)	2e-08	-1.52e-06	-8.293e-05	0.01246	-0.10090	-0.72358

Problem 5.7 Verify the correctness of the probabilistic switch $\mathbf{P}_{FM}$ in the example study by relating it to Eq. (5.57).

Problem 5.8 Modify the example in Fig. 5.4*a* in such a way that no repair is possible, that is, let the repair rate be $\delta = 0$. Let $X^{(s)}(t)$ denote the system's service rate at time t, and let $Y^{(s)}(t) = \int_0^\tau X^{(s)}(\tau)d\tau$ be the number of customers serviced in $[0, t)$. Derive all the intermediate probabilities, the approximate transient state probability vector $\boldsymbol{\pi}^{\approx}(t)$, the expected instantaneous reward rate $E[X^{(s)}(t)]$, the cumulative expected reward $E[Y^{(s)}(t)]$ in finite time, and the limit $E[Y^{(s)}(\infty)]$ for the modified model. Perform the analyses of the resulting model with the same parameters that have been used in the example presented this section.

Problem 5.9 Consider the steady-state case as $t \to \infty$ for the example in Fig. 5.4*a*. Derive the approximate steady-state probability vector $\boldsymbol{\pi}^{\approx}$ and calculate the expected number of customers in the system in steady state. Then, calculate the probability that the incoming customers find the system

Table 5.4 Parameter set for Table 5.3

Experiment	δ	γ	λ	μ
(1)	0.001	0.0001		
(2)	0.01	0.001	0.5	1
(3)	0.1	0.01		

full, in which case they must be rejected. Use this probability to calculate the *effective* arrival rate λ_e, defined as λ minus the rejected portion of arrivals. Use Little's law to calculate the average response time.

Problem 5.10 Apply the aggregation technique for the transient analysis of stiff Markov chains of this section to the example in Section 4.1 and derive the approximate transient probability vector. Assume $\boldsymbol{\pi}(0) = \left(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right)$, evaluate the approximate transient state probability vector at time instances $t \in \{1, 2, 3, 5, 10\}$, and compare the results to those obtained by applying Courtois's method and to the exact results.

6

Single Station Queueing Systems

A single station queueing system, as shown in Fig. 6.1, consists of a queueing buffer of finite or infinite size and one or more identical servers. Such an elementary queueing system is also referred to as a service station or, simply, as a node. A server can only serve one customer at a time and hence, it is

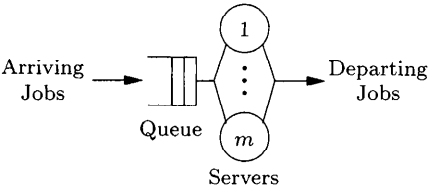


Fig. 6.1 Service station with m servers (a multiple server station).

either in a "busy" or an "idle" state. If all servers are busy upon the arrival of a customer, the newly arriving customer is buffered, assuming that buffer space is available, and waits for its turn. When the customer currently in service departs, one of the waiting customers is selected for service according to a *queueing (or scheduling) discipline.* An elementary queueing system is further described by an arrival process, which can be characterized by its sequence of interarrival time random variables $\{A_1, A_2, \cdots\}$. It is common to assume that the sequence of interarrival times is independent and identically distributed, leading to an arrival process that is known as a renewal process [Triv82]. Recent work on correlated interarrival times can be found in [Luca91] and [Luca93]. The distribution function of interarrival times can be continuous or discrete. We deal only with the former case in this book.

For information related to discrete interarrival time distributions, the reader may consult recent work on discrete queueing networks [Dadu96].

The average interarrival time is denoted by $E[A] = \overline{T}_A$ and its reciprocal by the average arrival rate λ:

$$\lambda = \frac{1}{\overline{T}_A}. \tag{6.1}$$

The most common interarrival time distribution is the exponential, in which case the arrival process is Poisson. The sequence $\{B_1, B_2, \cdots\}$ of service times of successive jobs also needs to be specified. We assume that this sequence is also a set of independent random variables with a common distribution function. The mean service time $E[B]$ is denoted by $\overline{T}_B$ and its reciprocal by the service rate μ:

$$\mu = \frac{1}{\overline{T}_B}. \tag{6.2}$$

6.1 KENDALL'S NOTATION

The following notation, known as Kendall's notation, is widely used to describe elementary queueing systems:

A/B/m - queueing discipline,

where A indicates the distribution of the interarrival times, B denotes the distribution of the service times, and m is the number of servers ($m \geq 1$). The following symbols are normally used for A and B:

M	Exponential distribution (memoryless property)
E_k	Erlang distribution with k phases
H_k	Hyperexponential distribution with k phases
C_k	Cox distribution with k phases
D	Deterministic distribution, i.e., the interarrival time or service time is constant
G	General distribution
GI	General distribution with independent interarrival times

Due to proliferation of high-speed networks, there is considerable interest in traffic arrival processes where successive arrivals are correlated. Such non-GI arrival processes include Markov modulated Poisson process (MMPP) [FiMe93] or batch Markovian arrival process (BMAP) [Luca91]. Queueing systems with such complex arrival processes have also been analyzed [Luca93]. Except for an example of the use of MMPP, we do not consider such queueing systems in this book.

The queueing discipline or service strategy determines which job is selected from the queue for processing when a server becomes available. Some commonly used queueing disciplines are:

FCFS (First-Come-First-Served): If no queueing discipline is given in the Kendall notation, then the default is assumed to be the FCFS discipline. The jobs are served in the order of their arrival.

LCFS (Last-Come-First-Served): The job that arrived last is served next.

SIRO (Service-In-Random-Order): The job to be served next is selected at random.

RR (Round Robin): If the servicing of a job is not completed at the end of a time slice of specified length, the job is preempted and returns to the queue, which is served according to FCFS. This action is repeated until the job service is completed.

PS (Processor Sharing): This strategy corresponds to round robin with infinitesimally small time slices. It is as if all jobs are served simultaneously and the service time is increased correspondingly.

IS (Infinite Server): There is an ample number of servers so that no queue ever forms.

Static Priorities: The selection depends on priorities that are permanently assigned to the job. Within a class of jobs with the same priority, FCFS is used to select the next job to be processed.

Dynamic Priorities: The selection depends on dynamic priorities that alter with the passing of time.

Preemption: If priority or LCFS discipline is used, then the job currently being processed is interrupted and preempted if there is a job in the queue with a higher priority.

As an example of Kendall's notation, the expression

M/G/1 - LCFS preemptive resume (PR)

describes an elementary queueing system with exponentially distributed interarrival times, arbitrarily distributed service times, and a single server. The queueing discipline is LCFS where a newly arriving job interrupts the job currently being processed and replaces it in the server. The servicing of the job that was interrupted is resumed only after all jobs that arrived after it have completed service.

Kendall's notation can be extended in various ways. An additional parameter is often introduced to represent the number of places in the queue (if the queue is finite) and we get the extended notation:

A/B/m/K - queueing discipline,

where K is the capacity of the station (queue + server). This means that if the number of jobs at server and queue is K, a newly arriving job is lost.

6.2 PERFORMANCE MEASURES

The different types of queueing systems are analyzed mathematically to determine performance measures from the description of the system. Because a queueing model represents a dynamic system, the values of the performance measures vary with time. Normally, however, we are content with the results in the steady-state. The system is said to be in steady state when all transient behavior has ended, the system has settled down, and the values of the performance measures are independent of time. The system is then said to be in statistical equilibrium, i.e., the rate at which jobs enter the system is equal to the rate at which jobs leave the system. Such a system is also called a *stable system.* Transient solutions of simple queueing systems are available in closed-form, but for more general cases, we need to resort to Markov chain techniques as described in Chapter 5. Recall that the generation and the solution of large Markov chains can be automated via stochastic reward nets [MuTr92].

The most important performance measures are:

Probability of the Number of Jobs in the System π_k: It is often possible to describe the behavior of a queueing system by means of the probability vector of the number of jobs in the system π_k. The mean values of most of the other interesting performance measures can be deduced from π_k:

$$\pi_k = P[\text{there are } k \text{ jobs in the system}].$$

Utilization ρ: If the queueing system consists of a single server, then the utilization ρ is the fraction of the time in which the server is busy, i.e., occupied. In case there is no limit on the number of jobs in the single server queue, the server utilization is given by:

$$\rho = \frac{\text{mean service time}}{\text{mean interarrival time}} = \frac{\text{arrival rate}}{\text{service rate}} = \frac{\lambda}{\mu}. \tag{6.3}$$

The utilization of a service station with multiple servers is the mean fraction of active servers. Since $m\mu$ is the overall service rate:

$$\rho = \frac{\lambda}{m\mu}, \tag{6.4}$$

and ρ can be used to formulate the condition for stationary behavior mentioned previously. The condition for stability is:

$$\rho < 1, \tag{6.5}$$

i.e., on average the number of jobs that arrive in a unit of time must be less than the number of jobs that can be processed. All the results given in Chapters 6-10 apply only to stable systems.

Throughput λ: The throughput of an elementary queueing system is defined as the mean number of jobs whose processing is completed in a single unit of time, i.e., the departure rate. Since the departure rate is equal to the arrival rate λ for a queueing system in statistical equilibrium, the throughput is given by:

$$\lambda = m \cdot \rho \cdot \mu \tag{6.6}$$

in accordance with Eq. (6.4). We note that in the case of finite buffer queueing system, throughput can be different from the external arrival rate.

Response Time T: The response time, also known as the sojourn time, is the total time that a job spends in the queueing system.

Waiting Time W: The waiting time is the time that a job spends in a queue waiting to be serviced. Therefore we have:

$$\text{Response time} = \text{waiting time} + \text{service time}.$$

Since W and T are usually random numbers, their mean is calculated. Then:

$$\overline{T} = \overline{W} + \frac{1}{\mu}. \tag{6.7}$$

The distribution functions of the waiting time, $F_W(x)$, and the response time, $F_T(x)$, are also sometimes required.

Queue Length Q: The queue length, Q, is the number of jobs in the queue.

Number of Jobs in the System K: The number of jobs in the queueing system is represented by K. Then:

$$\overline{K} = \sum_{k=1}^{\infty} k \cdot \pi_k. \tag{6.8}$$

The mean number of jobs in the queueing system $\overline{K}$ and the mean queue length $\overline{Q}$ can be calculated using one of the most important theorems of queueing theory, *Little's theorem*:

$$\overline{K} = \lambda \overline{T}, \tag{6.9}$$

$$\text{and} \quad \overline{Q} = \lambda \overline{W}. \tag{6.10}$$

Little's theorem is valid for all queueing disciplines and arbitrary GI/G/m systems. The proof is given in [Litt61].

6.3 THE M/M/1 SYSTEM

Recall that in this case, the arrival process is Poisson, the service times are exponentially distributed, and there is a single server. The system can be modeled as a birth-death process with birth rate (arrival rate) λ and a constant death rate (service rate) μ. We assume that $\lambda < \mu$ so the underlying CTMC is ergodic and hence the queueing system is stable. Then using Eq. (3.12), we obtain the steady-state probability of the system being empty:

$$\pi_0 = \frac{1}{1 + \sum_{k=1}^{\infty} \prod_{i=0}^{k-1} \frac{\lambda}{\mu}} = \frac{1}{1 + \sum_{k=1}^{\infty} \left(\frac{\lambda}{\mu}\right)^k},$$

which can be simplified to:

$$\pi_0 = \frac{1}{1 + \frac{\lambda/\mu}{1-\lambda/\mu}} = 1 - \frac{\lambda}{\mu}.$$

From Eq. (3.11) we get for the steady-state probability that there are k jobs in the system:

$$\pi_k = \pi_0 \left(\frac{\lambda}{\mu}\right)^k, \quad k \geq 0.$$

$$\pi_k = \left(1 - \frac{\lambda}{\mu}\right) \cdot \left(\frac{\lambda}{\mu}\right)^k.$$

or with the utilization $\rho = \lambda/\mu$:

$$\pi_0 = 1 - \rho \tag{6.11}$$

and:

$$\pi_k = (1 - \rho)\rho^k, \tag{6.12}$$

the probability mass function (pmf) of the modified geometric random variable. In Fig. 6.2, we plot this pmf for $\rho = 1/2$. The mean number of jobs is obtained using Eqs. (6.12) and (6.8):

$$\overline{K} = \frac{\rho}{1 - \rho}. \tag{6.13}$$

In Fig. 6.3, the mean number of jobs is plotted as a function of the utilization ρ. This is the typical behavior of all queueing systems.

From Eq. (1.10) we obtain the variance of the number of jobs in the system:

$$\sigma_K^2 = \frac{\rho}{(1 - \rho)^2} \tag{6.14}$$

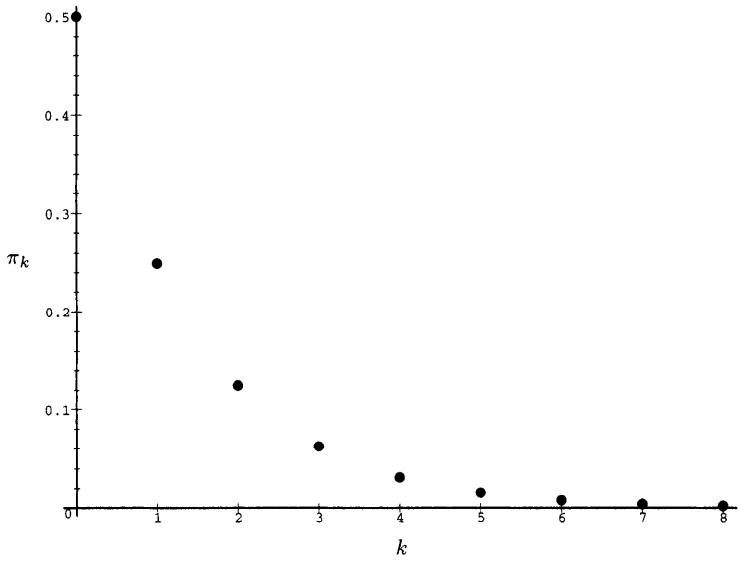


Fig. 6.2 The solution for π_k in a M/M/1 system.

and the coefficient of variation:

$$c_K = \frac{\sigma_K}{\overline{K}} = \frac{1}{\sqrt{\rho}}.$$

With Little's theorem (Eqs. (6.9) and (6.10)) we get for the mean response time:

$$\overline{T} = \frac{1/\mu}{1-\rho}, \tag{6.15}$$

with Eq. (6.7) for the mean waiting time:

$$\overline{W} = \frac{\rho/\mu}{1-\rho}, \tag{6.16}$$

and with Little's theorem again for the mean queue length:

$$\overline{Q} = \frac{\rho^2}{1-\rho}. \tag{6.17}$$

The same formulae are valid for M/G/1-PS and M/G/1-LCFS preemptive resume (see [Lave83]). For M/M/1-FCFS we can get a relation for the response time distribution if we consider the response time as the sum of $k+1$ independent exponentially distributed random variables [Triv82]:

$$X + X_1 + X_2 + \cdots + X_k,$$

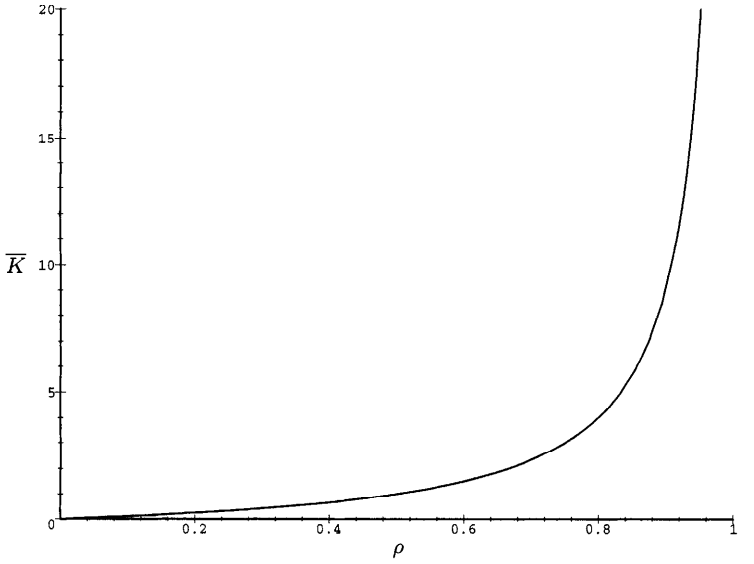


Fig. 6.3 The mean number of jobs $\overline{K}$ in a M/M/1 system.

where X is the service time of the tagged job, X_1 is the remaining service time of the job in service when the tagged job arrives, and $X_2, \ldots, X_k$ are the service times of the jobs in the queue; each of these is exponentially distributed with parameter μ. Note that X_1 is also exponentially distributed with rate μ due to the memoryless property of the exponential distribution. Noting that the Laplace-Stieltjes transform (LST) of the exponentially distributed service time (see Table 1.5) is:

$$L_X(s) = \frac{\mu}{\mu + s},$$

we get for the conditional LST of the response time:

$$L_{T|K}(s|k) = \left(\frac{\mu}{\mu + s}\right)^{k+1}.$$

Unconditioning using the steady-state probability Eq. (6.12), the LST of the response time is:

$$\begin{aligned} L_T(s) &= \sum_{k=0}^{\infty} \left(\frac{\mu}{\mu + s}\right)^{k+1} \cdot (1-\rho)\rho^k \\ L_T(s) &= \frac{\mu(1-\rho)}{s + \mu(1-\rho)}. \end{aligned} \tag{6.18}$$

Thus the response time T is exponentially distributed with the parameter $\mu(1-\rho)$:

$$F_T(x) = 1 - \mathrm{e}^{-\mu(1-\rho)x} \tag{6.19}$$

and the variance:

$$\mathrm{var}(T) = \frac{1}{\mu^2(1-\rho)^2}. \tag{6.20}$$

Similarly we get the distribution of the waiting time:

$$F_W(x) = \begin{cases} 1-\rho, & x = 0, \\ 1 - \rho \cdot \mathrm{e}^{-\mu(1-\rho)x}, & x > 0. \end{cases} \tag{6.21}$$

Thus $F_W(0) = P(W = 0) = 1 - \rho$ is the mass at origin, corresponding to the probability that an arriving customer does not have to wait in the queue.

6.4 THE M/M/∞ SYSTEM

In an M/M/∞ queueing system we have a Poisson arrival process with arrival rate λ and an infinite number of servers with service rate μ each. If there are k jobs in the system, then the overall service rate is $k\mu$ because each arriving job immediately gets a server and does not have to wait. Once again, the underlying CTMC is a birth-death process. From Eq. (3.11) we obtain the steady-state probability of k jobs in the system:

$$\begin{aligned} \pi_k &= \pi_0 \prod_{i=0}^{k-1} \frac{\lambda}{(i+1)\mu} \\ &= \pi_0 \left(\frac{\lambda}{\mu}\right)^k \frac{1}{k!}, \end{aligned}$$

with Eq. (3.12), we obtain the steady-state probability of no jobs in the system:

$$\begin{aligned} \pi_0 &= \frac{1}{1 + \sum_{k=1}^{\infty} \left(\frac{\lambda}{\mu}\right)^k \cdot \frac{1}{k!}} \\ &= \mathrm{e}^{-\frac{\lambda}{\mu}}, \end{aligned} \tag{6.22}$$

and finally:

$$\pi_k = \frac{\left(\frac{\lambda}{\mu}\right)^k}{k!} \cdot \mathrm{e}^{-\frac{\lambda}{\mu}}. \tag{6.23}$$

This is the Poisson pmf, and the expected number of jobs in the system is:

$$\overline{K} = \frac{\lambda}{\mu}. \tag{6.24}$$

With Little's theorem the mean response time as expected:

$$\overline{T} = \frac{1}{\mu}. \tag{6.25}$$

6.5 THE M/M/m SYSTEM

An M/M/m queueing system with arrival rate λ and service rate μ for each server can also be modeled as a birth-death process with:

$$\lambda_k = \lambda, \qquad k \geq 0,$$
$$\mu_k = \begin{cases} k\mu, & 0 \leq k \leq m, \\ m\mu, & m \leq k. \end{cases}$$

The condition for the queueing system to be stable (underlying CTMC to be ergodic) is $\lambda < m\mu$. The steady-state probabilities are given by (from Eq. (3.11)):

$$\pi_k = \begin{cases} \pi_0 \displaystyle\prod_{i=0}^{k-1} \frac{\lambda}{(i+1)\mu} = \pi_0 \left(\frac{\lambda}{\mu}\right)^k \cdot \frac{1}{k!}, & 0 \leq k \leq m, \\ \pi_0 \displaystyle\prod_{i=0}^{m-1} \frac{\lambda}{(i+1)\mu} \cdot \prod_{i=m}^{k-1} \frac{\lambda}{m\mu}, & k \geq m. \end{cases}$$

With an individual server utilization, $\rho = \lambda/(m\mu)$, we obtain:

$$\pi_k = \begin{cases} \pi_0 \dfrac{(m\rho)^k}{k!}, & 0 \leq k \leq m, \\ \pi_0 \dfrac{\rho^k m^m}{m!}, & k \geq m, \end{cases} \tag{6.26}$$

and from Eq. (3.12):

$$\pi_0 = \left[\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!} \frac{1}{1-\rho} \right]^{-1}. \tag{6.27}$$

The steady-state probability that an arriving customer has to wait in the queue is given by:

$$\begin{aligned} P_m = P(K \geq m) &= \sum_{k=m}^{\infty} \pi_k \\ &= \frac{(m\rho)^m}{m!(1-\rho)} \cdot \pi_0 \end{aligned} \tag{6.28}$$

Using Eqs. (6.26) and (6.8) we obtain for the mean number of jobs in the system:

$$\overline{K} = m\rho + \frac{\rho}{1-\rho} \cdot P_m, \tag{6.29}$$

and for the mean queue length:

$$\overline{Q} = \frac{\rho}{1-\rho} \cdot P_m. \tag{6.30}$$

From this the mean response time $\overline{T}$ and mean waiting time $\overline{W}$ by Little's theorem (Eqs. (6.9) and (6.10)) can be easily derived. A formula for the distribution of the waiting time is given in [GrHa85]:

$$F_W(x) = \begin{cases} 1 - P_m, & x = 0, \\ 1 - P_m \cdot \mathrm{e}^{-m\mu(1-\rho)x}, & x > 0. \end{cases} \tag{6.31}$$

Figure 6.4 shows the interesting and important property of an M/M/m system that the mean number of jobs in the system $\overline{K}$ increases with the number of servers m if the server utilization is constant but the mean queue length $\overline{Q}$ decreases.

6.6 THE M/M/1/K FINITE CAPACITY SYSTEM

In an M/M/1/K queueing system, the maximum number of jobs in the system is K, which implies a maximum queue length of $K-1$. An arriving job enters the queue if it finds fewer than K jobs in the system and is lost otherwise. This behavior can be modeled by a birth-death process with:

$$\begin{aligned} \lambda_k &= \begin{cases} \lambda, & 0 \leq k < K, \\ 0, & k \geq K, \end{cases} \\ \mu_k &= \mu \quad k = 1, \ldots, K. \end{aligned}$$

Using Eqs. (3.12), (3.11), and $a = \lambda/\mu$, we obtain the steady-state probability of k jobs in the system:

$$\pi_k = \begin{cases} \dfrac{(1-a)a^k}{1-a^{K+1}}, & 0 \leq k \leq K, \\ 0, & k > K. \end{cases} \tag{6.32}$$

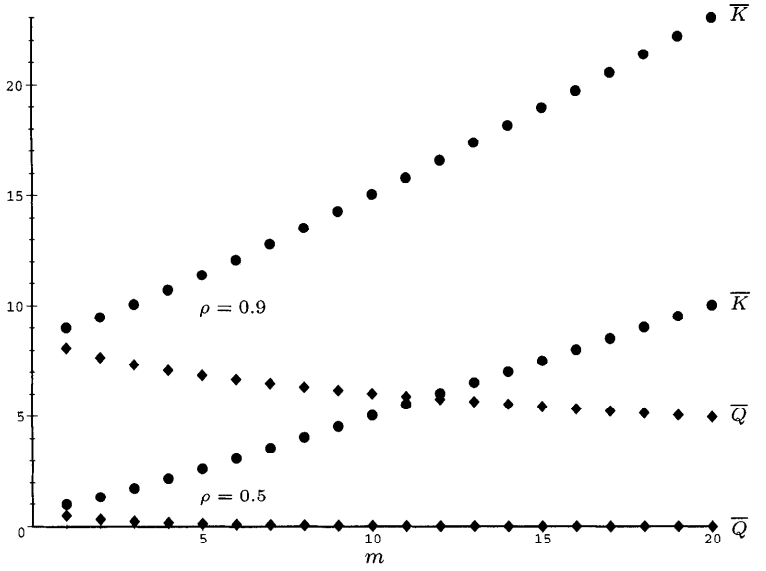


Fig. 6.4 Mean queue length $\overline{Q}$ and mean number of jobs in the system $\overline{K}$ as functions of the number of servers m.

Since there are no more than K customers in the system, the system is stable for all values of λ and μ. That is, we need not assume that $\lambda < \mu$ for the system to be stable. In other words, since the CTMC is irreducible and finite, it is ergodic. If $\lambda = \mu$, then $a = 1$ and:

$$\pi_0 = \frac{1}{K+1} = \pi_k, \quad k = 1, 2, \ldots, K. \tag{6.33}$$

The mean number of jobs in the system is given by (using Eq. (6.8)):

$$\overline{K} = \begin{cases} \dfrac{a}{1-a} - \dfrac{K+1}{1-a^{K+1}} \cdot a^{K+1}, & a \neq 1, \\ \dfrac{K}{2}, & a = 1. \end{cases} \tag{6.34}$$

Note that the utilization $\rho = 1 - \pi_0 \neq \lambda/\mu$ in this case; it is for this reason that we labeled the quantity $a = \lambda/\mu$ in the preceding equations. The throughput in this case is not equal to λ, but is equal to $\lambda(1 - \pi_K)$. Similar results have been derived for an M/M/m/K system (see [Klei75], [Alle90], [GrHa85], or [Triv82]).

6.7 MACHINE REPAIRMAN MODEL

Another special case of the birth-death process is obtained when the birth rate λ_j is of the form $(M-j)\lambda$, $j = 0, 1, \ldots, M$, and the death rate, $\mu_j = \mu$. This structure can be useful in the modeling of interactive computer systems where a an individual terminal user issues a request at the rate λ whenever it is in the "thinking state." If j out of the total of M terminals are currently waiting for a response to a pending request, the effective request rate is then $(M-j)\lambda$. The request completion rate is denoted by μ. Similarly, when M machines share a repair facility, the failure rate of each machine is λ and the repair rate is μ (see Fig. 6.5). Since the underlying CTMC is irreducible and

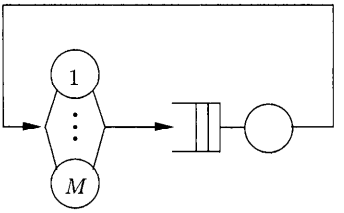


Fig. 6.5 Machine repairman model.

finite, it is ergodic implying that the queueing system is always stable. The expressions for steady-state probabilities are obtained using Eqs. (3.12) and (3.11) as:

$$\pi_k = \pi_0 \prod_{i=0}^{k-1} \frac{\lambda(M-i)}{\mu}, \quad 0 \le k \le M,$$

or:

$$\pi_k = \pi_0 \left(\frac{\lambda}{\mu}\right)^k \frac{M!}{(M-k)!}. \tag{6.35}$$

Hence:

$$\pi_0 = \frac{1}{\sum_{k=0}^{M} \left(\frac{\lambda}{\mu}\right)^k \frac{M!}{(M-k)!}}. \tag{6.36}$$

The utilization of the computer (or the repairman) is given by $\rho = (1-\pi_0)$ (see Eq. (6.11)) and the throughput by $\mu(1-\pi_0)$. With the average thinking time $1/\lambda$, we get for the mean response time of the computer:

$$\overline{T} = \frac{M}{\mu(1-\pi_0)} - \frac{1}{\lambda} \tag{6.37}$$

and, using Little's theorem, the average number of jobs in the computer:

$$\overline{K} = M - \frac{\mu(1-\pi_0)}{\lambda}. \tag{6.38}$$

6.8 CLOSED TANDEM NETWORK

Consider the closed tandem network in Fig. 6.6 with a total number of K

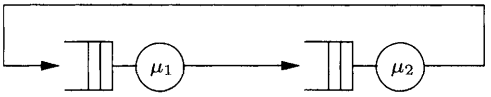


Fig. 6.6 Closed tandem network.

customers. This network is known as the cyclic queueing network. Assume that node 1 has service rate μ, and node 2 has service rate μ_2. The state space in this case is $\{(k_1, k_2) \mid k_1 \geq 0,\ k_2 \geq 0,\ k_1 + k_2 = K\}$ where k_i $(i = 1, 2)$ is the number of jobs at node i. Such multidimensional state spaces, we modify the earlier notation for steady-state probabilities. The joint probability of the state (k_1, k_2) will be denoted by $\pi(k_1, k_2)$ while the marginal pmf at node i will be denoted by $\pi_i(k_i)$. Although, the underlying CTMC appears to have a two-dimensional state space, in reality it is one-dimensional birth-death process because of the condition $k_1 + k_2 = K$. Thus the marginal pmf of node 2 can be derived from Eqs. (3.12) and (3.11):

$$\pi_2(k_2) = \pi_2(0) \cdot \left(\frac{\mu_1}{\mu_2}\right)^{k_2}, \quad k_2 \leq K; \quad \pi_2(k_2) = 0,\ k_2 > K.$$

$$\pi_2(0) = \frac{1}{1 + \sum_{i=1}^{K} \left(\frac{\mu_1}{\mu_2}\right)^i} = \frac{1}{\sum_{i=0}^{K} \left(\frac{\mu_1}{\mu_2}\right)^i},$$

and with $u = \mu_1/\mu_2$:

$$\pi_2(k_2) = \pi_2(0) \cdot u^{k_2}, \tag{6.39}$$

$$\pi_2(0) = \begin{cases} \dfrac{1-u}{1-u^{K+1}}, & u \neq 1, \\ \dfrac{1}{K+1}, & u = 1. \end{cases} \tag{6.40}$$

Similarly, we obtain for node 1:

$$\pi_1(k_1) = \pi_1(0) \cdot \frac{1}{u^{k_1}}, \tag{6.41}$$

$$\pi_1(0) = \begin{cases} \dfrac{1-\frac{1}{u}}{1-\left(\frac{1}{u}\right)^{K+1}}, & u \neq 1, \\ \dfrac{1}{K+1}, & u = 1. \end{cases} \tag{6.42}$$

The utilizations are given by:

$$\rho_1 1 - \pi_1(0), \qquad \rho_2 = 1 - \pi_2(0), \tag{6.43}$$

and the throughput:

$$\lambda = \lambda_1 = \lambda_2 = \rho_1\mu_1 = \rho_2\mu_2, \tag{6.44}$$

and the mean number of customers in node 2 with equation:

$$\begin{aligned}
\overline{K}_2 &= \pi_2(0) \cdot \sum_{k_2=0}^{K} k_2 \cdot u^{k_2} \\
&= \pi_2(0) \cdot u\frac{\partial}{\partial u} \cdot \sum_{k_2=0}^{K} u^{k_2} \\
&= \pi_2(0) \cdot u\frac{\partial}{\partial u} \cdot \frac{1-u^{k_2+1}}{1-u} \\
&= \frac{u}{1-u} - \frac{(K+1)\cdot u^{K+1}}{1-u^{K+1}}, \quad u \neq 1, \qquad (6.45) \\
\overline{K}_1 &= K - \overline{K}_2. \qquad (6.46)
\end{aligned}$$

6.9 THE M/G/1 SYSTEM

The mean waiting time $\overline{W}$ of an arriving job in an M/G/1 system has two components:

1. The mean remaining service time $\overline{W}_0$ of the job in service (if any),
2. The sum of the mean service times of the jobs in the queue.

We can sum these components to:

$$\overline{W} = \overline{W}_0 + \overline{Q} \cdot \overline{T}_B, \tag{6.47}$$

where the mean remaining service time $\overline{W}_0$ is given by:

$$\overline{W}_0 = P(\text{server is busy}) \cdot \overline{R} + P(\text{server is idle}) \cdot 0, \tag{6.48}$$

with the main remaining service time $\overline{R}$ of a busy server (the remaining service time of an idle server is obviously zero). When a job arrives, the job in service needs $\overline{R}$ time units on the average to be finished. This quantity is also called the mean residual life and is given by [Klei75]:

$$\overline{R} = \frac{\overline{T_B^2}}{2\overline{T}_B} = \frac{\overline{T}_B}{2}(1 + c_B^2). \tag{6.49}$$

For an M/M/1 system ($c_B^2 = 1$), we obtain:

$$\overline{R}_{\mathrm{M/M/1}} = \overline{T}_B = \frac{1}{\mu},$$

which is related to the memoryless property of the exponential distribution. The probability $P(\text{server is busy})$ is the utilization ρ by definition. From Eq. (6.47) and Little's theorem $\overline{Q} = \lambda \cdot \overline{W}$, we obtain:

$$\overline{W} = \frac{\overline{W}_0}{1-\rho} \tag{6.50}$$

and with Eq. (6.49) finally:

$$\overline{Q} = \frac{\rho^2}{(1-\rho)} \cdot \frac{(1+c_B^2)}{2}, \tag{6.51}$$

the well-known Pollaczek-Khintchine formula (see Eq. (3.28)) for the mean queue length.

For exponentially distributed service time we have $c_B^2 = 1$ and for deterministic service time we have $c_B^2 = 0$. Thus:

$$\overline{Q}_{\mathrm{M/M/1}} = \frac{\rho^2}{1-\rho} \qquad \text{(see Eq. (6.17))},$$

$$\overline{Q}_{\mathrm{M/D/1}} = \frac{\rho^2}{2(1-\rho)}.$$

The mean queue length is divided by two if we have deterministic service time instead of exponentially distributed service times. Using the theory of embedded DTMC (see Section 3.2.1), the Pollaczek-Khintchine transform equation can be derived (Eq. (3.25)):

$$G(z) = \tilde{B}(\lambda - \lambda z)\frac{(1-\rho)(1-z)}{\tilde{B}(\lambda - \lambda z) - z}, \tag{6.52}$$

where $G(z)$ is the z-transform of steady-state probabilities of the number of jobs in the system π_k and $\tilde{B}(s)$ is the LST of the service time.

As an example we consider the M/M/1 system with $\tilde{B}(s) = \mu/(s+\mu)$ (see Table 1.5). From Eq. (6.52) it follows that:

$$G(z) = \frac{1-\rho}{1-\rho z}$$

or:

$$G(z) = \sum_{k=0}^{\infty}(1-\rho)\rho^k \cdot z^k.$$

Hence we find the steady-state probability of k jobs in the system:

$$\pi_k = (1-\rho)\rho^k.$$

With Eqs. (6.52) and (1.57) we obtain for the mean number in the system [Klei75] (see also Eq. (3.28)):

$$\overline{K} = \rho + \frac{\rho^2}{1-\rho} \cdot \frac{1+c_B^2}{2} \tag{6.53}$$

or, using Eq. (6.7) and Little's theorem:

$$\overline{K} = \rho + \overline{Q}. \tag{6.54}$$

Hence:

$$\overline{Q} = \frac{\rho^2}{1-\rho} \cdot \frac{1+c_B^2}{2},$$

which is the same as Eq. (6.51).

Figure 6.7*a* shows that for an M/G/1 system the mean number of jobs $\overline{K}$ increases linearly with increasing squared coefficient of variation, and that the rate of increase is very sensitive to the utilization. From Fig. 6.7*b* we see how dramatically the mean number of jobs $\overline{K}$ increases if the utilization approaches 1 and that this is especially the case if the squared coefficient of variation c_B^2 is large.

6.10 THE GI/M/1 SYSTEM

We state the results for GI/M/1 systems, using the parameter σ given by:

$$\sigma = \tilde{A}(\mu - \mu\sigma) \tag{6.55}$$

where $\tilde{A}(s)$ is the LST of the interarrival time [Tane95]. For example, the mean number of jobs in the system:

$$\overline{K} = \frac{\rho}{1-\sigma}, \tag{6.56}$$

the variance of the number of jobs in the system:

$$\sigma_K^2 = \frac{\rho(1+\sigma-\rho)}{(1-\sigma)^2}, \tag{6.57}$$

the mean response time:

$$\overline{T} = \frac{1}{\mu} \cdot \frac{1}{1-\sigma}, \tag{6.58}$$

the mean queue length:

$$\overline{Q} = \frac{\rho \cdot \sigma}{1-\sigma}, \tag{6.59}$$

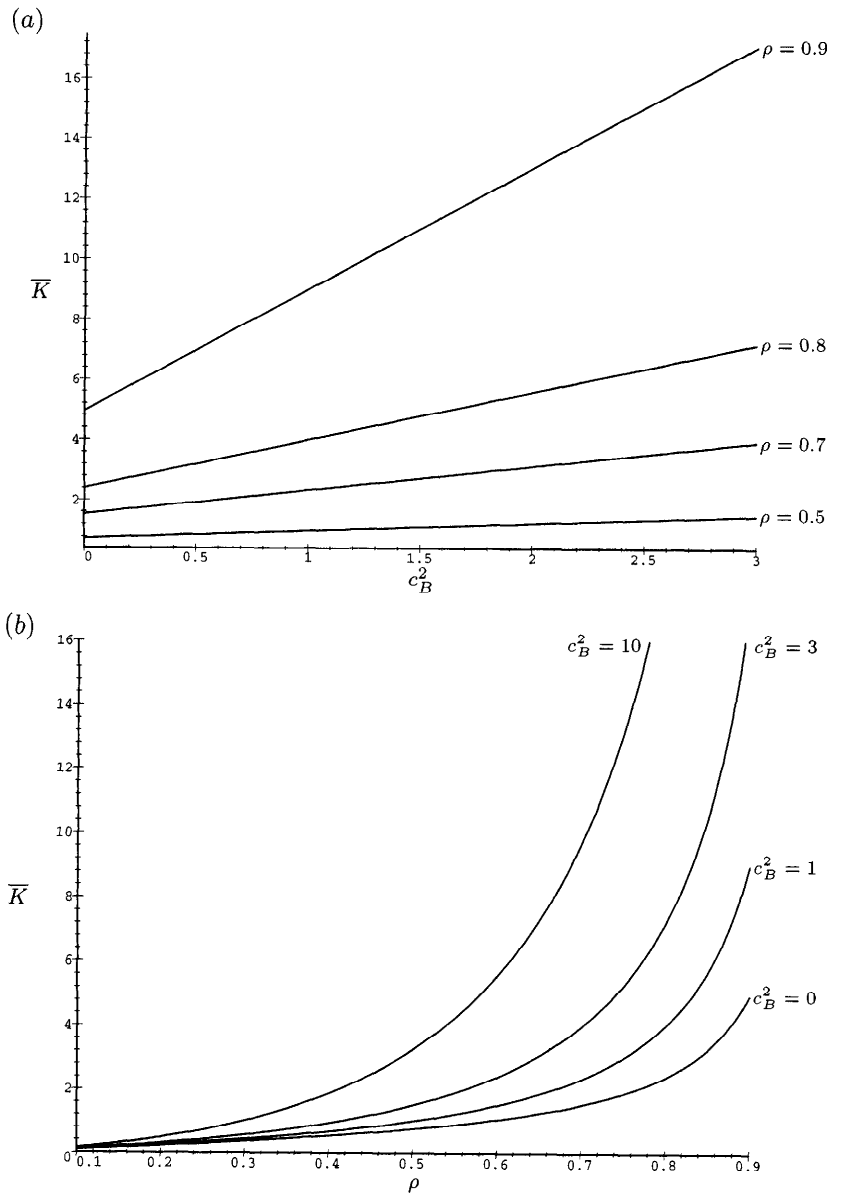


Fig. 6.7 (*a*) Mean number of jobs $\overline{K}$ in an M/G/1 system as a function of c_B^2 with parameter ρ. (*b*) Mean number of jobs $\overline{K}$ in a M/G/1 system as a function of ρ with parameter c_B^2.

the variance of the queue length:

$$\sigma_Q^2 = \frac{\rho\sigma(1+\sigma(1-\rho))}{(1-\sigma)^2}, \tag{6.60}$$

the mean waiting time:

$$\overline{W} = \frac{1}{\mu} \cdot \frac{\sigma}{1-\sigma}, \tag{6.61}$$

the steady-state probability of the number of jobs in the system:

$$\begin{aligned} \pi_k &= \rho(1-\sigma)\sigma^{k-1}, \quad k > 0, \\ \pi_0 &= 1-\rho, \end{aligned} \tag{6.62}$$

and the waiting time distribution:

$$F_W(x) = \begin{cases} 1-\sigma, & x = 0, \\ 1-\sigma \cdot \mathrm{e}^{-\mu(1-\sigma)x}, & x > 0. \end{cases} \tag{6.63}$$

In the case of an M/M/1 system, we have $A^{\tilde{}}(s) = \lambda/(s+\lambda)$ (see Table 1.5). Together with Eq. (6.55) we obtain:

$$\sigma = \frac{\lambda}{\mu} = \rho.$$

and with the Eqs. (6.56)-(6.63), the well-known M/M/1 formulae.

A more interesting example is the E_2/M/1 system. From Table 1.5 we have:

$$A^{\tilde{}}(s) = \left(\frac{\lambda}{s+\lambda}\right)^2,$$

and with Eq. (6.55) we have

$$\sigma = \rho + \frac{1}{2} - \sqrt{\rho - \frac{1}{4}}.$$

Using Eqs. (6.56)-(6.63), we obtain explicit formulae for E_2/M/1 performance measures.

The behavior of an M/G/1 and of a GI/M/1 system is very similar, especially if $c_X^2 \leq 1$. This is shown in Fig. 6.8 where we compare the mean number of jobs $\overline{K}$ for M/G/1 and GI/M/1 systems having the same coefficient of variation c_X^2. Note that c_X^2 in the case of GI/M/1 denotes the coefficient of variation of the interarrival times, while in the M/G/1 case it denotes the coefficient of variation of the service times. Note also that in the M/G/1 case, $\overline{K}$ depends only on the first two moments of the service time distribution, while in the GI/M/1 case, the dependence is more extensive. In Fig. 6.8, for the GI/M/1 system, the gamma distribution is used. For increasing value of c_X^2 the deviation increases.

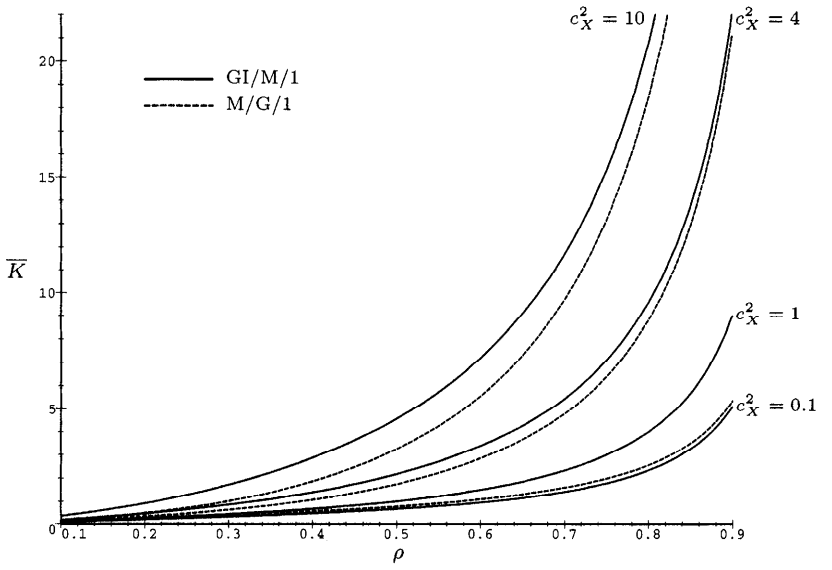


Fig. 6.8 Mean number of jobs $\overline{K}$ in an M/G/1 and a GI/M/1 system.

6.11 THE GI/M/m SYSTEM

Exact results for GI/M/m queueing systems are also available. See [Alle90], [Klei75], or [GrHa85]. The mean waiting time is given by:

$$\overline{W} = \frac{J}{m\mu(1-\sigma)^2}, \tag{6.64}$$

where:

$$\sigma = \tilde{A}(m\mu - m\mu\sigma) \tag{6.65}$$

and:

$$J = \left(\frac{1}{1-\sigma} + \sum_{k=0}^{m-2} R_k \right)^{-1}.$$

For R_k, see [Klei75] (page 408). We introduce only a heavy traffic approximation:

$$\overline{W} \approx \frac{\sigma_A^2 + \sigma_B^2/m^2}{2(1-\rho)} \cdot \lambda, \tag{6.66}$$

which is an upper bound [Klei75] and can be used for GI/G/1 and GI/G/m system as well (see subsequent sections). Here λ is the reciprocal of the mean interarrival time, σ_A^2 is the variance of interarrival time, and σ_B^2 is the variance of service time.

6.12 THE GI/G/1 SYSTEM

In the GI/G/1 case only approximation formulae and bounds exist. We can use M/G/1 and GI/M/1 results as upper or lower bounds, depending on the value of the coefficient of variation (see Table 6.1).

Table 6.1 Upper bounds (UB) and lower bounds (LB) for the GI/G/1 mean waiting time

c_A^2	c_B^2	M/G/1	GI/M/1
>1	>1	LB	LB
>1	<1	LB	UB
<1	>1	UB	LB
<1	<1	UB	UB

Another upper bound is given by Eq. (6.66) with $m = 1$ [Klei75]:

$$\overline{W} < \frac{\sigma_A^2 + \sigma_B^2}{2(1-\rho)} \cdot \lambda. \tag{6.67}$$

A modification of this upper bound is [Marc78]:

$$\overline{W} < \frac{1 + c_B^2}{(1/\rho^2) + c_B^2} \cdot \frac{\sigma_A^2 + \sigma_B^2}{2(1-\rho)} \cdot \lambda. \tag{6.68}$$

This formula is exact for M/G/1 and is a good approximation for GI/M/1 and GI/G/1 systems if ρ is not too small and c_A^2 or c_B^2 are not too big.

A lower bound is also known [Marc78]:

$$\overline{W} > \frac{\rho^2 \cdot c_B^2 + \rho(\rho - 2)}{2\lambda(1-\rho)}, \tag{6.69}$$

but more complex and better lower bounds are given in [Klei76].

Many approximation formulae for the mean waiting time are mentioned in the literature. Four of them that are either very simple and straightforward or good approximations are introduced here. First, the well-known Allen-Cunneen approximation formula for GI/G/m systems is [Alle90]:

$$\overline{W} \approx \frac{\rho/\mu}{1-\rho} \cdot \frac{c_A^2 + c_B^2}{2}. \tag{6.70}$$

This formula is exact for M/G/1 (Pollaczek-Khintchine formula) and a fair approximation elsewhere and is the basis for many other better approximations. A very good approximation is the Krämer/Langenbach-Belz formula, a direct extension of Eq. (6.70) via a correction factor:

$$\overline{W} \approx \frac{\rho/\mu}{1-\rho} \cdot \frac{c_A^2 + c_B^2}{2} \cdot G_{\mathrm{KLB}}, \tag{6.71}$$

with the correction factor:

$$G_{\mathrm{KLB}} = \begin{cases} \exp\left(-\frac{2}{3} \cdot \frac{1-\rho}{\rho} \cdot \frac{(1-c_A^2)^2}{c_A^2 + c_B^2}\right), & 0 \le c_A \le 1, \\ \exp\left(-(1-\rho)\frac{c_A^2 - 1}{c_A^2 + 4c_B^2}\right), & c_A > 1. \end{cases} \tag{6.72}$$

Another extension of the Allen-Cunneen formula is the approximation of Kulbatzki [Kulb89]:

$$\overline{W} \approx \frac{\rho/\mu}{1-\rho} \cdot \frac{c_A^{f(c_A,c_B,\rho)} + c_B^2}{2}, \tag{6.73}$$

with:

$$f(c_A, c_B, \rho) = \begin{cases} 1, & c_A \in \{0, 1\}, \\ \Big[\rho(14.1c_A - 5.9) + (-13.7c_A + 4.1)\Big]c_B^2 & \\ + \Big[\rho(-59.7c_A + 21.1) + (54.9c_A - 16.3)\Big]c_B & \\ + \Big[\rho(c_A - 4.5) + (-1.5c_A + 6.55)\Big], & 0 \le c_A \le 1, \\ -0.75\rho + 2.775, & c_A > 1, \end{cases} \tag{6.74}$$

It is interesting to note that Eq. (6.74) was obtained using simulation experiments. A good approximation for the case $c_A^2 < 1$ is the Kimura approximation [Kimu85]:

$$\overline{W} \approx \frac{c_A^2 + c_B^2}{2} \overline{W}_{\mathrm{M/M/m}} \left((1 - c_A^2) \exp\left(\frac{2(1-\rho)}{3\rho}\right) + c_A^2\right)^{-1}. \tag{6.75}$$

6.13 THE M/G/m SYSTEM

We obtain the Martin's approximation formula for M/G/m systems [Mart72] by an extension of the Eq. (6.47) for the mean waiting time for an M/G/1 system:

$$\overline{W} = \overline{W}_0 + \frac{\overline{Q}}{m} \cdot \overline{T}. \tag{6.76}$$

Because of the m servers, an arriving customer has to wait, on the average, only for the service of $\overline{Q}/m$ customers. The remaining service time in this case is:

$$\overline{W}_0 = P_m \cdot \overline{R}. \tag{6.77}$$

With Eq. (6.49) we obtain as an approximation:

$$\overline{R} \approx \overline{T}\frac{(1+c_B^2)}{2m}, \tag{6.78}$$

and with Eqs. (6.76) and (6.77) finally:

$$\overline{W} \approx \frac{P_m/\mu}{1-\rho} \cdot \frac{(1+c_B^2)}{2m}. \tag{6.79}$$

This is a special case of the Allen-Cunneen formula for GI/G/m systems (see next section) and is exact for M/M/m and M/G/1 systems.

For the waiting probability P_m we can use Eq. (6.28) for M/M/m systems or a simple yet good approximation from [Bolc83]:

$$P_m \approx \begin{cases} \dfrac{\rho^m + \rho}{2}, & \rho > 0.7, \\ \rho^{\frac{m+1}{2}}, & \rho < 0.7. \end{cases} \tag{6.80}$$

As an example, we compare the exact waiting probability with this approximation for $m = 5$ in Table 6.2.

Table 6.2 Exact (Eq. (6.28)) and approximate (Eq. (6.80)) values of the probability of waiting

ρ	0.2	0.4	0.6	0.7	0.8	0.9	0.95	0.99
$P_{m_{ex}}$	0	0.06	0.23	0.38	0.55	0.76	0.88	0.97
$P_{m_{app}}$	0	0.06	0.21	0.34	0.56	0.75	0.86	0.97

From Fig. 6.9 we see that the deviation for the mean number of jobs $\overline{K}$ in the system is very small if we use the approximation for P_m instead of the exact value.

A good approximation for the mean waiting time in M/G/m systems is due to Cosmetatos [Cosm76]:

$$\overline{W}_{\mathrm{M/G/m}} \approx c_B^2 \overline{W}_{\mathrm{M/M/m}} + (1 - c_B^2)\overline{W}_{\mathrm{M/D/m}}. \tag{6.81}$$

In Eq. (6.81), we can use Eq. (6.30) for $\overline{W}_{\mathrm{M/M/m}}$ and use:

$$\overline{W}_{\mathrm{M/D/m}} = \frac{1}{2} \cdot \frac{1}{nc_{Dm}} \cdot \overline{W}_{\mathrm{M/M/m}}$$

where:

$$nc_{Dm} = \left(1 + (1-\rho)(m-1)\frac{\sqrt{4+5m}-2}{16\rho m}\right)^{-1}. \tag{6.82}$$

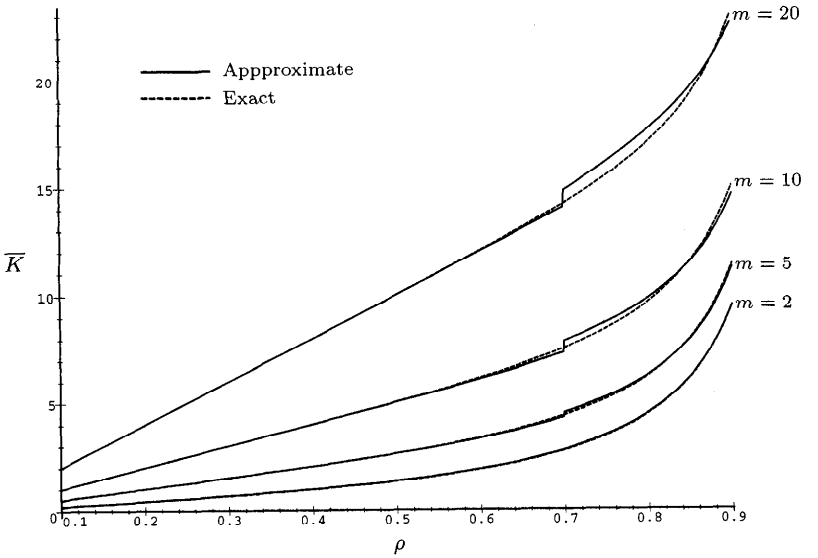


Fig. 6.9 Mean number of jobs in an M/M/m system with exact and approximative formulae for P_m.

For $\overline{W}_{\mathrm{M/D/m}}$ we can also use the Crommelin approximation formula [Crom34]:

$$\overline{W}_{\mathrm{M/D/m}} \approx \frac{1}{\mu}\sum_{\nu=1}^{\infty}\left(e^{-\nu\rho m}\left(\frac{(\nu\rho m)^{\nu m}}{(\nu m)!} - (1-1/\rho)\sum_{i=1}^{\nu m}\frac{(\nu\rho m)^i}{i!}\right) + \left(1-\frac{1}{\rho}\right)\right). \tag{6.83}$$

Boxma, Cohen, and Huffels [BCH79] also use the preceding formulae for $\overline{W}_{\mathrm{M/D/m}}$ as a basis for their approximation:

$$\overline{W}_{\mathrm{M/G/m}} \approx \frac{1}{2}(1+c_B^2)\frac{2\overline{W}_{\mathrm{M/D/m}}\overline{W}_{\mathrm{M/M/m}}}{2a\overline{W}_{\mathrm{M/D/m}} + (1-a)\overline{W}_{\mathrm{M/m/m}}}, \tag{6.84}$$

where:

$$a = \begin{cases} 1, & m = 1, \\ \dfrac{1}{m-1}\left(\dfrac{(c_B^2+1)}{\gamma_1} - m + 1\right), & m > 1, \end{cases}$$

and:

$$\gamma_1 \approx \frac{1-c_B^2}{m+1} + \frac{c_B^2}{m}.$$

Tijms [Tijm86] uses γ_1 from the BCH-formula (Eq. (6.84)) in his approximation:

$$\overline{W}_{\mathrm{M/G/m}} \approx \left((1-\rho)\gamma_1 m + \frac{\rho}{2}(c_B^2+1)\right)\overline{W}_{\mathrm{M/M/m}}. \tag{6.85}$$

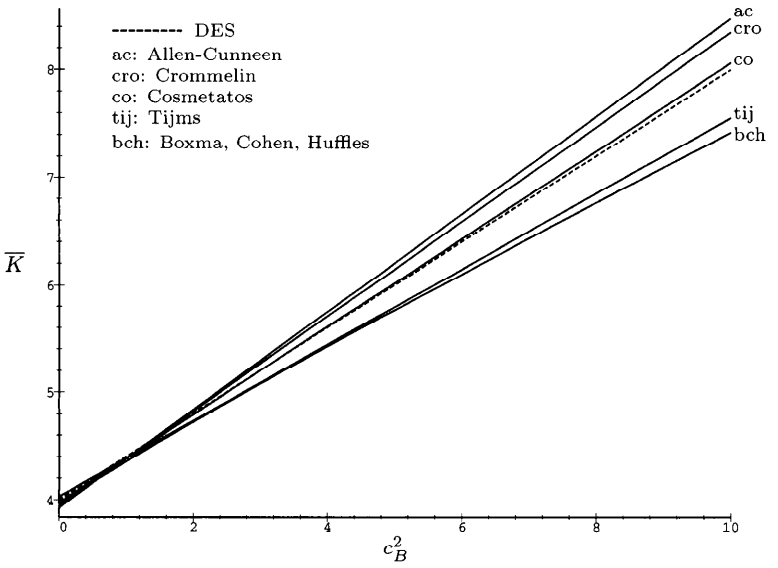


Fig. 6.10 Comparison of different M/G/5 approximations with DES ($\rho = 0.7$).

In Fig. 6.10 we compare five approximations introduced above with the results obtained from discrete-event simulation (DES). From the figure we see that for $\rho = 0.7$ all approximations are good for $c_B^2 < 2$, and that for higher values of c_B^2 the approximation due to Cosmetatos is very good and the others are fair.

6.14 THE GI/G/m SYSTEM

For GI/G/m systems only bounds and approximation formulae are available. These are extensions of M/G/m or GI/G/1 formulae. We begin with the well-known upper bound due to Kingman [King70]:

$$\overline{W} \leq \frac{\sigma_A^2 + \sigma_B^2/m + (m-1)/(m^2\cdot\mu^2)}{2(1-\rho)} \cdot \lambda, \tag{6.86}$$

and the lower bound of Brumelle [Brum71] and Marchal [Marc74]:

$$\overline{W} \geq \frac{\rho^2 c_B^2 - \rho(2-\rho)}{2\lambda(1-\rho)} - \frac{m-1}{m}\cdot\frac{c_B^2+1}{2\mu}. \tag{6.87}$$

As a heavy traffic approximation we have Eq. (6.66), which we already introduced for GI/M/m systems:

$$\overline{W} \approx \frac{\sigma_A^2 + \sigma_B^2/m^2}{2(1-\rho)} \cdot \lambda, \tag{6.88}$$

and the Kingman-Köllerström approximation [Köll74] for the waiting time distribution:

$$F_W(x) \approx 1 - \exp\left(-\frac{2(1-\rho)}{\sigma_A^2 + \sigma_B^2/m^2} \cdot \frac{1}{\lambda} x\right). \tag{6.89}$$

The most known approximation formula for GI/G/m systems is the Allen-Cunneen (A-C) formula [Alle90]. We already introduced it for the special case GI/G/1 (Eq. (6.70)). Note that the A-C formula is an extension of Martin's formula (Eq. (6.79)) where we replace the 1 in the term $(1 + c_B^2)$ by c_A^2 to consider approximately the influence of the distribution of interarrival times:

$$\overline{W} \approx \frac{P_m/\mu}{1-\rho} \cdot \frac{c_A^2 + c_B^2}{2m}. \tag{6.90}$$

For the probability of waiting we can either use Eq. (6.28) or the good approximation provided by Eq. (6.80). As in the GI/G/1 case, the Allen-Cunneen approximation was improved by Krämer/Langenbach-Belz [KrLa76] using a correction factor:

$$\overline{W} \approx \frac{P_m/\mu}{1-\rho} \cdot \frac{c_A^2 + c_B^2}{2m} \cdot G_{\mathrm{KLB}}, \tag{6.91}$$

$$G_{\mathrm{KLB}} = \begin{cases} \exp\left(-\frac{2}{3}\frac{1-\rho}{P_m}\frac{(1-c_A^2)^2}{c_A^2 + c_B^2}\right), & 0 \le c_A \le 1, \\ \exp\left(-(1-\rho)\frac{c_A^2 - 1}{c_A^2 + 4c_B^2}\right), & c_A > 1, \end{cases} \tag{6.92}$$

and by Kulbatzki [Kulb89] using the exponent $f(c_A, c_B, \rho)$ in place of 2 for c_A in Eq. (6.90):

$$\overline{W} \approx \frac{P_m/\mu}{1-\rho} \cdot \frac{c_A^{f(c_A,c_B,\rho)} + c_B^2}{2m}. \tag{6.93}$$

For the definition of $f(c_A, c_B, \rho)$, see Eq. (6.74). The Kulbatzki formula was further improved by Jaekel [Jaek91]. We start with the Kulbatzki GI/G/1-formula and use a heuristic correction factor to consider the number of servers m:

$$\overline{W} \approx \frac{\rho/\mu}{1-\rho} \cdot \frac{c_A^{f(c_A,c_B,\rho)} + c_B^2}{2} \cdot \rho^{\sqrt{0.5(m-1)}}. \tag{6.94}$$

This formula is applicable even if the values of m and the coefficients of variation are large.

In order to extend the Cosmetatos approximation [Cosm76] from M/G/m to GI/G/m systems, $\overline{W}_{\mathrm{M/M/m}}$ and $\overline{W}_{\mathrm{M/D/m}}$ need to be replaced by $\overline{W}_{\mathrm{GI/M/m}}$ and $\overline{W}_{\mathrm{GI/D/m}}$, respectively:

$$\overline{W}_{\mathrm{GI/G/m}} \approx c_B^2 \overline{W}_{\mathrm{GI/M/m}} + (1 - c_B^2)\overline{W}_{\mathrm{GI/D/m}}, \tag{6.95}$$

where $\overline{W}_{\mathrm{GI/M/m}}$ is given by Eq. (6.64) or by the following approximation:

$$\overline{W}_{\mathrm{GI/M/m}} \approx \begin{cases} \dfrac{1}{2}(c_A^2 + 1) \exp\left(-\dfrac{2}{3} \cdot \dfrac{1-\rho}{P_m} \cdot \dfrac{(1-c_A^2)^2}{1+c_A^2}\right) & \text{for } 0 \le c_A \le 1, \\ \dfrac{1}{2}\left(c_A^{f(c_A, c_B, \rho)} + 1\right) \overline{W}_{\mathrm{M/M/m}} & \text{for } c_A > 1, \end{cases} \tag{6.96}$$

and $\overline{W}_{\mathrm{GI/D/m}}$ by:

$$\overline{W}_{\mathrm{GI/D/m}} = \frac{1}{2} \cdot \frac{1}{nc_{Dm}} \cdot \overline{W}_{\mathrm{GI/M/m}}, \tag{6.97}$$

with nc_{Dm} from Eq. (6.82) or:

$$\begin{aligned} \overline{W}_{\mathrm{GI/D/m}} &\approx c_A^{h(\rho,m) f(c_A, 0, \rho)} \cdot \overline{W}_{\mathrm{M/D/m}} \\ h(\rho, m) &= 4\sqrt{(m-1)/(m+4)} \cdot (1-\rho) + 1 \end{aligned} \tag{6.98}$$

with $\overline{W}_{\mathrm{M/D/m}}$ from Eq. (6.82) or Eq. (6.83). A good approximation for the case $c_A^2 < 1$ is given by Kimura [Kimu85]:

$$\begin{aligned} \overline{W}_{\mathrm{GI/G/m}} &= \frac{c_A^2 + c_B^2}{2} \overline{W}_{\mathrm{M/M/m}} \\ &\quad \cdot \left(\frac{1 - c_A^2}{1 - 4c(m,\rho)} \exp\left(\frac{2(1-\rho)}{3\rho}\right) + \frac{1 - c_B^2}{1 + c(m,\rho)} + c_A^2 + c_B^2 - 1 \right)^{-1}, \\ c(m,\rho) &= (1-\rho)(m-1) \frac{\sqrt{4+5m} - 2}{16\rho m}. \end{aligned} \tag{6.99}$$

Finally the Boxma, Cohen, and Huffels formula [BCH79] can be extended to GI/G/m systems:

$$\overline{W}_{\mathrm{GI/G/m}} = \frac{1}{2}(1 + c_B^2) \frac{2\overline{W}_{\mathrm{GI/D/m}} \overline{W}_{\mathrm{GI/M/m}}}{2a\overline{W}_{\mathrm{GI/D/m}} + (1-a)\overline{W}_{\mathrm{GI/M/m}}} \tag{6.100}$$

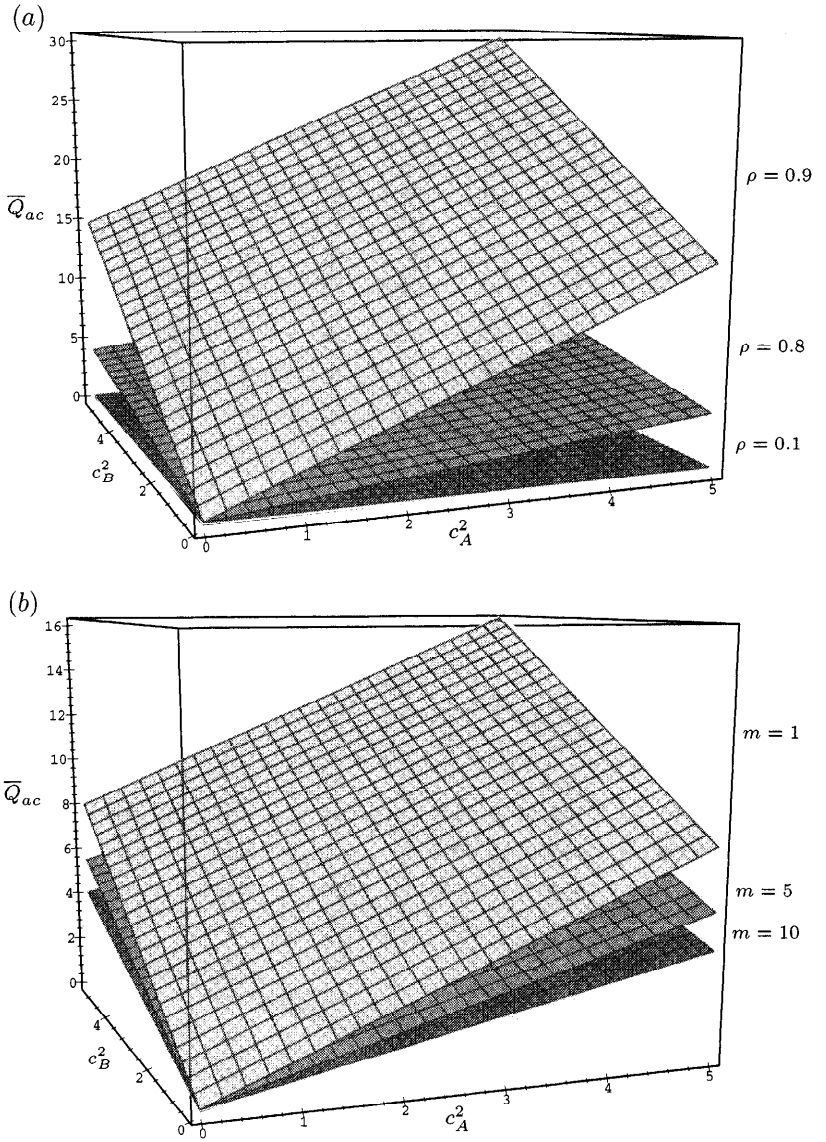


Fig. 6.11 (*a*) Mean queue length $\overline{Q}$ of a GI/G/10 system, and (*b*) mean queue length $\overline{Q}$ of a GI/G/m system with $\rho = 0.8$.

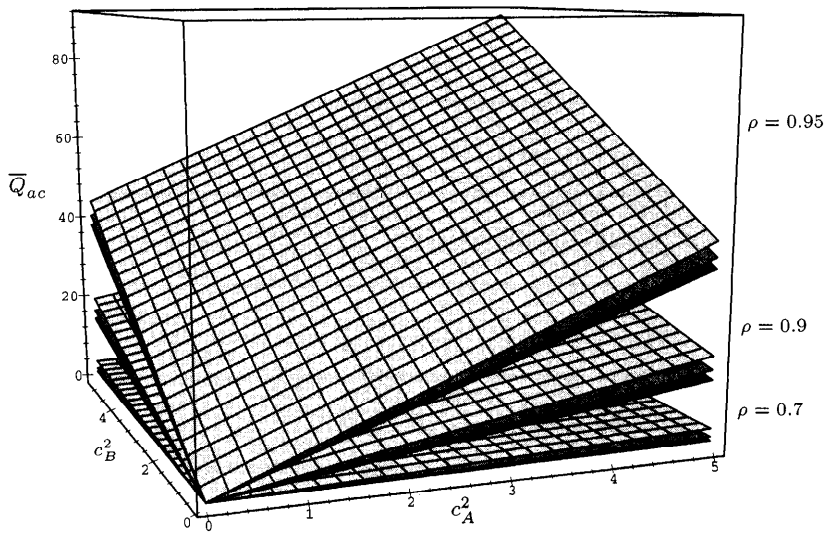


Fig. 6.12 Mean queue length of a GI/G/m system with $m = 1, 5, 10$.

as well as the Tijms formula [Tijm86]:

$$\overline{W}_{\mathrm{GI/G/m}} = \left((1-\rho)\gamma_1 m + \frac{\rho}{2}(c_B^2+1)\right)\overline{W}_{\mathrm{GI/M/m}}, \tag{6.101}$$

with a and γ_1 from Eq. (6.84).

In Figs. 6.11 and 6.12 the mean queue lengths as functions of the coefficients of variation c_A^2, c_B^2, the utilization ρ, and the number of servers m are shown in a very compact manner using the approximate formula of Allen-Cunneen. Fig. 6.13 shows that the approximations of Allen-Cunneen, Krämer/Langenbach-Belz, and Kulbatzki are close together, especially for higher values of the coefficients of variation c_X^2 with a large number of servers m.

6.15 PRIORITY QUEUEING

In a priority queueing system we assume that an arriving customer belongs to a priority class r ($r = 1, 2, \ldots, R$). The next customer to be served is the customer with the highest priority number r. Inside a priority class the queueing discipline is FCFS.

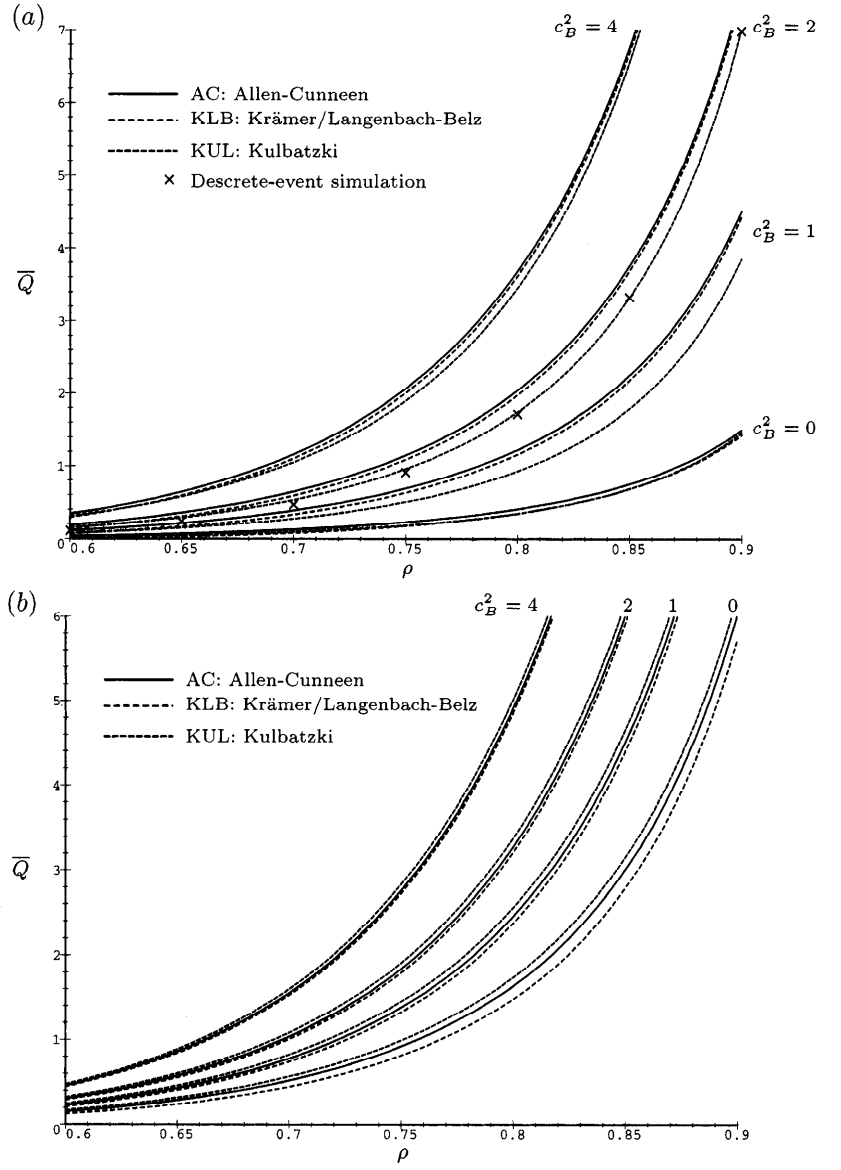


Fig. 6.13 (*a*) Mean queue length $\overline{Q}$ for a GI/G/10 system with $c_A^2 = 0.5$. (*b*) Mean queue length $\overline{Q}$ for a GI/G/10 system with $c_A^2 = 2.0$.

6.15.1 System without Preemption

Here we consider the case where a customer already in service is not preempted by an arriving customer with higher priority. The mean waiting time $\overline{W}_r$ of an arriving customer of priority class r has three components:

1. The mean remaining service time $\overline{W}_0$ of the job in service (if any).
2. Mean service time of customers in the queue that are served before the tagged customer. These are the customers in the queue of the same and higher priority as the tagged customer.
3. Mean service time of customers that arrive at the system while the tagged customer is in the queue and are served before him. These are customers with higher priority than the tagged customers.

Define:

$\overline{N}_{ir}$: Mean number of customers of class i found in the queue by the tagged (priority r) customer and receiving service before him,

$\overline{M}_{ir}$: Mean number of customers of class i who arrive during the waiting time of the tagged customer and receive service before him.

Then the mean waiting time of class r customers can be written as the sum of three components:

$$\overline{W}_r = \overline{W}_0 + \sum_{i=1}^{R} \overline{N}_{ir} \cdot \frac{1}{\mu_i} + \sum_{i=1}^{R} \overline{M}_{ir} \cdot \frac{1}{\mu_i}. \tag{6.102}$$

For multiple server systems ($m > 1$):

$$\overline{W}_r = \overline{W}_0 + \sum_{i=1}^{R} \frac{\overline{N}_{ir}}{m} \frac{1}{\mu_i} + \sum_{i=1}^{R} \frac{\overline{M}_{ir}}{m} \cdot \frac{1}{\mu_i}, \tag{6.103}$$

where $\overline{N}_{ir}$ and $\overline{M}_{ir}$ are given by:

$$\begin{aligned} \overline{N}_{ir} &= 0 \quad i < r, \\ \overline{M}_{ir} &= 0 \quad i \leq r, \end{aligned} \tag{6.104}$$

and, with Little's theorem:

$$\begin{aligned} \overline{N}_{ir} &= \lambda_i \overline{W}_i \quad i \geq r, \\ \overline{M}_{ir} &= \lambda_i \overline{W}_r \quad i > r. \end{aligned} \tag{6.105}$$

We can solve Eqs. (6.102) and (6.103) to obtain:

$$\overline{W}_r = \frac{\overline{W}_0}{(1-\sigma_r)(1-\sigma_{r+1})}, \tag{6.106}$$

where:

$$\sigma_r = \sum_{i=r}^{R} \rho_i. \tag{6.107}$$

The overall mean waiting time is:

$$\overline{W} = \sum_{i=1}^{R} \frac{\lambda_i}{\lambda} \cdot \overline{W}_i. \tag{6.108}$$

Values for $\overline{W}_0$ are given by Eqs. (6.48), (6.49), (6.77), and (6.78) and are the weighted sum of the $\overline{W}_{0i}$ of all the classes:

$$\overline{W}_{0,\mathrm{M/M/1}} = \sum_{i=1}^{R} \rho_i \frac{1}{\mu_i}, \tag{6.109}$$

$$\overline{W}_{0,\mathrm{M/G/1}} = \sum_{i=1}^{R} \rho_i \cdot \frac{1 + c_{B_i}^2}{2\mu_i}, \tag{6.110}$$

$$\overline{W}_{0,\mathrm{GI/G/1,AC}} \approx \sum_{i=1}^{R} \rho_i \cdot \frac{c_{A_i}^2 + c_{B_i}^2}{2\mu_i}, \tag{6.111}$$

$$\overline{W}_{0,\mathrm{GI/G/1,KLB}} \approx \sum_{i=1}^{R} \rho_i \cdot \frac{c_{A_i}^2 + c_{B_i}^2}{2\mu_i} \cdot G_{\mathrm{KLB}}, \tag{6.112}$$

$$\overline{W}_{0,\mathrm{GI/G/1,KUL}} \approx \sum_{i=1}^{R} \rho_i \cdot \frac{c_{A_i}^{f(c_{A_i},c_{B_i},\rho_i)} + c_{B_i}^2}{2\mu_i}, \tag{6.113}$$

$$\overline{W}_{0,\mathrm{M/M/m}} = \frac{P_m}{m\rho} \sum_{i=1}^{R} \rho_i \cdot \frac{1}{\mu_i}, \tag{6.114}$$

$$\overline{W}_{0,\mathrm{M/G/m}} \approx \frac{P_m}{2m\rho} \cdot \sum_{i=1}^{R} \rho_i \cdot \frac{1 + c_{B_i}^2}{\mu_i}, \tag{6.115}$$

$$\overline{W}_{0,\mathrm{GI/G/m,AC}} \approx \frac{P_m}{2m\rho} \cdot \sum_{i=1}^{R} \rho_i \cdot \frac{c_{A_i}^2 + c_{B_i}^2}{\mu_i}, \tag{6.116}$$

$$\overline{W}_{0,\mathrm{GI/G/m,KLB}} \approx \frac{P_m}{2m\rho} \sum_{i=1}^{R} \rho_i \cdot \frac{c_{A_i}^2 + c_{B_i}^2}{\mu_i} G_{\mathrm{KLB}}, \tag{6.117}$$

$$\overline{W}_{0,\mathrm{GI/G/m,KUL}} \approx \frac{P_m}{2m\rho} \sum_{i=1}^{R} \rho_i \cdot \frac{c_{A_i}^{f(c_{A_i},c_{B_i},\rho_i)} + c_{B_i}^2}{\mu_i}. \tag{6.118}$$

For $f(c_{A_i}, c_{B_i}, \rho_i)$, see Eq. (6.74); for $G_{\mathrm{KLB,GI/G/1}}$, see Eq. (6.72); and for $G_{\mathrm{KLB,GI/G/m}}$, see Eq. (6.92).

Also, the GI/G/m-FCFS Approximation of Cosmetatos can be extended to priority queues:

$$\overline{W}_{\mathrm{GI/G/m_r}} \approx c_{B_r}^2 \overline{W}_{\mathrm{GI/M/m_r}} + (1 - c_{B_r}^2)\overline{W}_{\mathrm{GI/D/m_r}}, \tag{6.119}$$

with:

$$\overline{W}_{\mathrm{GI/M/m_r}} = \frac{\overline{W}_{0,\mathrm{GI/M/m}}}{(1-\sigma_r)(1-\sigma_{r+1})},$$

$$\overline{W}_{\mathrm{GI/D/m_r}} = \frac{\overline{W}_{0,\mathrm{GI/D/m}}}{(1-\sigma_r)(1-\sigma_{r+1})},$$

$$\overline{W}_{0,\mathrm{GI/M/m}} \approx \frac{P_m}{2m\rho} \sum_{i=1}^{R} \rho_i \frac{c_{A_i}^{f(c_{A_i},1,\rho_i)} + 1}{\mu_i},$$

$$\overline{W}_{0,\mathrm{GI/D/m}} \approx \frac{P_m}{2m\rho} \sum_{i=1}^{R} \rho_i \frac{c_{A_i}^{f(c_{A_i},0,\rho_i)}}{\mu_i}.$$

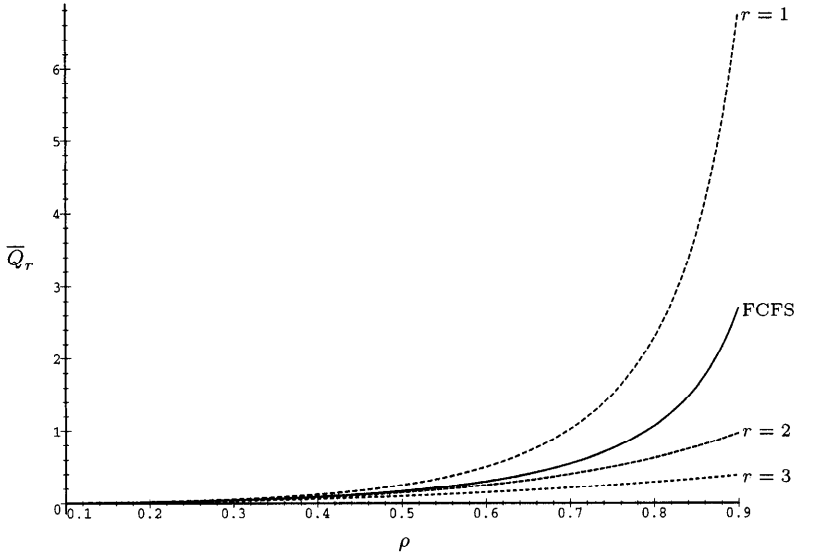


Fig. 6.14 Mean queue lengths $\overline{Q}_r$ for an M/M/1 priority system without preemption.

The M/G/m Cosmetatos approximation can similarly be extended. All GI/G/m approximations yield good results for the M/G/m–priority queues as well.

Figure 6.14 shows the mean queue length $\overline{Q}_r$ for different priority classes of a priority system without preemption and $R = 3$ priority classes together

with the mean queue length for the same system without priorities. It can be seen that the higher-priority jobs have a much lower queue length, and the lower-priority jobs have a much higher mean queue length than the FCFS system.

6.15.2 Conservation Laws

In priority queueing systems, the mean waiting time of jobs is dependent on their priority class. It is relatively short for jobs with high priority and considerably longer for jobs with low priority because there exists a fundamental conservation law ([Klei65, Schr70, HeSo84]):

$$\sum_{i=1}^{R} \rho_i \overline{W}_i = \frac{\rho \overline{W}_0}{1-\rho} = \rho \cdot \overline{W}_{\text{FCFS}} \tag{6.120}$$

or:

$$\frac{1}{\rho} \sum_{i=1}^{R} \rho_i \overline{W}_i = \frac{\overline{W}_0}{1-\rho} = \overline{W}_{\text{FCFS}}. \tag{6.121}$$

The conservation law to apply the following restrictions must be satisfied:

1. No service facility is idle as long as there are jobs in the queue, i.e., scheduling is work conserving [ReKo75].
2. No job leaves the system before its service is completed.
3. The distributions of the interarrival times and the service times are arbitrary with the restriction that the first moments of both the distributions and the second moment of the service time distribution exist.
4. The service times of the jobs are independent of the queueing discipline.
5. Preemption is allowed only when all jobs have the same exponential service time distribution and preemption is of the type preemptive resume.
6. For GI/G/m systems all classes have the same service times. This restriction is not necessary for GI/G/1 systems [HeSo84].

If for GI/G/m systems the service times of the classes differ, then the conservation law is an approximation [Jaek91].

6.15.3 System with Preemption

Here we consider the case that an arriving customer can preempt a customer of lower priority from service. The preempted customer will be continued later at the point where he was preempted (preemptive resume). Thus a customer

of class r is not influenced by customers of the classes $1, 2, \ldots, r-1$. Hence we need to consider only a system with the classes $r, r+1, \ldots, R$ to calculate the mean waiting time $\overline{W}_r$ of class r.

In the conservation law we can replace $\rho = \sum_{i=1}^{R} \rho_i$ by $\sigma_r = \sum_{i=r}^{R} \rho_i$ and $\overline{W}_{\text{FCFS}}$ by:

$$\overline{W}^r = \frac{\overline{W}_0^r}{1-\sigma_r}, \tag{6.122}$$

where $\overline{W}_0^r$ is the mean remaining service time for such a system and we obtain it by substitution of ρ by σ_r in the formulae. For example:

$$\overline{W}_{0_{\text{AC}}}^r = \frac{P_m^r}{2m\sigma_r} \cdot \sum_{i=r}^{R} \rho_i \frac{c_A^2 + c_B^2}{\mu_i}. \tag{6.123}$$

We sum only from r and R rather than from 1 to R. For P_m^r, we replace ρ by σ_r in the exact or approximative formula for P_m.

To obtain the mean waiting time of a customer of class r we apply the conservation law twice:

$$\begin{aligned} \sigma_r \cdot \overline{W}^r &= \sum_{i=r}^{R} \rho_i \overline{W}_i, \\ \sigma_{r+1} \cdot \overline{W}^{r+1} &= \sum_{i=r+1}^{R} \rho_i \cdot \overline{W}_i. \end{aligned} \tag{6.124}$$

By substitution we obtain:

$$\overline{W}_r = \frac{1}{\rho_r} \left(\sigma_r \overline{W}^r - \sigma_{r+1} \overline{W}^{r+1} \right). \tag{6.125}$$

Equation (6.125) is exact for M/M/1 and M/G/1 systems. It is also exact for M/M/m systems if the mean service time of all classes are the same. For other systems, Eq. (6.125) is a good approximation. From Fig. 6.15 it can be seen that even for heavy utilization the mean queue length for the highest priority class is negligible in the case of preemption, at the expense of poorer service for lower-priority classes.

6.15.4 System with Time-Dependent Priorities

Customers with low priorities have long waiting times especially when the utilization of the system is very high. Sometimes it is advantageous for a customer priority to increase with the time, so that he does not have to wait too long. This possibility can be considered if we do not have a fixed priority function:

$$q_r(t) = \text{Priority of class } r \text{ at time } t.$$

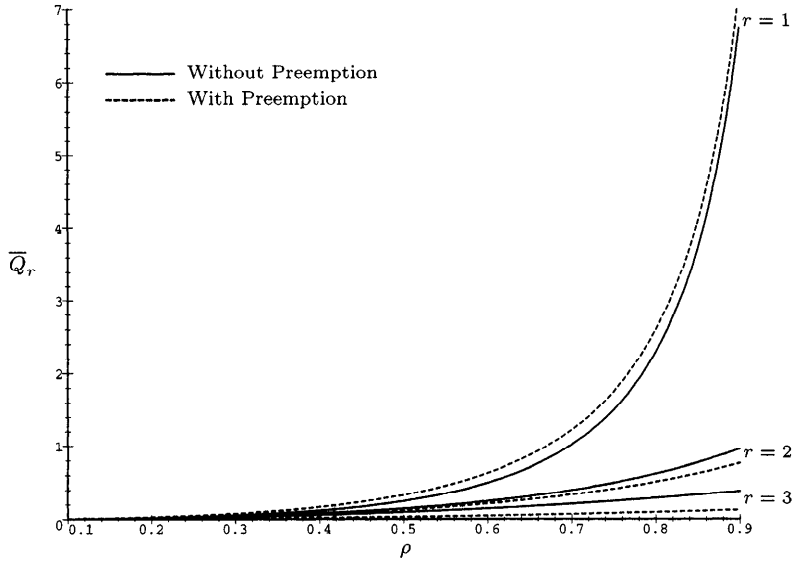


Fig. 6.15 Mean queue lengths for an M/M/1 priority system with and without preemption.

Such systems are more flexible but need more expense for the administration.

A popular and easy to handle priority function is the following:

$$q_r(t) = (t - t_0) \cdot b_r \tag{6.126}$$

(see Fig. 6.16*a*), with $0 \leq b_1 \leq b_2 \leq \ldots \leq b_r$. The customer enters the system at time t_0 and then increases the priority at the rate b_r. Customers of the same priority class have the same rate of increase b_r but different values of the arrival time t_0.

We only consider systems without preemption and provide the following recursive formula [BoBr84] for the mean waiting time of priority class-r customer:

$$\overline{W}_r = \frac{\dfrac{\overline{W}_0}{1-\rho} - \sum_{i=1}^{r-1} \rho_i \overline{W}_i \left(1 - \dfrac{b_i}{b_r}\right)}{1 - \sum_{i=r+1}^{R} \rho_i \left(1 - \dfrac{b_r}{b_i}\right)} \tag{6.127}$$

with the same mean remaining service times $\overline{W}_0$ as for the priority systems with fixed priorities. If these are exact, then Eq. (6.127) is also exact; otherwise it is an approximation formula.

Another important priority function is:

$$q_r(t) = r_r + t - t_0, \tag{6.128}$$

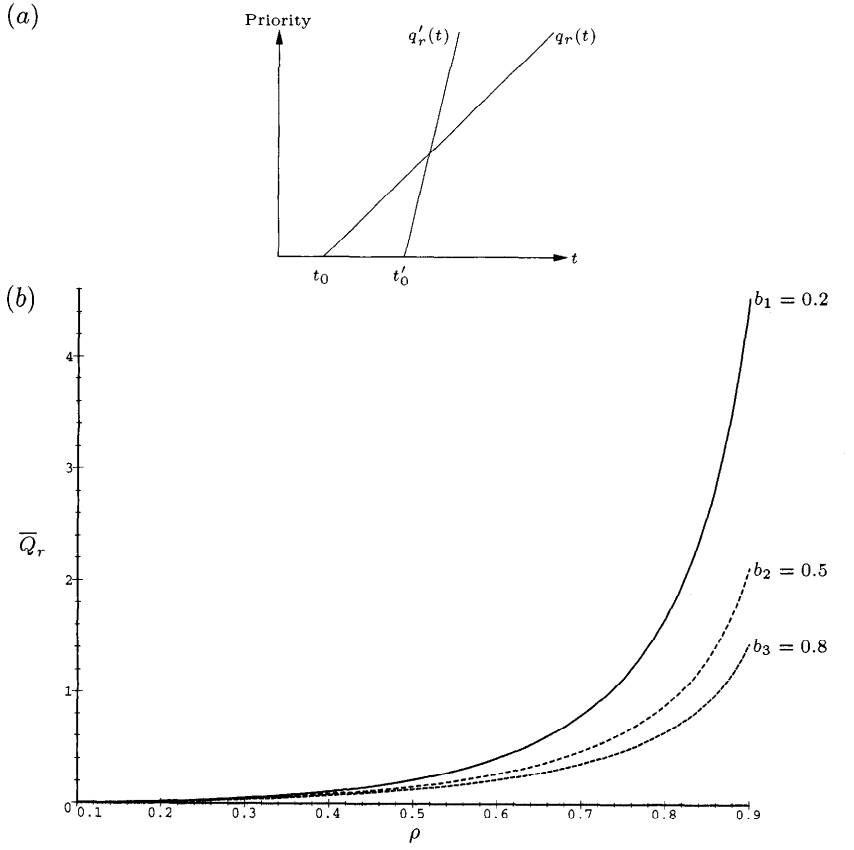


Fig. 6.16 (*a*) Priority function with slope b_r. (*b*) Mean queue length $\overline{Q}_r$ for an M/M/1 priority system with time-dependent priorities having slope b_r.

with $0 \leq r_1 \leq r_2 \leq \ldots \leq r_r$. Each priority class has a different starting priority r_r and a constant slope (see Fig. 6.17*a*). In [BoBr84] a heavy traffic approximation (as ρ approaches 1) is given:

$$\overline{W}_r \approx \frac{\overline{W}_0}{1-\rho} - P_m \cdot \sum_{i=1}^{R} \rho_i (r_r - r_i), \tag{6.129}$$

as well as a more accurate recursive formula:

$$\overline{W}_r \approx \frac{\overline{W}_0}{1-\rho} - \sum_{i=1}^{r-1} \rho_i \overline{W}_i \left(1 - \exp\left(\frac{P_m(r_i - r_r)}{\overline{W}_i}\right)\right). \tag{6.130}$$

Note that for $m = 1$, we have $P_m = \rho$.

From Figs. 6.16*b* and 6.17*b* it can be seen that systems with time-dependent priorities are more flexible than systems with static priorities. If the values of

b_r or r_r, respectively, are close together, then the systems behave like FCFS systems, and if they are very different, then they behave more like systems with static priorities.

In many real time systems, a job has to be serviced within an *upper time limit.* To achieve such a behavior, it is advantageous to use a priority function that increases from 0 to ∞ between the arrival time and the upper time limit u_r. A convenient priority function is:

$$q_r(t) = \begin{cases} (t - t_0)/(u_r - t + t_0) & t_0 < t \leq u_r + t_0, \\ \infty & u_r + t_0 \leq t. \end{cases} \tag{6.131}$$

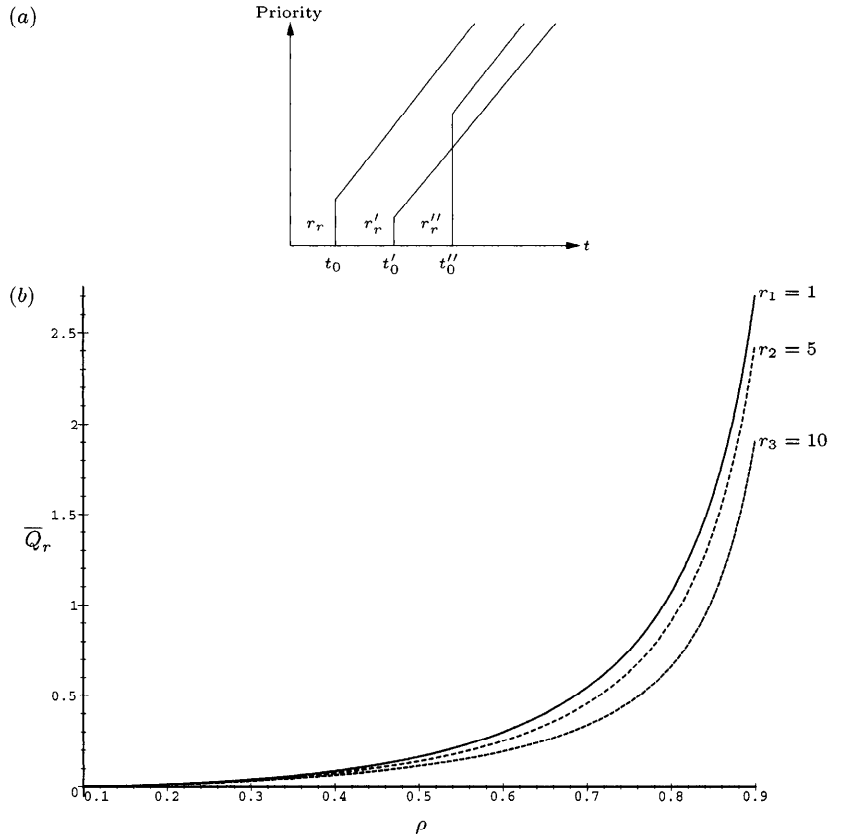


Fig. 6.17 (*a*) Priority function with starting priority r_r. (*b*) Mean queue lengths $\overline{Q}_r$ for an M/M/1 priority system with time-dependent priorities having starting priorities r_r.

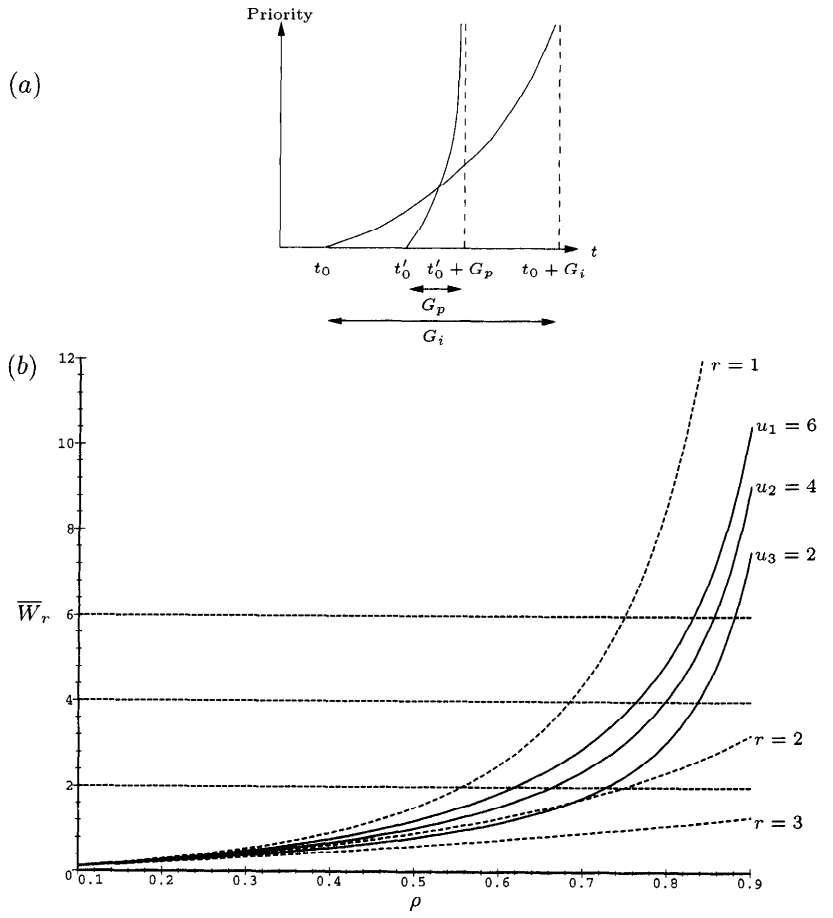


Fig. 6.18 (*a*) Priority function with upper time limit u_r. (*b*) Mean waiting time $\overline{W}_r$ for an M/M/1 priority system with static priorities and with upper time limits $u_r = 2, 4, 6$.

Again we get a heavy traffic approximation (as ρ approaches 1):

$$\overline{W}_r \approx \left(\frac{\overline{W}_0}{1-\rho} - P_m \sum_{i=1}^{r-1} \rho_i (u_i - u_r) \right) \left(1 - (1 - P_m) \sum_{i=r+1}^{R} \rho_i \left(1 - \frac{u_i}{u_r} \right) \right)^{-1} \tag{6.132}$$

and a recursive formula for more accurate results:

$$\overline{W}_r \approx \left(\frac{\overline{W}_0}{1-\rho} - \sum_{i=1}^{r-1} \rho_i \cdot \overline{W}_i \left(1 - \frac{u_r}{u_i}\right) \left(1 - P_m \exp\left(-\frac{\rho u_i}{\overline{\overline{W}}_i}\right)\right)\right) \cdot \left(1 - \sum_{i=r+1}^{R} \rho_i \left(1 - \frac{u_i}{u_r}\right) \left(1 - P_m \cdot \exp\left(-\frac{\rho u_r}{\overline{\overline{W}}_r}\right)\right)\right)^{-1}. \tag{6.133}$$

Also see [BoBr84] and [Jaek91] for other priority functions.

In Fig. 6.18*b*, the mean waiting times for a system with static priorities are compared to those of a priority system with upper time limits. The figure also contains the upper time limits. For the two higher priorities, the static priority system is better, but the upper time limit system is better over all priorities.

6.16 THE ASYMMETRIC SYSTEM

The calculation of the performance measures is fairly simple if there is only one server or several servers with identical service times arranged in parallel, and the interarrival time and service time are exponentially distributed (M/M/m queueing systems). However, heterogeneous multiple servers often occur in practice and here the servers have different service times. This situation occurs, for example, when machines of different age or manufacturer are running in parallel. It is therefore useful to be able to calculate the performance measures of such systems. One method for solving this problem is given in [BoSc91] in which the formulae derived provide results whose relative deviation from values obtained by simulation are quite acceptable. The heterogeneous multiple server is treated in more detail by [BFH88] and [Baer85] who achieve exact results. Also see [Triv82] and [GeTr83] for exact results.

We will assume throughout this section that the arrival process is Poisson with rate λ and that service times are exponentially distributed with rate μ_i on the ith server.

6.16.1 Approximate Analysis

In homogeneous (symmetric) queueing systems, the response time depends only on whether an arriving job finds some free server. A heterogeneous (asymmetric) multiple server system differs since which of the m servers processes the job is now important. The analysis in [BoSc91] is based on the following consideration: Let p_k $(k = 1, \ldots, m)$ be the probability that a job is assigned to the kth server. Because it can be assumed that a faster server processes more jobs than a slower server, it follows that the ratios of the probabilities p_k to each other are the same as the ratios of the service rates to each other. Here no distinction is made between different strategies for selecting a

free server. This approximation is justified by the fact that in a system with heavy traffic there is usually one free server and therefore a server selection strategy is not needed. Then p_k $(1 \leq k \leq m)$ is approximately given by:

$$p_k \approx \frac{\mu_k}{\sum_{i=1}^{m} \mu_i}. \tag{6.134}$$

In addition, p_k can be used to determine the utilization ρ_k of the kth server:

$$\rho_k \approx \frac{\lambda}{\mu_k} \cdot p_k = \frac{\lambda}{\mu_k} \cdot \frac{\mu_k}{\sum_{i=1}^{m} \mu_i} = \frac{\lambda}{\sum_{i=1}^{m} \mu_i}. \tag{6.135}$$

Thus the utilization of all the servers is the same and the overall utilization is given by:

$$\rho = \rho_k. \tag{6.136}$$

In [BoSc91] it is shown that the known formulae for symmetric M/M/m and GI/G/m systems can be applied to asymmetric M/M/m and GI/G/m systems if Eqs. (6.135) and (6.136) are used to calculate the utilization ρ. Good approximations to the performance measures are obtained provided the values of the service rates do not differ too much, i.e, $\mu_{\min}/\mu_{\max} \leq 10$ (see Table 6.3).

Table 6.3 Approximative and simulative results for (*a*) M/M/5 and (*b*) M/E_2/5 queueing systems

	λ	μ_k	$\rho(A)$	$\rho(S_1)$	$\rho(S_2)$	$\overline{Q}(A)$	$\overline{Q}(S_1)$	$\overline{Q}(S_2)$
(*a*)	73	10,15,20,20,25	0.811	0.799	0.819	2.472	2.367	2.495
	73	16,17,18,19,20	0.811	0.807	0.812	2.472	2.476	2.500
	73	5,10,18,22,35	0.811	0.808	0.844	2.472	2.443	2.626
	81.11	4,8,16,32,40	0.811	0.826	0.858	2.472	2.549	2.713
	81.11	8,9,20,31,32	0.811	0.807	0.839	2.472	2.456	2.606
	81.11	8,14,20,26,32	0.811	0.801	0.830	2.472	2.442	2.569
	81.11	10,15,20,25,30	0.811	0.799	0.824	2.472	2.425	2.523
(*b*)	81.11	10,15,20,25,30	0.811	0.800	0.825	1.854	1.854	1.962

Notes: A = approximation, S_1 = discrete-event simulation with random selection of a free server, and S_2 = discrete-event simulation with selection of the fastest free server.

6.16.2 Exact Analysis

This section describes the methods used for calculating the characteristics of asymmetric queueing systems described in [BFH88] and [Baer85]. These systems are treated as a CTMC, and the performance measures are calculated

by solving the underlying CTMCs. In this way exact results are obtained for elementary asymmetric M/M/m queueing systems. This method is used as follows: First, the steady-state probabilities are calculated for systems without queues, known as loss systems. Then the formulae given in [BFH88] and [Baer85] can be used to extend the results to asymmetric, non-lossy systems.

Two strategies for selecting a free server are investigated:

1. The job is assigned randomly to a free server as in [BFH88].
2. The fastest free server is selected as in [Baer85].

6.16.2.1 Analysis of M/M/m Loss Systems A loss system contains no queues. The name indicates that jobs that cannot find a free server are rejected and disappear from the system. A CTMC is constructed and solved to obtain a closed-form formula for its steady-state probabilities. It should be noted that asymmetric queueing systems differ from symmetric queueing systems in that it is not sufficient when producing the CTMC to describe the states by the number of servers occupied. Due to the different service rates, the state classification depends on which of the m servers are currently occupied. Thus each state is characterized by a set $g = (k_1, k_2, \ldots, k_i)$, where k_l belongs to g if the lth server is currently occupied. Here $i = |g|$ is the number of servers currently occupied ($1 \leq i \leq m$) and l is an index ($1 \leq l \leq m$).

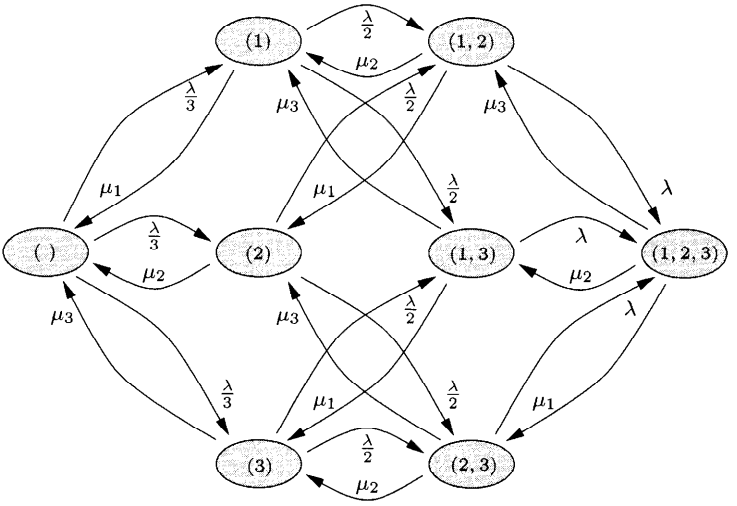


Fig. 6.19 CTMC for an asymmetric M/M/3 loss system with random selection of a free server.

Random Selection of a Free Server An M/M/3 system is used as an example. The state transition diagram of the loss system is shown in Fig. 6.19. The

steady-state probabilities are given by [BFH88]:

$$\pi_g = \frac{(m - |g|)!}{m!} \cdot \prod_{k \in g} \frac{\lambda}{\mu_k} \cdot \pi_\emptyset, \tag{6.137}$$

for all $g \subseteq G$ with $g \neq \emptyset$ and $G = \{1, 2, \ldots, m\}$. In order to determine π_0, we use the normalization condition:

$$\sum_{g \subseteq G} \pi_g = 1 \qquad \text{with} \qquad G = \{1, 2, \ldots, m\}. \tag{6.138}$$

The probability of loss $\pi^{(L)}$ is the probability that all servers are occupied:

$$\pi_m^{(L)} = \pi_{\{1,2,\ldots,m\}} = \pi_G. \tag{6.139}$$

The utilization ρ_k of the kth server is equal to the sum of the state probabilities of all states in which the kth server is occupied:

$$\rho_k^L = \sum_{g:\ k \in g} \pi_g. \tag{6.140}$$

Choice of the Fastest Free Server. Here it is always assumed that the individual servers are sorted in descending order of the service rates, i.e., the fastest server has the lowest index. Let $\boldsymbol{\mu}^m = (\mu_1, \mu_2, \cdots, \mu_m)$ and let $(\boldsymbol{\mu}^{k-1}, \mu_m) = (\mu_1, \mu_2, \cdots, \mu_{k-1}, \mu_m)$. The probability of loss $\pi_m^{(L)} = \pi_{\{1,\ldots,m\}}(\boldsymbol{\mu}^m)$ is given by [Baer85]:

$$\begin{aligned} \pi_m^{(L)} &= \pi_{\{1,\ldots,m\}}(\boldsymbol{\mu}^m) = B_m(\boldsymbol{\mu}^m) \\ &= B_{m-1}(\boldsymbol{\mu}^{m-1}) \cdot \left[1 + \frac{\mu_m}{\lambda} \cdot \prod_{k=1}^{m-1} \frac{B_k(\boldsymbol{\mu}^{k-1}, \mu_m)}{B_k(\boldsymbol{\mu}^{k-1}, \mu_k + \mu_m)}\right]^{-1}, \end{aligned} \tag{6.141}$$

with:

$$B_1(\boldsymbol{\mu}^1) = \pi_{\{1\}}(\mu_1) = \frac{\lambda}{\lambda + \mu_1}.$$

The utilization ρ_k of the individual servers is given by:

$$\rho_k = \frac{\lambda}{\mu_k}[B_{k-1}(\boldsymbol{\mu}^{k-1}) - B_k(\boldsymbol{\mu}^k)] \qquad \text{with} \qquad B_0(\boldsymbol{\mu}^0) = 1. \tag{6.142}$$

6.16.2.2 Extending to Non-Lossy System According to [BFH88], the next step is to use the results obtained so far to draw conclusions about the behavior of the corresponding non-lossy system. Here the method used to select a free server of the corresponding loss system does not matter. It is only necessary

that the values appropriate to the selected strategy are used in the formulae finally obtained. In what follows, a superscript W means that the variable is a characteristic of a non-lossy system; a superscript L means that the variable is a characteristic of the corresponding loss system. The steady-state probabilities are given by [BFH88]:

$$\pi_i^{(W)} = \left(\sum_{k=1}^{m} \frac{\mu_k}{\lambda} \right)^{m-i} \cdot \pi_m^{(W)} \qquad \text{for} \qquad i > m, \tag{6.143}$$

with:

$$\pi_m^{(W)} = \frac{\pi_m^{(L)}}{N}, \tag{6.144}$$

where:

$$N = 1 + \frac{\pi_m^{(L)} \cdot c}{1-c} \qquad \text{with} \qquad c = \frac{\lambda}{\sum_{k=1}^{m} \mu_k}. \tag{6.145}$$

It is now possible to calculate all the state probabilities of the asymmetric non-lossy system and all interesting performance measures. It should be noted that c is not the same as the utilization ρ. The method for calculating ρ is described in the following.

The probability P_m that an arriving job is queued is equal to the sum of the probabilities of all states in which there are more than m jobs in the system:

$$P_m = \sum_{i=m}^{\infty} \pi_i^{(W)} = \frac{1}{1-c} \cdot \pi_m^{(W)} = \frac{1}{1-c} \cdot \frac{\pi_m^{(L)}}{N}. \tag{6.146}$$

Analogous to the symmetric case, the mean queue length $\overline{Q}$ is given by:

$$\overline{Q} = \sum_{i=m+1}^{\infty} (i-m) \cdot \pi_i^{(W)} = \frac{P_m \cdot c}{1-c}. \tag{6.147}$$

The mean waiting time can be calculated using Little's theorem:

$$\overline{W} = \frac{\overline{Q}}{\lambda}. \tag{6.148}$$

To calculate the utilization ρ it is first necessary to calculate the specific utilization ρ_k for each server:

$$\rho_k^{(W)} = \frac{\rho_k^{(L)}}{N} + \sum_{i=m+1}^{\infty} \pi_i^{(W)} = \frac{\rho_k^{(L)} + \pi_m^{(L)} \cdot c/(1-c)}{N}. \tag{6.149}$$

Then ρ is given by:

$$\rho = \frac{m}{\sum_{k=1}^{m} \rho_k^{-1}}. \tag{6.150}$$

6.16.3 Exact Analysis of an Asymmetric M/M/2 System

In [Triv82] an exact solution is given for an asymmetric M/M/2 system with the strategy of selecting the fastest free server. Figure 6.20 shows the CTMC state transition diagram for this system. The state of the system is defined

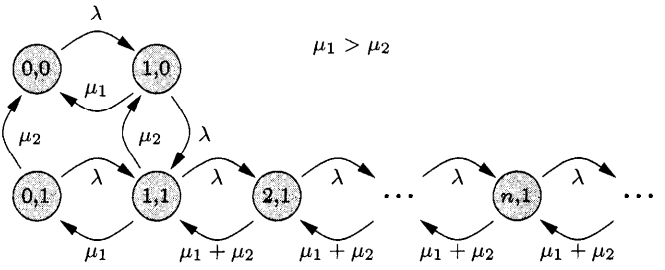


Fig. 6.20 CTMC for the M/M/2 asymmetric system.

to be the tuple (k_1, k_2) where $k_1 \geq 0$ denotes the number of jobs in the queue including any at the faster server, and $k_2 \in \{0, 1\}$ denotes the number of jobs at the slower server.

If we use Eq. (2.58) for the underlying CTMC, we can calculate the steady-state probabilities easily with:

$$c = \frac{\lambda}{\mu_1 + \mu_2} \tag{6.151}$$

$$\pi(k, 1) = c\pi(k-1, 1) = c^{k-1}\pi(1, 1), \qquad \text{for} \qquad k > 1. \tag{6.152}$$

Furthermore:

$$\begin{aligned} \pi(0,1) &= \frac{c}{1+2c}\frac{\lambda}{\mu_2}\pi(0,0), \\ \pi(1,0) &= \frac{1+c}{1+2c}\frac{\lambda}{\mu_1}\pi(0,0), \\ \pi(1,1) &= \frac{c}{1+2c}\frac{\lambda(\lambda+\mu_2)}{\mu_1\mu_2}\pi(0,0). \end{aligned} \tag{6.153}$$

Finally, using the normalization condition, we get:

$$\pi(0,0) = [1 + \frac{\lambda(\lambda+\mu_2)}{\mu_1\mu_2(1+2c)(1-c)}]^{-1}. \tag{6.154}$$

And with these, the utilizations of the two servers:

$$\begin{aligned} \rho_1 &= 1 - \pi(0,0) - \pi(0,1), \\ \rho_2 &= 1 - \pi(0,0) - \pi(1,0), \end{aligned} \tag{6.155}$$

and the mean number of jobs in system:

$$\overline{K} = \frac{1}{A(1-c)^2}, \tag{6.156}$$

where:

$$A = \left[\frac{\mu_1\mu_2(1+2c)}{\lambda(\lambda+\mu_2)} + \frac{1}{1-c}\right].$$

Example 6.1 To show how to use the introduced formulae for asymmetric M/M/m systems, we apply them to a simple M/M/2 example with $\lambda = 0.2$, $\mu_1 = 0.5$, and $\mu_2 = 0.25$.

Approximation [BoSc91] With Eq. (6.136) for the utilization:

$$\rho = \rho_1 = \rho_2 = \frac{\lambda}{\mu_1+\mu_2} = \frac{0.2}{0.5+0.25} = \underline{0.267},$$

with Eq. (6.29) for symmetric M/M/2 systems for mean number in system:

$$\overline{K} = 2\rho + \frac{2\rho^3}{1-\rho^2} = \underline{0.575},$$

and with Little's theorem (Eq. (6.9)):

$$\overline{T} = \underline{2.875}.$$

Exact [Triv82] With Eq. (6.151):

$$c = \underline{0.267},$$

with Eqs. (6.153), (6.154), and (6.155) for the utilizations of the servers:

$$\begin{aligned}
\pi(0,0) &= \left[1 + \frac{0.2(0.2+0.25)}{0.25\cdot 0.5(1+2c)(1-c)}\right]^{-1} \\
&= \underline{0.6096}, \\
\pi(0,1) &= \frac{0.267}{1+0.534}\cdot\frac{0.2}{0.25}\cdot 0.6056 = \underline{0.085}, \\
\pi(1,0) &= \frac{1.267}{1.534}\cdot\frac{0.2}{0.5}\cdot 0.6056 = \underline{0.2014}, \\
\rho_1 &= 1 - 0.6096 + 0.085 = \underline{0.305}, \\
\rho_2 &= 1 - 0.6096 + 0.2014 = \underline{0.189},
\end{aligned}$$

and with Eq. (6.156):

$$A = \frac{0.5 \cdot 0.25(1 + 0.534)}{0.2(0.2 + 0.25)} + \frac{1}{0.733} = \underline{3.495},$$

$$\overline{K} = \frac{1}{3.495 + 0.733^2} = \underline{0.533},$$

$$\overline{T} = \frac{\overline{K}}{\lambda} = \underline{2.663}.$$

Exact [BFH88] First we consider the asymmetric M/M/2 loss system with random selection of a free server: With Eq. (6.156) for the state probabilities:

$$\pi_1 = \frac{(2-1)!}{2!} \cdot \frac{\lambda}{\mu_1} \cdot \pi_0 = \frac{1}{2} \cdot \frac{0.2}{0.5} \cdot \pi_0 = 0.2\pi_0,$$

$$\pi_2 = \frac{(2-1)!}{2!} \cdot \frac{\lambda}{\mu_2} \cdot \pi_0 = \frac{1}{2} \cdot \frac{0.2}{0.25} \cdot \pi_0 = 0.4\pi_0,$$

$$\pi_{1,2} = \frac{(2-2)!}{2!} \cdot \frac{\lambda}{\mu_1} \cdot \frac{\lambda}{\mu_2} \cdot \pi_0 = 0.16\pi_0.$$

With the normalizing condition:

$$(1 + 0.2 + 0.4 + 0.16)\pi_0 = 1, \qquad \pi_0 = 0.568.$$

Now the probability of loss can be determined (Eq. (6.139)):

$$\pi_2^{(L)} = \pi_{1,2} = 0.16 \cdot 0.568 = \underline{0.091},$$

and also the utilizations of the servers (Eq. (6.140)):

$$\rho_1^L = \pi_1 + \pi_{1,2} = (0.2 + 1.6) \cdot 0.568 = \underline{0.204},$$

$$\rho_2^L = \pi_2 + \pi_{1,2} = (0.4 + 1.6) \cdot 0.568 = \underline{0.319}.$$

This result means that the slower server has a higher utilization as expected. Now we can use these results of the loss system to obtain performance measures for the non-lossy system. For c and N we get with Eq. (6.145):

$$c = \frac{0.2}{0.5 + 0.25} = 0.267, \qquad N = 1 + \frac{0.091 \cdot 0.267}{0.733} = \underline{1.033},$$

and for the probability of waiting (Eq. (6.146)):

$$P_2 = \frac{1}{0.733} \cdot \frac{0.091}{1.033} = \underline{0.120}.$$

The mean queue length and the mean waiting time (Eqs. (6.147), (6.148)):

$$\overline{Q} = \frac{0.120 \cdot 0.267}{0.733} = \underline{0.0437}, \qquad \overline{W} = \underline{0.219}.$$

Utilization of the servers (Eq. (6.149)):

$$\rho_1^{(W)} = \frac{0.204 + 0.091 \cdot 0.267/0.733}{1.033} = \underline{0.230},$$

$$\rho_2^{(W)} = \frac{0.319 + 0.091 \cdot 0.267/0.733}{1.033} = \underline{0.341}.$$

Utilization of the system (Eq. (6.150)):

$$\rho = \frac{2}{\frac{1}{0.230} + \frac{1}{0.341}} = \underline{0.275}.$$

Mean number of jobs in system:

$$\overline{K} = \rho_1 + \rho_2 + \overline{Q} = 0.230 + 0.341 + 0.0437 = \underline{0.6147}.$$

Mean response time:

$$\overline{T} = \frac{\overline{K}}{\lambda} = \underline{3.074}.$$

Exact [Baer85] The last case we consider is the asymmetric M/M/2 system with selection of the fastest free server using the formulae in [Baer85]. First we again consider the loss system and determine the probability of loss (Eq. (6.141)):

$$\begin{aligned}
\pi_2^{(L)} = B_2(\boldsymbol{\mu}^2) &= B_1(\boldsymbol{\mu}^1)\left[1 + \frac{\mu_2}{\lambda} \cdot \frac{B_1(\mu_2)}{B_1(\mu_1+\mu_2)}\right]^{-1} \\
&= \frac{\lambda}{\lambda+\mu_1} \cdot \left[1 + \frac{\mu_2}{\lambda} \cdot \frac{\frac{\lambda}{\lambda+\mu_2}}{\frac{\lambda}{\lambda+\mu_1+\mu_2}}\right]^{-1} \\
&= \frac{\lambda}{\lambda+\mu_1} \cdot \left[1 + \frac{0.25}{0.2} \cdot \frac{0.2+0.25+0.5}{0.2+0.25}\right]^{-1} \\
&= \underline{0.0785},
\end{aligned}$$

and the utilizations of the servers (Eq. (6.142)):

$$\rho_1^L = \frac{\lambda}{\mu_1}[B_0(\boldsymbol{\mu}^0) - B_1(\boldsymbol{\mu}^1)] = \frac{0.2}{0.5}\left[1 - \frac{0.2}{0.2+0.5}\right] = \underline{0.286},$$

$$\rho_2^L = \frac{0.2}{0.25}\left[B_1(\boldsymbol{\mu}^1) - B_2(\boldsymbol{\mu}^2)\right] = \frac{0.2}{0.25}\left[\frac{0.2}{0.2+0.5} - 0.0785\right] = \underline{0.166}.$$

Again we use the results of the loss system to obtain performance measures for the queueing system with selection of the fastest server. From Eq. (6.145)):

$$c = \underline{0.267}, \qquad N = 1 + \frac{0.0785 \cdot 0.267}{0.733} = \underline{1.0286}.$$

Probability of waiting (Eq. (6.146)):

$$P_2 = \frac{1}{0.733} \cdot \frac{0.0785}{1.0280} = \underline{0.104}.$$

Mean queue length (Eq. (6.147)):

$$\overline{Q} = \frac{0.104 \cdot 0.267}{0.733} = \underline{0.0379}.$$

Mean waiting time:

$$\overline{W} = \frac{\overline{Q}}{\lambda} = \underline{0.190}.$$

Utilizations of the servers (Eq. (6.149)):

$$\rho_1^{(W)} = \frac{0.286 + 0.0785 \cdot 0.267/0.733}{1.0286} = \underline{0.306},$$

$$\rho_2^{(W)} = \frac{0.166 + 0.0785 \cdot 0.267/0.733}{1.0286} = \underline{0.189}.$$

Utilization of the system (Eq. (6.150)):

$$\rho = \frac{2}{\frac{1}{0.306} + \frac{1}{0.189}} = \underline{0.234}.$$

Mean number of jobs in the system:

$$\overline{K} = \rho_1 + \rho_2 + \overline{Q} = 0.306 + 0.189 + 0.0379 = \underline{0.533}.$$

Mean system time:

$$\overline{T} = \frac{\overline{K}}{\lambda} = \underline{2.665}.$$

In Table 6.4, results of this example are summarized. In Table 6.5, the

Table 6.4 Results for several performance measures with different solution techniques and strategies for a asymmetric M/M/2 system ($\lambda = 0.2, \mu_1 = 0.5, \mu_2 = 0.25$)

Strategy Reference	Appr. [BoSc91]	FFS [Triv82]	FFS [Baer85]	Random [BFH88]
ρ	0.267	(0.267)	0.234	0.275
ρ_1	0.267	0.305	0.306	0.230
ρ_2	0.267	0.189	0.189	0.341
$\overline{K}$	0.575	0.533	0.533	0.615
$\overline{T}$	2.875	2.663	2.665	3.074

Notes: FFS = selection of the fastest free server and Random = random selection of the free server.

mean response time $\overline{T}$ for an asymmetric M/M/2 system is compared with

Table 6.5 Mean response times $\overline{T}, \overline{T}_1$, and $\overline{T}_2$ for an asymmetric M/M/2 system with strategy FFS and two M/M/1 systems ($\lambda = 0.2, \mu_2 = 0.25, \mu_1 = \alpha \cdot \mu_2$)

α	1	2	3	4	5
$\overline{T}_1$	20	3.33	1.818	1.25	0.55
$\overline{T}_2$	20	20	20	20	20
$\overline{T}$	4.762	2.662	1.875	1.459	1.20

the mean response times $\overline{T}_1$ (for a M/M/1 system with service rate μ_1) and $\overline{T}_2$ (for a M/M/1 system with service rate μ_2) [Triv82]. From Table 6.5 we see that sometimes it is better to disconnect the slower server if we wish to minimize the mean response time. This is the case only for low utilization of the servers and if the service rates differ considerably. This issue is further explored in [GeTr83].

6.17 SYSTEMS WITH BATCH SERVICE

An interesting and important way to service customers in queueing systems is the batch service. In such batch systems the service of all the customers of a batch is started at the same time. There are many applications of this kind of service, especially in manufacturing systems. Two policies are in use. In the case of a full batch policy (FB), the service of the batch is started when all b customers of the batch have arrived. In the case of the minimum batch policy (MB), a minimum number of a customers is sufficient to start the service of the batch. If there are more then b waiting customers, only b customers are collected to a batch and serviced together.

The extended Kendall notation for batch systems is:

$\mathrm{GI/G^{[a,b]}/m}$, Multiserver system with MB policy
$\mathrm{GI/G^{[b,b]}/m}$, Multiserver system with FB policy

A special case of the MB policy is the GREEDY policy where the service begins when $a = 1$ customer is in the queue. To obtain a formula for the mean waiting time and the mean queue length for the $\mathrm{GI/G^{[b,b]}/m}$ queueing system, we use the already introduced formulae for GI/G/m queueing systems and calculate the mean queue length $\hat{Q}$ for the batches. From the result we can calculate the mean queue length $\overline{Q}_{\text{batch}}$ for the individual customers. If the arrival rate of the individual customers is λ, then the arrival rate of the batches with batch size b is:

$$\hat{\lambda} = \frac{\lambda}{b}, \tag{6.157}$$

and the coefficient of variation for the batch interarrival time is:

$$\hat{c}_A^2 = \frac{c_A^2}{b} \tag{6.158}$$

using Eq. (1.52) with the coefficient of variation of the individual customers c_A. Using the queue length $\hat{Q}$ of the batches, the queue length of the individual customers is approximately given by [HaSp95, Kirs91]:

$$\overline{Q}_{\text{batch}} \approx b \cdot \hat{Q} + \frac{b-1}{2}. \tag{6.159}$$

The first part of the formula is due to the customers in the $\hat{Q}$ batches of the queue, and the second part is the mean number of customers who are still not combined to a batch. The accuracy of Eq. (6.159) depends mainly on the accuracy of the applied approximation formula for calculating $\hat{Q}$.

A solution for $\text{GI/G}^{[a,b]}\text{/m}$ queueing systems is given by a heuristic extension of Eq. (6.159) [Kirs91]:

$$\overline{Q}_{\text{batch}}^{[a,b]} \approx b \cdot \hat{Q} + P_m \cdot \frac{b-1}{2} + (1 - P_m) \cdot \frac{a-1}{2} \tag{6.160}$$

Eq. (6.160) includes Eq. (6.159) if $a = b$. For the probability of waiting P_m, we use the corresponding value for the M/M/m queueing system.

In the special case of an $\text{M/M}^{[b,b]}\text{/1}$ system, we obtain an $\text{E}_b\text{/M/1}$ queueing system for the calculation of $\hat{Q}$ since in this case the interarrival time of the batch is the sum of b exponentially distributed interarrival times. In the case of an $\text{M/M}^{[b,b]}\text{/m}$ system, we obtain an $\text{E}_b\text{/M/m}$ queueing system for which exact formulae (see Eqs. (6.59) and (6.64) respectively) are known.

From Fig. 6.21*a* it can be seen how the mean queue length $\overline{Q}_{\text{batch}}$ for individual jobs depends on the batch size b in a full batch system. Figure 6.21*b* shows that for increasing values of ρ, the mean queue length of a minimum batch system approaches that of a full batch system with the same batch size b.

Figure 6.22 shows that the queue length $\overline{Q}_{\text{batch}}$ of an individual job in a minimal batch system depends more on the batch size b than on the minimum batch size a, as also can be seen from Eq. (6.160). Moreover we realize that a has a noticeable influence only on $\overline{Q}_{\text{batch}}^{[a,b]}$ for small values of ρ.

Problem 6.1 Consider two M/M/1 systems with arrival rate $\lambda_1 = 0.5/\text{sec}$ and service rate $\mu_1 = 1/\text{sec}$ each and an M/M/2 system with arrival rate $\lambda_2 = 2\lambda_1 = 1/\text{sec}$ and service rate $\mu_2 = \mu_1 = 1/\text{sec}$ for each server. Compare utilization ρ_i, mean number of jobs in the system $\overline{K}_i$, mean waiting time $\overline{W}_i$, and mean response time $\overline{T}_i$ for both cases $(i = 1, 2)$. Discuss the results!

Problem 6.2 Consider an M/M/1/K system and determine the throughput and the mean number of jobs $\overline{K}$ as a function of the maximum number

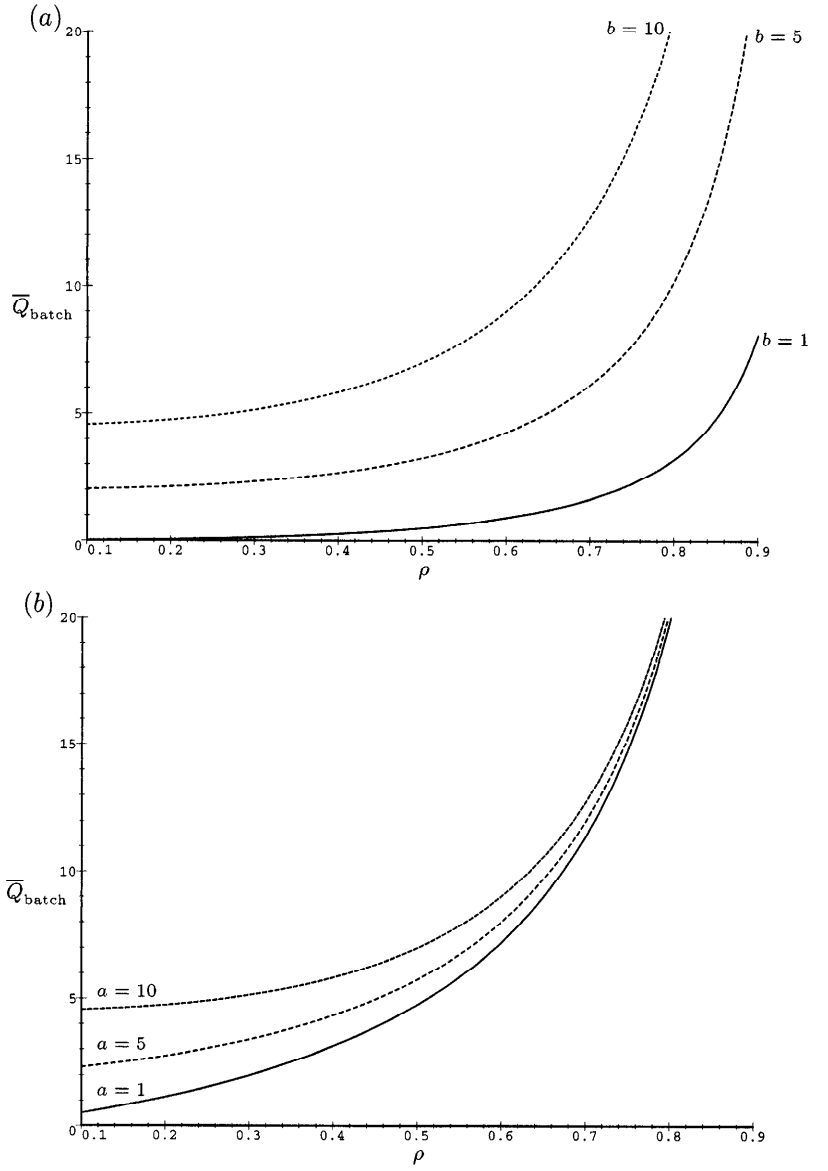


Fig. 6.21 (*a*) Mean queue length $\overline{Q}_{\text{batch}}$ for a full batch system $M/M^{[b,b]}/1\text{-FCFS}$ and (*b*) $\overline{Q}_{\text{batch}}^{[a,10]}$ for a minimum batch system.

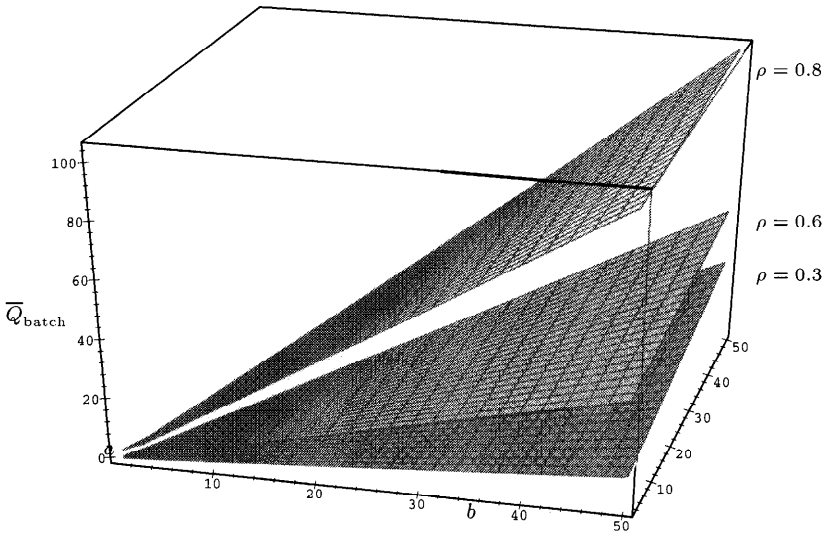


Fig. 6.22 Mean queue length $\overline{Q}_{\text{batch}}^{[a,b]}$ of a minimum batch system.

of jobs in the system K for the cases $\lambda = \mu/2$, $\lambda = \mu$, and $\lambda = 2\mu$ with $\mu = 1/\text{sec}$. Discuss the results.

Problem 6.3 Compare the mean number of jobs $\overline{K}$ of an M/D/1, an M/M/1, and an M/G/1 system ($c_B^2 = 2$) with $\lambda = 0.8/\text{sec}$ and $\mu = 1/\text{sec}$. Give a comprehensible explanation why the values are different although the arrival rate λ and the service rate μ are always the same.

Problem 6.4 Compare the mean number of jobs $\overline{K}$ of an M/G/1 and a GI/M/1 system with $\lambda = 0.9/\text{sec}$, $\mu = 1/\text{sec}$, $c_A^2 = 0.5$, and $c_B^2 = 0.5$, respectively. Discuss the results.

Problem 6.5 Consider a GI/G/1 system with $\rho = 0.8$, $c_A^2 = 0.5$, and $c_B^2 = 2$. Compute the mean queue length $\overline{Q}$ using the Allen-Cunneen, the Krämer/Langenbach-Belz, and the Kimura approximation. Also compute upper and lower bounds for the mean queue length (see Section 6.12).

Problem 6.6 Consider an M/M/1 priority system with 2 priority classes, and $\mu_1 = \mu_2 = 1/\text{sec}$ and $\lambda_1 = \lambda_2 = 0.4/\text{sec}$. Compute the mean waiting time $\overline{W}$ for the two classes for a system with and without preemption. Discuss the results and compare them to the corresponding M/M/1-FCFS system.

Problem 6.7 Consider an asymmetric M/M/2 system with $\lambda = 1/\text{sec}$, $\mu_1 = 1/\text{sec}$, and $\mu_2 = 1.5/\text{sec}$. Compute the utilizations ρ_1 and ρ_2 of the two servers and the mean number of jobs $\overline{K}$ using the approximation method and

the exact methods for the strategy random selection of the free server and the strategy selection of the fastest free server (see Section 6.16). Discuss the results.

Problem 6.8 Consider an $M/M^{[5,5]}/1$ batch system and an $M/M^{[2,5]}/1$ batch system with $\rho = 0.5$. Compute the mean queue length $\overline{Q}$ for both cases and discuss the results.

7

Queueing Networks

Queueing networks consisting of several service stations are more suitable for representing the structure of many systems with a large number of resources than models consisting of a single service station. In a queueing network at least two service stations are connected to each other. A station, i.e., a *node*, in the network represents a resource in the real system. Jobs in principle can be transferred between any two nodes of the network; in particular, a job can be directly returned to the node it has just left.

A queueing network is called *open* when jobs can enter the network from outside and jobs can also leave the network. Jobs can arrive from outside the network at every node and depart from the network from any node. A queueing network is said to be *closed* when jobs can neither enter nor leave the network. The number of jobs in a closed network is constant. A network in which a new job enters whenever a job leaves the system can be considered as a closed one.

In Fig. 7.1 an open queueing network model of a simple computer system is shown. An example of a closed queueing network model is shown in Fig. 7.2. This is the central-server model, a particular closed network that has been proposed by [Buze71] for the investigation of the behavior of multiprogramming system with a fixed degree of multiprogramming. The node with service rate μ_1 is the central-server representing the central processing unit (CPU). The other nodes model the peripheral devices: disk drives, printers, magnetic tape units, etc. The number of jobs in this closed model is equal to the degree of multiprogramming. A closed tandem queueing network with two nodes is shown in Fig. 7.3. A very frequently occurring queueing network is the machine repairman model, shown in Fig. 7.4.

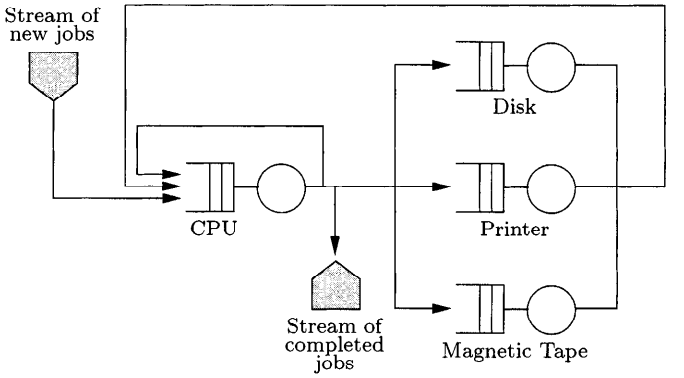


Fig. 7.1 Computer system shown as an open queueing network.

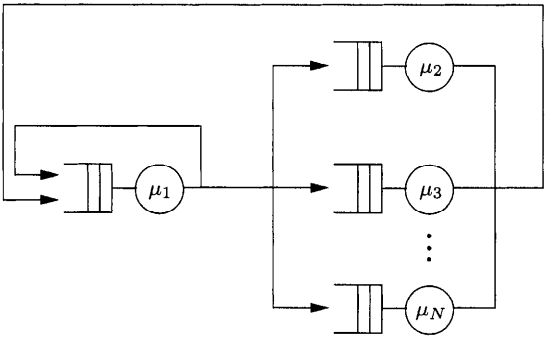


Fig. 7.2 Central-server model.

There are many systems that can be represented by the machine repairman model, for example, a simple terminal system where the M machines represent the M terminals and the repairman represents the computer. Another example is a system in which M machines operate independently of each other and are repaired by a single repairman if they fail. The M machines are modeled by a delay server or an infinite server node. A job does not have to wait; it is immediately accepted by one of the servers. When a machine fails, it sends a repair request to the repair facility. Depending on the service discipline, this request may have to wait until other requests have been dealt with. Since there are M machines, there can be at most M repair requests.

Fig. 7.3 Closed tandem network.

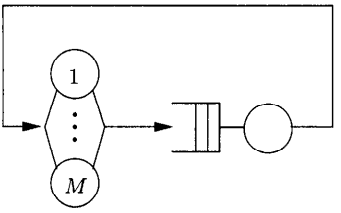


Fig. 7.4 Machine repairman model.

This is a closed two-node queueing network with M jobs that are either being processed, waiting, or are in the working machines.

7.1 DEFINITIONS AND NOTATION

We will consider both single class and multiclass networks.

7.1.1 Single Class Networks

The following symbols are used in the description of queueing networks:

N	Number of nodes
K	The constant number of jobs in a closed network
$(k_1, k_2, \ldots, k_N)$	The state of the network
k_i	The number of jobs at the ith node; for closed networks $\sum_{i=1}^{N} k_i = K$
m_i	The number of parallel servers at the ith node ($m_i \geq 1$)
μ_i	Service rate of the jobs at the ith node
$1/\mu_i$	The mean service time of the jobs at the ith node
p_{ij}	Routing probability, the probability that a job is transferred to the jth node after service completion at the ith node. (In open networks, the node with index 0 represents the external world to the network.)
p_{0j}	The probability that a job entering the network from outside first enters the jth node
p_{i0}	The probability that a job leaves the network just after completing service at node i ($p_{i0} = 1 - \sum_{j=1}^{N} p_{ij}$).
λ_{0i}	The arrival rate of jobs from outside to the ith node

λ The overall arrival rate from outside to an open network ($\lambda = \sum_{i=1}^{N} \lambda_{0i}$)

λ_i the overall arrival rate of jobs at the ith node

The *arrival rate* λ_i for node $i = 1, \ldots, N$ of an open network is calculated by adding the arrival rate from outside and the arrival rates from all the other nodes. Note that in statistical equilibrium the rate of departure from a node is equal to the rate of arrival, and the overall arrival rate at node i can be written as:

$$\lambda_i = \lambda_{0i} + \sum_{j=1}^{N} \lambda_j p_{ji}, \quad \text{for } i = 1, \ldots, N \tag{7.1}$$

for an open network. These are known as traffic equations. For closed networks these equations reduce to:

$$\lambda_i = \sum_{j=1}^{N} \lambda_j p_{ji}, \quad \text{for } i = 1, \ldots, N, \tag{7.2}$$

since no jobs enter the network from outside.

Another important network parameter is the *mean number of visits* (e_i) of a job to the ith node, also known as the *visit ratio* or *relative arrival rate*:

$$e_i = \frac{\lambda_i}{\lambda}, \quad \text{for } i = 1, \ldots, N, \tag{7.3}$$

where λ is the overall throughput of the network(see Eqs. (7.24) and (7.25)). The visit ratios can also be calculated directly from the routing probabilities using Eqs. (7.3) and (7.1), or (7.3) and (7.2). For open networks, since $\lambda_{0i} = \lambda \cdot p_{0i}$:

$$e_i = p_{0i} + \sum_{j=1}^{N} e_j p_{ji}, \quad \text{for } i = 1, \ldots, N, \tag{7.4}$$

and for closed networks:

$$e_i = \sum_{j=1}^{N} e_j p_{ji}, \quad \text{for } i = 1, \ldots, N. \tag{7.5}$$

Since there are only $(N - 1)$ independent equations for the visit ratios in closed networks, the e_i can only be determined up to a multiplicative constant. Usually we assume that $e_1 = 1$, although other possibilities are used as well. Using e_i, we can also compute the *relative utilization* x_i, which is given by:

$$x_i = \frac{e_i}{\mu_i}. \tag{7.6}$$

It is easy to show that [ChLa74] the ratio of server utilizations is given by:

$$\frac{x_i}{x_j} = \frac{\rho_i}{\rho_j}. \tag{7.7}$$

7.1.2 Multiclass Networks

The model type discussed in the previous section can be extended by including multiple job classes in the network. The job classes can differ in their service times and in their routing probabilities. It is also possible that a job changes its class when it moves from one node to another. If no jobs of a particular class enter or leave the network, i.e., the number of jobs of this class is constant, then the job class is said to be *closed*. A job class that is not closed is said to be *open*. If a queueing network contains both open and closed classes, then it is said to be a *mixed* network. Figure 7.5 shows a mixed network.

The following additional symbols are needed to describe queueing networks that contain multiple job classes, namely:

R The number of job classes in a network

k_{ir} The number of jobs of the rth class at the ith node; for a closed network:

$$\sum_{i=1}^{N} \sum_{r=1}^{R} k_{ir} = K \tag{7.8}$$

K_r The number of jobs of the rth class in the network; not necessarily constant, even in a closed network:

$$\sum_{i=1}^{N} k_{ir} = K_r \tag{7.9}$$

$\mathbf{K}$ The number of jobs in the various classes, known as the population vector ($\mathbf{K} = (K_1, \ldots, K_R)$)

$\mathbf{S}_i$ The state of the ith node ($\mathbf{S}_i = (k_{i1}, \ldots, k_{iR})$):

$$\sum_{i=1}^{N} \mathbf{S}_i = \mathbf{K} \tag{7.10}$$

$\mathbf{S}$ The overall state of the network with multiple classes ($\mathbf{S} = (\mathbf{S}_1, \ldots, \mathbf{S}_N)$)

μ_{ir} The service rate of the ith node for jobs of the rth class

$p_{ir,js}$ that a job of the rth class at the ith node is transferred to the sth class and the jth node (routing probability)

$p_{0,js}$ The probability in an open network that a job from outside the network enters the jth node as a job of the sth class

$p_{ir,0}$ The probability in an open network that a job of the rth class leaves the network after having been serviced at the ith node, so:

$$p_{ir,0} = 1 - \sum_{j=1}^{N} \sum_{s=1}^{R} p_{ir,js} \tag{7.11}$$

λ The overall arrival rate from outside to an open network

$\lambda_{0,ir}$ The arrival rate from outside to node i for class r jobs ($\lambda_{0,ir} = \lambda \cdot p_{0,ir}$)

λ_{ir} The arrival rate of jobs of the rth class at the ith node:

$$\lambda_{ir} = \lambda \cdot p_{0,ir} + \sum_{j=1}^{N} \sum_{s=1}^{R} \lambda_{js} \cdot p_{js,ir}; \tag{7.12}$$

for closed networks, $p_{0,ir} = 0$ $(1 < i < N, 1 < r < R)$ and we obtain:

$$\lambda_{ir} = \sum_{j=1}^{N} \sum_{s=1}^{R} \lambda_{js} \cdot p_{js,ir}. \tag{7.13}$$

The mean number of visits e_{ir} of a job of the rth class at the ith node of an open network can be determined from the routing probabilities similarly to Eq. (7.4):

$$e_{ir} = p_{0,ir} + \sum_{j=1}^{N} \sum_{s=1}^{R} e_{js} p_{js,ir}, \quad \text{for } i = 1, \ldots, N, \ r = 1, \ldots, R. \tag{7.14}$$

For closed networks, the corresponding equation is:

$$e_{ir} = \sum_{j=1}^{N} \sum_{s=1}^{R} e_{js} p_{js,ir}, \qquad \text{for } i = 1, \ldots, N, \ r = 1, \ldots, R. \tag{7.15}$$

Usually we assume that $e_{1r} = 1$, for $r = 1, \ldots, R$, although other settings are also possible.

7.2 PERFORMANCE MEASURES

7.2.1 Single Class Networks

Analytic methods to calculate state probabilities and other performance measures of queueing networks are described in the following sections. The determination of the *steady-state probabilities* $\pi(k_1, \ldots, k_N)$ of all possible states of the network can be regarded as the central problem of queueing theory. The

mean values of all other important performance measures of the network can be calculated from these. There are, however, simpler methods for calculating these characteristics directly without using these probabilities.

Note that we use a slightly different notation compared with that used in Chapters 2-5. In Chapters 2-5, $\pi_i(t)$ denoted the transient probability of the CTMC being in state i at time t and π_i as the steady-state probability in state i. Since we now deal with multidimension state spaces, $\pi(k_1, k_2, \ldots, k_N)$ will denote the steady-state probability of state $(k_1, k_2, \ldots, k_N)$.

The most important performance measures for queueing networks are:

Marginal Probabilities $\pi_i(k)$: For *closed* queueing networks, the *marginal probabilities* $\pi_i(k)$ that the ith node contains exactly $k_i = k$ jobs are calculated as follows:

$$\pi_i(k) = \sum_{\substack{\sum_{j=1}^{N} k_j = K \\ \& \; k_i = k}} \pi(k_1, \ldots, k_N). \tag{7.16}$$

Thus $\pi_i(k)$ is the sum of the probabilities of all possible states $(k_1, \ldots, k_N)$, $0 \leq k_i \leq K$ that satisfy the condition $\sum_{j=1}^{N} k_j = K$ where a fixed number of jobs, k, is specified for the ith node. The normalization condition that the sum of the probabilities of all possible states $(k_1, \ldots, k_N)$ that satisfy the condition $\sum_{j=1}^{N} k_j = K$ with $(0 \leq k_j \leq K)$ must be 1, that is:

$$\sum_{\sum_{j=1}^{N} k_j = K} \pi(k_1, \ldots, k_N) = 1. \tag{7.17}$$

Correspondingly, for *open* networks we have:

$$\pi_i(k) = \sum_{k_i = k} \pi(k_1, \ldots, k_N),$$

with the normalization condition:

$$\sum \pi(k_1, \ldots, k_N) = 1.$$

Now we can use the marginal probabilities to obtain other interesting performance measures for open and closed networks. For closed networks we have to take into consideration that $\pi_i(k) = 0 \quad \forall k > K$.

Utilization ρ_i: The *utilization* ρ_i of a single server node with index i is given by:

$$\rho_i = \sum_{k=1}^{\infty} \pi_i(k) \tag{7.18}$$

where ρ_i is the probability that the ith node is busy, that is:

$$\rho_i = 1 - \pi_i(0). \tag{7.19}$$

For nodes with multiple servers we have:

$$\rho_i = \frac{1}{m_i} \sum_{k=0}^{\infty} \min(m_i, k)\pi_i(k) = 1 - \sum_{k=0}^{m_i-1} \frac{m_i - k}{m_i} \cdot \pi_i(k), \tag{7.20}$$

and if the service rate is independent of the load we get (see Eq. (6.4)):

$$\rho_i = \frac{\lambda_i}{m_i \mu_i}. \tag{7.21}$$

Throughput λ_i: The *throughput* λ_i of an individual node with index i represents in general the rate at which jobs leave the node:

$$\lambda_i = \sum_{k=1}^{\infty} \pi_i(k)\mu_i(k), \tag{7.22}$$

where the service rate $\mu_i(k)$ is, in general, dependent on the load, i.e., on the number of jobs at the node. For example, a node with multiple servers ($m_i > 1$) can be regarded as a single server whose service rate depends on the load $\mu_i(k) = \min(k, m_i) \cdot \mu_i$, where μ_i is the service rate of an individual server. It is also true for load-independent service rates that (see Eqs. (6.4), and (6.5)):

$$\lambda_i = m_i \cdot \rho_i \cdot \mu_i. \tag{7.23}$$

We note that for a node in equilibrium, arrival rate and throughput are equal. Also note that when we consider nodes with finite buffers, arriving customers can be lost when the buffer is full. In this case, node throughput will be less than the arrival rate to the node.

Overall Throughput λ: The *overall throughput* λ of an open network is defined as the rate at which jobs leave the network. For a network in equilibrium this departure rate is equal to the rate at which jobs enter the network, that is:

$$\lambda = \sum_{i=1}^{N} \lambda_{0i}. \tag{7.24}$$

The overall throughput of a closed network is defined as the throughput of a particular node with index i for which $e_i = 1$. Then the overall throughput of jobs in closed networks is:

$$\lambda = \frac{\lambda_i}{e_i}. \tag{7.25}$$

Mean Number of Jobs $\overline{K}_i$: The *mean number of jobs* at the ith node is given by:

$$\overline{K}_i = \sum_{k=1}^{\infty} k \cdot \pi_i(k). \tag{7.26}$$

From Little's theorem (see Eq. (6.9)) it follows:

$$\overline{K}_i = \lambda_i \cdot \overline{T}_i, \tag{7.27}$$

where $\overline{T}_i$ denotes the mean response time.

Mean Queue Length $\overline{Q}_i$: The *mean queue length* at the ith node is determined by:

$$\overline{Q}_i = \sum_{k=m_i}^{\infty} (k - m_i) \cdot \pi_i(k), \tag{7.28}$$

or, using Little's theorem:

$$\overline{Q}_i = \lambda_i \overline{W}_i, \tag{7.29}$$

where $\overline{W}_i$ is the *mean waiting time.*

Mean Response Time $\overline{T}_i$: The *mean response time* of jobs at the ith node can be calculated using Little's theorem (see Eq. (6.9)) for a given mean number of jobs $\overline{K}_i$:

$$\overline{T}_i = \frac{\overline{K}_i}{\lambda_i}. \tag{7.30}$$

Mean Waiting Time $\overline{W}_i$: If the service rates are independent of the load, then the *mean waiting time* at node i is:

$$\overline{W}_i = \overline{T}_i - \frac{1}{\mu_i}. \tag{7.31}$$

7.2.2 Multiclass Networks

The extension of queueing networks to include multiple job classes leads to a corresponding extension of the performance measures. The *state probability* of a network with multiple job classes is represented by $\pi(\mathbf{S}_1, \ldots, \mathbf{S}_N)$. The normalization condition that the sum of the probabilities of all possible states $(\mathbf{S}_1, \ldots, \mathbf{S}_N)$ is 1 must also be satisfied here.

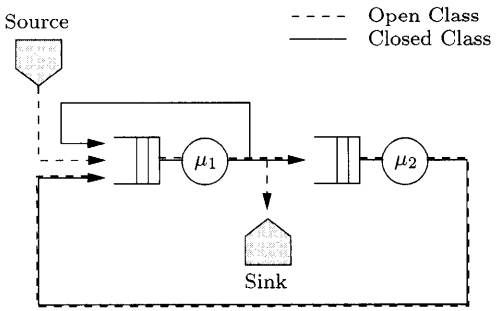


Fig. 7.5 A mixed network.

Marginal Probability $\pi_i(\mathbf{k})$: For closed networks the *marginal probability*, i.e., the probability that the ith node is in the state $\mathbf{S}_i = \mathbf{k}$, is given by:

$$\pi_i(\mathbf{k}) = \sum_{\substack{\sum_{j=1}^{N} \mathbf{S}_j = \mathbf{K} \\ \& \ \mathbf{S}_i = \mathbf{k}}} \pi(\mathbf{S}_1, \ldots, \mathbf{S}_N), \tag{7.32}$$

and for open networks:

$$\pi_i(\mathbf{k}) = \sum_{s_i = k} \pi(\mathbf{S}_1, \ldots, \mathbf{S}_N). \tag{7.33}$$

The following formulae for computing the performance measures can be applied to open and closed networks.

Utilization ρ_{ir}: The *utilization* of the ith node with respect to jobs of the rth class is:

$$\rho_{ir} = \frac{1}{m_i} \sum_{\substack{\text{all states } \mathbf{k} \\ \text{with } k_r > 0}} \pi_i(\mathbf{k}) \frac{k_{ir}}{k_i} \min(m_i, k_i), \quad k_i = \sum_{r=1}^{R} k_{ir}, \tag{7.34}$$

and if the service rates are independent on the load:

$$\rho_{ir} = \frac{\lambda_{ir}}{m_i \mu_{ir}}. \tag{7.35}$$

Throughput λ_{ir}: The *throughput* λ_{ir} is the rate at which jobs of the rth class are serviced and leave the ith node [BrBa80]:

$$\lambda_{ir} = \sum_{\substack{\text{all states } \mathbf{k} \\ \text{with } k_r > 0}} \pi_i(\mathbf{k}) \frac{k_{ir}}{k_i} \mu_i(k_i), \tag{7.36}$$

or if the service rates are independent on the load:

$$\lambda_{ir} = m_i \cdot \rho_{ir} \cdot \mu_{ir}. \tag{7.37}$$

Overall Throughput λ_r: The *overall throughput* of jobs of the rth class in closed networks with multiclasses is:

$$\lambda_r = \frac{\lambda_{ir}}{e_{ir}}, \tag{7.38}$$

and for open networks:

$$\lambda_r = \sum_{i=1}^{N} \lambda_{0,ir}. \tag{7.39}$$

Mean Number of Jobs $\overline{K}_{ir}$: The *mean number of jobs* of the rth class at the ith node is:

$$\overline{K}_{ir} = \sum_{\substack{\text{all states } \mathbf{k} \\ \text{with } k_r > 0}} k_r \cdot \pi_i(\mathbf{k}). \tag{7.40}$$

Little's theorem can also be used here:

$$\overline{K}_{ir} = \lambda_{ir} \cdot \overline{T}_{ir}. \tag{7.41}$$

Mean Queue Length $\overline{Q}_{ir}$: The *mean queue length* of class r jobs at the ith node can be calculated using Little's theorem as:

$$\overline{Q}_{ir} = \lambda_{ir} \overline{W}_{ir}. \tag{7.42}$$

Mean Response Time $\overline{T}_{ir}$: The *mean response time* of jobs of the rth class at the ith node can also be determined using Little's theorem (see Eq. (7.41)):

$$\overline{T}_{ir} = \frac{\overline{K}_{ir}}{\lambda_{ir}}. \tag{7.43}$$

Mean Waiting Time $\overline{W}_{ir}$: If the service rates are load-independent, then the *mean waiting time* is given by:

$$\overline{W}_{ir} = \overline{T}_{ir} - \frac{1}{\mu_{ir}}. \tag{7.44}$$

7.3 PRODUCT-FORM QUEUEING NETWORKS

In the rest of this chapter, we will discuss queueing networks that have a special structure such that their solutions can be obtained without generating their underlying state space. Such networks are known as product-form or separable networks.

7.3.1 Global Balance

The behavior of many queueing system models can be described using CTMCs. A CTMC is characterized by the transition rates between the states of the corresponding model (for more details on CTMC see Chapters 2-5). If the CTMC is ergodic, then a unique steady-state probability vector independent of the initial probability vector exists. The system of equations to determine the steady-state probability vector $\boldsymbol{\pi}$ is given by $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$ (see Eq. (2.58)), where $\mathbf{Q}$ is the infinitesimal generator matrix of the CTMC. This equation says that for each state of a queueing network in equilibrium, the flux out of a state is equal to the flux into that state. This conservation of flow in the steady state can be written as:

$$\sum_{j \in S} \pi_j q_{ji} = \pi_i \sum_{j \in S} q_{ij}, \quad \forall i \in S, \tag{7.45}$$

where q_{ij} is the transition rate from state i to state j. After subtracting $\pi_i \cdot q_{ii}$ from both sides of Eq. (7.45) and noting that $q_{ii} = -\sum_{j \neq i} q_{ij}$, we obtain the global balance equation (see Eq. (2.61)):

$$\forall i \in S: \quad \sum_{j \neq i} \pi_j q_{ji} - \pi_i \sum_{j \neq i} q_{ij} = 0, \tag{7.46}$$

which corresponds to the matrix equation $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$ (Eq. (2.58)).

In the following we use two simple examples to show how to write the global balance equations and use them to obtain performance measures.

Example 7.1 Consider the closed queueing network given in Fig. 7.6. The network consists of two nodes ($N = 2$) and three jobs ($K = 3$). The service times are exponentially distributed with mean values $1/\mu_1 = 5$ sec and $1/\mu_2 = 2.5$ sec, respectively. The service discipline at each node is FCFS. The state space of the CTMC consists of the following four states:

$$\{(3,0),(2,1),(1,2),(0,3)\}.$$

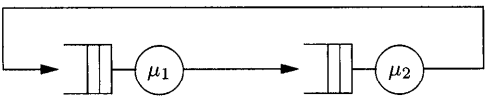


Fig. 7.6 A closed network.

The notation (k_1, k_2) says that there are k_1 jobs at node 1 and k_2 jobs at node 2, and $\pi(k_1, k_2)$ denotes the probability for that state in equilibrium. Consider state $(1,2)$. A transition from state $(1,2)$ to state $(0,3)$ takes place if the job at node 1 completes service (with corresponding rate μ_1). Therefore, μ_1 is the transition rate from state $(1,2)$ to state $(0,3)$ and, similarly, μ_2 is

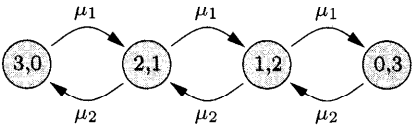


Fig. 7.7 State transition diagram for Example 7.1.

the transition rate from state $(1,2)$ to state $(2,1)$. The flux into a state of the model is just given by all arcs into the corresponding state, and the flux out of that state is determined from the set of all outgoing arcs from the state. The corresponding state transition diagram is shown in Fig. 7.7. The global balance equations for this example are:

$$\begin{aligned}
\pi(3,0)\mu_1 &= \pi(2,1)\mu_2, \\
\pi(2,1)(\mu_1+\mu_2) &= \pi(3,0)\mu_1 + \pi(1,2)\mu_2, \\
\pi(1,2)(\mu_1+\mu_2) &= \pi(2,1)\mu_1 + \pi(0,3)\mu_2, \\
\pi(0,3)\mu_2 &= \pi(1,2)\mu_1.
\end{aligned}$$

Rewriting this system of equations in the form $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$ we have:

$$\mathbf{Q} = \begin{pmatrix} -\mu_1 & \mu_1 & 0 & 0 \\ \mu_2 & -(\mu_1+\mu_2) & \mu_1 & 0 \\ 0 & \mu_2 & -(\mu_1+\mu_2) & \mu_1 \\ 0 & 0 & \mu_2 & -\mu_2 \end{pmatrix}$$

and the steady-state probability vector $\boldsymbol{\pi} = (\pi(3,0), \pi(2,1), \pi(1,2), \pi(0,3))$. Once the steady-state probabilities are known, all other performance measures and marginal probabilities $\pi_i(k)$ can be computed. If we use $\mu_1 = 0.2$ and $\mu_2 = 0.4$, then the generator matrix $\mathbf{Q}$ has the following values:

$$\mathbf{Q} = \begin{pmatrix} -0.2 & 0.2 & 0 & 0 \\ 0.4 & -0.6 & 0.2 & 0 \\ 0 & 0.4 & -0.6 & 0.2 \\ 0 & 0 & 0.4 & -0.4 \end{pmatrix}.$$

Using one of the steady-state solution methods introduced in Chapter 3, the steady-state probabilities are computed to be:

$$\pi(3,0) = \underline{0.5333}, \quad \pi(2,1) = \underline{0.2667}, \quad \pi(1,2) = \underline{0.1333}, \quad \pi(0,3) = \underline{0.0667}$$

and are used to determine all other performance measures of the network, as follows:

- Marginal probabilities (see Eq. (7.16)):

$$\pi_1(0) = \pi_2(3) = \pi(0,3) = \underline{0.0667}, \quad \pi_1(1) = \pi_2(2) = \pi(1,2) = \underline{0.133},$$

$$\pi_1(2) = \pi_2(1) = \pi(2,1) = \underline{0.2667}, \quad \pi_1(3) = \pi_2(0) = \pi(3,0) = \underline{0.5333}.$$

- Utilizations (see Eq. (7.20)):

$$\rho_1 = 1 - \pi_1(0) = \underline{0.9333}, \quad \rho_2 = 1 - \pi_2(0) = \underline{0.4667}.$$

- Throughput (see Eq. (7.23)):

$$\lambda = \lambda_1 = \lambda_2 = \rho_1\mu_1 = \rho_2\mu_2 = \underline{0.1867}.$$

- Mean number of jobs (see Eq. (7.26)):

$$\overline{K}_1 = \sum_{k=1}^{3} k\pi_1(k) = \underline{2.2667}, \quad \overline{K}_2 = \sum_{k=1}^{3} k\pi_2(k) = \underline{0.7333}.$$

- Mean response time of the jobs (see Eq. (7.30)):

$$\overline{T}_1 = \frac{\overline{K}_1}{\lambda_1} = \underline{12.1429}, \quad \overline{T}_2 = \frac{\overline{K}_2}{\lambda_2} = \underline{3.9286}.$$

Example 7.2 Now we modify the closed network of Example 7.1 so that the service time at node 1 is Erlang-2 distributed, while the service time at node 2 remains exponentially distributed. The modified network is shown in Fig. 7.8. Service rates of the different phases at node 1 are given by $\mu_{11} = \mu_{12} = 0.4sec^{-1}$, and the service rate of node 2 is $\mu_2 = 0.4$. There are $K = 2$ jobs in the system. A state $(k_1, l; k_2)$, $l = 0, 1, 2, \ldots$, of the network is now

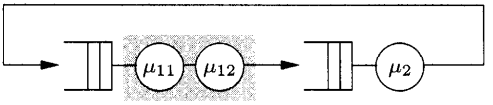


Fig. 7.8 Network with Erlang-2 distributed server.

not only given by the number k_i of jobs at the nodes, but also by the phase l in which a job is being served at node 1. This state definition leads to exactly five states in the CTMC underlying the network.

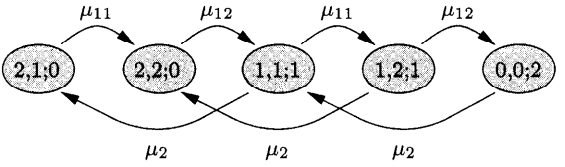


Fig. 7.9 State transition diagram for Example 7.2.

The state transition diagram of the CTMC is shown in Fig. 7.9. The following global balance equations can be derived:

$$\begin{aligned}
\pi(2,1;0)\mu_{11} &= \pi(1,1;1)\mu_2,\\
\pi(2,2;0)\mu_{12} &= \pi(2,1;0)\mu_{11} + \pi(1,2;1)\mu_2,\\
\pi(1,1;1)(\mu_{11}+\mu_2) &= \pi(2,2;0)\mu_{12} + \pi(0,0;2)\mu_2,\\
\pi(1,2;1)(\mu_{12}+\mu_2) &= \pi(1,1;1)\mu_{11},\\
\pi(0,0;2)\mu_2 &= \pi(1,2;1)\mu_{12}.
\end{aligned}$$

The generator matrix is:

$$\mathbf{Q} = \begin{pmatrix}
-\mu_{11} & \mu_{11} & 0 & 0 & 0\\
0 & -\mu_{12} & \mu_{12} & 0 & 0\\
\mu_2 & 0 & -(\mu_{11}+\mu_2) & \mu_{11} & 0\\
0 & \mu_2 & 0 & -(\mu_{12}+\mu_2) & \mu_{12}\\
0 & 0 & \mu_2 & 0 & -\mu_2
\end{pmatrix}.$$

After inserting the values $\mu_{11} = \mu_{12} = \mu_2 = 0.4$, $\mathbf{Q}$ becomes:

$$\mathbf{Q} = \begin{pmatrix}
-0.4 & 0.4 & 0 & 0 & 0\\
0 & -0.4 & 0.4 & 0 & 0\\
0.4 & 0 & -0.8 & 0.4 & 0\\
0 & 0.4 & 0 & -0.8 & 0.4\\
0 & 0 & 0.4 & 0 & -0.4
\end{pmatrix}.$$

Solving the system of equations $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$, we get:

$$\begin{aligned}
\pi(2,1;0) &= \underline{0.2219}, \quad \pi(2,2;0) = \underline{0.3336},\\
\pi(1,1;1) &= \underline{0.2219}, \quad \pi(1,2;1) = \underline{0.1102},\\
\pi(0,0;2) &= \underline{0.1125}.
\end{aligned}$$

These state probabilities are used to determine the marginal probabilities:

$$\begin{aligned}
\pi_1(0) = \pi_2(2) &= \pi(0,0;2) = \underline{0.1125},\\
\pi_1(1) = \pi_2(1) &= \pi(1,1;1) + \pi(1,2;1) = \underline{0.3321},\\
\pi_1(2) = \pi_2(0) &= \pi(2,1;0) + \pi(2,2;0) = \underline{0.5555}.
\end{aligned}$$

The computation of other performance measures is done in the same way as in Example 7.1.

7.3.2 Local Balance

Numerical techniques based on the solution of the global balance equations (see Eq. (2.61)) can in principle always be used, but for large networks this technique is very expensive because the number of equations can be extremely

large. For such large networks, we therefore look for alternative solution techniques.

In this chapter we show that efficient and exact solution algorithms exist for a large class of queueing networks. These algorithms avoid the generation and solution of global balance equations. If all nodes of the network fulfill certain assumptions concerning the distributions of the interarrival and service times and the queueing discipline, then it is possible to derive local balance equations, which describe the system behavior in an unambiguous way. These local balance equations allow an essential simplification with respect to the global balance equations because each equation can be split into a number of single equations, each one related to each individual node.

Queueing networks that have an unambiguous solution of the local balance equations are called *product-form networks.* The steady-state solution to such networks' state probabilities consist of multiplicative factors, each factor relating to a single node. Before introducing the different solution methods for product-form networks, we explain the local balance concept in more detail. This concept is the theoretical basis for the applicability of analysis methods.

Consider global balance equations (Eq. (2.61)) for a CTMC:

$$\forall i \in S: \quad \sum_{j \in S} \pi_j q_{ji} = \pi_i \sum_{j \in S} q_{ij}$$

or:

$$\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0},$$

with the normalization condition:

$$\sum_{i \in S} \pi_i = 1.$$

Chandy [Chan72] noticed that under certain conditions the global balance equations can be split into simpler equations, known as *local balance equations.*

Local balance property for a node means: The departure rate from a state of the queueing network due to the departure of a job from node i equals the arrival rate to this state due to an arrival of a job to this node.

This can also be extended to queueing networks with several job classes in the following way:

The departure rate from a state of the queueing network due to the departure of a *class r-job* from node i equals the arrival rate to this state due to an arrival of a *class r-job* to this node.

In the case of non-exponentially distributed service times, arrivals and departures to phases, instead of nodes, have to be considered.

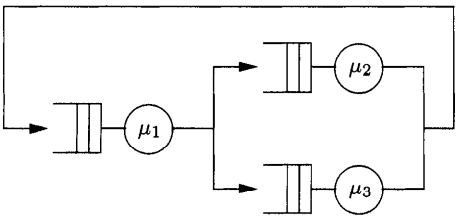


Fig. 7.10 A closed queueing network.

Example 7.3 Consider a closed queueing network (see Fig. 7.10) consisting of $N = 3$ nodes with exponentially distributed service times and the following service rates: $\mu_1 = 4sec^{-1}$, $\mu_2 = 1sec^{-1}$, and $\mu_3 = 2sec^{-1}$. There are $K = 2$ jobs in the network and the routing probabilities are given as: $p_{12} = 0.4$, $p_{13} = 0.6$, and $p_{21} = p_{31} = 1$. The following states are possible in the network:

$$(2,0,0), \quad (0,2,0), \quad (0,0,2), \quad (1,1,0), \quad (1,0,1), \quad (0,1,1).$$

The state diagram of the underlying CTMC is shown in Fig. 7.11.

We set the overall flux into a state equal to the overall flux out of the state for each state to get global balance equations:

$$\begin{aligned}
&(1) && \pi(2,0,0)(\mu_1 p_{12} + \mu_1 p_{13}) = \pi(1,0,1)\mu_3 p_{31} + \pi(1,1,0)\mu_2 p_{21},\\
&(2) && \pi(0,2,0)\mu_2 p_{21} = \pi(1,1,0)\mu_1 p_{12},\\
&(3) && \pi(0,0,2)\mu_3 p_{31} = \pi(1,0,1)\mu_1 p_{13},\\
&(4) && \pi(1,1,0)(\mu_2 p_{21} + \mu_1 p_{13} + \mu_1 p_{12}) = \pi(0,2,0)\mu_2 p_{21} + \pi(2,0,0)\mu_1 p_{12}\\
&&& \qquad + \pi(0,1,1)\mu_3 p_{31},\\
&(5) && \pi(1,0,1)(\mu_3 p_{31} + \mu_1 p_{12} + \mu_1 p_{13}) = \pi(0,0,2)\mu_3 p_{31} + \pi(0,1,1)\mu_2 p_{21}\\
&&& \qquad + \pi(2,0,0)\mu_1 p_{13},\\
&(6) && \pi(0,1,1)(\mu_3 p_{31} + \mu_2 p_{21}) = \pi(1,1,0)\mu_1 p_{13} + \pi(1,0,1)\mu_1 p_{12}.
\end{aligned}$$

To determine the local balance equations we start, for example, with the state $(1,1,0)$. The departure rate out of this state, because of the departure of a job from node 2, is given by $\pi(1,1,0)\cdot\mu_2\cdot p_{21}$. This rate is equated to the arrival rate into this state $(1,1,0)$ due to the arrival of a job at node 2; $\pi(2,0,0)\cdot\mu_1\cdot p_{12}$. Therefore, we get the local balance equation:

$$(4') \quad \pi(1,1,0)\cdot\mu_2\cdot p_{21} = \pi(2,0,0)\cdot\mu_1\cdot p_{12}.$$

Correspondingly, the departure rate of a serviced job at node 1 from state $(1,1,0)$ equals the arrival rate of a job, arriving at node 1 into state $(1,1,0)$:

$$(4'') \quad \pi(1,1,0)\cdot\mu_1\cdot(p_{13} + p_{12}) = \pi(0,1,1)\cdot\mu_3\cdot p_{31} + \pi(0,2,0)\cdot\mu_2\cdot p_{21}.$$

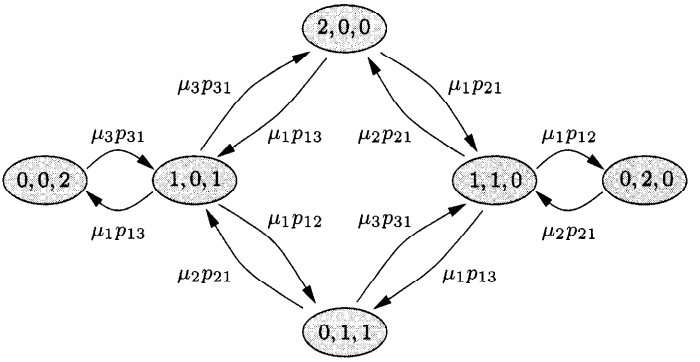


Fig. 7.11 The CTMC for Example 7.3.

By adding these two local balance equations, (4′) and (4″), we get the global balance Eq. (4). Furthermore, we see that the global balance Eqs. (1), (2), and (3) are also local balance equations at the same time. The rest of the local balance equations are given by:

$$
\begin{aligned}
&(5') \quad \pi(1,0,1)\mu_1(p_{12}+p_{13}) = \pi(0,1,1)\mu_2 p_{21} + \pi(0,0,2)\mu_3 p_{31},\\
&(5'') \quad \pi(1,0,1)\mu_3 p_{31} = \pi(2,0,0)\mu_1 p_{13},\\
&(6') \quad \pi(0,1,1)\mu_2 p_{21} = \pi(1,0,1)\mu_1 p_{12},\\
&(6'') \quad \pi(0,1,1)\mu_3 p_{31} = \pi(1,1,0)\mu_1 p_{13},
\end{aligned}
$$

with (5′) + (5″) = (5) and (6′) + (6″) = (6), respectively.

Noting that $p_{12} + p_{13} = 1$ and $p_{21} = p_{31} = 1$, the following relations can be derived from these local balance equations:

$$
\begin{aligned}
&\pi(1,0,1) = \pi(2,0,0)\frac{\mu_1}{\mu_3}p_{13}, \qquad \pi(1,1,0) = \pi(2,0,0)\frac{\mu_1}{\mu_2}p_{12},\\
&\pi(0,0,2) = \pi(2,0,0)\left(\frac{\mu_1}{\mu_3}p_{13}\right)^2, \quad \pi(0,2,0) = \pi(2,0,0)\left(\frac{\mu_1}{\mu_2}p_{12}\right)^2,\\
&\pi(0,1,1) = \pi(2,0,0)\frac{\mu_1^2}{\mu_2\mu_3}p_{12}p_{13}.
\end{aligned}
$$

Imposing the normalization condition, that the sum of probabilities of all states in the network is 1, for $\pi(2,0,0)$ we get the following expression:

$$
\pi(2,0,0) = \left[1 + \mu_1\left(\frac{p_{13}}{\mu_3} + \frac{p_{12}}{\mu_2} + \frac{\mu_1 p_{13}^2}{\mu_3^2} + \frac{\mu_1 p_{12}^2}{\mu_2^2} + \frac{\mu_1 p_{12} p_{13}}{\mu_2\mu_3}\right)\right]^{-1}.
$$

After inserting the values, we get the following results:

$$
\begin{aligned}
&\pi(2,0,0) = \underline{0.103}, \quad \pi(0,0,2) = \underline{0.148}, \quad \pi(1,0,1) = \underline{0.123},\\
&\pi(0,2,0) = \underline{0.263}, \quad \pi(1,1,0) = \underline{0.165}, \quad \pi(0,1,1) = \underline{0.198}.
\end{aligned}
$$

As can be seen in this example, the structure of the local balance equations is much simpler than the global balance equations. However, not every network has a solution of the local balance equations but there always exists a solution of the global balance equations. Therefore, local balance can be considered as a sufficient (but not necessary) condition for the global balance. Furthermore, if there exists a solution for the local balance equations, the model is then said to have the *local balance property.* Then this solution is also the unique solution of the system of global balance equations.

The computational effort for the numerical solution of the local balance equations of a queueing network is still very high but can be reduced considerably with the help of a characteristic property of local-balanced queueing networks: For the determination of the state probabilities it is not necessary to solve the local balance equations for the whole network. Instead, the state probabilities of the queueing network in these cases can be determined very easily from the state probabilities of individual single nodes of the network. If each node in the network has the local balance property, then the following two very important implications are true:

- The overall network also has the local balance property as proven in [CHT77].

- There exists a product-form solution for the network, that is:

$$\pi(\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_N) = \frac{1}{G} \left[\pi(\mathbf{S}_1) \cdot \pi(\mathbf{S}_2) \cdot \ldots \cdot \pi(\mathbf{S}_N)\right], \tag{7.47}$$

  in the sense that the expression for the state probability of the network is given by the product of marginal state probabilities of each individual node. The proof of this fact can be found in [Munt73]. The normalization constant G is chosen in such a way that the sum of probabilities over all states in the network equals 1.

Equation (7.47) says that in networks having the local-balance property, the nodes behave as if they were single queueing systems. This characteristic means that the nodes of the network can be examined in isolation from the rest of the network. Networks of the described type belong to the class of so-called *separable networks* or *product-form networks.* Now we need to examine for which types of elementary queueing systems a solution of the local balance equation exists. If the network consists of only these types of nodes then we know, because of the preceding conclusion, that the whole network has the local-balance property and the network has a product-form solution. The local-balance equations for a single node can be presented in a simplified form as follows [SaCh81]:

$$\pi(\mathbf{S}) \cdot \mu_r(\mathbf{S}) = \pi(\mathbf{S} - \mathbf{1}_r) \cdot \lambda_r. \tag{7.48}$$

In this equation, $\mu_r(\mathbf{S})$ is the rate with which class-r jobs in state $\mathbf{S}$ are serviced at the node, λ_r is the rate at which class-r jobs arrive at the node, and $(\mathbf{S} - \mathbf{1}_r)$ describes the state of the node after a single class-r job leaves it.

It can be shown that for the following types of queueing systems the local balance property holds [Chan72]:

Type-1: **M/M/m–FCFS**. The service rates for different job classes must be equal. Examples of Type-1 nodes are input/output (I/O) devices or disks.

Type-2: **M/G/1–PS**. The CPU of a computer system can very often be modeled as a Type-2 node.

Type-3: **M/G/∞ (infinite server)**. Terminals can be modeled as Type-3 nodes.

Type-4: **M/G/1–LCFS PR**. There is no practical example for the application of Type-4 nodes in computer systems.

For Type-2, Type-3, and Type-4 nodes, different job classes can have *different general service time distributions*, provided that these have rational Laplace transform. In practice, this requirement is not an essential limitation as any distribution can be approximated as accurately as necessary using a Cox distribution.

In the next section we consider product-form solutions of separable networks in more detail.

Problem 7.1 Consider the closed queueing network given in Fig. 7.12. The network consists of two nodes ($N = 2$) and three jobs ($K = 3$). The

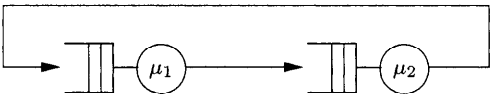


Fig. 7.12 A simple queueing network example for Problem 7.1.

service times are exponentially distributed with mean values $1/\mu_1 = 5$ sec and $1/\mu_2 = 2.5$ sec. The service discipline at each node is FCFS, and the routing probability $q = 0.1$.

(a) Determine the local balance equations.

(b) From the local balance equations, derive the global balance equations.

(c) Determine the steady-state probabilities using the local balance equations.

7.3.3 Product-Form

The term *product-form* was introduced by [Jack63] and [GoNe67a], who considered open and closed queueing networks with exponentially distributed interarrival and service times. The queueing discipline at all stations was assumed to be FCFS. As the most important result for the queueing theory, it is shown that for these networks the solution for the steady-state probabilities can be expressed as a product of factors describing the state of each node. This solution is called *product-form solution.* In [BCMP75] these results were extended to open, closed, and mixed networks with several job classes, non-exponentially distributed service times and different queuing disciplines. In this section we consider these results in more detail and give algorithms to compute performance measures of product-form queueing networks.

A necessary and sufficient condition for the existence of product-form solutions is given in the previous section but repeated here in a slightly different way:

Local Balance Property: Steady-state probabilities can be obtained by solving steady-state (global) balance equations. These equations balance the rate at which the CTMC leaves that state with the rate at which the CTMC enters it. The problem is that the number of equations increases exponentially in the number of states. Therefore, a new set of balance equations, the so-called *local balance equations*, is defined. With these, the rate at which jobs enter a *single* node of the network is equated to the rate at which they leave it. Thus local balance is concerned with a local situation and reduces the computational effort.

Moreover, there exist two other characteristics that apply to a queueing network with product-form solution:

M $\Rightarrow$ M-Property (Markov Implies Markov): A service station has the M $\Rightarrow$ M-property if and only if the station transforms a Poisson arrival process into a Poisson departure process. In [Munt73] it is shown that a queueing network has a product-form solution if all nodes of the network have the M $\Rightarrow$ M-property.

Station-Balance Property: A service discipline is said to have station-balance property if the service rates at which the jobs in a position of the queue are served are proportional to the probability that a job enters this position. In other words, the queue of a node is partitioned into positions and the rate at which a job enters this position is equal to the rate with which the job leaves this position. In [CHT77] it is shown that networks that have the station-balance property have a product-form solution. The opposite does not hold.

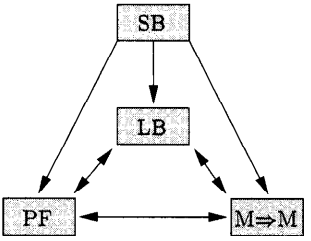


Fig. 7.13 Relation between SB, LB, PF, and $M \Rightarrow M$.

The relation between station balance (SB), local balance (LB), product-form property (PF), and Markov implies Markov property ($M \Rightarrow M$) is shown in Fig. 7.13.

7.3.4 Jackson Networks

The breakthrough in the analysis of queueing networks was achieved by the works of Jackson [Jack57, Jack63]. He examined open queueing networks and found product-form solutions. The networks examined fulfill the following assumptions:

- There is only one job class in the network.
- The overall number of jobs in the network is unlimited.
- Each of the N nodes in the network can have Poisson arrivals from outside. A job can leave the network from any node.
- All service times are exponentially distributed.
- The service discipline at all nodes is FCFS.
- The ith node consists of $m_i \geq 1$ identical service stations with the service rates μ_i, $i = 1, \ldots, N$. The arrival rates λ_{0i}, as well as the service rates, can depend on the number k_i of jobs at the node. In this case we have *load-dependent service rates* and *load-dependent arrival rates.*

Note: A service station with more than one server and a constant service rate μ_i is equivalent to a service station with exactly one server and load-dependent service rates:

$$\mu_i(k) = \begin{cases} k_i \cdot \mu_i, & k_i \leq m_i, \\ m_i \cdot \mu_i, & k_i \geq m_i. \end{cases} \tag{7.49}$$

Jackson's Theorem: If in an open network ergodicity ($\lambda_i < \mu_i \cdot m_i$) holds for all nodes $i = 1, \ldots, N$ (the arrival rates λ_i can be computed using Eq. (7.1)), then the steady-state probability of the network can be expressed as the product of the state probabilities of the individual nodes, that is:

$$\pi(k_1, k_2, \ldots, k_N) = \pi_1(k_1) \cdot \pi_2(k_2) \cdot \ldots \cdot \pi_N(k_N). \tag{7.50}$$

The nodes of the network can be considered as independent M/M/m queues with arrival rate λ_i and service rate μ_i. To prove this theorem, [Jack63] has shown that Eq. (7.50) fulfills the global balance equations. Thus, the marginal probabilities $\pi_i(k_i)$ can be computed with the well-known formulae for M/M/m systems (see Eqs. (6.26), (6.27)):

$$\pi_i(k_i) = \begin{cases} \pi_i(0) \dfrac{(m_i \rho_i)^{k_i}}{k_i!}, & k_i \leq m_i, \\ \pi_i(0) \dfrac{m_i^{m_i} \rho_i^{k_i}}{m_i!}, & k_i > m_i, \end{cases} \tag{7.51}$$

where $\pi_i(0)$ is given by the condition $\sum_{k_i=0}^{\infty} \pi_i(k_i) = 1$:

$$\pi_i(0) = \left(\sum_{k_i=0}^{m_i - 1} \frac{(m_i \rho_i)^{k_i}}{k_i!} + \frac{(m_i \rho_i)^{m_i}}{m_i!(1 - \rho_i)} \right)^{-1}, \quad \rho_i = \frac{\lambda_i}{m_i \mu_i} < 1. \tag{7.52}$$

Proof: We verify that Eq. (7.50) fulfills the following global balance equations:

$$\begin{aligned} &\left(\sum_{i=1}^{N} \lambda_{0i} + \sum_{i=1}^{N} \alpha_i(k_i) \mu_i \right) \pi(k_1, \ldots, k_N) = \\ &= \sum_{i=1}^{N} \lambda_{0i} \gamma(k_i) \pi(k_1, \ldots, k_i - 1, \ldots, k_N) \\ &\quad + \sum_{i=1}^{N} \alpha_i(k_i + 1) \mu_i \left(1 - \sum_{j=1}^{N} p_{ij} \right) \pi(k_1, \ldots, k_i + 1, \ldots, k_N) \\ &\quad + \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_j(k_j + 1) \mu_j p_{ji} \pi(k_1, \ldots, k_j + 1, \ldots, k_i - 1, \ldots, k_N). \end{aligned} \tag{7.53}$$

The indicator function $\gamma(k_i)$ is given by:

$$\gamma(k_i) = \begin{cases} 0, & k_i = 0, \\ 1, & k_i > 0. \end{cases} \tag{7.54}$$

The function:

$$\alpha_i(k_i) = \begin{cases} k_i, & k_i \le m_i, \\ m_i, & k_i \ge m_i \end{cases} \tag{7.55}$$

gives the load-dependent service rate multiplier.

For the proof we use the following relations:

$$\begin{aligned} \frac{\pi(k_1,\ldots,k_i+1,\ldots,k_N)}{\pi(k_1,\ldots,k_i,\ldots,\ldots,k_N)} &= \frac{\pi_1(k_1)\cdots\pi_i(k_i+1)\cdots\pi_N(k_N)}{\pi_1(k_1)\cdots\pi_i(k_i)\cdots\pi_N(k_N)} \\ &= \frac{\lambda_i}{\mu_i\alpha_i(k_i+1)}, \\ \frac{\pi(k_1,\cdots,k_i-1,\cdots,k_N)}{\pi(k_1,\cdots,k_i,\cdots,k_N)} &= \frac{\mu_i\alpha_i(k_i)}{\lambda_i}, \\ \frac{\pi(k_1,\ldots,k_j+1,\ldots,k_i-1,\ldots,k_N)}{\pi(k_1,\ldots,k_j,\ldots,k_i,\ldots,k_N)} &= \frac{\lambda_j\mu_i\alpha_i(k_i)}{\lambda_i\mu_j\alpha_j(k_j+1)}. \end{aligned} \tag{7.56}$$

If we divide Eq. (7.53) by $\pi(k_1,\ldots,k_N)$ and insert Eq. (7.56), then, by using the relation $\gamma(k_i)\cdot\alpha_i(k_i) \equiv \alpha_i(k_i)$, we get:

$$\begin{aligned} \sum_{i=1}^{N}\lambda_{0i} + \sum_{i=1}^{N}\alpha_i(k_i)\mu_i &= \sum_{i=1}^{N}\left(1-\sum_{j=1}^{N}p_{ij}\right)\lambda_i \\ &\quad + \sum_{i=1}^{N}\frac{\lambda_{0i}\mu_i\alpha_i(k_i)}{\lambda_i} + \sum_{i=1}^{N}\sum_{j=1}^{N}\frac{\mu_i\alpha_i(k_i)}{\lambda_i}p_{ji}\lambda_j. \end{aligned} \tag{7.57}$$

The first term can be rewritten as:

$$\sum_{i=1}^{N}\left(1-\sum_{j=1}^{N}p_{ij}\right)\lambda_i = \sum_{i=1}^{N}\lambda_{0i},$$

and the last one as:

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{\mu_i\alpha_i(k_i)}{\lambda_i}p_{ji}\lambda_j = \sum_{i=1}^{N}\mu_i\alpha_i(k_i) - \sum_{i=1}^{N}\frac{\lambda_{0i}\mu_i\alpha_i(k_i)}{\lambda_i}. \tag{7.58}$$

Substituting these results on the right side of Eq. (7.57), we get the same result as on the left side. *q.e.d.*

The algorithm based on Jackson's theorem for computing the steady-state probabilitics can now be described in the following three steps:

STEP 1 For all nodes, $i = 1,\ldots,N$, compute the arrival rates λ_i of the open network by solving the traffic equations, Eq. (7.1).

STEP 2 Consider each node i as an M/M/m queueing system. Check the ergodicity, Eq. (6.5), and compute the state probabilities and performance measures of each node using the formulae given in Section 6.5.

STEP 3 Using Eq. (7.50), compute the steady-state probabilities of the overall network.

We note that for a Jackson network with feedback input, processes to the nodes will, in general, not be Poisson and yet the nodes behave like independent M/M/m nodes. Herein lies the importance of Jackson's theorem [BeMe78]. We illustrate the procedure with the following example.

Example 7.4 Consider the queueing network given in Fig. 7.14, which consists of $N = 4$ single server FCFS nodes. The service times of the jobs at

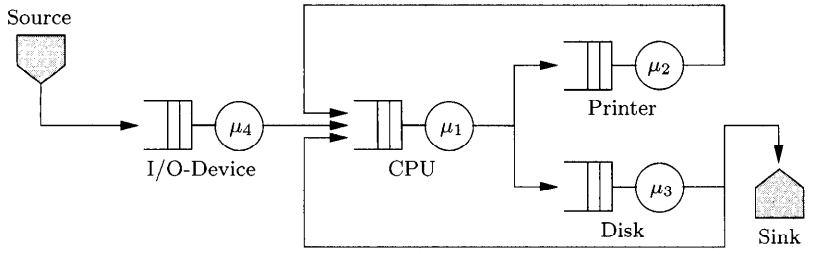


Fig. 7.14 Open queueing network model of a computer system.

each node are exponentially distributed with respective means:

$$\frac{1}{\mu_1} = 0.04\,\text{sec}, \quad \frac{1}{\mu_2} = 0.03\,\text{sec}, \quad \frac{1}{\mu_3} = 0.06\,\text{sec}, \quad \frac{1}{\mu_4} = 0.05\,\text{sec}.$$

The interarrival time is also exponentially distributed with the parameter:

$$\lambda = \lambda_{04} = 4 \text{ jobs/sec}.$$

Furthermore, the routing probabilities are given as follows:

$$p_{12} = p_{13} = 0.5, \quad p_{41} = p_{21} = 1, \quad p_{31} = 0.6, \quad p_{30} = 0.4.$$

Assume that we wish to compute the steady-state probability of state $(k_1, k_2, k_3, k_4) = (3, 2, 4, 1)$ with the help of the Jackson's method. For this we follow the three steps given previously:

STEP 1 Compute the arrival rates from the traffic equations, Eq. (7.1):

$$\lambda_1 = \lambda_2 p_{21} + \lambda_3 p_{31} + \lambda_4 p_{41} = \underline{20}, \quad \lambda_2 = \lambda_1 p_{12} = \underline{10},$$
$$\lambda_3 = \lambda_1 p_{13} = \underline{10}, \qquad \lambda_4 = \lambda_{04} = \underline{4}.$$

STEP 2 Compute the state probabilities and important performance measures for each node. For the *utilization* of a single server we use Eq. (6.3):

$$\rho_1 = \frac{\lambda_1}{\mu_1} = \underline{0.8}, \quad \rho_2 = \frac{\lambda_2}{\mu_2} = \underline{0.3}, \quad \rho_3 = \frac{\lambda_3}{\mu_3} = \underline{0.6}, \quad \rho_4 = \frac{\lambda_4}{\mu_4} = \underline{0.2}.$$

Thus, ergodicity ($\rho_i < 1$) is fulfilled for all nodes. The *mean number of jobs* at the nodes is given by Eq. (6.13):

$$\overline{K}_1 = \frac{\rho_1}{1-\rho_1} = \underline{4}, \quad \overline{K}_2 = \underline{0.429}, \quad \overline{K}_3 = \underline{1.5}, \quad \overline{K}_4 = \underline{0.25}.$$

Mean response times, from Eq. (6.15):

$$\overline{T}_1 = \frac{1/\mu_1}{1-\rho_1} = \underline{0.2}, \quad \overline{T}_2 = \underline{0.043}, \quad \overline{T}_3 = \underline{0.15}, \quad \overline{T}_4 = \underline{0.0625}.$$

The *mean overall response time* of a job is given by using Little's theorem in the following way:

$$\overline{T} = \frac{\overline{K}}{\lambda} = \frac{1}{\lambda}\sum_{1=1}^{4} \overline{K}_i = \underline{1.545}.$$

Mean waiting times, from Eq. (6.16):

$$\overline{W}_1 = \frac{\rho_1/\mu_1}{1-\rho_1} = \underline{0.16}, \quad \overline{W}_2 = \underline{0.013}, \quad \overline{W}_3 = \underline{0.09}, \quad \overline{W}_4 = \underline{0.0125}.$$

Mean queue lengths, from Eq. (6.17):

$$\overline{Q}_1 = \frac{\rho_1^2}{1-\rho_1} = \underline{3.2}, \quad \overline{Q}_2 = \underline{0.129}, \quad \overline{Q}_3 = \underline{0.9}, \quad \overline{Q}_4 = \underline{0.05}.$$

The necessary marginal probabilities can be computed using Eq. (6.12):

$$\pi_1(3) = (1-\rho_1)\rho_1^3 = \underline{0.1024}, \quad \pi_2(2) = (1-\rho_2)\rho_2^2 = \underline{0.063},$$
$$\pi_3(4) = (1-\rho_3)\rho_3^4 = \underline{0.0518}, \quad \pi_4(1) = (1-\rho_4)\rho_4 = \underline{0.16}.$$

STEP 3 Computation of the state probability $\pi(3,2,4,1)$ using Eq. (7.50):

$$\pi(3,2,4,1) = \pi_1(3)\cdot\pi_2(2)\cdot\pi_3(4)\cdot\pi_4(1) = \underline{0.0000534}.$$

7.3.5 Gordon/Newell Networks

Gordon and Newell [GoNe67a] considered closed queueing networks for which they made the same assumptions as in open queueing networks, except that

no job can enter or leave the system ($\lambda_{0i} = \lambda_{i0} = 0$). This restriction means that the number K of jobs in the system is always constant:

$$K = \sum_{i=1}^{N} k_i.$$

Thus, the number of possible states is finite, and it is given by the binomial coefficient:

$$\binom{N+K-1}{N-1},$$

which describes the number of ways of distributing K jobs on N nodes. The theorem of Gordon/Newell says that the probability for each network state in equilibrium is given by the following product-form expression:

$$\pi(k_1, \ldots, k_N) = \frac{1}{G(K)} \prod_{i=1}^{N} F_i(k_i). \tag{7.59}$$

Here $G(K)$ is the so-called *normalization constant.* It is given by the condition that the sum of all network state probabilities equals 1:

$$G(K) = \sum_{\sum_{i=1}^{N} k_i = K} \prod_{i=1}^{N} F_i(k_i). \tag{7.60}$$

The $F_i(k_i)$ are functions that correspond to the state probabilities $\pi_i(k_i)$ of the ith node and are given by:

$$F_i(k_i) = \left(\frac{e_i}{\mu_i}\right)^{k_i} \cdot \frac{1}{\beta_i(k_i)}, \tag{7.61}$$

where the visit ratios e_i are computed using Eq. (7.5). The function $\beta_i(k_i)$ is given by:

$$\beta_i(k_i) = \begin{cases} k_i!, & k_i \leq m_i, \\ m_i! \cdot m_i^{k_i - m_i}, & k_i \geq m_i, \\ 1, & m_i = 1. \end{cases} \tag{7.62}$$

For various applications a more general form of the function $F_i(k_i)$ is advantageous. In this generalized function, the service rates depend on the number of jobs at the node. For this function we have:

$$F_i(k_i) = \frac{e_i^{k_i}}{A_i(k_i)}, \tag{7.63}$$

with

$$A_i(k_i) = \begin{cases} \prod_{j=1}^{k_i} \mu_i(j), & k_i > 0, \\ 1, & k_i = 0. \end{cases} \tag{7.64}$$

With relation (7.49) it can easily be seen that the case of constant service rates, Eq. (7.61), is a special case of Eq. (7.63).

Proof: Gordon/Newell have shown in [GoNe67a] that Eq. (7.59) fulfills the global balance equations (see Eq. (7.53)) that have now the following form:

$$\begin{aligned} &\left(\sum_{i=1}^{N} \gamma(k_i)\alpha_i(k_i)\mu_i\right) \pi(k_1, \ldots, k_N) = \\ &\sum_{j=1}^{N}\sum_{i=1}^{N} \gamma(k_i)\alpha_j(k_j+1)\mu_j p_{ji} \pi(k_1, \ldots, k_i - 1, \ldots, k_j + 1, \ldots, k_N), \end{aligned} \tag{7.65}$$

where the left side describes the departure rate out of state $(k_1, \ldots, k_N)$ and the right side describes the arrival rate from successor states into this state. The function $\gamma(k_i)$ and $\alpha_i(k_i)$ are given by Eqs. (7.54) and (7.55).

We define now a variable transformation as follows:

$$\pi(k_1, \ldots, k_N) = \frac{Q(k_1, \ldots, k_N)}{\prod_{i=1}^{N} \beta_i(k_i)}, \tag{7.66}$$

and:

$$\begin{aligned} &\pi(k_1, \ldots, k_i - 1, \ldots, k_j + 1, \ldots, k_N) = \\ &\frac{\frac{\alpha_i(k_i)}{\alpha_j(k_j+1)} Q(k_1, \ldots, k_i - 1, \ldots, k_j + 1, \ldots, k_N)}{\prod_{i=1}^{N} \beta_i(k_i)}. \end{aligned} \tag{7.67}$$

If we substitute these equations into Eq. (7.65) we get:

$$\begin{aligned} &\frac{\sum_{i=1}^{N} \gamma(k_i)\alpha_i(k_i)\mu_i Q(k_1, \ldots, k_N)}{\prod_{i=1}^{N} \beta_i(k_i)} = \\ &\frac{\sum_{j=1}^{N}\sum_{i=1}^{N} \gamma(k_i)\alpha_j(k_j+1)\mu_j p_{ji} \frac{\alpha_i(k_i)}{\alpha_j(k_j+1)} Q(k_1, \ldots, k_i - 1, \ldots, k_j + 1, \ldots, k_N)}{\prod_{i=1}^{N} \beta_i(k_i)}. \end{aligned}$$

This equation can be simplified using the relation $\gamma(k_i)\alpha_i(k_i) \equiv \alpha_i(k_i)$:

$$\sum_{i=1}^{N} \alpha_i(k_i)\mu_i Q(k_1, \ldots, k_N) = \sum_{j=1}^{N}\sum_{i=1}^{N} \alpha_i(k_i)\mu_j p_{ji} Q(k_1, \ldots, k_i - 1, \ldots, k_j + 1, \ldots, k_N), \tag{7.68}$$

and $Q(k_1, \ldots, k_N)$ can be written in the form:

$$Q(k_1, \ldots, k_N) = \left\{ \prod_{i=1}^{N} x_i^{k_i} \right\} \cdot c, \tag{7.69}$$

with the relative utilization $x_i = e_i/\mu_i$ (Eq. (7.6)).

Here c is a constant. Substituting this into (7.68) we have:

$$\begin{aligned} &\sum_{i=1}^{N} \alpha_i(k_i)\mu_i (x_1^{k_1} \cdots x_N^{k_N}) \cdot c \\ &= \sum_{j=1}^{N}\sum_{i=1}^{N} \alpha_i(k_i)\mu_j p_{ji} (x_1^{k_1} \cdots x_i^{k_i - 1} \cdots x_j^{k_j + 1} \cdots x_N^{k_N}) \cdot c \\ &= \sum_{j=1}^{N}\sum_{i=1}^{N} \alpha_i(k_i)\mu_j p_{ji} (x_1^{k_1} \cdots x_N^{k_N}) \frac{x_j}{x_i} \cdot c, \end{aligned}$$

$$\sum_{i=1}^{N} \alpha_i(k_i)\mu_i = \sum_{j=1}^{N}\sum_{i=1}^{N} \alpha_i(k_i)\mu_j p_{ji} \frac{x_j}{x_i}.$$

This expression can be rewritten as:

$$\sum_{i=1}^{N} \alpha_i(k_i) \left(\mu_i - \sum_{j=1}^{N} \mu_j p_{ji} \frac{x_j}{x_i} \right) = 0.$$

Since at least one $\alpha_i(k_i)$ will be non-zero, it follows that the factor in the square brackets must be zero. Thus, we are led to consider the system of linear algebraic equations for x_i:

$$\mu_i x_i = \sum_{j=1}^{N} \mu_j x_j p_{ji},$$

where $x_i = e_i/\mu_i$ (Eq. (7.6)) and:

$$e_i = \sum_{j=1}^{N} e_j p_{ji}.$$

This is the traffic equation for closed networks (Eq. (7.5)). That means that Eq. (7.69) is correct and with Eqs. (7.66) and (7.61) we obtain Eq. (7.59). *q.e.d.*

Thus the Gordon/Newell theorem yields a product-form solution. In the general form it says that the state probabilities $\pi(k_1, k_2, \cdots, k_N)$ are given as the product of the functions $F_i(k_i)$, $i = 1 \ldots, N$, defined for single nodes. It is interesting to note that if we substitute in Eq. (7.59) $F_i(k_i)$ by $L_i \cdot F_i(k_i), 1 \leq i \leq N$, then this has no influence on the solution of the state probabilities $\pi(k_1, k_2, \ldots, k_N)$ as long as L_i is a positive real number. Furthermore, the use of $\lambda \cdot e_i$, $i = 1, \ldots, N$, with an arbitrary constant $\lambda > 0$, has no influence on the results because the visit ratios e_i are relative values [ShBu77]. (Also see Problems 2 and 3 on *page* 444 of [Triv82].)

The Gordon/Newell method for computing the state probabilities can be summarized in the following four steps:

STEP 1 Compute the visit ratios e_i for all nodes $i = 1, \ldots, N$ of the closed network using Eq. (7.5).

STEP 2 For all $i = 1, \ldots, N$, compute the functions $F_i(k_i)$ using Eq. (7.61) or Eq. (7.63) (in the case of load-dependent service rates).

STEP 3 Compute the normalization constant $G(K)$ using Eq. (7.60).

STEP 4 Compute the state probabilities of the network using Eq. (7.59). From the marginal probabilities, which can be determined from the state probabilities using Eq. (7.16), all other required performance measures can be determined.

An example of the application of the Gordon/Newell theorem follows:

Example 7.5 Consider the closed queueing network shown in Fig. (7.15) with $N = 3$ nodes and $K = 3$ jobs. The queueing discipline at all nodes is FCFS. The routing probabilities are given as follows:

$$\begin{aligned} p_{11} &= 0.6, \quad p_{21} = 0.2, \quad p_{31} = 0.4, \\ p_{12} &= 0.3, \quad p_{22} = 0.3, \quad p_{32} = 0.1, \\ p_{13} &= 0.1, \quad p_{23} = 0.5, \quad p_{33} = 0.5. \end{aligned}$$

The service time at each node is exponentially distributed with the rates:

$$\mu_1 = 0.8\,\mathrm{sec}^{-1}, \quad \mu_2 = 0.6\,\mathrm{sec}^{-1}, \quad \mu_3 = 0.4\,\mathrm{sec}^{-1}. \tag{7.70}$$

This network consists of

$$\binom{N+K-1}{N-1} = \underline{10}$$

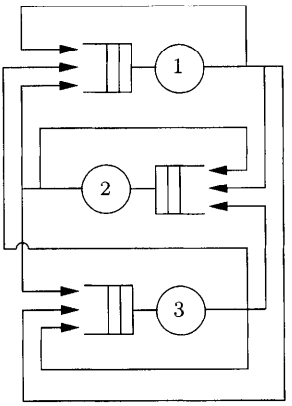


Fig. 7.15 Closed queueing network.

states, namely:

$$(3,0,0),\quad (2,1,0),\quad (2,0,1),\quad (1,2,0),\quad (1,1,1),$$
$$(1,0,2),\quad (0,3,0),\quad (0,2,1),\quad (0,1,2),\quad (0,0,3).$$

We wish to compute the state probabilities using the Gordon/Newell theorem. For this we proceed in the following four steps:

STEP 1 Determine the visit ratios at each node using Eq. (7.5):

$$e_1 = e_1p_{11} + e_2p_{21} + e_3p_{31} = \underline{1},$$
$$e_2 = e_1p_{12} + e_2p_{22} + e_3p_{32} = \underline{0.533},$$
$$e_3 = e_1p_{13} + e_2p_{23} + e_3p_{33} = \underline{0.733}.$$

STEP 2 Determine the functions $F_i(k_i)$ for $i = 1,2,3$ using Eq. (7.61):

$$F_1(0) = (e_1/\mu_1)^0 = \underline{1}, \qquad F_1(1) = (e_1/\mu_1)^1 = \underline{1.25},$$
$$F_1(2) = (e_1/\mu_1)^2 = \underline{1.5625}, \quad F_1(3) = (e_1/\mu_1)^3 = \underline{1.953},$$

and correspondingly:

$$F_2(0) = \underline{1}, \quad F_2(1) = \underline{0.889}, \quad F_2(2) = \underline{0.790}, \quad F_2(3) = \underline{0.702},$$
$$F_3(0) = \underline{1}, \quad F_3(1) = \underline{1.833}, \quad F_3(2) = \underline{3.361}, \quad F_3(3) = \underline{6.162}.$$

STEP 3 Determine the normalization constant using Eq. (7.60):

$$\begin{aligned} G(3) = {} & F_1(3)F_2(0)F_3(0) + F_1(2)F_2(1)F_3(0) + F_1(2)F_2(0)F_3(1) \\ & + F_1(1)F_2(2)F_3(0) + F_1(1)F_2(1)F_3(1) + F_1(1)F_2(0)F_3(2) \end{aligned}$$

$$+ F_1(0)F_2(3)F_3(0) + F_1(0)F_2(2)F_3(1) + F_1(0)F_2(1)F_3(2)$$
$$+ F_1(0)F_2(0)F_3(3) = \underline{24.733}.$$

STEP 4 Determine the state probabilities using the Gordon/Newell theorem, Eq. (7.59):

$$\pi(3,0,0) = \frac{1}{G(3)} F_1(3) \cdot F_2(0) \cdot F_3(0) = \underline{0.079},$$
$$\pi(2,1,0) = \frac{1}{G(3)} F_1(2) \cdot F_2(1) \cdot F_3(0) = \underline{0.056}.$$

In the same way we compute:

$$\pi(2,0,1) = \underline{0.116}, \quad \pi(1,2,0) = \underline{0.040}, \quad \pi(1,1,1) = \underline{0.082}, \quad \pi(1,0,2) = \underline{0.170},$$
$$\pi(0,3,0) = \underline{0.028}, \quad \pi(0,2,1) = \underline{0.058}, \quad \pi(0,1,2) = \underline{0.121}, \quad \pi(0,0,3) = \underline{0.249}.$$

Using Eq. (7.16), all marginal probabilities can now be determined:

$$\pi_1(0) = \pi(0,3,0) + \pi(0,2,1) + \pi(0,1,2) + \pi(0,0,3) = \underline{0.457},$$
$$\pi_1(1) = \pi(1,2,0) + \pi(1,1,1) + \pi(1,0,2) = \underline{0.292},$$
$$\pi_1(2) = \pi(2,1,0) + \pi(2,0,1) = \underline{0.172},$$
$$\pi_1(3) = \pi(3,0,0) = \underline{0.079},$$

$$\pi_2(0) = \pi(2,0,1) + \pi(1,0,2) + \pi(0,0,3) + \pi(3,0,0) = \underline{0.614},$$
$$\pi_2(1) = \pi(2,1,0) + \pi(1,1,1) + \pi(0,1,2) = \underline{0.259},$$
$$\pi_2(2) = \pi(1,2,0) + \pi(0,2,1) = \underline{0.098},$$
$$\pi_2(3) = \pi(0,3,0) = \underline{0.028},$$

$$\pi_3(0) = \pi(3,0,0) + \pi(2,1,0) + \pi(1,2,0) + \pi(0,3,0) = \underline{0.203},$$
$$\pi_3(1) = \pi(2,0,1) + \pi(1,1,1) + \pi(0,2,1) = \underline{0.257},$$
$$\pi_3(2) = \pi(1,0,2) + \pi(0,1,2) = \underline{0.291},$$
$$\pi_3(3) = \pi(0,0,3) = \underline{0.249}.$$

For the other performance measures, we get:

- Utilization at each node, Eq. (7.19):

$$\rho_1 = 1 - \pi_1(0) = \underline{0.543}, \quad \rho_2 = \underline{0.386}, \quad \rho_3 = \underline{0.797}.$$

- Mean number of jobs at each node, Eq. (7.26):

$$\overline{K}_1 = \sum_{k=1}^{3} k \cdot \pi_1(k) = \underline{0.873} \quad \overline{K}_2 = \underline{0.541}, \quad \overline{K}_3 = \underline{1.585}.$$

- Throughputs at each node, Eq. (7.23):

$$\lambda_1 = m_1\rho_1\mu_1 = \underline{0.435}, \quad \lambda_2 = \underline{0.232}, \quad \lambda_3 = \underline{0.319}.$$

- Mean response time at each node, Eq. (7.43):

$$\overline{T}_1 = \frac{\overline{K}_1}{\lambda_1} = \underline{2.009}, \quad \overline{T}_2 = \underline{2.337}, \quad \overline{T}_3 = \underline{4.976}.$$

7.3.6 BCMP Networks

The results of Jackson and Gordon/Newell were extended by Baskett, Chandy, Muntz, and Palacios in their classic article [BCMP75], to queueing networks with several job classes, different queueing strategies, and generally distributed service times. The considered networks can be open, closed, or mixed.

Following [BrBa80], an allowed state in a queueing model without class switching is characterized by four conditions:

1. The number of jobs in each class at each node is always nonnegative, i.e.:

$$k_{ir} \geq 0, \quad 1 \leq r \leq R, \quad 1 \leq i \leq N.$$

2. For all jobs the following condition must hold:

$$k_{ir} > 0 \quad \text{if there exists a way for class-}r\text{ jobs to node } i \text{ with a non-zero probability.}$$

3. For a closed network, the number of jobs in the network is given by:

$$K = \sum_{r=1}^{R}\sum_{i=1}^{N} k_{ir}.$$

4. The sum of class-r jobs in the network is constant at any time, i.e.:

$$K_r = \sum_{i=1}^{N} k_{ir} = \text{const.} \quad 1 \leq r \leq R.$$

If class switching is allowed, then Conditions 1-3 are fulfilled, but Condition 4 can not be satisfied because the number of jobs within a class is no longer constant but depends on the time when the system is looked at and can have the values $k \in \{0, \ldots, K\}$. In order to avoid this situation, the concept of chains is introduced.

7.3.6.1 The Concept of Chains Consider the routing matrix $\mathbf{P} = [p_{ir,js}]$, $i, j = 1, \ldots, N$, and $r, s = 1, \ldots, R$ of a closed queueing network. The routing matrix $\mathbf{P}$ defines a finite-state DTMC whose states are pairs of form (i, r) (node number i, class index r). We call (i, r) a *node-class pair.* This state space can be partitioned into disjoint sets Γ_i, $i = 1, \ldots, U$:

$$\Gamma = \Gamma_1 + \Gamma_2 + \ldots + \Gamma_U,$$

where Γ_i is a closed communicating class of recurrent states of the DTMC.

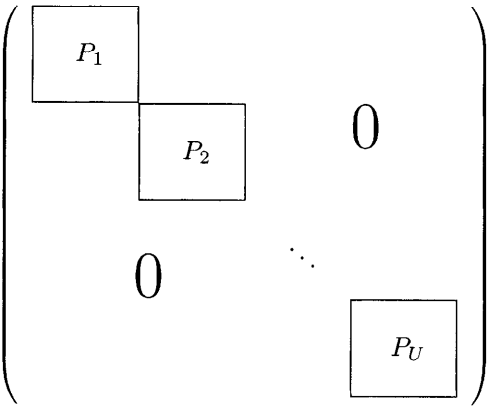


Fig. 7.16 Modified routing matrix.

With a possible relabeling of nodes, we get the routing matrix of Fig. 7.16. Where submatrices P_i contains the transition probabilities in set Γ_i and each P_i is disjoint and assumed to be ergodic. Let chain C_i denote the set of job classes in Γ_i. Because of the disjointness of the chains C_i, it is impossible for a job to switch from one chain to another. If a job starts in a chain, it will never leave this chain. With the help of this partitioning technique, the number of jobs in each chain is always constant in closed networks. Therefore, Condition 4 for queueing networks is fulfilled for the chains. Muntz [Munt73] proved that a closed queueing network with U chains is equivalent to a closed queueing network with U job classes. If there is no class switching allowed, then the number of chains equals the number of classes. But if class switching is allowed, the number of chains is smaller than the number of classes. This means that the dimension of the population vector is reduced. An extension of the chains concept to open networks is possible.

The procedure to find the chains from a given solution matrix is the following:

STEP 1 Construct R sets that satisfy the following condition:

$$E_r := \left\{ s : \begin{array}{l}\text{node-class pair } (j,s) \text{ can be reached from} \\ \text{pair } (i,r) \text{ in a finite number of steps}\end{array} \right\},$$

$$1 \leq r \leq R, \quad 1 \leq s \leq R.$$

STEP 2 Eliminate all subsets and identical sets so that we have:

$$U \leq R \quad \text{sets to obtain the chains} \quad C_1, \ldots, C_U.$$

STEP 3 Compute the number of jobs in each chain

$$K_q^* = \sum_{r \epsilon C_q} K_r, \quad 1 \leq q \leq U.$$

The set of all possible states in a closed multiclass network is given by the following binomial coefficient:

$$\prod_{q=1}^{U} \binom{N \cdot |C_q| + K_q^* - 1}{N \cdot |C_q| - 1},$$

where $|C_q|$ is the number of elements in C_q, $\quad 1 \leq q \leq U$.

For open networks, the visit ratios in a chain are given by:

$$2e_{ir} = p_{0,ir} + \sum_{\substack{s \in C_q \\ j=1,\ldots,N}} e_{js} p_{js,ir} \qquad \text{for} \quad \begin{array}{l} r \in C_q, \\ i = 1, \ldots N, \\ 1 \leq q \leq U, \end{array} \tag{7.71}$$

and for closed networks:

$$e_{ir} = \sum_{\substack{s \in C_q \\ j=1,\ldots,N}} e_{js} p_{js,ir} \qquad \text{for} \quad \begin{array}{l} r \in C_q, \\ i = 1, \ldots N, \\ 1 \leq q \leq U. \end{array} \tag{7.72}$$

Example 7.6 Let the following routing matrix $\mathbf{P}$ be given:

$$\mathbf{P} = \begin{array}{c} \\ (1,1) \\ (1,2) \\ (1,3) \\ (2,1) \\ (2,2) \\ (2,3) \\ (3,1) \\ (3,2) \\ (3,3) \end{array} \begin{array}{c} \begin{array}{ccccccccc} (1,1) & (1,2) & (1,3) & (2,1) & (2,2) & (2,3) & (3,1) & (3,2) & (3,3) \end{array} \\ \begin{pmatrix} 0 & 0.4 & 0 & 0 & 0.3 & 0 & 0 & 0.3 & 0 \\ 0.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0 & 0.7 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 \\ 0 & 0 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0.7 \\ 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0 & 0 & 0.6 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

By using the chaining technique we get:

$$E_1 = \{1,2\}, \quad E_2 = \{1,2\}, \quad E_3 = \{3\},$$

meaning that $R = 3$ classes are reduced to $U = 2$ chains. By eliminating the subsets and identical sets we get two chains:

$$C_1 = \{1,2\}, \quad C_2 = \{3\}.$$

Then the reorganized routing matrix $\mathbf{P}'$ has the form:

$$\mathbf{P}' = \begin{array}{c} \\ (1,1) \\ (1,2) \\ (2,1) \\ (2,2) \\ (3,1) \\ (3,2) \\ (1,3) \\ (2,3) \\ (3,3) \end{array} \begin{array}{c} \begin{array}{ccccccccc} (1,1) & (1,2) & (2,1) & (2,2) & (3,1) & (3,2) & (1,3) & (2,3) & (3,3) \end{array} \\ \begin{pmatrix} 0 & 0.4 & 0 & 0.3 & 0 & 0.3 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0 \\ 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0.6 & 0 \end{pmatrix} \end{array}.$$

If a job starts in one of the chains C_1 or C_2, then it can not leave them.

Because of the switch from classes to chains, it is necessary to transfer class measures into chain measures [BrBa80] (chain measures are marked with a *):

- Number of visits: In a chain a job can reach different node-class pairs (i, j) (i: node index, j: class index) where:

$$e_{iq}^* = \frac{\sum_{r \epsilon C_q} e_{ir}}{\sum_{r \epsilon C_q} e_{1r}}. \tag{7.73}$$

- The number of jobs per class is constant in a chain, but within the chain jobs can change their class. Therefore, if a job of chain q visits node i, it is impossible to know to which class it belongs because we consider a chain to be a single entity. If we make the transformation

$$\text{class} \quad \longrightarrow \quad \text{chain},$$

the information about the class of a job is lost. Because different job classes have different service times, this missing information is exchanged by the scale factor α. For the service rate in a chain we get:

$$s_{iq}^* = \frac{1}{\mu_{iq}^*} = \sum_{r \epsilon C_q} s_{ir} \cdot \alpha_{ir}, \tag{7.74}$$

$$\alpha_{ir} = \frac{e_{ir}}{\sum_{s \in C_q} e_{is}}. \tag{7.75}$$

In Chapter 8 we introduce several algorithms to calculate the performance measures of single and multiclass queueing networks without class switching (such as mean value analysis or convolution). Using the concept of chains, it is possible to also use these algorithms for queueing networks with class switching. To do so, we proceed in the following steps:

STEP 1 Calculate the number of visits e_{ir} in the original network.

STEP 2 Determine the chains $C_1 \ldots C_U$, and calculate the number of jobs K_q^* in each chain.

STEP 3 Compute the number of visits e_{iq}^* for each chain, Eq. (7.73).

STEP 4 Determine the scale factors α_{ir}, Eq. (7.75).

STEP 5 Calculate the service times s_{iq}^* for each chain, Eq. (7.74).

STEP 6 Derive the performance measures per chain [BrBa80] with one of the algorithms introduced later (mean value analysis, convolution, etc.).

STEP 7 Calculate the performance measures per class from the performance measures per chain:

$$\overline{T}_{ir}(\mathbf{K}^*) = s_{ir} \cdot \left(1 + \overline{K}_i(\mathbf{K}^* - \mathbf{1}_q)\right), \quad r \in C_q$$

$$\lambda_{ir}(\mathbf{K}^*) = \alpha_{ir} \cdot \lambda_{iq}^*,$$
$$\rho_{ir}(\mathbf{K}^*) = s_{ir} \cdot \lambda_{ir}(\mathbf{K}^*),$$

where:

$\mathbf{K}^* = (K_1^*, \ldots, K_U^*)$: Population vector containing the number of jobs in each chain,

$\mathbf{K}^* - \mathbf{1}_q$: $\mathbf{K}^*$ with one job less in chain q,

$\overline{K}_i(\mathbf{K}^* - \mathbf{1}_q)$: Mean number of jobs at node i if the number of jobs in the chains is given by $(\mathbf{K}^* - \mathbf{1}_q)$.

All other performance measures can easily be obtained using the formulae of Section 7.2.

In the following section we introduce the BCMP theorem, which is the basis of all analysis techniques to come. If class switching is not allowed in the network, then the BCMP theorem can be applied directly. In the case of class switching, it needs to be applied in combination with the concept of chains.

7.3.6.2 BCMP Theorem The theorems of Jackson and Gordon/Newell have been extended by [BCMP75] to networks with several job classes and different service strategies and interarrival/service time distributions, and also to mixed networks that contain open and closed classes. The networks considered by BCMP must fulfill the following assumptions:

- Queueing discipline: The following disciplines are allowed at network nodes: FCFS, PS, LCFS-PR, IS (infinite server).

- Distribution of the service times: The service times of an FCFS node must be exponentially distributed and class-independent (i.e. $\mu_{i1} = \mu_{i2} = \ldots = \mu_{iR} = \mu_i$), while PS, LCFS-PR and IS nodes can have any kind of service time distribution with a rational Laplace transform. For the latter three queueing disciplines, the mean service time for different job classes can be different.

- Load-dependent service rates: The service rate of an FCFS node is only allowed to depend on the number of jobs at this node, whereas in a PS, LCFS-PR and IS node the service rate for a particular job class can also depend on the number of jobs of that class at the node but not on the number of jobs in another class.

- Arrival processes: In open networks two kinds of arrival processes can be distinguished from each other.

  Case 1: The arrival process is Poisson where all jobs arrive at the network from one source with an overall arrival rate λ, where λ can depend

on the number of jobs in the network. The arriving jobs are distributed over the nodes in the network in accordance to the probability $p_{0,ir}$ where:

$$\sum_{i=1}^{N}\sum_{r=1}^{R} p_{0,ir} = 1.$$

Case 2: The arrival process consists of U independent Poisson arrival streams where the U job sources are assigned to the U chains. The arrival rate λ_u from the uth source can be load dependent. A job arrives at the ith node with probability $p_{0,ir}$ so that:

$$\sum_{\substack{r\in\mathcal{C}_u \\ i=1,\ldots,N}} p_{0,ir} = 1, \quad \text{for all } u = 1,\ldots,U.$$

These assumptions lead to the four product-form node types and the local balance conditions for BCMP networks (see Section 7.3.2), that is:

Type-1: –/M/m – FCFS	Type-2: –/G/1 – PS
Type-3: –/G/∞ (IS)	Type-4: –/G/1 – LCFS PR

Note that we use –/M/m notation since we know that, in general, the arrival process to a node in a BCMP network will not be Poisson.

The BCMP Theorem says that networks with the characteristics just described have product-form solution:

> For open, closed, and mixed queueing networks whose nodes consist only of the four described node types, the steady-state probabilities have the following product-form:

$$\pi(\mathbf{S}_1,\ldots,\mathbf{S}_N) = \frac{1}{G(\mathbf{K})} d(\mathbf{S}) \prod_{i=1}^{N} f_i(\mathbf{S}_i), \tag{7.76}$$

where,

$G(\mathbf{K})$ = the normalization constant,

$d(\mathbf{S})$ = a function of the number of jobs in the network, $\mathbf{S} = (\mathbf{S}_1,\ldots,\mathbf{S}_N)$

and:

$$d(\mathbf{S}) = \begin{cases} \displaystyle\prod_{i=0}^{K(\mathbf{S})-1} \lambda(i), & \text{open network with arrival process 1,} \\ \displaystyle\prod_{u=1}^{U}\prod_{i=0}^{K_u(\mathbf{S})-1} \lambda_u(i), & \text{open network with arrival process 2,} \\ 1, & \text{closed networks.} \end{cases}$$

$f_i(\mathbf{S}_i)$ = a function which depends on the type and state of each node i and:

$$f_i(S_i) = \begin{cases} \left(\frac{1}{\mu_i}\right)^{k_i} \prod_{j=1}^{k_i} e_{is_{ij}}, & \text{Type-1,} \\ k_i! \prod_{r=1}^{R} \prod_{l=1}^{u_{ir}} \frac{1}{k_{irl}!} \left(\frac{e_{ir}A_{irl}}{\mu_{irl}}\right)^{k_{irl}}, & \text{Type-2,} \\ \prod_{r=1}^{R} \prod_{l=1}^{u_{ir}} \frac{1}{k_{irl}!} \left(\frac{e_{ir}A_{irl}}{\mu_{irl}}\right)^{k_{irl}}, & \text{Type-3,} \\ \prod_{j=1}^{k_i} \frac{e_{ir_j}A_{ir_jm_j}}{\mu_{ir_jm_j}}, & \text{Type-4.} \end{cases} \tag{7.77}$$

Variables in for Eq. (7.77) have the following meanings:

s_{ij}: Class of the job that is at the jth position in the FCFS queue.

μ_{irl}: Mean service rate in the lth phase ($l = 1, \ldots, u_{ir}$) in a Cox distribution (see Chapter 1).

u_{ir}: Maximum number of exponential phases.

A_{irl}: $\prod_{j=0}^{l-1} a_{irj}$, probability that a class-r job at the ith node reaches the lth service phase ($A_{ir1} = 1$ because of $a_{ir0} = 1$).

a_{irj}: Probability that a class-r job at the ith node moves to the $(j+1)$th phase.

k_{irl}: Number of class-r jobs in the lth phase of node i.

For the load-dependent case, $f_i(\mathbf{S}_i)$ is of the form:

$$f_i(S_i) = \left(\frac{1}{\mu_i}\right)^{k_i} \prod_{j=1}^{k_i} e_{is_{ij}}.$$

Proof: The proof of this theorem is very complex and therefore only the basic idea is given here (for the complete proof see [Munt72]). In order to find a solution for the steady-state probabilities $\pi(\mathbf{S})$, the following global balance equations have to be solved:

$$\pi(\mathbf{S}) \left[\begin{array}{l}\text{state transition rate} \\ \text{from state } \mathbf{S}\end{array}\right] = \sum_{\tilde{\mathbf{S}}} \pi(\tilde{\mathbf{S}}) \left[\begin{array}{l}\text{state transition rate from} \\ \text{state } \tilde{\mathbf{S}} \text{ to state } \mathbf{S}\end{array}\right], \tag{7.78}$$

with the normalization condition:

$$\sum_{\mathbf{S}} \pi(\mathbf{S}) = 1. \tag{7.79}$$

Now we insert Eq. (7.76) into Eq. (7.78) to verify that Eq. (7.78) can be written as a system of balance equations that [Chan72] calls *local balance equations.* All local balance equations can be transformed into a system of $N-1$ independent equations. To get unambiguity for the solution the normalization condition, Eq. (7.79), has to be used.

Now we wish to give two simplified versions of the BCMP theorem for open and closed networks.

BCMP Version 1: For a *closed* queueing network fulfilling the assumptions of the BCMP theorem, the steady-state state probabilities have the form:

$$\pi(\mathbf{S}_1, \ldots, \mathbf{S}_N) = \frac{1}{G(\mathbf{K})} \prod_{i=1}^{N} F_i(\mathbf{S}_i), \tag{7.80}$$

where the normalization constant is defined as:

$$G(\mathbf{K}) = \sum_{\sum_{i=1}^{N} \mathbf{S}_i = \mathbf{K}} \prod_{i=1}^{N} F_i(\mathbf{S}_i), \tag{7.81}$$

and the function $F_i(\mathbf{S}_i)$ is given by:

$$F_i(\mathbf{S}_i) = \begin{cases} k_i! \dfrac{1}{\beta_i(k_i)} \cdot \left(\dfrac{1}{\mu_i}\right)^{k_i} \cdot \displaystyle\prod_{r=1}^{R} \frac{1}{k_{ir}!} e_{ir}^{k_{ir}}, & \text{Type-1,} \\ k_i! \displaystyle\prod_{r=1}^{R} \frac{1}{k_{ir}!} \cdot \left(\frac{e_{ir}}{\mu_{ir}}\right)^{k_{ir}}, & \text{Type-2,4,} \\ \displaystyle\prod_{r=1}^{R} \frac{1}{k_{ir}!} \cdot \left(\frac{e_{ir}}{\mu_{ir}}\right)^{k_{ir}}, & \text{Type-3.} \end{cases} \tag{7.82}$$

The quantity $k_i = \sum_{r=1}^{R} k_{ir}$ gives the overall number of jobs of all classes at node i. The visit ratios e_{ir} can be determined with Eq. (7.72), while the function $\beta_i(k_i)$ is given in Eq. (7.62).

For FCFS-nodes ($m_i = 1$) with load-dependent service rates $\mu_i(j)$ we get:

$$F_i(\mathbf{S}_i) = \frac{k_i!}{\displaystyle\prod_{j=1}^{k_i} \mu_i(j)} \prod_{r=1}^{R} \frac{1}{k_{ir}!} e_{ir}^{k_{ir}}. \tag{7.83}$$

BCMP Version 2: For an *open* queueing network fulfilling the assumptions of the BCMP theorem and load-independent arrival and service rates, we have:

$$\pi(k_1, \ldots, k_N) = \prod_{i=1}^{N} \pi_i(k_i), \tag{7.84}$$

where:

$$\pi_i(k_i) = \begin{cases} (1-\rho_i)\rho_i^{k_i}, & \text{Type-1,2,4 } (m_i = 1), \\ e^{-\rho_i}\dfrac{\rho_i^{k_i}}{k_i!}, & \text{Type-3,} \end{cases} \tag{7.85}$$

and:

$$k_i = \sum_{i=i}^{R} k_{ir},$$

$$\rho_i = \sum_{r=1}^{R} \rho_{ir},$$

with:

$$\rho_{ir} = \begin{cases} \lambda_r \dfrac{e_{ir}}{\mu_i}, & \text{Type-1 } (m_i = 1), \\ \lambda_r \dfrac{e_{ir}}{\mu_{ir}}, & \text{Type-2,3,4.} \end{cases} \tag{7.86}$$

Furthermore we have:

$$\overline{K}_{ir} = \frac{\rho_{ir}}{1-\rho_i}. \tag{7.87}$$

For Type-1 nodes with more than one service unit ($m_i > 1$), Eq. (6.26) can be used to compute the probabilitics $\pi_i(k_i)$. For thc existence of thc steady-state probabilities $\pi_i(k_i)$, ergodicity condition ($\rho_i < 1$) has to be fulfilled for all i, $i = 1, \ldots, N$.

The algorithm to determine the performance measures using the BCMP theorem can now be given in the following five steps:

STEP 1 Compute the visit ratios e_{ir} for all $i = 1, \ldots, N$ and $r = 1, \ldots, R$ using Eq. (7.71).

STEP 2 Compute the utilization of each node using Eq. (7.86)

STEP 3 Compute the other performance measures with the equations given in Section 7.2.

STEP 4 Compute the marginal probabilities of the network using Eq. (7.85).

STEP 5 Compute the state probabilities using Eq. (7.84)

Example 7.7 Consider the open network given in Fig. 7.17 with $N = 3$ nodes and $R = 2$ job classes. The first node is of Type-2 and the second and

third nodes are of Type-4. The service times are exponentially distributed with the rates:

$$\mu_{11} = 8\,\text{sec}^{-1}, \quad \mu_{21} = 12\,\text{sec}^{-1}, \quad \mu_{31} = 16\,\text{sec}^{-1},$$
$$\mu_{12} = 24\,\text{sec}^{-1}, \quad \mu_{22} = 32\,\text{sec}^{-1}, \quad \mu_{32} = 36\,\text{sec}^{-1}.$$

The interarrival times are also exponentially distributed with the rates:

$$\lambda_1 = \lambda_2 = 1 \text{ job/sec.}$$

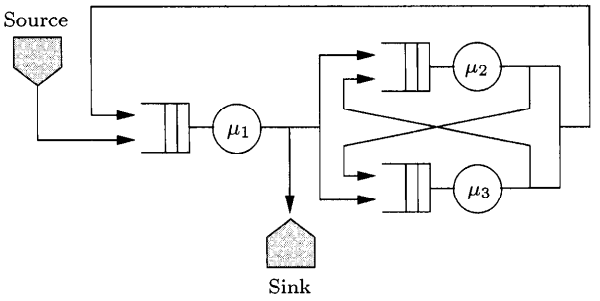


Fig. 7.17 Open queueing network.

The routing probabilities are given by:

$$p_{0,11} = 1, \quad p_{21,11} = 0.6, \quad p_{0,12} = 1, \quad p_{22,12} = 0.7,$$
$$p_{11,21} = 0.4, \quad p_{21,31} = 0.4, \quad p_{12,22} = 0.3, \quad p_{22,32} = 0.3,$$
$$p_{11,31} = 0.3, \quad p_{31,11} = 0.5, \quad p_{12,32} = 0.6, \quad p_{32,12} = 0.4,$$
$$p_{11,0} = 0.3, \quad p_{31,21} = 0.5, \quad p_{12,0} = 0.1, \quad p_{32,22} = 0.6,$$

which means that class switching is not allowed in the network. We wish to compute the probability for the state $(k_1, k_2, k_3) = (3, 2, 1)$ by using the BCMP theorem, Eq. (7.84).

STEP 1 Compute the visit ratios e_{ir} for all $i = 1, \ldots, N$ and $r = 1, \ldots, R$ using Eq. (7.71).

$$e_{11} = p_{0,11} + e_{11}p_{11,11} + e_{21}p_{21,11} + e_{31}p_{31,11} = \underline{3.333},$$
$$e_{21} = p_{0,21} + e_{11}p_{11,21} + e_{21}p_{21,21} + e_{31}p_{31,21} = \underline{2.292},$$
$$e_{31} = p_{0,31} + e_{11}p_{11,31} + e_{21}p_{21,31} + e_{31}p_{31,31} = \underline{1.917}.$$

In the same way we get:

$$e_{12} = \underline{10}, \quad e_{22} = \underline{8.049}, \quad e_{32} = \underline{8.415}.$$

STEP 2 Compute the utilization of each node using Eq. (7.86):

$$\rho_1 = \lambda_1 \frac{e_{11}}{\mu_{11}} + \lambda_2 \frac{e_{12}}{\mu_{12}} = \rho_{11} + \rho_{12} = \underline{0.833},$$

$$\rho_2 = \lambda_1 \frac{e_{21}}{\mu_{21}} + \lambda_2 \frac{e_{22}}{\mu_{22}} = \rho_{21} + \rho_{22} = \underline{0.442},$$

$$\rho_3 = \lambda_1 \frac{e_{31}}{\mu_{31}} + \lambda_2 \frac{e_{32}}{\mu_{32}} = \rho_{31} + \rho_{32} = \underline{0.354}.$$

STEP 3 Compute the other performance measures of the network. In our case we use Eq. (7.87) to determine the mean number of jobs at each node:

$$\overline{K}_{11} = \frac{\rho_{11}}{1-\rho_1} = \underline{2.5}, \quad \overline{K}_{21} = \frac{\rho_{21}}{1-\rho_2} = \underline{0.342}, \quad \overline{K}_{31} = \frac{\rho_{31}}{1-\rho_3} = \underline{0.186},$$

$$\overline{K}_{12} = \underline{2.5}, \quad \overline{K}_{22} = \underline{0.5}, \quad \overline{K}_{32} = \underline{0.362}.$$

STEP 4 Determine the marginal probabilities using Eq. (7.85):

$$\pi_1(3) = (1-\rho_1)\rho_1^3 = \underline{0.0965}, \quad \pi_2(2) = (1-\rho_2)\rho_2^2 = \underline{0.1093},$$
$$\pi_3(1) = (1-\rho_3)\rho_3 = \underline{0.2287}.$$

STEP 5 Compute the state probabilities for the network using Eq. (7.84):

$$\pi(3,2,1) = \pi_1(3) \cdot \pi_2(2) \cdot \pi_3(1) = \underline{0.00241}.$$

An example of the BCMP theorem for closed networks, Eq. (7.80), is not given in this chapter. As shown in Example 7.7, the direct use of the BCMP theorem will require that all states of the network have to be considered in order to compute the normalization constant. This is a very complex procedure and only suitable for small networks because for bigger networks the number of possible states in the network becomes prohibitively large. In Chapter 8 we provide efficient algorithms to analyze closed product-form queueing networks. In these algorithms the states of the queueing network are not explicitly involved in the computation and therefore these algorithms provide much shorter computation time.

Several researchers have extended the class of product-form networks to nodes of the following types:

- In [Spir79] the SIRO (service **i**n **r**andom **o**rder) is examined and it is shown that -/M/1-SIRO nodes fulfill the local balance property and therefore have a product-form solution.

- In [Noet79] the LBPS (**l**ast **b**atch **p**rocessor **s**haring) strategy is introduced and it is show that -/M/1-LBPS node types have product-form solutions. In this strategy the processor is assigned between the two last

batch jobs. If the last batch consists only of one job, then we get the LCFS-PR strategy and if the batch consists of all jobs, we get PS.

- In [ChMa83] the WEIRDP-strategy (**weird**, **p**arameterized strategy) is considered, where the first job in the queue is assigned $100 \cdot p$ % of the processor and the rest of the jobs are assigned $100 \cdot (1-p)$ % of the processor. It is shown that **-**/M/1-WEIRDP nodes have product-form solution.

- In [CHT77] it is shown that -/G/1-PS, -/G/∞-IS, and -/G/1-LCFS-PR nodes with arbitrary differentiable service time distribution also have product-form solution.

Furthermore, in [Tows80] and [Krze87], the class of product-form networks is extended to networks where the probability to enter a particular node depends on the number of jobs at that node. In this case, we have so-called *load-dependent routing probabilities.*

Problem 7.2 Consider an open queueing network with $N = 3$ nodes, FCFS queueing discipline, and exponentially distributed service times with the mean values:

$$\frac{1}{\mu_1} = 0.08\,\text{sec}, \quad \frac{1}{\mu_2} = 0.06\,\text{sec}, \quad \frac{1}{\mu_3} = 0.04\,\text{sec}.$$

Only at the first node do jobs arrive from outside with exponentially distributed interarrival times and the rate $\lambda_{01} = 4$ jobs/sec. Node 1 is a multiple server node with $m_1 = 2$ server, nodes 2 and 3 are single server nodes. The routing probabilities are given as follows:

$$\begin{aligned} &p_{11} = 0.2, \quad p_{21} = 1, \quad p_{31} = 0.5, \\ &p_{12} = 0.4, \quad p_{30} = 0.5, \\ &p_{13} = 0.4. \end{aligned}$$

(a) Draw the queueing network.

(b) Determine the steady-state probability for the state $\mathbf{k} = (4, 3, 2)$.

(c) Determine all performance measures.

Note: Use the Jackson's theorem.

Problem 7.3 Determine the CPU utilization and other performance measures of a central server model with $N = 3$ nodes and $K = 4$ jobs. Each node has only one server and the queueing discipline at each node is FCFS. The exponentially distributed service times have the respective means:

$$\frac{1}{\mu_1} = 2\ \text{msec}, \quad \frac{1}{\mu_2} = 5\ \text{msec}, \quad \frac{1}{\mu_3} = 5\ \text{msec},$$

and the routing probabilities are:

$$p_{11} = 0.3, \quad p_{12} = 0.5, \quad p_{13} = 0.2, \quad p_{21} = p_{31} = 1.$$

Problem 7.4 Consider a closed queueing network with $K = 3$ nodes, $N = 3$ jobs, and the service discipline FCFS. The first node has $m_1 = 2$ servers and the other two nodes have one server each. The routing probabilities are given by:

$$\begin{aligned} p_{11} &= 0.6, \quad p_{21} = 0.5, \quad p_{31} = 0.4, \\ p_{12} &= 0.3, \quad p_{22} = 0.0, \quad p_{32} = 0.6, \\ p_{12} &= 0.1, \quad p_{23} = 0.5, \quad p_{33} = 0.0. \end{aligned}$$

The service times are exponentially distributed with the rates

$$\mu_1 = 0.4\,\text{sec}^{-1}, \quad \mu_2 = 0.6\,\text{sec}^{-1}, \quad \mu_3 = 0.3\,\text{sec}^{-1}.$$

(a) Draw the queueing network.

(b) What is the steady-state probability that there are two jobs at node 2?

(c) Determine all performance measures.

(d) Solve this problem also with the local and global balance equations respectively and compare the solution complexity for the two different methods.

Note: Use the Gordon/Newell Theorem for parts (*a*), (*b*) and (*c*).

Problem 7.5 Consider a closed queueing network with $N = 3$ nodes, $K = 3$ jobs, and exponentially distributed service times with the rates:

$$\mu_1 = 0.72\,\text{sec}^{-1}, \quad \mu_2 = 0.64\,\text{sec}^{-1}, \quad \mu_3 = 1\,\text{sec}^{-1},$$

and the routing probabilities:

$$p_{31} = 0.4, \quad p_{32} = 0.6, \quad p_{13} = p_{23} = 1.$$

Determine all performance measures.

Problem 7.6 Consider the following routing matrix **P**:

P	(1,1)	(1,2)	(1,3)	(1,4)	(2,1)	(2,2)	(2,3)	(2,4)
(1,1)	0.5	0	0	0	0.25	0	0.25	0
(1,2)	0	0.5	0	0.5	0	0	0	0
(1,3)	0.5	0	0	0	0	0	0.5	0
(1,4)	0	0	0	0.5	0	0.25	0	0.25
(2,1)	0.5	0	0	0	0.5	0	0	0
(2,2)	0	0.5	0	0	0	0.5	0	0
(2,3)	0	0	1	0	0	0	0	0
(2,4)	0	0	0	1	0	0	0	0

At the beginning, the jobs are distributed as follows over the four different classes:

$$K_1' = K_2' = K_3' = K_4' = 5.$$

(a) Determine the disjoint chains.

(b) Determine the number of states in the CTMC underlying the network.

(c) Determine the visit ratios.

Problem 7.7 Consider an open network with $N = 2$ nodes and $R = 2$ job classes. Node 1 is of Type-2 and node 2 of Type-4. The service rates are

$$\mu_{11} = 4\,\text{sec}^{-1}, \quad \mu_{12} = 5\,\text{sec}^{-1}, \quad \mu_{21} = 6\,\text{sec}^{-1}, \quad \mu_{22} = 2\,\text{sec}^{-1},$$

and the arrival rates per class are

$$\lambda_1 = \lambda_2 = 1\,\text{sec}^{-1}.$$

The routing probabilities are given as:

$$\begin{aligned} &p_{0,11} = p_{0,12} = 1, \quad p_{21,11} = p_{22,12} = 1, \\ &p_{11,21} = 0.6, \quad p_{12,22} = 0.4 \quad p_{11,0} = 0.4, \quad p_{12,0} = 0.6. \end{aligned}$$

With the BCMP theorem, Version 2, determine

(a) All visit ratios.

(b) The utilizations of node 1 and node 2.

(c) The steady-state probability for state (2,1).

8

Algorithms for Product-Form Networks

Although product-form solutions can be expressed very easily as formulae, the computation of state probabilities in a closed queueing network is very time consuming if a straightforward computation of the normalization constant using Eq. (7.3.5) is carried out. As seen in Example 7.7, considerable computation is needed to analyze even a single class network with a small number of jobs, primarily because the formula makes a pass through all the states of the underlying CTMC. Therefore we need to develop efficient algorithms to reduce the computation time [Buze71].

Many efficient algorithms for calculating performance measures of closed product-form queueing networks have been developed. The most important ones are the *convolution algorithm* and the *mean value analysis* (MVA) [ReLa80]. The convolution algorithm is an efficient iterative technique for calculating the normalization constant, which is needed when all performance measures are computed using a set of simple equations. In contrast, the mean value analysis is an iterative technique where the mean values of the performance measures can be computed directly without computing the normalization constant. We also introduce the *RECAL algorithm* [CoGe86] (**re**cursion by **ch**ain **al**gorithm), which is well suited for networks with many job classes. The fourth and last method for analyzing product-form networks presented in detail is the so-called *flow-equivalent server method* [CHW75b, ABP85]. This method is well suited when we are especially interested in computing the performance of a single station or a part of the network.

There are several other algorithms that we do not cover in detail due to space limitations. Based on the mean value analysis, [ChSa80] and [SaCh81] developed an algorithm called *LBANC* (local balance algorithm for normal-

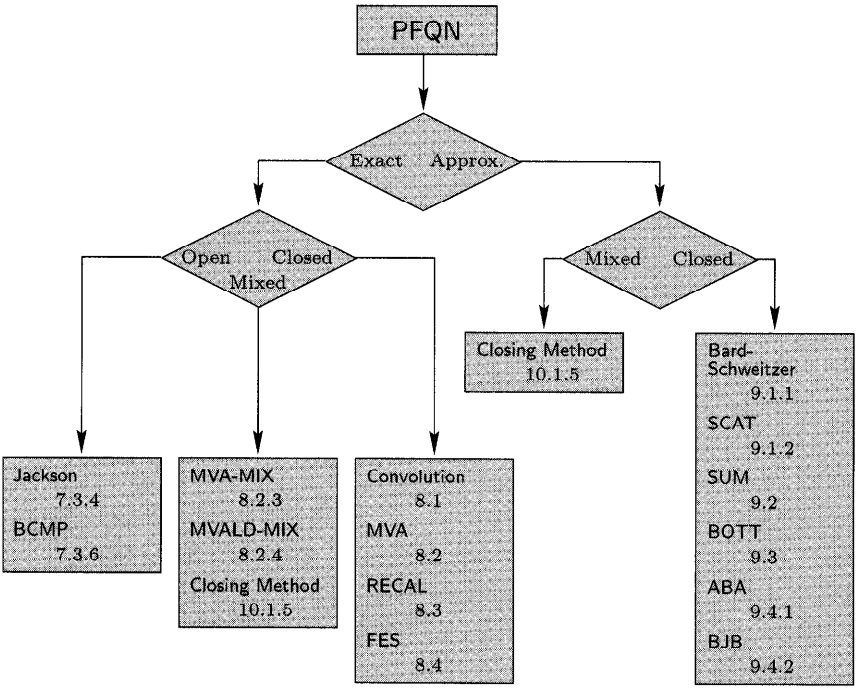


Fig. 8.1 Flowchart describing sections of Chapters 8 and 9.

izing constants). This algorithm iteratively computes the normalization constant and the performance measures. It is very well suited for networks with a small number of nodes but a large number of jobs. The *CCNC algorithm* (coalesce computation of normalizing constants) [ChSa80] is especially used when storage space is limited. That the convolution algorithm, the mean value analysis, and LBANC can be derived from each other has been shown by [Lam81]. The *DAC algorithm* (distribution analysis by chain) [SoLa89] is used to directly compute the state probabilities of a product-form queueing network. A generalization of the MVA to higher moments is introduced in [Stre86]. Other algorithms for determining the normalization constant are the algorithms of [Koba78] and [Koba79] (which are based on the enumeration of polynomials of Polya), and the *partial fraction method* of [Moor72].

For very large networks, execution time of the preceding algorithms is not acceptable. Many approximation algorithms have been developed for this purpose. Exact and approximate algorithms are discussed in this chapter and in Chapter 9 and are summarized in Fig. 8.1.

8.1 THE CONVOLUTION ALGORITHM

The convolution algorithm was one of the first efficient algorithms for analyzing closed product-form queueing networks and is still in use today. The name of this technique reflects the method of determining the normalization constant $G(K)$ from the functions $F_i(k_i)$, which is similar to the convolution of two probability mass functions. Once the normalization constant is computed, the system performance measures of interest can be easily derived [Buze71]. We recall that the convolution operator $\otimes$ is defined in the following way: Let A, B, C be vectors of length $K+1$. Then the convolution $C = A \otimes B$ is defined as:

$$C(k) = \sum_{j=0}^{k} A(j) \cdot B(k-j), \quad k = 0, \ldots, K. \tag{8.1}$$

First we consider the convolution algorithm for single class closed queueing networks. Then this technique is extended to multiclass closed queueing networks.

8.1.1 Single Class Closed Networks

According to the BCMP theorem, Eq. (7.80), the state probabilities of a closed single class product-form queueing network with N nodes can be expressed in the following way:

$$\pi(k_1, \ldots, k_N) = \frac{1}{G(K)} \prod_{i=1}^{N} F_i(k_i),$$

where the functions $F_i(k_i)$ are defined in Eq. (7.61), and:

$$G(K) = \sum_{\sum_{i=1}^{N} k_i = K} \prod_{i=1}^{N} F_i(k_i)$$

is the normalization constant of the network.

The computation of $G(K)$ is carried out by iterating over the number of nodes in the network and over the number of possible jobs at each node. For this purpose the following auxiliary functions $G_n(k)$, $n = 1, \ldots, N$ and $k = 0, \ldots, K$ are defined as follows:

$$G_n(k) = \sum_{\sum_{i=1}^{n} k_i = k} \prod_{i=1}^{n} F_i(k_i). \tag{8.2}$$

The desired normalization constant is then:

$$G(K) = G_N(K). \tag{8.3}$$

From Eq. (8.2) for $n > 1$, it follows that:

$$G_n(k) = \sum_{j=0}^{k} \left(\sum_{\substack{\sum_{i=1}^{n} k_i = k \\ \& k_n = j}} \prod_{i=1}^{n} F_i(k_i) \right) = \sum_{j=0}^{k} F_n(j) \left(\sum_{\substack{\sum_{i=1}^{n} k_i = k_j \\ \& k_n = 0}} \prod_{i=1}^{n-1} F_i(k_i) \right)$$

$$= \sum_{j=0}^{k} F_n(j) \cdot G_{n-1}(k-j). \tag{8.4}$$

For $n = 1$ we have:

$$G_1(k) = F_1(k), \quad k = 1, \ldots, K. \tag{8.5}$$

The initial condition is of the form:

$$G_n(0) = 1, \quad n = 1, \ldots, N. \tag{8.6}$$

The convolution method for computing the normalization constant $G(K)$ is fully determined by Eqs. (8.4) to (8.6). The $(K+1)$-dimensional vectors:

$$G_n = \begin{pmatrix} G_n(0) \\ \vdots \\ G_n(K) \end{pmatrix}, \quad n = 1, \ldots, N,$$

are therefore determined by the convolution:

$$G_n = F_n \otimes G_{n-1}$$

where:

$$F_n = \begin{pmatrix} F_n(0) \\ \vdots \\ F_n(K) \end{pmatrix}.$$

The computation of $G_n(k)$ is easily visualized as in Fig. 8.2. In this figure we show the values of the functions needed in the calculation of $G_N(K)$. The real objective of the algorithm is to determine the last value in the last column because this value is the normalization constant $G(K)$ that we are looking for. The values $G_N(k)$ for $k = 1, \ldots, K-1$ in the last column are also useful in determining the system performance measures. Buzen [Buze71, Buze73] has developed algorithms that are based on the normalization constant $G(K)$ and the functions $F_i(k_i)$ to compute all important performance measures of the queueing network without needing the underlying CTMC state probabilities.

Computation of the performance measures using the normalization constant is now illustrated.

	1	⋯	$n-1$		n	⋯	N
0	1	⋯	$G_{n-1}(0)(\cdot F_n(k))$		1		$1 = G(1)$
1	$F_1(1)$	⋯	$G_{n-1}(1)(\cdot F_n(k-1))$		$G_n(1)$		
⋮	⋮		⋮		⋮		
$k-1$	$F_1(k-1)$	⋯	$G_{n-1}(k-1)(\cdot F_n(1))$	$\sum$			
k	$F_1(k)$	⋯	$G_{n-1}(k)(\cdot F_n(0))$	→	$G_n(k)$	⋯	$G_N(k) = G(k)$
⋮	⋮				⋮		
K	$F_1(K)$				$G_n(K)$		$G_N(K) = G(K)$

Fig. 8.2 Computation of $G_n(k)$.

(a) The *marginal probability* that there are exactly $k_i = k$ jobs at the ith node is given by Eq. (7.16):

$$\pi_i(k) = \sum_{\substack{\sum_{j=1}^{N} k_j = K \\ \& k_i = k}} \pi(k_1, \ldots, k_N).$$

If we substitute Eq. (7.59) we get:

$$\pi_i(k) = \sum_{\substack{\sum_{j=1}^{N} k_j = K \\ \& k_i = k}} \frac{1}{G(K)} \prod_{j=1}^{N} F_j(k_j) = \frac{F_i(k)}{G(K)} \sum_{\substack{\sum_{j=1}^{N} k_j = K \\ \& k_i = k}} \prod_{\substack{j=1 \\ \& j \neq i}}^{N} F_j(k_j)$$

$$= \frac{F_i(k)}{G(K)} \cdot G_N^{(i)}(K-k). \tag{8.7}$$

Then $G_N^{(i)}(k)$ can be interpreted as the normalization constant of the network with k jobs and node i removed from the network.

$$G_N^{(i)}(k) = \sum_{\substack{\sum_{j=1}^{N} k_j = K \\ \& k_i = K-k}} \prod_{\substack{j=1 \\ \& j \neq i}}^{N} F_j(k_j). \tag{8.8}$$

By definition we have:

$$G_{N-1}(k) = G_{N-1}^{(N)}(k) = G_N^{(N)}(k) \quad \text{for } k = 0, \ldots, K.$$

Because of Eq. (8.7), for node N we have [Buze71, Buze73]:

$$\pi_N(k) = \frac{F_N(k)}{G(K)} \cdot G_{N-1}(K-k). \tag{8.9}$$

In [BBS77], a simple algorithm for computing the $G_N^{(i)}(k)$ for $k = 0, \ldots, K$ is presented. Since the sum of all marginal probabilities is 1, it follows from Eq. (8.7):

$$\sum_{j=0}^{K} \pi_i(j) = \sum_{j=0}^{K} \frac{F_i(j)}{G(K)} G_N^{(i)}(K-j) = 1,$$

and therefore we have:

$$G(K) = \sum_{j=0}^{K} F_i(j) \cdot G_N^{(i)}(K-j), \quad j = 1, \ldots, N. \tag{8.10}$$

With the help of these equations we can derive an iterative formula for computing the $G_N^{(i)}(k)$, for $k = 0, \ldots, K$:

$$G_N^{(i)}(k) = G(k) - \sum_{j=1}^{k} F_i(j) \cdot G_N^{(i)}(k-j), \tag{8.11}$$

with the initial condition:

$$G_N^{(i)}(0) = G(0) = 1, \quad i = 1, \ldots, N. \tag{8.12}$$

In the case of $m_i = 1$ the preceding formulae can be considerably simplified.

Because of Eq. (8.11) we have:

$$\begin{aligned} G_N^{(i)}(k) &= G(k) - \sum_{j=0}^{k-1} F_i(j+1) G_N^{(i)}(k-1-j) \\ &= G(k) - \frac{e_i}{\mu_i} \sum_{j=0}^{k-1} F_i(j) G_N^{(i)}(k-1-j) \\ &= G(k) - \frac{e_i}{\mu_i} G(k-1). \end{aligned}$$

After inserting this result in Eq. (8.7), we get for the marginal probabilities:

$$\pi_i(k) = \left(\frac{e_i}{\mu_i}\right)^k \cdot \frac{1}{G(K)} \cdot \left(G(K-k) - \frac{e_i}{\mu_i} \cdot G(K-k-1)\right), \tag{8.13}$$

where $G(k) = 0$ for $k < 0$.

(b) The *throughput* of node i in the load-dependent or load-independent case is given by the formula:

$$\lambda(K) = \frac{G(K-1)}{G(K)} \quad \text{and} \quad \lambda_i(K) = e_i \cdot \frac{G(K-1)}{G(K)}. \tag{8.14}$$

Proof: By definition, the throughput is given by Eq. (7.22):

$$\begin{aligned}\lambda_i(k) &= \sum_{k=1}^{K} \pi_i(k)\mu_i(k) = \sum_{k=1}^{K} \frac{F_i(k)}{G(K)} G_N^{(i)}(K-k)\mu_i(k) \\ &= \sum_{k=1}^{K} \frac{e_i}{\mu_i(k)} \frac{F_i(k-1)}{G(K)} G_N^{(i)}(K-k)\mu_i(k) \\ &= \frac{e_i}{G(K)} \sum_{k=1}^{K} F_i(k-1) G_N^{(i)}(K-k) \\ &= \frac{e_i}{G(K)} \sum_{k=0}^{K-1} F_i(k) G_N^{(i)}(K-1-k) = e_i \frac{G(K-1)}{G(K)}. \quad q.e.d.\end{aligned}$$

(c) The *utilization* of a node in the load-independent case can be determined by inserting Eq. (8.14) in the well known relation $\rho_i = \lambda_i/(m_i\mu_i)$:

$$\rho_i = \frac{e_i}{m_i\mu_i} \cdot \frac{G(K-1)}{G(K)}. \tag{8.15}$$

(d) The *mean number of jobs* for a single server node can be computed from following simplified equation [Buze71]:

$$\overline{K}_i = \sum_{k=1}^{K} \left(\frac{e_i}{\mu_i}\right)^k \cdot \frac{G(K-k)}{G(K)}. \tag{8.16}$$

(e) The *mean response time* of jobs at node i can be determined with the help of Little's theorem. For a single server node we have:

$$\overline{T}_i = \frac{\overline{K}_i}{\lambda_i} = \sum_{k=1}^{K} \left(\frac{e_i}{\mu_i}\right)^k \cdot \frac{G(K-k)}{e_i \cdot G(K-1)}. \tag{8.17}$$

The use of the convolution method for determining the normalization constant is now demonstrated by a simple example. We also show how to compute other performance measures from these results.

Example 8.1 Consider the following closed queueing network (Fig. 8.3) with $N = 3$ nodes and $K = 3$ jobs. The first node has $m_1 = 2$ and the second node has $m_2 = 3$ identical service stations. For the third node we have $m_3 = 1$. The service time at each node is exponentially distributed with respective rates:

$$\mu_1 = 0.8\,\text{sec}^{-1}, \quad \mu_2 = 0.6\,\text{sec}^{-1}, \quad \mu_3 = 0.4\,\text{sec}^{-1}.$$

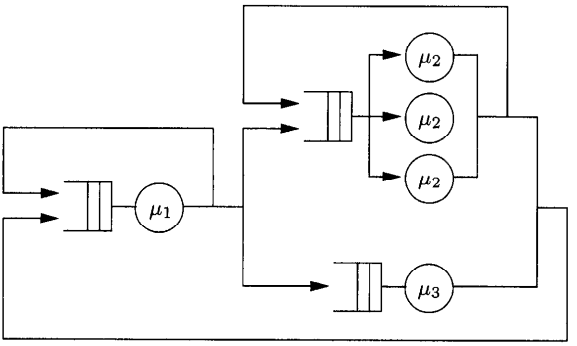


Fig. 8.3 A closed queueing network.

The visit ratios are given as follows:

$$e_1 = 1, \quad e_2 = 0.667, \quad e_3 = 0.2.$$

From Eq. (7.61) it follows for the functions $F_i(k_i)$, $i = 1, 2, 3$:

$$\begin{aligned}
F_1(0) &= \underline{1}, & F_2(0) &= \underline{1}, & F_3(0) &= \underline{1}, \\
F_1(1) &= \underline{1.25}, & F_2(1) &= \underline{1.111}, & F_3(1) &= \underline{0.5}, \\
F_1(2) &= \underline{0.781}, & F_2(2) &= \underline{0.617}, & F_3(2) &= \underline{0.25}, \\
F_1(3) &= \underline{0.488}, & F_2(3) &= \underline{0.229}, & F_3(3) &= \underline{0.125}.
\end{aligned}$$

In Table 8.1 the intermediate results as well as the final result of computing $G(K)$ are shown.

Table 8.1 Computation of the normalization constant

	1	2	N=3
0	1	1	1
1	1.25	2.361	2.861
2	0.781	2.787	4.218
K=3	0.488	2.356	4.465

Thus the normalization constant $G(K) = \underline{4.465}$. The marginal probabilities for the single server node 3 can be computed using Eq. (8.13):

$$\pi_3(0) = \left(\frac{e_3}{\mu_3}\right)^0 \cdot \frac{1}{G(3)} \cdot \left(G(3) - \frac{e_3}{\mu_3} G(2)\right) = \underline{0.528},$$

$$\pi_3(1) = \left(\frac{e_3}{\mu_3}\right)^1 \cdot \frac{1}{G(3)} \cdot \left(G(2) - \frac{e_3}{\mu_3} G(1)\right) = \underline{0.312},$$

$$\pi_3(2) = \left(\frac{e_3}{\mu_3}\right)^2 \cdot \frac{1}{G(3)} \cdot \left(G(1) - \frac{e_3}{\mu_3} G(0)\right) = \underline{0.132},$$

$$\pi_3(3) = \left(\frac{e_3}{\mu_3}\right)^3 \cdot \frac{1}{G(3)} \cdot \left(G(0) - \frac{e_3}{\mu_3} \cdot 0\right) = \underline{0.028}.$$

For the computation of the marginal probabilities of nodes 1 and 2, we need the values $G_N^{(i)}(k)$ for $k = 0, 1, 2, 3$ and $i = 1, 2$. With Eq. (8.11) we get:

$$G_N^{(1)}(0) = \underline{1},$$
$$G_N^{(1)}(1) = G(1) - F_1(1) G_N^{(1)}(0) = \underline{1.611},$$
$$G_N^{(1)}(2) = G(2) - (F_1(1) G_N^{(1)}(1) + F_1(2) G_N^{(1)}(0)) = \underline{1.423},$$
$$G_N^{(1)}(3) = G(3) - (F_1(1) G_N^{(1)}(2) + F_1(2) G_N^{(1)}(1) + F_1(3) G_N^{(1)}(0)) = \underline{0.940}.$$

In the same way we compute:

$$G_N^{(2)}(0) = \underline{1}, \quad G_N^{(2)}(1) = \underline{1.656}, \quad G_N^{(2)}(2) = \underline{1.75}, \quad G_N^{(2)}(3) = \underline{1.316}.$$

With Eq. (8.7) the marginal probabilities are:

$$\pi_1(0) = \frac{F_1(0)}{G(3)} G_N^{(1)}(3) = \underline{0.211}, \quad \pi_1(1) = \frac{F_1(1)}{G(3)} G_N^{(1)}(2) = \underline{0.398},$$
$$\pi_1(2) = \frac{F_1(2)}{G(3)} G_N^{(1)}(1) = \underline{0.282}, \quad \pi_1(3) = \frac{F_1(3)}{G(3)} G_N^{(1)}(0) = \underline{0.109},$$

and

$$\pi_2(0) = \underline{0.295}, \quad \pi_2(1) = \underline{0.412}, \quad \pi_2(2) = \underline{0.242}, \quad \pi_2(3) = \underline{0.051}.$$

The throughputs can be computed using Eq. (8.14):

$$\lambda_1 = e_1 \frac{G(2)}{G(3)} = \underline{0.945}, \quad \lambda_2 = e_2 \frac{G(2)}{G(3)} = \underline{0.630}, \quad \lambda_3 = e_3 \frac{G(2)}{G(3)} = \underline{0.189}.$$

The utilizations are given by Eq. (7.21):

$$\rho_1 = \frac{\lambda_1}{m_1 \mu_1} = \underline{0.590}, \quad \rho_2 = \frac{\lambda_2}{m_2 \mu_2} = \underline{0.350}, \quad \rho_3 = \frac{\lambda_3}{\mu_3} = \underline{0.473}.$$

The mean number of jobs at the multiserver nodes is given by Eq. (7.26):

$$\overline{K}_1 = \pi_1(1) + 2\pi_1(2) + 3\pi_1(3) = \underline{1.290},$$
$$\overline{K}_2 = \pi_2(1) + 2\pi_2(2) + 3\pi_2(3) = \underline{1.050},$$

where we use Eq. (8.16) for the single server node 3:

$$\overline{K}_3 = \left(\frac{e_3}{\mu_3}\right) \frac{G(2)}{G(3)} + \left(\frac{e_3}{\mu_3}\right)^2 \frac{G(1)}{G(3)} + \left(\frac{e_3}{\mu_3}\right)^3 \frac{G(0)}{G(3)} = \underline{0.660}.$$

For the mean response time we use Eq. (7.43):

$$\overline{T}_1 = \frac{\overline{K}_1}{\lambda_1} = \underline{1.366}, \quad \overline{T}_2 = \frac{\overline{K}_2}{\lambda_2} = \underline{1.667}, \quad \overline{T}_3 = \frac{\overline{K}_3}{\lambda_3} = \underline{3.496}.$$

8.1.2 Multiclass Closed Networks

The convolution method introduced in Section 8.1.1 can be extended to the case of multiclass closed product-form queueing networks. According to the BCMP theorem, Eq. (7.80), the state probabilities for closed product-form queueing networks are given by:

$$\pi(\mathbf{S}_1, \ldots, \mathbf{S}_N) = \frac{1}{G(\mathbf{K})} \prod_{i=1}^{N} F_i(\mathbf{S}_i),$$

where the functions $F_i(\mathbf{S}_i)$ are defined in Eq. (7.82). The determination of the normalization constant of the network:

$$G(\mathbf{K}) = \sum_{\sum_{i=1}^{N} \mathbf{S}_i = \mathbf{K}} \prod_{i=1}^{N} F_i(\mathbf{S}_i),$$

is analogous to the single class case. For now, we assume that class switching is not allowed and thus the number of jobs in each class is constant. Corresponding to Eq. (8.2), we define for $\mathbf{k} = \mathbf{0}, \ldots, \mathbf{K}$ the auxiliary functions:

$$G_n(\mathbf{k}) = \sum_{\sum_{i=1}^{n} \mathbf{S}_i = \mathbf{k}} \prod_{i=1}^{n} F_i(\mathbf{S}_i) \tag{8.18}$$

and determine for $n = 1, \ldots, N$ the matrix G_n by convolution:

$$G_n = F_n \otimes G_{n-1},$$

with the initial condition $G_1(.) = F_1(.)$. For $n > 1$ we also have:

$$G_n(\mathbf{k}) = \sum_{j_1=0}^{k_1} \cdots \sum_{j_R=0}^{k_R} F_n(\mathbf{j}) \cdot G_{n-1}(\mathbf{k} - \mathbf{j}). \tag{8.19}$$

The computation of the normalization constant was simplified by [MuWo74b] and [Wong75] by introducing a much simpler iterative formula for $G(\mathbf{K}) = G_N(\mathbf{K})$. For this purpose, we define the vector:

$$\mathbf{k}^{(n)} = \sum_{i=1}^{n} \mathbf{S}_i = \left(k_1^{(n)}, \ldots, k_R^{(n)}\right),$$

where $k_r^{(n)}, 1 \leq r \leq R$, is the overall number of jobs of class r at the nodes $1, 2, \ldots, n-1, n$. We then have:

$$k_r^{(n)} = \sum_{i=1}^{n} k_{ir} \quad \text{and} \quad \mathbf{k}^{(N)} = \mathbf{K}.$$

With the help of these expressions, it is possible to rewrite Eq. (8.18) in the following manner:

$$G_n(\mathbf{k}^{(n)}) = \sum_{\sum_{i=1}^{n} \mathbf{S}_i = \mathbf{k}^{(n)}} \prod_{i=1}^{n} F_i(\mathbf{S}_i). \tag{8.20}$$

Thus:

$$G_n(\mathbf{k}^{(n)}) = \sum_{\mathbf{k}^{(n-1)} + \mathbf{S}_n = \mathbf{k}^{(n)}} \sum_{\sum_{i=1}^{n-1} \mathbf{S}_i = \mathbf{k}^{(n-1)}} \prod_{i=1}^{n-1} F_i(\mathbf{S}_i) \cdot F_n(\mathbf{S}_n)$$

$$= \sum_{\mathbf{k}^{(n-1)} + \mathbf{S}_n = \mathbf{k}^{(n)}} G_{n-1}(\mathbf{k}^{(n-1)}) \cdot F_n(\mathbf{S}_n). \tag{8.21}$$

Equation (8.21) together with the initial conditions $G_1(.) = F_1(.)$ completes the description of the algorithm of [MuWo74b] and [Wong75]. For the normalization constant we have:

$$G(\mathbf{K}) = G_N(\mathbf{k}^{(N)}). \tag{8.22}$$

The computation of the performance measures is as follows:

(a) According to Eq. (7.32) the *marginal probability* that there are exactly $\mathbf{S}_i = \mathbf{k}$ jobs at node i is given by:

$$\pi_i(\mathbf{k}) = \sum_{\substack{\sum_{j=1}^{N} \mathbf{S}_j = \mathbf{K} \\ \& \mathbf{S}_i = \mathbf{k}}} \pi(\mathbf{S}_1, \ldots, \mathbf{S}_N) = \frac{1}{G(\mathbf{K})} \sum_{\substack{\sum_{j=1}^{N} \mathbf{S}_j = \mathbf{K} \\ \& \mathbf{S}_i = \mathbf{k}}} \prod_{j=1}^{N} F_j(\mathbf{S}_j)$$

$$= \frac{F_i(\mathbf{k})}{G(\mathbf{K})} \sum_{\substack{\sum_{j=1}^{N} \mathbf{S}_j = \mathbf{K} \\ \& \mathbf{S}_i = \mathbf{k}}} \prod_{\substack{j=1 \\ \& j \neq i}}^{N} F_j(\mathbf{S}_j) = \frac{F_i(\mathbf{k})}{G(\mathbf{K})} G_N^{(i)}(\mathbf{K} - \mathbf{k}). \tag{8.23}$$

Then $G_N^{(i)}(\mathbf{K})$ can again be interpreted as the normalization constant of the network without node i:

$$G_N^{(i)}(\mathbf{k}) = \sum_{\substack{\sum_{j=1}^{N} \mathbf{S}_j = \mathbf{K} \\ \& \mathbf{S}_i = \mathbf{K}-\mathbf{k}}} \prod_{\substack{j=1 \\ \& j \neq i}}^{N} F_j(\mathbf{S}_j). \tag{8.24}$$

In the same way as in the single class case (Eq. (8.11)), the iterative formula for computing the normalization constant $G_N^{(i)}(\mathbf{k})$ is given as follows:

$$G_N^{(i)}(\mathbf{k}) = G(\mathbf{k}) - \sum_{\mathbf{j}=\mathbf{0}}^{\mathbf{k}} \delta(\mathbf{j}) \cdot F_i(\mathbf{j}) \cdot G_N^{(i)}(\mathbf{k}-\mathbf{j}), \tag{8.25}$$

with $\delta(\mathbf{j})$ defined by:

$$\delta(\mathbf{j}) = \begin{cases} 0, & \text{if } \mathbf{j} = \mathbf{0}, \\ 1, & \text{otherwise} \end{cases} \tag{8.26}$$

and affecting the computation only for $\mathbf{j} = \mathbf{0}$. The initial condition is:

$$G_N^{(i)}(\mathbf{0}) = G(\mathbf{0}) = 1, \quad i = 1, \ldots, N. \tag{8.27}$$

(b) The *throughput* λ_{ir} of class-r jobs at the ith node can be expressed as follows [Wong75]:

$$\lambda_{ir} = e_{ir} \frac{G(K_1, \ldots, K_r - 1, \ldots, K_R)}{G(K_1, \ldots, K_r, \ldots, K_R)},$$

where e_{ir} is the solution of Eq. (7.72). With $(\mathbf{K} - \mathbf{1}_r) = (K_1, \ldots, K_r - 1, \ldots, K_R)$, we have the following simpler form:

$$\lambda_{ir} = e_{ir} \frac{G(\mathbf{K} - \mathbf{1}_r)}{G(\mathbf{K})}. \tag{8.28}$$

This formula holds for load-dependent and load-independent nodes of all four types. The proof can be found in [BrBa80].

(c) The *utilization* ρ_{ir} is given by Eq. (7.34). If the service rates are constant, then we have the following simplification because of $\lambda_{ir}/(m_i \mu_{ir})$:

$$\rho_{ir} = \frac{\lambda_{ir}}{m_i \mu_{ir}} = \frac{e_{ir}}{m_i \mu_{ir}} \cdot \frac{G(\mathbf{K} - \mathbf{1}_r)}{G(\mathbf{K})}. \tag{8.29}$$

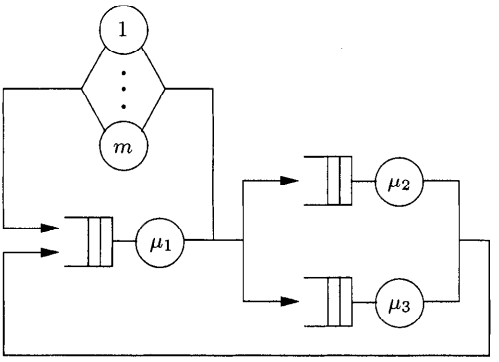


Fig. 8.4 An example of a two-class closed network.

Table 8.2 Computation of the functions $F_i(\mathbf{S}_i), i = 1,2,3,4,$

$\mathbf{S}_i$	$F_1(\mathbf{S}_1)$	$F_2(\mathbf{S}_2)$	$F_3(\mathbf{S}_3)$	$F_4(\mathbf{S}_4)$
(0,0)	1	1	1	1
(1,0)	1	1.6	3.2	2.4
(0,1)	2	2	3	4.8
(1,1)	4	6.4	19.2	11.52
(0,2)	4	4	9	11.52
(1,2)	12	19.2	86.4	27.648

Example 8.2 Consider the network given in Fig. 8.4 consisting of $N = 4$ nodes and $R = 2$ job classes. In class 1 the number of jobs is $K_1 = 1$ and in class 2, $K_2 = 2$. It is assumed that class switching is not allowed. The first node is of Type-2, the second and third nodes are of Type-4, and the fourth node is of Type-3 with mean service times:

$$\frac{1}{\mu_{11}} = 1\,\text{sec}, \quad \frac{1}{\mu_{21}} = 4\,\text{sec}, \quad \frac{1}{\mu_{31}} = 8\,\text{sec}, \quad \frac{1}{\mu_{41}} = 12\,\text{sec},$$
$$\frac{1}{\mu_{12}} = 2\,\text{sec}, \quad \frac{1}{\mu_{22}} = 5\,\text{sec}, \quad \frac{1}{\mu_{32}} = 10\,\text{sec}, \quad \frac{1}{\mu_{42}} = 16\,\text{sec}.$$

The visit ratios are given by:

$$e_{11} = 1, \quad e_{21} = 0.4, \quad e_{31} = 0.4, \quad e_{41} = 0.2,$$
$$e_{12} = 1, \quad e_{22} = 0.4, \quad e_{32} = 0.3, \quad e_{42} = 0.3.$$

First, with the help of Eq. (7.82), the functions $F_i(\mathbf{S}_i), i = 1,2,3,4,$ are computed (Table 8.2). Then for determining the normalization constant we compute the $G_n(\mathbf{k}^{(n)})$ from Eq. (8.21) where $G_1(.) = F_1(.)$. For $n = 2$ we

have:

$$
\begin{aligned}
G_2(0,0) &= G_1(0,0)F_2(0,0) && = \underline{1},\\
G_2(1,0) &= G_1(0,0)F_2(1,0) + G_1(1,0)F_2(0,0) && = \underline{2.6},\\
G_2(0,1) &= G_1(0,0)F_2(0,1) + G_1(0,1)F_2(0,0) && = \underline{4},\\
G_2(1,1) &= G_1(0,0)F_2(1,1) + G_1(1,1)F_2(0,0)\\
&\quad + G_1(1,0)F_2(0,1) + G_1(0,1)F_2(1,0) && = \underline{15.6},\\
G_2(0,2) &= G_1(0,0)F_2(0,2) + G_1(0,2)F_2(0,0)\\
&\quad + G_1(0,1)F_2(0,1) && = \underline{12},\\
G_2(1,2) &= G_1(0,0)F_2(1,2) + G_1(1,2)F_2(0,0)\\
&\quad + G_1(1,1)F_2(0,1) + G_1(0,1)F_2(1,1)\\
&\quad + G_1(1,0)F_2(0,2) + G_1(0,2)F_2(1,0) && = \underline{62.4}.
\end{aligned}
$$

In the same way we can compute the values for $G_3(\mathbf{k}^{(3)})$ and $G_4(\mathbf{k}^{(4)})$ as summarized in Table 8.3. So the normalization constant is $G(\mathbf{K}) = \underline{854.424}$.

Table 8.3 Computation of $G_i(\mathbf{k}^{(i)})$

$\mathbf{k}^{(n)}, 1 \le n \le N$	$G_2(\mathbf{k}^{(2)})$	$G_3(\mathbf{k}^{(3)})$	$G_4(\mathbf{k}^{(4)})$
(0,0)	1	1	1
(1,0)	2.6	5.8	8.2
(0,1)	4	7	11.8
(1,1)	15.6	55.4	111.56
(0,2)	12	33	78.12
(1,2)	62.4	334.2	854.424

Now, with the help of Eq. (7.47), we can compute the marginal probability of 0 jobs of class 1 and 2 jobs of class 2 at node 4:

$$\pi_4(0,2) = \frac{F_4(0,2)}{G(\mathbf{K})} G_N^{(4)}(1,0) = \underline{0.0782},$$

where $G_N^{(4)}(1,0)$ can be obtained from Eq. (8.26):

$$G_N^{(4)}(1,0) = G(1,0) - F_4(1,0) \cdot G_N^{(4)}(0,0) = \underline{5}.$$

Similarly other marginal probabilities at the nodes can be calculated:

$$\pi_4(1,2) = \frac{F_4(1,2)}{G(\mathbf{K})} G_N^{(4)}(0,0) = \underline{0.0324}, \quad \text{with } G_N^{(4)}(0,0) = \underline{1},$$

$$\pi_4(1,1) = \frac{F_4(1,1)}{G(\mathbf{K})} G_N^{(4)}(0,1) = \underline{0.0944}, \quad \text{with } G_N^{(4)}(0,1) = \underline{7},$$

$$\pi_4(0,1) = \frac{F_4(0,1)}{G(\mathbf{K})} G_N^{(4)}(1,1) = \underline{0.3112}, \quad \text{with } G_N^{(4)}(1,1) = \underline{55.4},$$

$$\pi_4(1,0) = \frac{F_4(1,0)}{G(\mathbf{K})} G_N^{(4)}(0,2) = \underline{0.0927}, \quad \text{with } G_N^{(4)}(0,2) = \underline{33},$$

$$\pi_4(0,0) = \frac{F_4(0,0)}{G(\mathbf{K})} G_N^{(4)}(1,2) = \underline{0.3911}, \quad \text{with } G_N^{(4)}(1,2) = \underline{334.2},$$

$$\pi_3(1,2) = \frac{F_3(1,2)}{G(\mathbf{K})} G_N^{(3)}(0,0) = \underline{0.1011}, \quad \text{with } G_N^{(3)}(0,0) = \underline{1},$$

$$\pi_3(1,0) = \frac{F_3(1,0)}{G(\mathbf{K})} G_N^{(3)}(0,2) = \underline{0.1600}, \quad \text{with } G_N^{(3)}(0,2) = \underline{42.72},$$

$$\pi_3(1,1) = \frac{F_3(1,1)}{G(\mathbf{K})} G_N^{(3)}(0,1) = \underline{0.1977}, \quad \text{with } G_N^{(3)}(0,1) = \underline{8.8}.$$

$$\vdots$$

The rest of the marginal probabilities are computed in the same way. The mean number of jobs can be determined with the help of Eq. (7.40). For example, for node 3 the mean number of class 1 and class 2 jobs is respectively given by:

$$\overline{K}_{31} = \pi_3(1,0) + \pi_3(1,1) + \pi_3(1,2) = \underline{0.4588},$$
$$\overline{K}_{32} = \pi_3(0,1) + \pi_3(1,1) + 2\pi_3(0,2) + 2\pi_3(1,2) = \underline{0.678}.$$

For the IS-node 4 the mean number of jobs by class type is:

$$\overline{K}_{41} = \frac{\lambda_{41}}{\mu_{41}} = \underline{0.219}, \quad \overline{K}_{42} = \frac{\lambda_{42}}{\mu_{42}} = \underline{0.627}.$$

The throughputs at each node by class type can be computed with Eq. (8.28):

$$\lambda_{11} = e_{11}\frac{G(0,2)}{G(1,2)} = \underline{0.0914}, \quad \lambda_{12} = e_{12}\frac{G(1,1)}{G(1,2)} = \underline{0.1306},$$

$$\lambda_{21} = e_{21}\frac{G(0,2)}{G(1,2)} = \underline{0.0366}, \quad \lambda_{22} = e_{22}\frac{G(1,1)}{G(1,2)} = \underline{0.0522},$$

$$\lambda_{31} = e_{31}\frac{G(0,2)}{G(1,2)} = \underline{0.0366}, \quad \lambda_{32} = e_{32}\frac{G(1,1)}{G(1,2)} = \underline{0.0392},$$

$$\lambda_{41} = e_{41}\frac{G(0,2)}{G(1,2)} = \underline{0.0183}, \quad \lambda_{42} = e_{42}\frac{G(1,1)}{G(1,2)} = \underline{0.0392}.$$

For the computation of the utilizations of each node by class type, Eq. (8.29) is used to get:

$$\rho_{11} = \underline{0.0914}, \quad \rho_{21} = \underline{0.1463}, \quad \rho_{31} = \underline{0.2926},$$
$$\rho_{12} = \underline{0.2611}, \quad \rho_{22} = \underline{0.2611}, \quad \rho_{32} = \underline{0.3917}.$$

The algorithm we presented in the preceding text is applicable only to networks without class switching. For networks with class switching, [Munt72] proved that a closed queueing network with U ergodic chains is equivalent to a closed network with U job classes without class switching. Therefore, the computation of the normalization constant for networks with class switching is an extension of the technique for networks without class switching. More details are given in [BBS77] and [BrBa80] and Section 7.3.6. In [ReKo75] and [Saue83] the convolution method has been extended for analyzing open queueing networks with load-dependent service rates and for analyzing closed queueing networks with load-dependent routing probabilities. The methods can also be extended for networks with *class specific* service rates. These are networks in which the service rates depend not only on the overall number of jobs of each class at the node, but also on the number of jobs at the node. For this case, [LaLi83] modified the convolution algorithm so as to make the best use of storage space. Their algorithm is known as *tree-convolution.*

Because the computation of the normalization constant can cause numerical problems, other techniques were developed that allow the calculation of the performance measures without using the normalization constant. One of the key development in this regard is the *mean value analysis* (MVA), which we discuss next.

8.2 THE MEAN VALUE ANALYSIS

The MVA was developed by Reiser and Lavenberg [ReLa80] for the analysis of closed queueing networks with product-form solution. The advantage of this method is that the performance measures can be computed without explicitly computing the normalization constant. The method is based on two fundamental equations and it allows us to compute the mean values of measures of interest such as the mean waiting time, throughput, and the mean number of jobs at each node. In these computations only mean values are computed (hence the name). For the case of multiserver nodes ($m_i > 1$), it is necessary, however, to compute the marginal probabilities.

The MVA method is based on two simple laws:

1. *Little's theorem*, which is introduced in Eq. (6.9) to express a relation between the mean number of jobs, the throughput, and the mean response time of a node or the overall system:

$$\overline{K} = \lambda \cdot \overline{T}. \tag{8.30}$$

2. *Theorem of the distribution at arrival time* (in short, *arrival theorem*), proven by [LaRe80] and [SeMi81], for all networks that have a product-form solution. The arrival theorem says that in a closed product-form queueing network, the pmf of the number of jobs seen at the time of arrival to a node i when there are k jobs in the network is equal to the

pmf of the number of jobs at this node with one less job in the network ($= (k-1)$). This property has an intuitive justification [LZGS84]. At the moment a job arrives at a node, it is certain that this job itself is not already in the queue of this node. Thus, there are only $k-1$ other jobs that could possibly interfere with the new arrival. The number of these at the node is simply the number there when only those $(k-1)$ jobs are in the network.

At first we introduce the MVA for single class closed queueing networks and explain it in more detail. This algorithm is then extended to multiclass networks, mixed networks, and networks with load-dependent service rates.

8.2.1 Single Class Closed Networks

The fundamental equation of the mean value analysis is based on the *arrival theorem* for closed product-form networks [ReLa80, SeMi81] and it relates the mean response time of a job at the ith node and the mean number of jobs at that node with one job less in the network, that is:

$$\overline{T}_i(K) = \frac{1}{\mu_i} \cdot \left[1 + \overline{K}_i(K-1)\right], \qquad i = 1, \ldots, N. \tag{8.31}$$

For single server stations ($m_i = 1$) with an FCFS strategy, it is easy to give an intuitive explanation for Eq. (8.31) because for each FCFS node i the mean response time $\overline{T}_i(K)$ of a job in a network with K jobs is given by the mean service time $(1/\mu_i)$ of that job plus the sum of the mean service times of all jobs that are ahead of this job in the queue. Equation (8.31) can also be derived without using the arrival theorem. For this purposes, we use the formulae for computing the utilization (Eq. (8.15)) and the mean number of jobs (Eq. (8.16)):

$$\rho_i(K) = \frac{e_i}{\mu_i} \cdot \frac{G(K-1)}{G(K)},$$

and:

$$\overline{K}_i(K) = \sum_{k=1}^{K} \left(\frac{e_i}{\mu_i}\right)^k \cdot \frac{G(K-k)}{G(K)}.$$

From Eq. (8.16) it follows:

$$\overline{K}_i(K-1) = \sum_{k=1}^{K-1} \left(\frac{e_i}{\mu_i}\right)^k \cdot \frac{G(K-k-1)}{G(K-1)} = \sum_{k=2}^{K} \left(\frac{e_i}{\mu_i}\right)^{k-1} \cdot \frac{G(K-k)}{G(K-1)}.$$

If we transform Eq. (8.15) for $G(K-1)$ and insert the result of the preceding equation, then, after rearranging the equation, we get:

$$\rho_i(K) \cdot \overline{K}_i(K-1) = \sum_{k=2}^{K} \left(\frac{e_i}{\mu_i}\right)^k \cdot \frac{G(K-k)}{G(K)},$$

and substituting this equation to Eq. (8.16), we have:

$$\begin{aligned}\overline{K}_i(K) &= \frac{e_i}{\mu_i} \cdot \frac{G(K-1)}{G(K)} + \rho_i(K) \cdot \overline{K}_i(K-1) \\ &= \rho_i(K) + \rho_i(K) \cdot \overline{K}_i(K-1) = \rho_i(K) \cdot \left[1 + \overline{K}_i(K-1)\right].\end{aligned}$$

If we assume constant service rates and $m_i = 1$, then we get the desired result by using $\rho_i(K) = \lambda_i(K)/\mu_i$ and Little's theorem:

$$\overline{T}_i(K) = \frac{\overline{K}_i(K)}{\lambda_i(K)} = \frac{\rho_i(K)}{\lambda_i(K)} \cdot \left[1 + \overline{K}_i(K-1)\right] = \frac{1}{\mu_i} \cdot \left[1 + \overline{K}_i(K-1)\right].$$

For computing the mean response time at the ith node we need two other equations in addition to Eq. (8.31) to describe the MVA completely. Both equations can be derived from Little's theorem. The first one is for determining the overall throughput of the network:

$$\lambda(k) = \frac{k}{\sum_{i=1}^{N} e_i \cdot \overline{T}_i(k)}, \tag{8.32}$$

whereas the other one determines the mean number of jobs at the ith node:

$$\overline{K}_i(k) = \lambda(k) \cdot \overline{T}_i(k) \cdot e_i, \tag{8.33}$$

where e_i is the visit ratio at the ith node.

The three equations, (8.31), (8.32), and (8.33), allow an iterative computation of the mean response time, mean number of jobs, and throughput of the closed product-form queueing network. The iteration is done over the number of jobs k in the network. Equation (8.31) is valid for FCFS single server nodes, PS-nodes, and LCFS PR-nodes. The description of the MVA is complete if we extend Eq. (8.31) to the case of IS-nodes and FCFS-nodes with multiple servers. In the case of IS-nodes we have:

$$\overline{T}_i(K) = \frac{1}{\mu_i}. \tag{8.34}$$

For the latter case, consider a job that arrives at -/M/m-FCFS node containing $j-1$ jobs, given that the network population is $k-1$. This event occurs with probability $\pi_i(j-1 \mid k-1)$. Then we have:

$$\overline{T}_i(k) = \sum_{j=1}^{k} \frac{j}{\mu_i \cdot \alpha_i(j)} \cdot \pi_i(j-1 \mid k-1),$$

$$\alpha_i(j) = \begin{cases} j, & \text{if } j \le m_i, \\ m_i, & \text{otherwise.} \end{cases}$$

To obtain an expression for the probability $\pi_i(j \mid k)$, we use the formulae presented in the convolution algorithm for computing the performance measures with help of the normalization constant, namely:

$$\begin{aligned}
\pi_i(j \mid k) &= F_i(j) \cdot \frac{G_N^{(i)}(k-j)}{G(k)} \quad \text{(see Eq. (8.7))} \\
&= \frac{e_i}{\mu_i \cdot \alpha_i(j)} \cdot F_i(j-1) \cdot \frac{G_N^{(i)}((k-1)-(j-1))}{G(k)} \\
&= \frac{1}{\mu_i \cdot \alpha_i(j)} \cdot \underbrace{\frac{e_i \cdot G(k-1)}{G(k)}}_{\lambda_i(k)} \cdot \underbrace{\frac{G_N^{(i)}((k-1)-(j-1))}{G(k-1)} \cdot F_i(j-1)}_{\pi_i(j-1|k-1) \quad \text{see Eq. (8.7)}} \\
&= \frac{\lambda_i(k)}{\mu_i \cdot \alpha_i(j)} \cdot \pi_i(j-1 \mid k-1), \\
\pi_i(0 \mid 0) &= 1, \\
\pi_i(0 \mid k) &= 1 - \sum_{j=1}^{k} \pi_i(j \mid k) = 1 - \sum_{j=1}^{k} \frac{\lambda_i(k)}{\mu_i \cdot \alpha_i(j)} \cdot \pi_i(j-1 \mid k-1).
\end{aligned}$$

By induction and rearrangement of the equations, the following can be obtained:

$$\begin{aligned}
\overline{T}_i(k) &= \sum_{j=1}^{k} \frac{j}{\mu_i \cdot \alpha_i(j)} \cdot \pi_i(j-1 \mid k-1) \\
&= \frac{1}{m_i \cdot \mu_i} \cdot \left(1 + \overline{K}_i(k-1) + \sum_{j=0}^{m_i-2} (m_i - j - 1) \cdot \pi_i(j \mid k-1) \right),
\end{aligned} \tag{8.35}$$

$$\begin{aligned}
\pi_i(0 \mid k) &= 1 - \sum_{j=1}^{k} \frac{\lambda_i(k)}{\mu_i \cdot \alpha_i(j)} \cdot \pi_i(j-1 \mid k-1) \\
&= 1 - \frac{1}{m_i} \cdot \left(\frac{e_i \cdot \lambda(k)}{\mu_i} + \sum_{j=1}^{m_i-1} (m_i - j) \cdot \pi_i(j \mid k) \right),
\end{aligned} \tag{8.36}$$

$$\pi_i(j \mid k) = \frac{\lambda_i(k)}{\mu_i \cdot \alpha_i(j)} \cdot \pi_i(j-1 \mid k-1). \tag{8.37}$$

Now the MVA for closed single class product-form queueing networks can be described as follows:

STEP 1 Initialization. For $i = 1, \ldots, N$ and $j = 1, \ldots, (m_i - 1)$:

$$\overline{K}_i(0) = 0, \quad \pi_i(0 \mid 0) = 1, \quad \pi_i(j \mid 0) = 0.$$

STEP 2 Iteration over the number of jobs $k = 1, \ldots, K$.

STEP 2.1 For $i = 1, \ldots, N$, compute the mean response time of a job at the ith node:

$$\overline{T}_i(k) = \begin{cases} \dfrac{1}{\mu_i}\left[1 + \overline{K}_i(k-1)\right], & \text{Type-1,2,4 } (m_i = 1), \\ \dfrac{1}{\mu_i \cdot m_i}\left[1 + \overline{K}_i(k-1) + \sum_{j=0}^{m_i-2}(m_i - j - 1)\cdot \pi_i(j \mid k-1)\right], & \text{Type-1 } (m_i > 1), \\ \dfrac{1}{\mu_i}, & \text{Type-3,} \end{cases} \tag{8.38}$$

where the conditional probabilities are computed using Eqs. (8.37) and (8.36).

STEP 2.2 Compute the overall throughput:

$$\lambda(k) = \frac{k}{\sum_{i=1}^{N} e_i \cdot \overline{T}_i(k)}. \tag{8.39}$$

The throughput of each node can be computed using the following equation:

$$\lambda_i(k) = e_i \cdot \lambda(k), \tag{8.40}$$

where the e_i can be determined with Eq. (7.5).

STEP 2.3 For $i = 1, \ldots, N$, compute the mean number of jobs at the ith node:

$$\overline{K}_i(k) = e_i \cdot \lambda(k) \cdot \overline{T}_i(k). \tag{8.41}$$

The other performance measures, e.g., utilization, mean waiting time, mean queue length, etc., can be derived from the calculated measures using the well-known equations.

The disadvantage of the MVA is its extremely high memory requirement. The memory requirement can be considerably reduced by using approximation techniques (e.g., SCAT, or self-correcting approximation technique), discussed in Chapter 9. Another disadvantage of the MVA is the fact that it is not possible to compute state probabilities. In [AkBo83], the MVA has been extended for computing the normalization constant and for computing the state probabilities. Only Step 2.2 needs to be extended to include the equation:

$$G(k) = \frac{G(k-1)}{\lambda(k)}, \tag{8.42}$$

with the initial condition $G(0) = 1$. This follows immediately from Eq. (8.14). When the iteration stops, we have the normalization constant $G(K)$ that

can be used to compute the state probabilities with the help of the BCMP theorem, Eq. (7.80). Two applications of the mean value analysis are now given.

Example 8.3 The central-server model shown in Fig. 8.5 has $N = 4$ nodes and $K = 6$ jobs. The service time of a job at the ith node, $i = 1, 2, 3, 4$, is exponentially distributed with the following mean values:

$$\frac{1}{\mu_1} = 0.02\,\text{sec}, \quad \frac{1}{\mu_2} = 0.2\,\text{sec}, \quad \frac{1}{\mu_3} = 0.4\,\text{sec}, \quad \frac{1}{\mu_4} = 0.6\,\text{sec}.$$

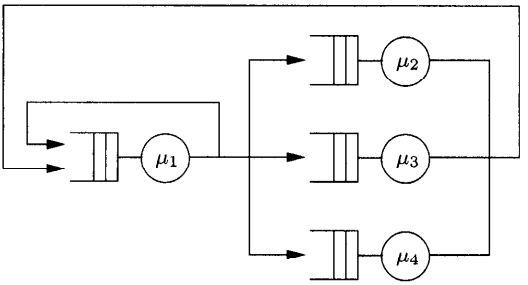


Fig. 8.5 The central-server model.

The visit ratios are given as follows:

$$e_1 = 1, \quad e_2 = 0.4, \quad e_3 = 0.2, \quad e_4 = 0.1.$$

With the help of the MVA, the performance measures and normalization constant of the network are computed in three steps:

STEP 1 Initialization:

$$\overline{K}_1(0) = \overline{K}_2(0) = \overline{K}_3(0) = \overline{K}_4(0) = 0, \quad G(0) = 1.$$

STEP 2 Iteration over the number of jobs in the network starting with $k = 1$:

STEP 2.1 Mean response times, Eq. (8.38):

$$\overline{T}_1(1) = \frac{1}{\mu_1}\left[1 + \overline{K}_1(0)\right] = \underline{0.02}, \quad \overline{T}_2(1) = \frac{1}{\mu_2}\left[1 + \overline{K}_2(0)\right] = \underline{0.2},$$
$$\overline{T}_3(1) = \frac{1}{\mu_3}\left[1 + \overline{K}_3(0)\right] = \underline{0.4}, \quad \overline{T}_4(1) = \frac{1}{\mu_4}\left[1 + \overline{K}_4(0)\right] = \underline{0.6}.$$

STEP 2.2 Throughput, Eq. (8.39), and normalization constant, Eq. (8.42):

$$\lambda(1) = \frac{1}{\sum_{i=1}^{4} e_i \overline{T}_i(1)} = \underline{4.167}, \quad G(1) = \frac{G(0)}{\lambda(1)} = \underline{0.24}.$$

Table 8.4 Performance measures after completing six iteration steps.

Node	1	2	3	4
Mean response time $\overline{T}_i$	0.025	0.570	1.140	1.244
Throughput λ_i	9.920	3.968	1.984	0.992
Mean number of jobs $\overline{K}_i$	0.244	2.261	2.261	1.234
Utilization ρ_i	0.198	0.794	0.794	0.595

STEP 2.3 Mean number of jobs, Eq. (8.41):

$$\overline{K}_1(1) = \lambda(1)\overline{T}_1(1)e_1 = \underline{0.083}, \quad \overline{K}_2(1) = \lambda(1)\overline{T}_2(1)e_2 = \underline{0.333},$$
$$\overline{K}_3(1) = \lambda(1)\overline{T}_3(1)e_3 = \underline{0.333}, \quad \overline{K}_4(1) = \lambda(1)\overline{T}_4(1)e_4 = \underline{0.25}.$$

Iteration for $k = 2$:

STEP 2.1 Mean response times:

$$\overline{T}_1(2) = \frac{1}{\mu_1}\left[1 + \overline{K}_1(1)\right] = \underline{0.022}, \quad \overline{T}_2(2) = \frac{1}{\mu_2}\left[1 + \overline{K}_2(1)\right] = \underline{0.267},$$
$$\overline{T}_3(2) = \frac{1}{\mu_3}\left[1 + \overline{K}_3(1)\right] = \underline{0.533}, \quad \overline{T}_4(2) = \frac{1}{\mu_4}\left[1 + \overline{K}_4(1)\right] = \underline{0.75}.$$

STEP 2.2 Throughput and normalization constant:

$$\lambda(2) = \frac{2}{\sum_{i=1}^{4} e_i\overline{T}_i(2)} = \underline{6.452}, \quad G(2) = \frac{G(1)}{\lambda(2)} = \underline{3.72 \cdot 10^{-2}}.$$

STEP 2.3 Mean number of jobs:

$$\overline{K}_1(2) = \lambda(2)\overline{T}_1(2)e_1 = \underline{0.140}, \quad \overline{K}_2(2) = \lambda(2)\overline{T}_2(2)e_2 = \underline{0.688},$$
$$\overline{K}_3(2) = \lambda(2)\overline{T}_3(2)e_3 = \underline{0.688}, \quad \overline{K}_4(2) = \lambda(2)\overline{T}_4(2)e_4 = \underline{0.484}.$$

$$\vdots$$

After six steps, the iteration stops and we get the final results as summarized in the Table 8.4. The normalization constant of the network is $G(K) = 5.756 \cdot 10^{-6}$. With Eq. (7.80) we can compute the steady-sate probability that the network is, for example, in the state $(3, 1, 1, 1)$:

$$\pi(3,1,1,1) = \frac{1}{G(K)} \prod_{i=1}^{4} \left(\frac{e_i}{\mu_i}\right)^{k_i} = \underline{5.337 \cdot 10^{-4}}.$$

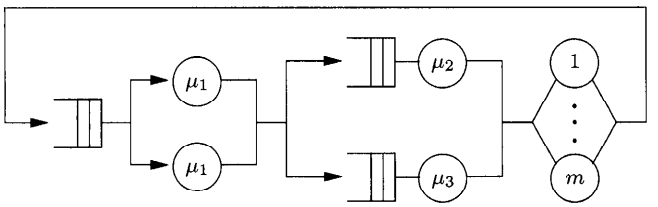


Fig. 8.6 A closed queueing network.

Example 8.4 As another example, consider the closed queueing network given in Fig. 8.6 with $K = 3$ jobs. At the first node we have $m_1 = 2$ identical processors having exponentially distributed service times with mean $1/\mu_1 = 0.5$ sec. Node 2 and node 3 have exponentially distributed service times with means $1/\mu_2 = 0.6$ sec and $1/\mu_3 = 0.8$ sec, respectively. At the fourth node (terminals), the mean service time is $1/\mu_4 = 1$ sec. The routing probabilities are as follows:

$$p_{12} = p_{13} = 0.5 \quad \text{and} \quad p_{24} = p_{34} = p_{41} = 1.$$

From Eq. (7.5) we compute the visit ratios:

$$e_1 = 1, \quad e_2 = 0.5, \quad e_3 = 0.5, \quad e_4 = 1.$$

The analysis of the network is carried out in the following steps:

STEP 1 Initialization:

$$\overline{K}_1(0) = \overline{K}_2(0) = \overline{K}_3(0) = 0, \quad \pi_1(0 \mid 0) = 1, \quad \pi_1(1 \mid 0) = 0.$$

STEP 2 Iteration over the number of jobs in the network starting with $k = 1$:

STEP 2.1 Mean response times:

$$\overline{T}_1(1) = \frac{1}{m_1\mu_1}\left[1 + \overline{K}_1(0) + \pi_1(0 \mid 0)\right] = \underline{0.5}, \quad \overline{T}_2(1) = \frac{1}{\mu_2}\left[1 + \overline{K}_2(0)\right] = \underline{0.6},$$

$$\overline{T}_3(1) = \frac{1}{\mu_3}\left[1 + \overline{K}_3(0)\right] = \underline{0.8}, \qquad \overline{T}_4(1) = \frac{1}{\mu_4} = \underline{1}.$$

STEP 2.2 Throughput:

$$\lambda(1) = \frac{1}{\sum_{i=1}^{4} e_i \overline{T}_i(1)} = \underline{0.454}.$$

STEP 2.3 Mean number of jobs:

$$\overline{K}_1(1) = \lambda(1)\overline{T}_1(1)e_1 = \underline{0.227}, \quad \overline{K}_2(1) = \lambda(1)\overline{T}_2(1)e_2 = \underline{0.136},$$
$$\overline{K}_3(1) = \lambda(1)\overline{T}_3(1)e_3 = \underline{0.182}, \quad \overline{K}_4(1) = \lambda(1)\overline{T}_4(1)e_4 = \underline{0.454}.$$

Iteration for $k = 2$:

STEP 2.1 Mean response times:

$$\overline{T}_1(2) = \frac{1}{m_1\mu_1}\left[1 + \overline{K}_1(1) + \pi_1(0 \mid 1)\right] = \underline{0.5},$$

with:

$$\pi_1(0 \mid 1) = 1 - \frac{1}{m_1}\left[\frac{e_1}{\mu_1}\lambda(1) + \pi_1(1 \mid 1)\right] = \underline{0.773},$$

and:

$$\pi_1(1 \mid 1) = \frac{e_1}{\mu_1}\lambda(1)\pi_1(0 \mid 0) = \underline{0.227}, \quad \overline{T}_2(2) = \frac{1}{\mu_2}\left[1 + \overline{K}_2(1)\right] = \underline{0.682},$$
$$\overline{T}_3(2) = \frac{1}{\mu_3}\left[1 + \overline{K}_3(1)\right] = \underline{0.946}, \quad \overline{T}_4(2) = \underline{1}.$$

STEP 2.2 Throughput:

$$\lambda(2) = \frac{2}{\sum_{i=1}^{4} e_i\overline{T}_i(2)} = \underline{0.864}.$$

STEP 2.3 Mean number of jobs:

$$\overline{K}_1(2) = \lambda(2)\overline{T}_1(2)e_1 = \underline{0.432}, \quad \overline{K}_2(2) = \lambda(2)\overline{T}_2(2)e_2 = \underline{0.295},$$
$$\overline{K}_3(2) = \lambda(2)\overline{T}_3(2)e_3 = \underline{0.409}, \quad \overline{K}_4(2) = \lambda(2)\overline{T}_4(2)e_4 = \underline{0.864}.$$

Iteration for $k = 3$:

STEP 2.1 Mean response times:

$$\overline{T}_1(3) = \frac{1}{m_1\mu_1}\left[1 + \overline{K}_1(2) + \pi_1(0 \mid 2)\right] = \underline{0.512},$$

with:

$$\pi_1(0 \mid 2) = 1 - \frac{1}{m_1}\left[\frac{e_1}{\mu_1}\lambda(2) + \pi_1(1 \mid 2)\right] = \underline{0.617},$$

and:

$$\pi_1(1 \mid 2) = \frac{e_1}{\mu_1}\lambda(2)\pi_1(0 \mid 1) = \underline{0.334},$$
$$\overline{T}_2(3) = \frac{1}{\mu_2}\left[1 + \overline{K}_2(2)\right] = \underline{0.776},$$
$$\overline{T}_3(3) = \frac{1}{\mu_3}\left[1 + \overline{K}_3(2)\right] = \underline{1.127}, \quad \overline{T}_4(3) = \underline{1}.$$

STEP 2.2 Throughput:

$$\lambda(3) = \frac{3}{\sum_{i=1}^{4} e_i \overline{T}_i(3)} = \underline{1.217}.$$

STEP 2.3 Mean number of jobs:

$$\overline{K}_1(3) = \lambda(3)\overline{T}_1(3)e_1 = \underline{0.624}, \quad \overline{K}_2(3) = \lambda(3)\overline{T}_2(3)e_2 = \underline{0.473},$$
$$\overline{K}_3(3) = \lambda(3)\overline{T}_3(3)e_3 = \underline{0.686}, \quad \overline{K}_4(3) = \lambda(3)\overline{T}_4(3)e_4 = \underline{1.217}.$$

The throughput at each node can be computed with Eq. (8.40):

$$\lambda_1 = \lambda(3)\cdot e_1 = \underline{1.218}, \quad \lambda_2 = \lambda(3)\cdot e_2 = \underline{0.609},$$
$$\lambda_3 = \lambda(3)\cdot e_3 = \underline{0.609}, \quad \lambda_4 = \lambda(3)\cdot e_4 = \underline{1.218}.$$

For determining the utilization of each node, we use Eq. (7.21):

$$\rho_1 = \frac{\lambda_1}{m_1\mu_1} = \underline{0.304}, \quad \rho_2 = \frac{\lambda_2}{\mu_2} = \underline{0.365}, \quad \rho_3 = \frac{\lambda_3}{\mu_3} = \underline{0.487}.$$

8.2.2 Multiclass Closed Networks

The algorithm for computing the performance measures of single class closed queueing networks can easily be extended to the multiclass case in the following way [ReLa80]:

STEP 1 Initialization. For $i = 1, \ldots, N, r = 1, \ldots, R, j = 1, \ldots, (m_i - 1)$:

$$\overline{K}_{ir}(0, 0 \ldots, 0) = 0, \quad \pi_i(0 \mid \mathbf{0}) = 1, \quad \pi_i(j \mid \mathbf{0}) = 0.$$

STEP 2 Iteration: $\mathbf{k} = \mathbf{0}, \ldots, \mathbf{K}$:

STEP 2.1 For $i = 1, \ldots, N$ and $r = 1, \ldots, R$, compute the mean response time of class-r jobs at the ith node:

$$\overline{T}_{ir}(\mathbf{k}) = \begin{cases} \dfrac{1}{\mu_{ir}} \left[1 + \sum\limits_{s=1}^{R} \overline{K}_{is}(\mathbf{k} - \mathbf{1}_r) \right] & \text{Type-1,2,4} \\ & (m_i = 1), \\ \dfrac{1}{\mu_{ir} \cdot m_i} \left[1 + \sum\limits_{s=1}^{R} \overline{K}_{is}(\mathbf{k} - \mathbf{1}_r) \right. & \\ \qquad \left. + \sum\limits_{j=0}^{m_i - 2} (m_i - j - 1) \pi_i(j \mid \mathbf{k} - \mathbf{1}_r) \right], & \text{Type-1} \\ & (m_i > 1), \\ \dfrac{1}{\mu_{ir}}, & \text{Type-3.} \end{cases} \tag{8.43}$$

Here $(\mathbf{k} - \mathbf{1}_r) = (k_1, \ldots, k_r - 1, \ldots, k_R)$ is the population vector with one class-r job less in the system.

The probability that there are j jobs at the ith node ($j = 1, \ldots, (m_i - 1)$) given that the network is in state $\mathbf{k}$ is given by:

$$\pi_i(j \mid \mathbf{k}) = \frac{1}{j} \left[\sum_{r=1}^{R} \frac{e_{ir}}{\mu_{ir}} \lambda_r(\mathbf{k}) \pi_i(j - 1 \mid \mathbf{k} - \mathbf{1}_r) \right], \tag{8.44}$$

and for $j = 0$ by:

$$\pi_i(0 \mid \mathbf{k}) = 1 - \frac{1}{m_i} \left[\sum_{r=1}^{R} \frac{e_{ir}}{\mu_{ir}} \lambda_r(\mathbf{k}) + \sum_{j=1}^{m_i - 1} (m_i - j) \pi_i(j \mid \mathbf{k}) \right], \tag{8.45}$$

where e_{ir} can be computed by Eq. (7.72).

STEP 2.2 For $r = 1, \ldots, R$, compute the throughput:

$$\lambda_r(\mathbf{k}) = \frac{k_r}{\sum\limits_{i=1}^{N} e_{ir} \overline{T}_{ir}(k)}. \tag{8.46}$$

STEP 2.3 For $i = 1, \ldots, N$ and $r = 1, \ldots, R$, compute the mean number of class-r jobs at the ith node:

$$\overline{K}_{ir}(\mathbf{k}) = \lambda_r(\mathbf{k}) \cdot \overline{T}_{ir}(\mathbf{k}) \cdot e_{ir}. \tag{8.47}$$

With these extensions the MVA algorithm for multiclass closed product-form queueing networks is completely specified.

Akyildiz and Bolch [AkBo83] extended the MVA for computing the normalization constant and the state probabilities. For this purpose, Step 2.2 in the preceding algorithm is expanded to include the formula:

$$C(\mathbf{k}) = \frac{G(\mathbf{k} - \mathbf{1}_r)}{\lambda_r(\mathbf{k})}, \tag{8.48}$$

with the initial condition $G(\mathbf{0}) = 1$. After the iteration stops we get the normalization constant $G(\mathbf{K})$ that can be used to compute the steady-state probabilities with the help of the BCMP theorem, Eq. (7.80).

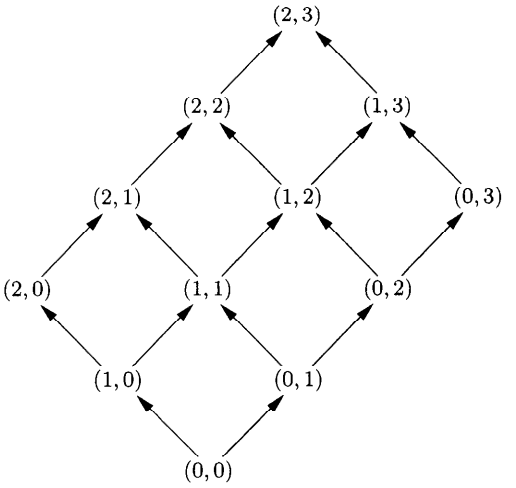


Fig. 8.7 Sequence of intermediate values.

To explain iteration Step 2, we use Fig. 8.7 where the sequence of intermediate values is shown for a network with $R = 2$ job classes. The first job class has $K_1 = 2$ jobs and the second class has $K_2 = 3$ jobs. The mean value algorithm starts with the trivial case where no job is in the system, that is, the population vector (0,0). From this, solutions for all population vectors that consist of exactly one job are computed; in our example these are the population vectors $(1,0)$ and $(0,1)$. Then the solution for all population vectors with exactly two jobs are computed, and so on, until the final result for $\mathbf{K} = (2,3)$ is reached. In general, to compute the solution for a population vector $\mathbf{k}$ we need R intermediate solutions as input, namely the solutions for all population vectors $\mathbf{k} - \mathbf{1}_r$, $r = 1, \ldots, R$.

Now, the algorithm is illustrated by means of an example:

Example 8.5 Consider the queueing network shown in Fig. 8.8 with $N = 3$ nodes and $R = 2$ job classes. Class 1 contains $K_1 = 2$ jobs and class 2 contains $K_2 = 1$ jobs. Class switching of the jobs is not allowed. The service time at the ith, $i = 1, 2, 3$, node is exponentially distributed with mean values:

$$\frac{1}{\mu_{11}} = 0.2\,\text{sec}, \quad \frac{1}{\mu_{21}} = 0.4\,\text{sec}, \quad \frac{1}{\mu_{31}} = 1\,\text{sec},$$
$$\frac{1}{\mu_{12}} = 0.2\,\text{sec}, \quad \frac{1}{\mu_{22}} = 0.6\,\text{sec}, \quad \frac{1}{\mu_{32}} = 2\,\text{sec}.$$

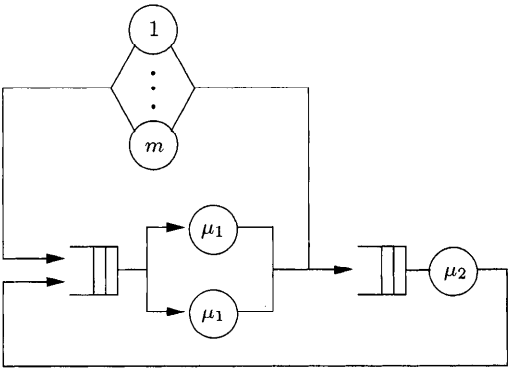


Fig. 8.8 Another closed queueing network.

The visit ratios are given as follows:

$$e_{11} = 1, \quad e_{21} = 0.6, \quad e_{31} = 0.4, \quad e_{12} = 1, \quad e_{22} = 0.3, \quad e_{32} = 0.7.$$

The queueing discipline at node 1 is FCFS and at node 2, processor sharing. The terminal station is modeled as an IS-node. We analyze the network using the MVA in the following three steps:

STEP 1 Initialization:

$$\overline{K}_{ir}(\mathbf{0}) = 0 \quad \text{for } i = 1,2,3 \text{ and } r = 1,2, \quad \pi_1(0 \mid \mathbf{0}) = 1, \quad \pi_1(1 \mid \mathbf{0}) = 0.$$

STEP 2 Iterate over the number of jobs in the network beginning with the population vector $\mathbf{k} = (1,0)$:

STEP 2.1 Mean response times, Eq. (8.43):

$$\overline{T}_{11}(1,0) = \frac{1}{2 \cdot \mu_{11}} \left[1 + \overline{K}_{11}(0,0) + \overline{K}_{12}(0,0) + \pi_1(0 \mid 0,0)\right] = \underline{0.2},$$

$$\overline{T}_{21}(1,0) = \frac{1}{\mu_{21}} \left[1 + \overline{K}_{21}(0,0) + \overline{K}_{22}(0,0)\right] = \underline{0.4},$$

$$\overline{T}_{31}(1,0) = \frac{1}{\mu_{31}} = \underline{1}.$$

STEP 2.2 Throughput for class 1 using Eq. (8.46):

$$\lambda_1(1,0) = \frac{1}{\sum_{i=1}^{3} e_{i1} \overline{T}_{i1}(1,0)} = \underline{1.190}.$$

Note that the class 2 throughput for this population vector is 0.

STEP 2.3 Mean number of jobs, Eq. (8.47):

$$\overline{K}_{11}(1,0) = \lambda_1(1,0)\overline{T}_{11}(1,0)e_{11} = \underline{0.238},$$
$$\overline{K}_{21}(1,0) = \lambda_1(1,0)\overline{T}_{21}(1,0)e_{21} = \underline{0.286},$$
$$\overline{K}_{31}(1,0) = \lambda_1(1,0)\overline{T}_{31}(1,0)e_{31} = \underline{0.476}.$$

Iteration for $\mathbf{k} = (0,1)$:

STEP 2.1 Mean response time:

$$\overline{T}_{12}(0,1) = \frac{1}{2\cdot\mu_{12}}\left[1 + \overline{K}_{11}(0,0) + \overline{K}_{12}(0,0) + \pi_1(0 \mid 0,0)\right] = \underline{0.2},$$
$$\overline{T}_{22}(0,1) = \frac{1}{\mu_{22}}\left[1 + \overline{K}_{21}(0,0) + \overline{K}_{22}(0,0)\right] = \underline{0.6},$$
$$\overline{T}_{32}(0,1) = \frac{1}{\mu_{32}} = \underline{2}.$$

STEP 2.2 Throughput for class 2:

$$\lambda_2(0,1) = \frac{1}{\sum_{i=1}^{3} e_{i2}\overline{T}_{i2}(0,1)} = \underline{0.562}.$$

Note that the class 1 throughput for this population vector is 0.

STEP 2.3 Mean number of jobs:

$$\overline{K}_{12}(0,1) = \lambda_2(0,1)\overline{T}_{12}(0,1)e_{12} = \underline{0.112},$$
$$\overline{K}_{22}(0,1) = \lambda_2(0,1)\overline{T}_{22}(0,1)e_{22} = \underline{0.101},$$
$$\overline{K}_{32}(0,1) = \lambda_2(0,1)\overline{T}_{32}(0,1)e_{32} = \underline{0.787}.$$

Iteration for $\mathbf{k} = (1,1)$:

STEP 2.1 Mean response times:

Class 1:

$$\overline{T}_{11}(1,1) = \frac{1}{2\cdot\mu_{11}}\left[1 + \overline{K}_{12}(0,1) + \pi_1(0 \mid 0,1)\right] = \underline{0.2},$$

where:

$$\pi_1(0 \mid 0,1) = 1 - \frac{1}{m_1}\left[\frac{e_{12}}{\mu_{12}}\lambda_2(0,1) + \pi_1(1 \mid 0,1)\right] = \underline{0.888},$$

and:

$$\pi_1(1 \mid 0,1) = \frac{e_{12}}{\mu_{12}}\lambda_2(0,1)\pi_1(0 \mid 0,0) = \underline{0.112},$$
$$\overline{T}_{21}(1,1) = \frac{1}{\mu_{21}}\left[1 + \overline{K}_{21}(0,1) + \overline{K}_{22}(0,1)\right] = \underline{0.44},$$
$$\overline{T}_{31}(1,1) = \frac{1}{\mu_{31}} = \underline{1}.$$

Class 2:

$$\overline{T}_{12}(1,1) = \frac{1}{2 \cdot \mu_{12}}\left[1 + \overline{K}_{11}(1,0) + \pi_1(0 \mid 1,0)\right] = \underline{0.2},$$

with:

$$\pi_1(0 \mid 1,0) = 1 - \frac{1}{m_1}\left[\frac{e_{11}}{\mu_{11}}\lambda_1(1,0) + \pi_1(1 \mid 1,0)\right] = \underline{0.762},$$

and:

$$\pi_1(1 \mid 1,0) = \frac{e_{11}}{\mu_{11}}\lambda_1(1,0)\pi_1(0 \mid 0,0) = \underline{0.238},$$
$$\overline{T}_{22}(1,1) = \frac{1}{\mu_{22}}\left[1 + \overline{K}_{21}(1,0)\right] = \underline{0.772}, \quad \overline{T}_{32}(1,1) = \frac{1}{\mu_{32}} = \underline{2}.$$

STEP 2.2 Throughputs:

$$\lambda_1(1,1) = \frac{1}{\sum_{i=1}^{3} e_{i1}\overline{T}_{i1}(1,1)} = \underline{1.157},$$
$$\lambda_2(1,1) = \frac{1}{\sum_{i=1}^{3} e_{i2}\overline{T}_{i2}(1,1)} = \underline{0.546}.$$

STEP 2.3 Mean number of jobs:

$$\overline{K}_{11}(1,1) = \lambda_1(1,1)\overline{T}_{11}(1,1)e_{11} = \underline{0.231},$$
$$\overline{K}_{12}(1,1) = \lambda_2(1,1)\overline{T}_{12}(1,1)e_{12} = \underline{0.109},$$
$$\overline{K}_{21}(1,1) = \lambda_1(1,1)\overline{T}_{21}(1,1)e_{21} = \underline{0.305},$$
$$\overline{K}_{22}(1,1) = \lambda_2(1,1)\overline{T}_{22}(1,1)e_{22} = \underline{0.126},$$
$$\overline{K}_{31}(1,1) = \lambda_1(1,1)\overline{T}_{31}(1,1)e_{31} = \underline{0.463},$$
$$\overline{K}_{32}(1,1) = \lambda_2(1,1)\overline{T}_{32}(1,1)e_{32} = \underline{0.764}.$$

Iteration for $\mathbf{k} = (2,0)$:

STEP 2.1 Mean response times:

$$\overline{T}_{11}(2,0) = \frac{1}{2 \cdot \mu_{11}} \left[1 + \overline{K}_{11}(1,0) + \pi_1(0 \mid 1,0)\right] = \underline{0.2},$$
$$\overline{T}_{21}(2,0) = \frac{1}{\mu_{21}} \left[1 + \overline{K}_{21}(1,0)\right] = \underline{0.514},$$
$$\overline{T}_{31}(2,0) = \frac{1}{\mu_{31}} = \underline{1}.$$

STEP 2.2 Throughput:

$$\lambda_1(2,0) = \frac{2}{\sum_{i=1}^{3} e_{i1} \overline{T}_{i1}(2,0)} = \underline{2.201}.$$

STEP 2.3 Mean number of jobs:

$$\overline{K}_{11}(2,0) = \lambda_1(2,0) \overline{T}_{11}(2,0) e_{11} = \underline{0.440},$$
$$\overline{K}_{21}(2,0) = \lambda_1(2,0) \overline{T}_{21}(2,0) e_{21} = \underline{0.679},$$
$$\overline{K}_{31}(2,0) = \lambda_1(2,0) \overline{T}_{31}(2,0) e_{31} = \underline{0.881}.$$

Iteration for $\mathbf{k} = (2,1)$:

STEP 2.1 Mean response times:

Class 1:

$$\overline{T}_{11}(2,1) = \frac{1}{2 \cdot \mu_{11}} \left[1 + \overline{K}_{11}(1,1) + \overline{K}_{12}(1,1) + \pi_1(0 \mid 1,1)\right] = \underline{0.203},$$

where:

$$\pi_1(0 \mid 1,1) = 1 - \frac{1}{m_1} \left[\frac{e_{11}}{\mu_{11}} \lambda_1(1,1) + \frac{e_{12}}{\mu_{12}} \lambda_2(1,1) + \pi_1(1 \mid 1,1)\right] = \underline{0.685},$$

and:

$$\pi_1(1 \mid 1,1) = \frac{e_{11}}{\mu_{11}} \lambda_1(1,1) \pi_1(0 \mid 0,1) + \frac{e_{12}}{\mu_{12}} \lambda_2(1,1) \cdot \pi_1(0 \mid 1,0) = \underline{0.289},$$
$$\overline{T}_{21}(2,1) = \frac{1}{\mu_{21}} \left[1 + \overline{K}_{21}(1,1) + \overline{K}_{22}(1,1)\right] = \underline{0.573}, \quad \overline{T}_{31}(2,1) = \frac{1}{\mu_{31}} = \underline{1}.$$

Class 2:

$$\overline{T}_{12}(2,1) = \frac{1}{2 \cdot \mu_{12}} \left[1 + \overline{K}_{11}(2,0) + \pi_1(0 \mid 2,0)\right] = \underline{0.205},$$

where:

$$\pi_1(0 \mid 2,0) = 1 - \frac{1}{m_1}\left[\frac{e_{11}}{\mu_{11}}\lambda_1(2,0) + \pi_1(1 \mid 2,0)\right] = \underline{0.612},$$

and:

$$\pi_1(1 \mid 2,0) = \frac{e_{11}}{\mu_{11}}\lambda_1(2,0)\pi_1(0 \mid 1,0) = \underline{0.335},$$

$$\overline{T}_{22}(2,1) = \frac{1}{\mu_{22}}\left[1 + \overline{K}_{21}(2,0)\right] = \underline{1.008}, \quad \overline{T}_{32}(2,1) = \frac{1}{\mu_{32}} = \underline{2}.$$

STEP 2.2 Throughputs:

$$\lambda_1(2,1) = \frac{2}{\sum_{i=1}^{3} e_{i1}\overline{T}_{i1}(2,1)} = \underline{2.113}, \quad \lambda_2(2,1) = \frac{1}{\sum_{i=1}^{3} e_{i2}\overline{T}_{i2}(2,1)} = \underline{0.524}.$$

STEP 2.3 Mean number of jobs:

$$\overline{K}_{11}(2,1) = \lambda_1(2,1)\overline{T}_{11}(2,1)e_{11} = \underline{0.428},$$
$$\overline{K}_{12}(2,1) = \lambda_2(2,1)\overline{T}_{12}(2,1)e_{12} = \underline{0.108},$$
$$\overline{K}_{21}(2,1) = \lambda_1(2,1)\overline{T}_{21}(2,1)e_{21} = \underline{0.726},$$
$$\overline{K}_{22}(2,1) = \lambda_2(2,1)\overline{T}_{22}(2,1)e_{22} = \underline{0.158},$$
$$\overline{K}_{31}(2,1) = \lambda_1(2,1)\overline{T}_{31}(\text{queueing} = \underline{0.845},$$
$$\overline{K}_{32}(2,1) = \lambda_2(2,1)\overline{T}_{32}(2,1)e_{32} = \underline{0.734}.$$

We can see that the MVA is easy to implement and is very fast. For networks with multiple server nodes ($m_i > 1$) and several job classes, this technique has some disadvantages such as stability problems, accumulation of roundoff errors, and large storage requirements. The storage requirement is $O\left(N \cdot \prod_{r=1}^{R}(K_r + 1)\right)$. For comparison, the storage requirement of the convolution algorithm is $O\left(\prod_{r=1}^{R}(K_r + 1)\right)$. The time requirement in both cases is approximately the same: $2 \cdot R(N-1) \cdot \prod_{r=1}^{R}(K_r + 1)$ [CoGe86]. Recall that N is the number of nodes in the network and K_r is the number of jobs of class r.

8.2.3 Mixed Networks

In [ZaWo81] the MVA is extended to the analysis of mixed product-form queueing networks. But the networks are restricted to consist only of single server nodes. Before we introduce the MVA for mixed queueing networks,

we consider the arrival theorem for open queueing networks. This theorem says that the probability that a job entering node i will find the network in state $(k_1 \ldots, k_i, \ldots, k_N)$ is equal to the steady-state probability for this state. This theorem is also called PASTA theorem.[1] Therefore we have for the mean response time of a job in an open network:

$$\overline{T}_{ir}(\mathbf{k}) = \begin{cases} \dfrac{1}{\mu_{ir}} \left(1 + \sum_{s=1}^{R} \overline{K}_{is}\right), & \text{Type-1,2,4 } (m_i = 1), \\ \dfrac{1}{\mu_{ir}}, & \text{Type-3.} \end{cases} \tag{8.49}$$

As from [ZaWo81], we have, for any class r and s, $\overline{K}_{is} = \dfrac{\rho_{is}}{\rho_{ir}} \overline{K}_{ir}$ and, therefore, with the help of the relation $\overline{K}_{ir}(\mathbf{k}) = \lambda_{ir} \cdot \overline{T}_{ir}(\mathbf{k})$, it follows from Eq. (8.49):

$$\overline{K}_{ir}(\mathbf{k}) = \begin{cases} \rho_{ir} \left(1 + \sum_{s=1}^{R} \dfrac{\rho_{is}}{\rho_{ir}} \overline{K}_{ir}\right) = \dfrac{\rho_{ir}}{1 - \sum_{s=1}^{R} \rho_{is}}, & \text{Type-1,2,4 } (m_i = 1), \\ \rho_{ir}, & \text{Type-3,} \end{cases} \tag{8.50}$$

with $\rho_{ir} = \lambda_{ir}/\mu_{ir}$, where $\lambda_{ir} = \lambda_r \cdot e_{ir}$ is given as an input parameter.

These results for open queueing networks are now used for analyzing mixed queueing networks. The arrival theorem for mixed product-form queueing networks says that jobs of the open job classes when arriving at a node see the number of jobs in equilibrium, while jobs of the closed job classes when arriving at that node will see the number of jobs at that node in equilibrium with one job less in its own job class in the network [ReLa80, SeMi81].

If we index the open job classes with $op = 1, \ldots, OP$ and the closed job classes with $cl = 1, \ldots, CL$, then we have for the mean number of jobs in an *open* class r, taking into account Eq. (8.50):

$$\begin{aligned} \overline{K}_{ir}(\mathbf{k}) &= \lambda_{ir} \cdot \frac{1}{\mu_{ir}} \left[1 + \sum_{cl=1}^{CL} \overline{K}_{i,cl}(\mathbf{k}) + \sum_{op=1}^{OP} \overline{K}_{i,op}(\mathbf{k})\right] \\ &= \frac{\rho_{ir} \left[1 + \sum_{cl=1}^{CL} \overline{K}_{i,cl}(\mathbf{k})\right]}{1 - \sum_{op=1}^{OP} \rho_{i,op}}, \end{aligned} \tag{8.51}$$

[1] PASTA: **P**oisson **a**rrivals **s**ee **t**ime **a**verages (see [Wolf82]).

where $\mathbf{k}$ is the population vector of the *closed* job classes. Equation (8.51) is valid for Type-1,2,4 single server nodes, while for Type-3 nodes we have:

$$\overline{K}_{ir}(\mathbf{k}) = \rho_{ir}. \tag{8.52}$$

For the mean response time of a job in a *closed* class r at node i we have:

$$\begin{aligned}
\overline{T}_{ir}(\mathbf{k}) &= \frac{1}{\mu_{ir}} \left[1 + \sum_{cl=1}^{CL} \overline{K}_{i,cl}(\mathbf{k} - \mathbf{1}_r) + \sum_{op=1}^{OP} \overline{K}_{i,op}(\mathbf{k} - \mathbf{1}_r) \right] \\
&= \frac{1}{\mu_{ir}} \left[1 + \sum_{cl=1}^{CL} \overline{K}_{i,cl}(\mathbf{k} - \mathbf{1}_r) + \frac{\left[1 + \sum_{cl=1}^{CL} \overline{K}_{i,cl}(\mathbf{k} - \mathbf{1}_r) \right] \sum_{op=1}^{OP} \rho_{i,op}}{1 - \sum_{op=1}^{OP} \rho_{i,op}} \right] \\
&= \frac{1}{\mu_{ir}} \frac{\left[1 + \sum_{cl=1}^{CL} \overline{K}_{i,cl}(\mathbf{k} - \mathbf{1}_r) \right]}{1 - \sum_{op=1}^{OP} \rho_{i,op}}.
\end{aligned} \tag{8.53}$$

Equation (8.53) is valid for Type-1,2,4 single server nodes. For Type-3 nodes we have:

$$\overline{T}_{ir}(\mathbf{k}) = \frac{1}{\mu_{ir}} \tag{8.54}$$

With these formulae the MVA for mixed product-form queueing networks can be described completely. We assume for now that all arrival and service rates are load independent.

The algorithm is as follows:

STEP 1 Initialization. For all nodes $i = 1, \ldots, N$, compute the utilization of the open class jobs $op = 1, \ldots, OP$ in the mixed network:

$$\rho_{i,op} = \frac{1}{\mu_{i,op}} \lambda_{op} \cdot e_{i,op}, \tag{8.55}$$

and check for the ergodicity condition ($\rho_{i,\mathrm{op}} \leq 1$). Set $\overline{K}_{i,cl}(\mathbf{0}) = 0$ for all $i = 1, \ldots, N$ and all closed classes $cl = 1, \ldots, CL$.

STEP 2 Construct a closed queueing network that contains only the jobs of the closed job classes and solve the model with the extended version of the MVA, which considers also the influence of jobs of the open job classes. The results are the performance measures for the closed job classes of the mixed queueing network.

Iteration: $\mathbf{k} = \mathbf{0}, \ldots, \mathbf{K}$:

STEP 2.1 For $i = 1, \ldots, N$ and $r = 1, \ldots, CL$, compute the mean response times with Eqs. (8.53) and (8.54).

STEP 2.2 For $r = 1, \ldots, CL$, compute the throughputs with Eq. (8.46).

STEP 2.3 For $i = 1, \ldots, N$ and $r = 1, \ldots, CL$, evaluate the mean number of jobs with Eq. (8.47).

STEP 3 With the help of the solutions of the closed model and the equations given in Section 7.2, compute the performance measures for the open job classes of the network starting with Eqs. (8.51) and (8.52) for computing the mean number of jobs $\overline{K}_{ir}$, $i = 1, \ldots, N$ and $r = 1, \ldots, OP$.

If we use the iterative formula (8.48) for computing the normalization constant in Step 2, then the steady-state probabilities can be derived from the BCMP theorem, Eq. (7.76) [AkBo83]:

$$\pi(\mathbf{S} = \mathbf{S}_1 \ldots, \mathbf{S}_N) = \frac{1}{G(\mathbf{K})} \prod_{j=0}^{\mathbf{K(S)}-1} \lambda(j) \prod_{i=1}^{N} F_i(\mathbf{S}_i), \tag{8.56}$$

where $\lambda(j)$ is the (possibly state dependent) arrival rate.

Example 8.6 As an example of a mixed queueing network, consider the model given in Fig. 8.9 with $N = 2$ nodes and $R = 4$ job classes. Class 1 and 2 are open, the classes 3 and 4 are closed. Node 1 is of Type-2 and node 2 is of Type-4.

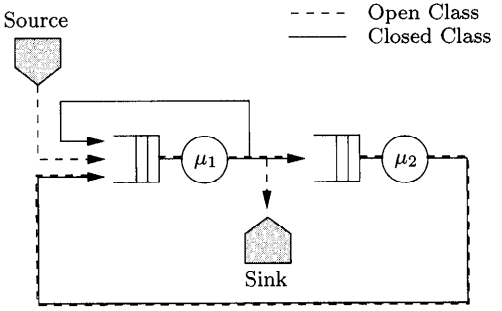


Fig. 8.9 A mixed queueing network.

The mean service times are given as follows:

$$\frac{1}{\mu_{11}} = 0.4\,\text{sec}, \quad \frac{1}{\mu_{12}} = 0.8\,\text{sec}, \quad \frac{1}{\mu_{13}} = 0.3\,\text{sec}, \quad \frac{1}{\mu_{14}} = 0.5\,\text{sec},$$
$$\frac{1}{\mu_{21}} = 0.6\,\text{sec}, \quad \frac{1}{\mu_{22}} = 1.6\,\text{sec}, \quad \frac{1}{\mu_{23}} = 0.5\,\text{sec}, \quad \frac{1}{\mu_{24}} = 0.8\,\text{sec}.$$

The arrival rates by class for the open class jobs are $\lambda_1 = 0.5$ jobs/sec and $\lambda_2 = 0.25$ jobs/sec. In each of the closed classes there is $K_3 = K_4 = 1$ job. The routing probabilities are:

$$\begin{aligned}
&p_{0,11} = 1, \quad p_{11,11} = 0, \quad p_{11,21} = 0.5, \quad p_{21,11} = 1,\\
&p_{0,12} = 1, \quad p_{12,12} = 0, \quad p_{12,22} = 0.6, \quad p_{22,12} = 1,\\
&p_{0,13} = 0, \quad p_{13,13} = 0.5, \quad p_{13,23} = 0.5, \quad p_{23,13} = 1,\\
&p_{0,14} = 0, \quad p_{14,14} = 0.6, \quad p_{14,24} = 0.4, \quad p_{24,14} = 1.
\end{aligned}$$

Thus, by Eq. (6.27) the visit ratios are computed to be:

$$\begin{aligned}
&e_{11} = 2, \quad e_{12} = 2.5, \quad e_{13} = 1, \quad e_{14} = 1,\\
&e_{21} = 1, \quad e_{22} = 1.5, \quad e_{23} = 0.5, \quad e_{24} = 0.4.
\end{aligned}$$

The determination of the performance measures of this mixed queueing network is carried out with the MVA in the following three steps:

STEP 1 Initialization. Compute the utilization of both nodes by the open job classes:

$$\begin{aligned}
\rho_{11} &= \lambda_1 e_{11} \cdot \frac{1}{\mu_{11}} = \underline{0.4}, \quad \rho_{21} = \lambda_1 e_{21} \cdot \frac{1}{\mu_{21}} = \underline{0.3},\\
\rho_{12} &= \lambda_2 e_{12} \cdot \frac{1}{\mu_{12}} = \underline{0.5}, \quad \rho_{22} = \lambda_2 e_{22} \cdot \frac{1}{\mu_{22}} = \underline{0.6}.
\end{aligned}$$

Set $\overline{K}_{11}(\underline{0}) = \overline{K}_{12}(\underline{0}) = \overline{K}_{21}(\underline{0}) = \overline{K}_{22}(\underline{0}) = 0$.

STEP 2 Using MVA, analyze the closed queueing network model, obtained by leaving out the jobs of the open job classes.

Mean value iteration for $\mathbf{k} = (1,0)$:

Mean response times, Eq. (8.53):

$$\overline{T}_{13}(1,0) = \frac{1}{\mu_{13}} \cdot \frac{1 + \overline{K}_{13}(\underline{0})}{1 - (\rho_{11} + \rho_{12})} = \underline{3}, \quad \overline{T}_{23}(1,0) = \frac{1}{\mu_{23}} \cdot \frac{1 + \overline{K}_{23}(\underline{0})}{1 - (\rho_{21} + \rho_{22})} = \underline{5}.$$

Throughput, Eq. (8.46):

$$\lambda_3(1,0) = \frac{K_3}{\sum_{i=1}^{2} e_{i3} \overline{T}_{i3}(1,0)} = \underline{0.182}.$$

Mean number of jobs, Eq. (8.47):

$$\begin{aligned}
\overline{K}_{13}(1,0) &= \lambda_3(1,0) \cdot e_{13} \cdot \overline{T}_{13}(1,0) = \underline{0.545},\\
\overline{K}_{23}(1,0) &= \lambda_3(1,0) \cdot e_{23} \cdot \overline{T}_{23}(1,0) = \underline{0.454}.
\end{aligned}$$

Mean value iteration for $\mathbf{k} = (0, 1)$:

$$\begin{aligned} \overline{T}_{14}(0,1) &= \underline{5}, & \overline{T}_{24}(0,1) &= \underline{8}, \\ \lambda_4(0,1) &= \underline{0.122}, \\ \overline{K}_{14}(0,1) &= \underline{0.610}, & \overline{K}_{24}(0,1) &= \underline{0.390}. \end{aligned}$$

Mean value iteration for $\mathbf{k} = (1, 1)$:

$$\begin{aligned} &\overline{T}_{13}(1,1) = \underline{4.829},\ \overline{T}_{23}(1,1) = \underline{6.951},\ \overline{T}_{14}(1,1) = \underline{7.727},\ \overline{T}_{24}(1,1) = \underline{11.636}. \\ &\lambda_3(1,1) = \underline{0.120}, \quad \lambda_4(1,1) = \underline{0.081}. \\ &\overline{K}_{13}(1,1) = \underline{0.582},\ \overline{K}_{23}(1,1) = \underline{0.418},\ \overline{K}_{14}(1,1) = \underline{0.624},\ \overline{K}_{24}(1,1) = \underline{0.376}. \end{aligned}$$

STEP 3 With these results and using Eq. (8.51), the performance measures of the open classes can be computed:

$$\begin{aligned} \overline{K}_{11}(1,1) &= \frac{\rho_{11}\left(1 + \overline{K}_{13}(1,1) + \overline{K}_{14}(1,1)\right)}{0.1} = \underline{8.822}, \\ \overline{K}_{21}(1,1) &= \frac{\rho_{21}\left(1 + \overline{K}_{23}(1,1) + \overline{K}_{24}(1,1)\right)}{0.1} = \underline{5.383}, \\ \overline{K}_{12}(1,1) &= \frac{\rho_{12}\left(1 + \overline{K}_{13}(1,1) + \overline{K}_{14}(1,1)\right)}{0.1} = \underline{11.028}, \\ \overline{K}_{22}(1,1) &= \frac{\rho_{22}\left(1 + \overline{K}_{23}(1,1) + \overline{K}_{24}(1,1)\right)}{0.1} = \underline{10.766}. \end{aligned}$$

For the computation of the mean response times we use, Eq. (7.43):

$$\begin{aligned} \overline{T}_{11}(1,1) &= \underline{8.822}, & \overline{T}_{12}(1,1) &= \underline{17.645}, \\ \overline{T}_{21}(1,1) &= \underline{10.766}, & \overline{T}_{22}(1,1) &= \underline{28.710}. \end{aligned}$$

Now all the other performance measures can be computed.

There are some suggestions to extend this algorithm to mixed queueing networks that are allowed to contain additional multiple server nodes. The most important approaches can be found in [KTK81] and [BBA84].

8.2.4 Networks with Load-Dependent Service

The MVA can be extended to examine product-form queueing networks where the service rates at node depends on the number of jobs at that node, [Reis81]. Let $\mu_i(j)$ denote the load-dependent service rate of the ith node when there are j jobs in it.

8.2.4.1 Closed Networks The algorithm for single class queueing networks with nodes having load-dependent service rates can be described in the following steps:

STEP 1 Initialization. For $i = 1, \ldots, N$: $\pi_i(0 \mid 0) = 1$.

STEP 2 Iteration: $k = 1, \ldots, K$.

STEP 2.1 For $i = 1, \ldots, N$, compute the mean response time:

$$\overline{T}_i(k) = \sum_{j=1}^{k} \frac{j}{\mu_i(j)} \pi_i(j-1 \mid k-1). \tag{8.57}$$

STEP 2.2 Compute the throughput:

$$\lambda(k) = \frac{k}{\sum_{i=1}^{N} e_i \cdot \overline{T}_i(k)}. \tag{8.58}$$

STEP 2.3 For $i = 1, \ldots, N$, compute the conditional probabilities of j jobs at the ith node given that there are k jobs in the network:

$$\pi_i(j \mid k) = \begin{cases} \dfrac{\lambda(k)}{\mu_i(j)} \pi_i(j-1 \mid k-1) e_i, & \text{for } j = 1, \ldots, k, \\ 1 - \displaystyle\sum_{l=1}^{k} \pi_i(l \mid k), & \text{for } j = 0. \end{cases} \tag{8.59}$$

The extension of this algorithm to multiclass queueing networks where the service rate of a node depends on the overall number of jobs at the node can be done very easily. Let $\mu_{ir}(j)$ be the the service rate of class-r jobs at the ith node when there are j jobs in it. Then the MVA can be presented in the following steps:

STEP 1 Initialization. For $i = 1, \ldots, N$: $\pi_i(0 \mid \mathbf{0}) = 1$

STEP 2 Iteration: $\mathbf{k} = \mathbf{0}, \ldots, \mathbf{K}$.

STEP 2.1 For $i = 1, \ldots, N$ and $r = 1, \ldots, R$, compute the mean response times:

$$\overline{T}_{ir}(\mathbf{k}) = \sum_{j=1}^{k} \frac{j}{\mu_{ir}(j)} \pi_i(j-1 \mid \mathbf{k} - \mathbf{1}_r), \quad \text{where } k = \sum_{r=1}^{R} k_r. \tag{8.60}$$

STEP 2.2 For $r = 1, \ldots, R$, compute the throughputs:

$$\lambda_r(\mathbf{k}) = \frac{k_r}{\sum_{i=1}^{N} e_{ir} \cdot \overline{T}_{ir}(\mathbf{k})}. \tag{8.61}$$

STEP 2.3 For $i = 1, \ldots, N$, determine the marginal probabilities:

$$\pi_i(j \mid \mathbf{k}) = \begin{cases} \sum_{r=1}^{R} \frac{\lambda_r(\mathbf{k})}{\mu_{ir}(j)} \pi_i(j-1 \mid \mathbf{k} - \mathbf{1}_r) e_{ir}, & \text{for } j = 1, \ldots, k, \\ 1 - \sum_{l=1}^{k} \pi_i(l \mid \mathbf{k}), & \text{for } j = 0. \end{cases} \tag{8.62}$$

Example 8.7 Consider again the example given in Fig. 8.6 where the multiple-server node 1 is replaced by a general load-dependent server. Load-dependent service rates are:

$$\begin{aligned} &\mu_1(1) = 2, \quad \mu_2(1) = 1.667, \quad \mu_3(1) = 1.25, \quad \mu_4(1) = 1, \\ &\mu_1(2) = 4, \quad \mu_2(2) = 1.667, \quad \mu_3(2) = 1.25, \quad \mu_4(2) = 2, \\ &\mu_1(3) = 4, \quad \mu_2(3) = 1.667, \quad \mu_3(3) = 1.25, \quad \mu_4(3) = 3. \end{aligned}$$

STEP 1 Initialization:

$$\pi_i(0 \mid 0) = 1 \quad \text{for } i = 1, 2, 3, 4.$$

STEP 2 Iteration over the number of jobs in the network starting with $k = 1$:

STEP 2.1 Mean response times, Eq. (8.57):

$$\begin{aligned} &\overline{T}_1(1) = \frac{1}{\mu_1(1)} \pi_1(0 \mid 0) = \underline{0.5}, \quad \overline{T}_2(1) = \frac{1}{\mu_2(1)} \pi_2(0 \mid 0) = \underline{0.6}, \\ &\overline{T}_3(1) = \frac{1}{\mu_3(1)} \pi_3(0 \mid 0) = \underline{0.8}, \quad \overline{T}_4(1) = \frac{1}{\mu_4(1)} \pi_4(0 \mid 0) = \underline{1}. \end{aligned}$$

STEP 2.2 Throughput, Eq. (8.61) and normalizing constant, Eq. (8.42):

$$\lambda(1) = \frac{1}{\sum_{i=1}^{4} e_i \overline{T}_i(1)} = \underline{0.454}, \quad G(1) = \frac{G(0)}{\lambda(1)} = \underline{2.203}.$$

STEP 2.3 Marginal probabilities, Eq. (8.62):

$$\pi_1(1 \mid 1) = \frac{\lambda(1)}{\mu_1(1)}\pi_1(0 \mid 0)e_1 = \underline{0.227},$$
$$\pi_1(0 \mid 1) = 1 - \pi_1(1 \mid 1) = \underline{0.773},$$
$$\pi_2(0 \mid 1) = \underline{0.864}, \quad \pi_2(1 \mid 1) = \underline{0.136},$$
$$\pi_3(0 \mid 1) = \underline{0.818}, \quad \pi_3(1 \mid 1) = \underline{0.182},$$
$$\pi_4(0 \mid 1) = \underline{0.545}, \quad \pi_4(1 \mid 1) = \underline{0.454}.$$

Iteration for $k = 2$:

STEP 2.1 Mean response times:

$$\overline{T}_1(2) = \frac{1}{\mu_1(1)}\pi_1(0 \mid 1) + \frac{2}{\mu_1(2)}\pi_1(1 \mid 1) = \underline{0.5},$$
$$\overline{T}_2(2) = \frac{1}{\mu_2(1)}\pi_2(0 \mid 1) + \frac{2}{\mu_2(2)}\pi_2(1 \mid 1) = \underline{0.682},$$
$$\overline{T}_3(2) = \frac{1}{\mu_3(1)}\pi_3(0 \mid 1) + \frac{2}{\mu_3(2)}\pi_3(1 \mid 1) = \underline{0.946},$$
$$\overline{T}_4(2) = \frac{1}{\mu_4(1)}\pi_4(0 \mid 1) + \frac{2}{\mu_4(2)}\pi_4(1 \mid 1) = \underline{1}.$$

STEP 2.2 Throughput and normalizing constant:

$$\lambda(2) = \frac{2}{\sum_{i=1}^{4} e_i \overline{T}_i(2)} = \underline{0.864}, \quad G(2) = \frac{G(1)}{\lambda(2)} = \underline{2.549}.$$

STEP 2.3 Marginal probabilities:

$$\pi_1(2 \mid 2) = \frac{\lambda(2)}{\mu_1(2)}\pi_1(1 \mid 1)e_1 = \underline{0.049},$$
$$\pi_1(1 \mid 2) = \frac{\lambda(2)}{\mu_1(1)}\pi_1(0 \mid 1)e_1 = \underline{0.334},$$
$$\pi_1(0 \mid 2) = 1 - \sum_{l=1}^{2} \pi_1(l \mid 2) = \underline{0.617}.$$
$$\pi_2(2 \mid 2) = \underline{0.035}, \quad \pi_2(1 \mid 2) = \underline{0.224}, \quad \pi_2(0 \mid 2) = \underline{0.741},$$
$$\pi_3(2 \mid 2) = \underline{0.063}, \quad \pi_3(1 \mid 2) = \underline{0.283}, \quad \pi_3(0 \mid 2) = \underline{0.654},$$
$$\pi_4(2 \mid 2) = \underline{0.196}, \quad \pi_4(1 \mid 2) = \underline{0.472}, \quad \pi_4(0 \mid 2) = \underline{0.332}.$$

Iteration for $k = 3$:

STEP 2.1 Mean response times:

$$\overline{T}_1(3) = \frac{1}{\mu_1(1)}\pi_1(0 \mid 2) + \frac{2}{\mu_1(2)}\pi_1(1 \mid 2) + \frac{3}{\mu_1(3)}\pi_1(2 \mid 2) = \underline{0.512},$$
$$\overline{T}_2(3) = \underline{0.776}, \quad \overline{T}_3(3) = \underline{1.127}, \quad \overline{T}_4(3) = \underline{1}.$$

STEP 2.2 Throughput and normalizing constant:

$$\lambda(3) = \underline{1.218}, \quad G(3) = \frac{G(2)}{\lambda(3)} = \underline{2.093}.$$

STEP 2.3 Marginal probabilities:

$$\pi_1(3 \mid 3) = \frac{\lambda(3)}{\mu_1(3)}\pi_1(2 \mid 2)e_1 = \underline{0.015},$$
$$\pi_1(2 \mid 3) = \frac{\lambda(3)}{\mu_1(2)}\pi_1(1 \mid 2)e_1 = \underline{0.102},$$
$$\pi_1(1 \mid 3) = \frac{\lambda(3)}{\mu_1(1)}\pi_1(0 \mid 2)e_1 = \underline{0.375},$$
$$\pi_1(0 \mid 3) = 1 - \sum_{l=1}^{3}\pi_1(l \mid 3) = \underline{0.507}.$$
$$\pi_2(3 \mid 3) = \underline{0.013}, \quad \pi_2(2 \mid 3) = \underline{0.082},$$
$$\pi_2(1 \mid 3) = \underline{0.271}, \quad \pi_2(0 \mid 3) = \underline{0.634},$$
$$\pi_3(3 \mid 3) = \underline{0.031}, \quad \pi_3(2 \mid 3) = \underline{0.138},$$
$$\pi_3(1 \mid 3) = \underline{0.319}, \quad \pi_3(0 \mid 3) = \underline{0.512},$$
$$\pi_4(3 \mid 3) = \underline{0.080}, \quad \pi_4(2 \mid 3) = \underline{0.287},$$
$$\pi_4(1 \mid 3) = \underline{0.404}, \quad \pi_4(0 \mid 3) = \underline{0.229}.$$

Now the iteration stops and all other performance measures can be computed. For the mean number of jobs at the nodes we have, for example:

$$\overline{K}_1(3) = \sum_{k=1}^{3} k \cdot \pi_1(k \mid 3) = \underline{0.624},$$
$$\overline{K}_2(3) = \underline{0.473}, \quad \overline{K}_3(3) = \underline{0.686}, \quad \overline{K}_4(3) = \underline{1.217}.$$

The MVA is extended by [Saue83] to deal with other kinds of load dependence:

- Networks in which the service rate of a node depends on the number of jobs of the different classes at that node (not just on the total number of jobs at the node).
- Networks with load-dependent routing probabilities.

In [TuSa85] and [HBAK86], the MVA is extended to the *tree mean value analysis*, which is well suited for very large networks with only a few jobs. Such networks arise especially while modeling distributed systems.

8.2.4.2 Mixed Networks The MVALDMX algorithm (**m**ean **v**alue **a**nalysis **l**oad **d**ependent **mix**ed) was first presented in [BBA84] and can be used to analyze mixed BCMP networks with load-dependent service stations. Apart from the algorithm presented in [BBA84], another suggestion is made in [KTK81] to extend the MVA to analyze multiclass load-dependent queueing networks. In order to extend the MVA, the equations derived in the previous sections, and especially the equation for the mean response time, need to be extended. Since a mixed network contains open as well as closed job classes, we need to consider the open classes in the network when computing the performance measures for the closed classes. Furthermore the load-dependency must be explicitly accounted for. To do so we need to consider marginal probabilities in the corresponding equations.

Bruell, Balbo, and Afshari [BBA84] introduce the *capacity function* to describe the behavior of load-dependent service stations. The capacity function $c_i(j)$ of node i is the number of jobs completed by that station if there are j jobs in the queue.

The inverse function of the capacity function, $C_i(j)$, is called the *load-dependence factor*.

The *load-dependent mean service time* $s_{ir}(j)$ for class-r jobs at service station i is given by the relation

$$s_{ir}(j) = s_{ir} \cdot C_i(j). \tag{8.63}$$

The function $s_{ir}(j)$ can be interpreted as the *nominal service time*. Using the capacity function $c_i(j)$, we can model a general load-dependency, which means, $c_i(j)$ can be any function with:

$$c_i : \mathbb{N} \to \mathbb{R}^+ \text{ and } c_i(1) = 1.$$

In the following, load-dependency shall be restricted to multiple server nodes. The load-dependency factor for an -/M/m node is then:

$$C_i(k) = \frac{1}{c_i(k)} = \begin{cases} \dfrac{1}{k}, & \text{if } k < m_i, \\ \dfrac{1}{m_i}, & \text{if } k \geq m_i. \end{cases} \tag{8.64}$$

In the MVA for mixed queueing networks, and especially in the MVALDMX algorithm, the procedure is to consider the performance measures for open and closed classes separately from each other. At first the performance measures of the closed classes are computed and then the performance measures of the open classes are computed. This separation can be achieved in the MVALDMX algorithm by checking the marginal probabilities of different job combinations. In [BBA84] a recursive equation is given for the computation of the conditional probabilities $\pi_i(k, \mathbf{K})$ such that there are exactly k jobs of the closed job classes at node i for a given population $\mathbf{K}$. This equation is only valid for the closed job classes. This recursive equation is the only one necessary for the analysis of mixed, load-dependent queueing networks:

$$\pi_i(k, \mathbf{K}) = \sum_{cl=1}^{CL} s_{i,cl} \cdot EC_i(k) \cdot \lambda_{i,cl}(\mathbf{K}) \cdot \pi_i(k-1, \mathbf{K} - \mathbf{1}_{cl}). \tag{8.65}$$

Here, $\mathbf{K}$ is the population vector of jobs in the closed classes, and $\mathbf{1}_{cl}$ is the vector whose components are all zero except the clth component, which is one. The variable k is the overall number of jobs in closed classes in a certain population vector during the iteration, and EC_i is the inverse of the effective capacity function. The reduction of the mixed network to a closed network is given by modifying the load-dependence factor $C_i(k)$. In a mixed queueing network, the service stations are used by jobs belonging to open as well as closed classes. The availability of open job classes in the network reduces the capacities of the service stations, which means that the closed job classes do not have the full capacity available any more as previously defined in the capacity function.

It is for this reason that we need to define the *effective capacity* function:

$$ec_i(k) = \frac{1}{EC_i(k)} = c_i(k) \cdot \frac{E_i(k-1)}{E_i(k)}. \tag{8.66}$$

This function describes the capacity that is effectively available for the closed classes if we consider the jobs of open classes. This step makes a separate analysis of the open and closed classes possible. As we can see, $E_i(k)$ (which is a function that helps in computing the effective capacity of station i) is of basic importance for this reduction. The computation of the auxiliary parameters can be done for each service station i independently from the other service stations.

$$\forall k > m_i - 1 \quad : \quad E_i(k) = \frac{1}{1 - L_i \cdot C_i(m_i)} \cdot E_i(k-1),$$

$$L_i = \sum_{op=1}^{OP} \lambda_{i,op} \cdot s_{i,op},$$

$$\forall k \le m_i - 1 : \begin{cases} E_i(k) = E_{i1}(k) + E_{i2}(k) - E_{i3}(k) \\ E_{i1}(k) = \begin{cases} \dfrac{1}{1 - L_i C_i(m_i)} \cdot \dfrac{C_i(m_i)}{C_i(k)} \cdot E_{i1}(k-1), & k > 0, \\ \dfrac{1}{1 - L_i C_i(m_i)} \cdot \displaystyle\prod_{j=1}^{m_i - 1} \dfrac{C_i(j)}{C_i(m_i)}, & k = 0, \end{cases} \\ E_{i2}(k) = \displaystyle\sum_{j=0}^{m_i - 2} F_{i2}(k,j), \\ E_{i3}(k) = \displaystyle\sum_{j=0}^{m_i - 2} F_3(k,j), \\ F_{i2}(k,j) = \begin{cases} \dfrac{k+j}{j} L_i \cdot C_i(k+j) \cdot F_{i2}(k, j-1), & j > 0, \\ 1, & j = 0, \end{cases} \\ F_{i3}(k,j) = \begin{cases} \dfrac{k+j}{j} \cdot L_i \cdot C_i(m_i) \cdot F_{i3}(k, j-1), & \begin{array}{l} k \ge 0, \\ j > 0, \end{array} \\ \dfrac{C_i(m_i)}{C_i(k)} \cdot F_{i3}(k-1, 0), & \begin{array}{l} k > 0, \\ j > 0, \end{array} \\ \displaystyle\prod_{l=1}^{m_i - 1} \dfrac{C_i(l)}{C_i(m_i)}, & \begin{array}{l} k = 0, \\ j = 0. \end{array} \end{cases} \end{cases}$$

Here L_i denotes the load factor of open classes with regard to station i. Now we have all necessary equations to describe the MVA algorithm for mixed product-form queueing networks with multiple server nodes. In the algorithm, K denotes the overall number of jobs in the closed job classes of the network and $\mathbf{k}$ denotes the state vector for an iteration step for the closed job classes.

STEP 1 Initialization. For all nodes $i = 1, \ldots, N$, of the network compute the following values:

STEP 1.1 Load factor L_i of the open job classes:

$$L_i = \sum_{op=1}^{OP} \lambda_{i,op} \cdot s_{i,op}.$$

STEP 1.2 Let $K = \displaystyle\sum_{cl=1}^{CL} K_{cl}$ and compute for all $k = 1, \ldots, K+1$ the auxiliary functions $E_i(k)$ and $EC_i(k)$.

STEP 1.3 $\pi_i(0, \mathbf{0}) = 1.0$.

STEP 2 Construct a closed model that contains only jobs of closed job classes of the original network and solve this closed model. The results that we get from this model are the performance measures for the *closed* classes of the mixed network. For all vectors $\mathbf{k} = \mathbf{0}, \ldots, \mathbf{K}$, do:

STEP 2.1 For all $cl = 1, \ldots, CL$ and $i = 1, \ldots, N$, compute the mean response time:

$$\overline{T}_{i,cl}(\mathbf{k}) = \begin{cases} s_{i,cl} \cdot \sum_{j=1}^{k} j \cdot EC_i(j) \cdot \pi_i(j-1, \mathbf{k} - \mathbf{1}_{cl}), & \text{Type-1,2,4,} \\ s_{i,cl}, & \text{Type-3.} \end{cases}$$

STEP 2.2 For all $cl = 1, \ldots, CL$ and $i = 1, \ldots, N$, compute the throughput:

$$\lambda_{i,cl}(\mathbf{k}) = \frac{k_{cl} \cdot e_{i,cl}}{\sum_{j=1}^{N} e_{j,cl} \cdot \overline{T}_{j,cl}(\mathbf{k})}.$$

STEP 2.3 For all $cl = 1, \ldots, CL$ and $i = 1, \ldots, N$, compute the number of jobs:

$$\overline{K}_{i,cl}(\mathbf{k}) = \overline{T}_{i,cl}(\mathbf{k}) \cdot \lambda_{i,cl}(\mathbf{k}).$$

STEP 2.4 Adapt the marginal probabilities for Type-1,2,4 nodes:

$k = \sum_{cl=1}^{CL} k_{cl}.$

For $i = 1, \ldots, N$ do:

$sum = 0.0$

for $k = 1, \ldots, k$ do:

$$\pi_i(k, \mathbf{k}) = \sum_{cl=1}^{CL} s_{i,cl} \cdot EC_i(k) \cdot \lambda_{i,cl}(\mathbf{k}) \cdot \pi_i(k-1, \mathbf{k} - \mathbf{1}_{cl}),$$

$\text{sum} = \text{sum} + \pi_i(k, \mathbf{k}),$

$\pi_i(0, \mathbf{k}) = 1.0 - \text{sum}.$

STEP 2.5 Compute all other performance measures using the well-known formulae (see Section 7.1)

STEP 3 Compute from the solution of the *closed* model, the performance measures for the open classes of the network. For $op = 1, \ldots, OP$ and $i =$

$1, \ldots, N$, compute:

$$\overline{K}_{i,op}(\mathbf{K}) = \begin{cases} \lambda_{i,op} \cdot s_{i,op} \cdot \sum_{k=0}^{K} (k+1) \cdot EC_i(k+1) \cdot \pi_i(k, \mathbf{K}), & \text{Type-1,2,4,} \\ \lambda_{i,op} \cdot s_{i,op}, & \text{Type-3.} \end{cases}$$

$$\overline{T}_{i,op}(\mathbf{K}) = \begin{cases} \dfrac{\overline{K}_{i,op}(\mathbf{K})}{\lambda_{i,op}}, & \text{Type-1,2,4,} \\ s_{i,op}, & \text{Type-3.} \end{cases}$$

In the following example, we show how to apply the method to a simple queueing network:

Example 8.8 Consider the network given in Fig. 8.9. The mean service times are given as follows:

$$\frac{1}{\mu_{11}} = \underline{0.4}\,\text{sec}, \quad \frac{1}{\mu_{12}} = \underline{0.8}\,\text{sec}, \quad \frac{1}{\mu_{13}} = \underline{0.3}\,\text{sec}, \quad \frac{1}{\mu_{14}} = \underline{0.5}\,\text{sec},$$
$$\frac{1}{\mu_{21}} = \underline{0.6}\,\text{sec}, \quad \frac{1}{\mu_{22}} = \underline{1.6}\,\text{sec}, \quad \frac{1}{\mu_{23}} = \underline{0.5}\,\text{sec}, \quad \frac{1}{\mu_{24}} = \underline{0.8}\,\text{sec}.$$

The arrival rates for the open class jobs are $\lambda_1 = 0.5$ jobs/sec and $\lambda_2 = 0.25$ jobs/sec. In each of the closed classes there is $K_3 = K_4 = 1$ job. The routing probabilities are:

$$\begin{aligned} &p_{0,11} = 1, \quad p_{11,11} = 0, \quad p_{11,21} = 0.5, \quad p_{21,11} = 1, \\ &p_{0,12} = 1, \quad p_{12,12} = 0, \quad p_{12,22} = 0.6, \quad p_{22,12} = 1, \\ &p_{0,13} = 0, \quad p_{13,13} = 0.5, \quad p_{13,23} = 0.5, \quad p_{23,13} = 1, \\ &p_{0,14} = 0, \quad p_{14,14} = 0.6, \quad p_{14,24} = 0.4, \quad p_{24,14} = 1. \end{aligned}$$

STEP 1 For both nodes of the mixed network compute:

STEP 1.1

$$L_1 = \sum_{op=1}^{OP} \lambda_{1,op} \cdot s_{1,op} = \lambda_{11} \cdot s_{11} + \lambda_{12} \cdot s_{12} = \underline{0.9}\ \text{jobs},$$

$$L_2 = \sum_{op=1}^{OP} \lambda_{2,op} \cdot s_{2,op} = \lambda_{21} \cdot s_{21} + \lambda_{22} \cdot s_{22} = \underline{0.9}\ \text{jobs}.$$

STEP 1.2 Set: $K = \sum_{cl=1}^{CL} K_{cl} = K_3 + K_4 = \underline{2}$.

For $k = 1, \ldots, K^{CL} + 1$ compute $E_i(k)$ and $EC_i(k)$:

$$
\begin{aligned}
i = 1: \quad & E_{11}(1) = E_{11}(1) + E_{12}(1) - E_{13}(1) = \underline{100}, \\
& E_{11}(2) = E_{11}(2) + E_{12}(2) - E_{13}(2) = \underline{1000}, \\
& E_{11}(3) = E_{11}(3) + E_{12}(3) - E_{13}(3) = \underline{10000}, \\
& EC_1(k) = \frac{C_1(m_1)}{1 - L_1^{op} \cdot C_1(m_1)} = \frac{1}{1 - L_1^{op}} = \underline{10}, \\
i = 2: \quad & \text{computed in a similar manner as for } i = 1.
\end{aligned}
$$

STEP 1.3

$$\pi_1(0, (0,0)) = \pi_2(0, (0,0)) = \underline{1.0}.$$

STEP 2 Analysis of the closed model, which we get after removing all open job classes in the network.

Iteration for $\mathbf{k} = (1,0)$:

STEP 2.1 Compute the mean response times:

$$
\begin{aligned}
& \overline{T}_{13}(1,0) = s_{13} \cdot \sum_{r=1}^{1} r \cdot EC_1(r) \cdot \pi_1(0, (0,0)) = \underline{3}, \\
& \overline{T}_{23}(1,0) = \ldots = \underline{5}.
\end{aligned}
$$

STEP 2.2 Compute the throughputs:

$$
\begin{aligned}
& \lambda_{13}(1,0) = \frac{k_3 \cdot e_{13}}{\sum_{j=1}^{2} e_{j3} \cdot \overline{T}_{j3}(1,0)} = \underline{0.182}, \\
& \lambda_{23}(1,0) = \ldots = \underline{0.091}.
\end{aligned}
$$

STEP 2.3 Compute the mean queue lengths:

$$
\begin{aligned}
& \overline{K}_{13}(1,0) = \lambda_{13}(1,0) \cdot \overline{T}_{13}(1,0) = \underline{0.545}, \\
& \overline{K}_{23}(1,0) = \ldots = \underline{0.455}.
\end{aligned}
$$

STEP 2.4 Adapt the marginal probabilities:

$$k = \sum_{cl=1}^{CL} k_{cl} = \underline{1}.$$

$i = 1$: sum $= 0.0$,
for $k = 1$:

$$\pi_1(1,(1,0)) = \sum_{cl=1}^{CL} s_{1,cl} \cdot EC_1(1) \cdot \lambda_{1,cl}(1,0) \cdot \pi_1(0, \mathbf{k} - \mathbf{1}_{cl}))$$

$$= \ldots = \underline{0.546},$$

$$\text{sum} = \text{sum} + \pi_1(1,(1,0)) = \underline{0.546},$$

$$\pi_1(0,(1,0)) = 1.0 - \text{sum} = \underline{0.454},$$

$i = 2$: sum $= 0.0$,
for $k = 1$:

$$\pi_2(1,(1,0)) = \ldots = \underline{0.454},$$

$$\pi_2(0,(1,0)) = \ldots = \underline{0.546}.$$

Iteration for $\mathbf{k} = (0,1)$:

STEP 2.1 Mean response times:

$$\overline{T}_{14}(0,1) = \ldots = \underline{5}, \quad \overline{T}_{24}(0,1) = \ldots = \underline{8}.$$

STEP 2.2 Throughputs:

$$\lambda_{14}(0,1) = \underline{0.122}, \quad \lambda_{24}(0,1) = \underline{0.049}.$$

STEP 2.3 Mean queue lengths:

$$\overline{K}_{14}(0,1) = \underline{0.610}, \quad \overline{K}_{24}(0,1) = \underline{0.392}.$$

STEP 2.4 Adapt the marginal probabilities:

$$k = \sum_{cl=1}^{CL} k_{cl} = \underline{1}.$$

$$\pi_1(1,(0,1)) = \underline{0.610}, \quad \pi_1(0,(0,1)) = \underline{0.390},$$
$$\pi_2(1,(0,1)) = \underline{0.390}, \quad \pi_2(0,(0,1)) = \underline{0.610}.$$

Iteration for $\mathbf{k} = (1,1)$:

STEP 2.1 Mean response times:

$$\overline{T}_{13}(1,1) = \underline{4.829}, \quad \overline{T}_{23}(1,1) = \underline{6.951},$$
$$\overline{T}_{14}(1,1) = \underline{7.727}, \quad \overline{T}_{24}(1,1) = \underline{11.636}.$$

STEP 2.2 Throughputs:

$$\lambda_{13}(1,1) = \underline{0.120}, \quad \lambda_{23}(1,1) = \underline{0.060},$$
$$\lambda_{14}(1,1) = \underline{0.081}, \quad \lambda_{24}(1,1) = \underline{0.032}.$$

STEP 2.3 Mean queue lengths:

$$\overline{K}_{13}(1,1) = \underline{0.582}, \quad \overline{K}_{23}(1,1) = \underline{0.418},$$
$$\overline{K}_{14}(1,1) = \underline{0.624}, \quad \overline{K}_{24}(1,1) = \underline{0.376}.$$

STEP 2.4 Adapt the marginal probabilities:

$$k = \sum_{cl=1}^{CL} k_{cl} = \underline{2}.$$

$$\begin{aligned} i = 1: \quad & \text{sum} = 0.0, \\ & \pi_1(1,(1,1)) = \underline{0.324}, \\ & \text{sum} = \text{sum} + 0.324 = \underline{0.324}. \\ & \pi_1(2,(1,1)) = \underline{0.441}, \\ & \text{sum} = \text{sum} + 0.441 = \underline{0.765}, \\ & \pi_1(0,(1,1)) = 1 - \text{sum} = \underline{0.235}, \end{aligned}$$

$$\begin{aligned} i = 2: \quad & \text{sum} = 0.0, \\ & \pi_2(1,(1,1)) = \underline{0.332}, \\ & \pi_2(2,(1,1)) = \underline{0.234}, \\ & \pi_2(0,(1,1)) = \underline{0.434}. \end{aligned}$$

STEP 3 Now we can compute the performance measures for the open job classes. First the mean number of jobs by class and node:

$$\begin{aligned} \overline{K}_{11}(1,1) &= \lambda_{11} \cdot s_{11} \cdot \sum_{k=0}^{2} (k+1) \cdot EC_1(k+1) \cdot P_1^{cl}(k,(1,1)) \\ &= \underline{8.822}, \\ \overline{K}_{21}(1,1) &= \underline{5.383}, \quad \overline{K}_{12}(1,1) = \underline{11.028}, \quad \overline{K}_{22}(1,1) = \underline{10.766}, \end{aligned}$$

Then the mean response time by class and node:

$$\overline{T}_{11}(1,1) = \frac{\overline{K}_{11}(1,1)}{\lambda_{11}} = \underline{8.822},$$
$$\overline{T}_{12}(1,1) = \underline{17.648}, \quad \overline{T}_{21}(1,1) = \underline{16.766}, \quad \overline{T}_{22}(1,1) = \underline{28.710}.$$

8.3 THE RECAL METHOD

Another technique for the exact solution of closed product-form queueing networks is the RECAL (**re**cursion by **c**hain **al**gorithm) algorithm. RECAL is very well suited for networks with a large number of job classes but a small number of nodes. For the explanation of this method we assume that we have a product-form network $\mathcal{N}$ consisting only of single server or infinite server nodes. We consider only the case of a multiple class network ($R > 1$) and introduce an extended normalizing constant $G_R(\mathbf{v})$ for the network with the vector $\mathbf{v} = (v_1, \ldots, v_N)$. Here, $G_R(\mathbf{0})$ is the standard normalizing constant $G(K)$, called G for short in the following. In [CoGe86] a recursive expression for computing $G = G_R(\mathbf{0})$ is given, which relates the normalizing constant of a network with r classes and the normalizing constants of a set of networks with each of the $(r-1)$ classes. The RECAL algorithm uses a simplified version of this general relation. This simplification can be given when each job class contains exactly one job, because if $K_r = 1$ for $r = 1, \ldots, R$, then the normalizing constant $G_R(\mathbf{0})$ is recursively given by the expression [CoGe86]:

$$G_r(\mathbf{v}_r) = \sum_{i=1}^{N} (1 + v_{ir}\delta_i) \frac{e_{ir}}{\mu_{ir}} G_{r-1}(\mathbf{v}_r + \mathbf{1}_i) \quad \text{for } \mathbf{v}_r \in \mathcal{F}_r, \tag{8.67}$$

with:

$\mathbf{v}_r = (v_{1r}, \ldots, v_{Nr})$,

$$\mathcal{F}_r = \begin{cases} \left\{ \mathbf{v}_r \;\middle|\; v_{ir} \geq 0 \text{ for } i = 1, \ldots, N, \sum_{i=1}^{N} v_{ir} = \sum_{s=r+1}^{R} K_s \right\}, & 0 \leq r \leq R-1, \\ \{0\}, & r = R, \end{cases}$$

$$\delta_i = \begin{cases} 1, & \text{Type-1,2,4,} \\ 0, & \text{Type-3,} \end{cases}$$

$\mathbf{1}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ is an N-dimensional vector with 1 at the ith position.

With the help of the normalizing constant $G_R(\mathbf{0})$, the throughput and the mean number of class-R jobs can be computed because for $K_R = 1$ the correctness of the following equations are proven in [CoGe86]:

$$\lambda_{iR} = \begin{cases} e_{iR} \cdot \sum_{j=1}^{N} \dfrac{G_{R-1}(\mathbf{1}_j)}{G_R(\mathbf{0}) \cdot (N + K - 1)}, & \text{if there are no Type-3 nodes in the network,} \\ e_{iR} \cdot \dfrac{G_{R-1}(\mathbf{1}_x)}{G_R(\mathbf{0})} & \text{for a Type-3 node with index } x, \end{cases} \tag{8.68}$$

and:

$$\overline{K}_{ir} = \frac{g_{r-1}(\mathbf{1}_i)}{g_r(\mathbf{0})} \cdot \frac{e_{iR}}{\mu_{iR}}. \tag{8.69}$$

The basic idea of the RECAL algorithm is to generate a fictitious network containing one job per job class in order to be able to compute the performance measures using Eqs. (8.67), (8.68), and (8.69). Therefore, it must be possible to partition each class r into K_r identical subclasses with exactly one job per job class. This partitioning changes the state space and the normalizing constant of the network, but the performance measures remain the same.

Let $\mathcal{N}^*$ denote the fictitious network that we get after partitioning the job classes into subclasses. The overall number of jobs in this fictitious network:

$$K^* = \sum_{r=1}^{R} K_r = K,$$

and the number of job classes is, by definition of $\mathcal{N}^*$, exactly $R^* = K^*$. Each class $r = 1, \ldots, R^*$ contains exactly $K_r^* = 1$ job. Let the jobs be numbered $1, 2, \ldots, K^*$ and let the function $c(k)$ denote the class index of the kth node in the original network.

Using Eq. (8.67), the normalizing constant $G^*_{R^*}(\mathbf{0})$ of the fictitious network $\mathcal{N}^*$ is recursively given by:

$$G_k^*(\mathbf{v}_k) = \sum_{i=1}^{N} (1 + v_{ik}\delta_i) \frac{e_{ic(k)}}{\mu_{ic(k)}} G_{k-1}^*(\mathbf{v}_k + \mathbf{1}_i), \tag{8.70}$$

for $k = 1, \ldots, K^*$ and $\mathbf{v}_k \in \mathcal{F}_k^*$ with:

$$\mathcal{F}_k^* = \left\{ \mathbf{v}_k \;\middle|\; v_{ik} \geq 0 \quad \text{for } i = 1, \ldots, N; \; \sum_{i=1}^{N} v_{ik} = K^* - k \right\},$$

and the initial conditions:

$$G_0^*(\mathbf{v}_0) = 1 \quad \text{for all } \mathbf{v}_0 \in \mathcal{F}_0^*.$$

With Eqs. (8.68) and (8.69) the performance measures $\lambda^*_{iR^*}$ and $\overline{K}^*_{iR^*}$ relating to a class-$c(R^*)$ job in the original network $\mathcal{N}$ can be computed:

$$\lambda^*_{iR^*} = \begin{cases} e_{ic(R^*)} \cdot \displaystyle\sum_{j=1}^{N} \frac{G^*_{R^*-1}(\mathbf{1}_j)}{G^*_{R^*}(\mathbf{0}) \cdot (N + K^* - 1)}, & \text{if there are no Type-3 nodes in the network,} \\ e_{ic(R^*)} \cdot \dfrac{G^*_{R^*-1}(\mathbf{1}_x)}{G^*_{R^*}(\mathbf{0})}, & \text{for a Type-3 node with index } x, \end{cases} \tag{8.71}$$

and:

$$\overline{K}^*_{iR^*} = \frac{G^*_{R^*-1}(\mathbf{1}_i)}{G^*_{R^*}(\mathbf{0})} \cdot \frac{e_{ic(R^*)}}{\mu_{ic(R^*)}}. \tag{8.72}$$

From the fact that all the jobs of a given class in the network $\mathcal{N}$ are identical, the throughputs and the mean number of jobs in class $c(R^*)$ of the network $\mathcal{N}$ are given by:

$$\lambda_{ic(R^*)} = K_{c(R^*)} \cdot \lambda^*_{iR^*}, \tag{8.73}$$

$$\overline{K}_{ic(R^*)} = K_{c(R^*)} \cdot \overline{K}^*_{iR^*}. \tag{8.74}$$

Performance measures for classes other than $c(R^*)$ can be obtained by relabeling the jobs in the fictitious network $\mathcal{N}^*$ with the goal that the assignment $c(R^*)$ belongs to another class in the original network $\mathcal{N}$. At the beginning, the numbering of jobs in $\mathcal{N}^*$ is chosen with the following assignment to the jobs in $\mathcal{N}$:

$$c^{(1)}(k) = \begin{cases} 1, & k = 1, \ldots, K_1 - 1, \\ i, & 2 \le i \le R \text{ and} \\ & k = 1 + \sum_{r=1}^{i-1}(K_r - 1), \ldots, \sum_{r=1}^{i}(K_r - 1), \\ R - k + 1 + \sum_{r=1}^{R}(K_r - 1), & k = 1 + \sum_{r=1}^{R}(K_r - 1), \ldots, K^*. \end{cases} \tag{8.75}$$

After computing $G^*_{R^*}(\mathbf{0})$ using Eq. (8.70), performance measures for class $s = 1$ in the original network $\mathcal{N}$ can be determined using Eqs. (8.71)-(8.74), due to Eq. (8.75). For the computation of performance measures of class-$(s+1)$ jobs in the network $\mathcal{N}$, the jobs in $\mathcal{N}^*$ have to be relabeled afterwards in the following way:

$$c^{(s+1)}(k) = \begin{cases} c^{(s)}(k), & k = 1, \ldots, s - 1 + \sum_{r=1}^{R}(K_r - 1), \\ s, & k = s + \sum_{r=1}^{R}(K_r - 1), \\ c^{(s)}(k) + 1, & k = s + 1 + \sum_{r=1}^{R}(K_r - 1), \ldots, K^*. \end{cases} \tag{8.76}$$

Now, the performance measures for class-$(s+1)$ jobs can be computed after computing $G^*_{R^*}(\mathbf{0})$ anew. The RECAL algorithm can be described in the following four steps:

STEP 1 Initialization:

$$G_0^*(\mathbf{v}_0) = 1 \quad \text{for all } \mathbf{v}_0 \in \mathcal{F}_0^*.$$

STEP 2 Number the jobs in the fictitious network $\mathcal{N}^*$ according to the assignment $c^{(1)}(k)$ given in Eq. (8.75).

STEP 3 Compute and store the values of $G_x^*(\mathbf{v}_x)$ for all $v_x \in \mathcal{F}_x^*$ with $x = K^* - R$ by using Eq. (8.70).

STEP 4 Iteration over all classes $s = 1, \ldots, R$ in the network $\mathcal{N}$:

STEP 4.1 Compute $G_{K^*}^*(\mathbf{0})$ with Eq. (8.70) by using the stored values for the $G_x^*(\mathbf{v}_x)$.

STEP 4.2 Compute the performance measures of class s using Eqs. (8.71)-(8.74). If $s = R$ then the algorithm stops.

STEP 4.3 For computing the performance measures of class-$s + 1$ jobs, relabel the jobs in $\mathcal{N}^*$ corresponding to the assignment (8.76).

STEP 4.4 Increase x by 1 and compute and store the values of the $G_x^*(\mathbf{v}_x)$ anew for all $\mathbf{v}_x \in \mathcal{F}_x^*$ by using Eq. (8.70).

By storing the values $G_x^*(\mathbf{v}_x)$ computed in Step 3 and Step 4.4, it is not necessary to compute Eq. (8.70) in Step 4.1 anew for all $k = 1, \ldots, K^*$ when evaluating the performance measures. Instead, Eq. (8.70) needs to be calculated only for $k = x, \ldots, K^*$ with $x = s + \sum_{r=1}^{R}(K_r - 1)$. The algorithm is illustrated by the following example:

Example 8.9 Consider a closed queueing network with $N = 2$ nodes and $R = 3$ job classes and with $K = 5$ jobs distributed as follows: $K_1 = 2, K_2 = 1, K_3 = 2$. We have $K = R^* = K^* = 5$. Class switching is not allowed. Node 1 is a Type-2 node and node 2 is a Type-1 node. The service rates are given as follows:

$$\mu_{11} = 200, \quad \mu_{12} = 50, \quad \mu_{13} = 2, \quad \mu_{21} = \mu_{22} = \mu_{23} = 4,$$

and the visit ratios are:

$$e_{11} = e_{12} = e_{13} = 1, \quad e_{21} = 0.72, \quad e_{22} = 0.63, \quad e_{23} = 0.4.$$

The computation of the performance measures is carried using the RECAL algorithm as follows:

STEP 1 Initialization:

$$G_0^*(\mathbf{v}_0) = 1 \quad \text{for all } \mathbf{v}_0 \in \mathcal{F}_0^*,$$

with $\mathcal{F}_0^* = \left\{ \mathbf{v}_0 \;\middle|\; v_{i0} \geq 0; \sum_{i=1}^{2} v_{i0} = R^* \right\}$.

STEP 2 Labeling the jobs in the network $\mathcal{N}^*$ with Eq. (8.75) yields:

$$c^{(1)}(k) = \begin{cases} 1, & k = 1, \\ 3, & k = 2, \\ 3, & k = 3, \\ 2, & k = 4, \\ 1, & k = 5. \end{cases}$$

STEP 3 Compute $G_x^*(v_x)$ for all $\mathbf{v}_x \in \mathcal{F}_x^*$ and $x = K^* - R = 2$. We get:

$$\mathcal{F}_2^* = \left\{ \mathbf{v}_2 \;\middle|\; v_{i2} \geq 0, \; \sum_{i=1}^{2} v_{i2} = K^* - 2 = 3 \right\} = \{(0,3),(1,2),(2,1),(3,0)\}.$$

With Eq. (8.70) we have:

$$\begin{aligned}
G_2^*(0,3) &= \frac{e_{13}}{\mu_{13}} \cdot G_1^*(1,3) + 4 \cdot \frac{e_{23}}{\mu_{23}} \cdot G_1^*(0,4) = \underline{0.727}, \\
G_2^*(1,2) &= 2 \cdot \frac{e_{13}}{\mu_{13}} \cdot G_1^*(2,2) + 3 \cdot \frac{e_{23}}{\mu_{23}} \cdot G_1^*(1,3) = \underline{0.774}, \\
G_2^*(2,1) &= 3 \cdot \frac{e_{13}}{\mu_{13}} \cdot G_1^*(3,1) + 2 \cdot \frac{e_{23}}{\mu_{23}} \cdot G_1^*(2,2) = \underline{0.681}, \\
G_2^*(3,0) &= 4 \cdot \frac{e_{13}}{\mu_{13}} \cdot G_1^*(4,0) + \frac{e_{23}}{\mu_{23}} \cdot G_1^*(3,1) = \underline{0.448}, \\
G_1^*(0,4) &= \frac{e_{11}}{\mu_{11}} \cdot G_0^*(1,4) + 5 \cdot \frac{e_{21}}{\mu_{21}} \cdot G_0^*(0,5) = \underline{0.905}, \\
G_1^*(1,3) &= 2 \cdot \frac{e_{11}}{\mu_{11}} \cdot G_0^*(2,3) + 4 \cdot \frac{e_{21}}{\mu_{21}} \cdot G_0^*(1,4) = \underline{0.73}, \\
G_1^*(2,2) &= 3 \cdot \frac{e_{11}}{\mu_{11}} \cdot G_0^*(3,2) + 3 \cdot \frac{e_{21}}{\mu_{21}} \cdot G_0^*(2,3) = \underline{0.555}, \\
G_1^*(3,1) &= 4 \cdot \frac{e_{11}}{\mu_{11}} \cdot G_0^*(4,1) + 2 \cdot \frac{e_{21}}{\mu_{21}} \cdot G_0^*(3,2) = \underline{0.38}, \\
G_1^*(4,0) &= 5 \cdot \frac{e_{11}}{\mu_{11}} \cdot G_0^*(5,0) + \frac{e_{21}}{\mu_{21}} \cdot G_0^*(5,0) = \underline{0.205}.
\end{aligned}$$

STEP 4 Iteration over all job classes s of the given network $\mathcal{N}$, starting with $s = 1$:

STEP 4.1 Computation of $G_5^*(\mathbf{0})$ using Eq. (8.70). For this computation we additionally need:

$$G_3^*(0,2) = \frac{e_{13}}{\mu_{13}} \cdot G_2^*(1,2) + 3 \cdot \frac{e_{23}}{\mu_{23}} \cdot G_2^*(0,3) = \underline{0.605},$$

$$G_3^*(1,1) = 2 \cdot \frac{e_{13}}{\mu_{13}} \cdot G_2^*(2,1) + 2 \cdot \frac{e_{23}}{\mu_{23}} \cdot G_2^*(1,2) = \underline{0.836},$$

$$G_3^*(2,0) = 3 \cdot \frac{e_{13}}{\mu_{13}} \cdot G_2^*(3,0) + \frac{e_{23}}{\mu_{23}} \cdot G_2^*(2,1) = \underline{0.740},$$

$$G_4^*(0,1) = \frac{e_{12}}{\mu_{12}} \cdot G_3^*(1,1) + 2 \cdot \frac{e_{22}}{\mu_{22}} \cdot G_3^*(0,2) = \underline{0.207},$$

$$G_4^*(1,0) = 2 \cdot \frac{e_{12}}{\mu_{12}} \cdot G_3^*(2,0) + \frac{e_{22}}{\mu_{22}} \cdot G_3^*(1,1) = \underline{0.161}.$$

From this it follows:

$$G_5^*(\mathbf{0}) = \frac{e_{11}}{\mu_{11}} \cdot G_4^*(1,0) + \frac{e_{21}}{\mu_{21}} \cdot G_4^*(0,1) = \underline{0.038}.$$

STEP 4.2 Computation of the class 1 performance measures. Because of Eqs. (8.71) and (8.72) we have:

$$\lambda_{15}^* = e_{11} \cdot \sum_{j=1}^{2} \frac{G_4^*(\mathbf{1}_j)}{G_5^*(\mathbf{0}) \cdot (2+5-1)} = \underline{1.611},$$

$$\lambda_{25}^* = e_{21} \cdot \sum_{j=1}^{2} \frac{G_4^*(\mathbf{1}_j)}{G_5^*(\mathbf{0}) \cdot (2+5-1)} = \underline{1.160},$$

$$\overline{K}_{15}^* = \frac{1}{G_5^*(\mathbf{0})} \cdot G_4^*(1,0) \cdot \frac{e_{11}}{\mu_{11}} = \underline{0.021},$$

$$\overline{K}_{25}^* = \frac{1}{G_5^*(\mathbf{0})} \cdot G_4^*(0,1) \cdot \frac{e_{21}}{\mu_{21}} = \underline{0.979}.$$

These results are needed to compute the throughputs and mean number of jobs in class 1 using Eqs. (8.73) and (8.74).

$$\lambda_{11} = K_1 \cdot \lambda_{15}^* = \underline{3.222}, \quad \overline{K}_{11} = K_1 \cdot \overline{K}_{15}^* = \underline{0.042}, \tag{8.77}$$

$$\lambda_{21} = K_1 \cdot \lambda_{25}^* = \underline{2.320}, \quad \overline{K}_{21} = K_1 \cdot \overline{K}_{25}^* = \underline{1.958}. \tag{8.78}$$

STEP 4.3 Relabel the jobs in $\mathcal{N}^*$ in accordance to Eq. (8.76):

$$c^{(2)}(k) = \begin{cases} 1, & k = 1, \\ 3, & k = 2, \\ 1, & k = 3, \\ 3, & k = 4, \\ 2, & k = 5. \end{cases}$$

STEP 4.4 Increase x by 1, i.e., $x = 3$ and compute and store $G_x^*(\mathbf{v}_x)$ for all $\mathbf{v}_x \in \mathcal{F}_3^* = \{(0,2),(1,1),(2,0)\}$:

$$G_3^*(0,2) = \frac{e_{11}}{\mu_{11}} \cdot G_2^*(1,2) + 3 \cdot \frac{e_{21}}{\mu_{21}} \cdot G_2^*(0,3) = \underline{0.396},$$

$$G_3^*(1,1) = 2 \cdot \frac{e_{11}}{\mu_{11}} \cdot G_2^*(2,1) + 2 \cdot \frac{e_{21}}{\mu_{21}} \cdot G_2^*(1,2) = \underline{0.285},$$

$$G_3^*(2,0) = 3 \cdot \frac{e_{11}}{\mu_{11}} \cdot G_2^*(3,0) + \frac{e_{21}}{\mu_{21}} \cdot G_2^*(2,1) = \underline{0.129}.$$

Iteration for $s = 2$:

STEP 4.1 Computation of $G_5^*(\mathbf{0})$ with Eq. (8.70). For this computation we additionally need:

$$G_4^*(0,1) = \frac{e_{13}}{\mu_{13}} \cdot G_3^*(1,1) + 2 \cdot \frac{e_{23}}{\mu_{23}} \cdot G_3^*(0,2) = \underline{0.222},$$

$$G_4^*(1,0) = 2 \cdot \frac{e_{13}}{\mu_{13}} \cdot G_3^*(2,0) + \frac{e_{23}}{\mu_{23}} \cdot G_3^*(1,1) = \underline{0.158}.$$

Hence:

$$G_5^*(\mathbf{0}) = \frac{e_{12}}{\mu_{12}} \cdot G_4^*(1,0) + \frac{e_{22}}{\mu_{22}} \cdot G_4^*(0,1) = \underline{0.038}.$$

STEP 4.2 Computation of the performance measures for class 2 using Eq. (8.71) and Eq. (8.72):

$$\lambda_{15}^* = e_{12} \cdot \sum_{j=1}^{2} \frac{G_4^*(\mathbf{1}_j)}{G_5^*(\mathbf{0}) \cdot 6} = \underline{1.661},$$

Furthermore:

$$\lambda_{25}^* = \underline{1.046}, \quad \overline{K}_{15}^* = \underline{0.083}, \quad \overline{K}_{25}^* = \underline{0.917}.$$

Because of $K_2 = 1$ with Eqs. (8.73), (8.74) we get:

$$\lambda_{12} = \lambda_{15}^*, \quad \lambda_{22} = \lambda_{25}^*, \quad \overline{K}_{12} = \overline{K}_{15}^*, \quad \overline{K}_{22} = \overline{K}_{25}^*.$$

STEP 4.3 Relabel the jobs:

$$c^{(3)}(k) = \begin{cases} 1, & k = 1, \\ 3, & k = 2, \\ 1, & k = 3, \\ 2, & k = 4, \\ 3, & k = 5. \end{cases}$$

STEP 4.4 Increase x by 1, i.e., $x = 4$, and compute and store $G_x^*(\mathbf{v}_x)$ for all $\mathbf{v}_x \in \mathcal{F}_4^* = \{(0,1),(1,0)\}$:

$$G_4^*(0,1) = \frac{e_{12}}{\mu_{12}} \cdot G_3^*(1,1) + 2 \cdot \frac{e_{22}}{\mu_{22}} \cdot G_3^*(0,2) = \underline{0.131},$$

$$G_4^*(1,0) = 2 \cdot \frac{e_{12}}{\mu_{12}} \cdot G_3^*(2,0) + \frac{e_{22}}{\mu_{22}} \cdot G_3^*(1,1) = \underline{0.050}.$$

Iteration for $s = 3$:

STEP 4.1 Computation of $G_5^*(\mathbf{0})$:

$$G_5^*(\mathbf{0}) = \frac{e_{13}}{\mu_{13}} \cdot G_4^*(1,0) + \frac{e_{23}}{\mu_{23}} \cdot G_4^*(0,1) = \underline{0.038}.$$

STEP 4.2 Computation of the class-3 performance measures:

$$\lambda_{15}^* = \underline{0.790}, \quad \lambda_{25}^* = \underline{0.316}, \quad \overline{K}_{15}^* = \underline{0.658}, \quad \overline{K}_{25}^* = \underline{0.343}.$$

Hence:

$$\lambda_{13} = \underline{1.580}, \quad \lambda_{23} = \underline{0.632}, \quad \overline{K}_{13} = \underline{1.315}, \quad \overline{K}_{15} = \underline{0.685}.$$

Now the iteration ends and all other performance measures can be obtained using formulae from Section 7.1.

If the number of job classes in the network is very large, then the RECAL algorithm is much more efficient than the convolution algorithm or the MVA. But if there are only a few job classes and many nodes in the network, then RECAL is not very well suited (see Figs. 8.10 and 8.11). If we assume that $K_r = \kappa$ for all r and N is fixed, then the time complexity and memory requirement grow polynomially with the number of job classes in the network. In this case, the memory requirement is:

$$O\left(\frac{\kappa^{N-1}+1}{(N-1)!} R^{N-1}\right),$$

and the time requirement in number of operations is:

$$O\left(\frac{4N-1}{(N+1)!} R^{N+1}\right).$$

The algorithm can be extended further to analyze Type-1 nodes with multiple servers and to analyze nodes with load-dependent service rates. Based on the RECAL algorithm, [CSL89] developed the MVAC algorithm (**m**ean **v**alue **a**nalysis by **c**hain), which directly computes the mean values of performance measures. The RECAL method was extended to the *tree RECAL technique* by [McKe88].

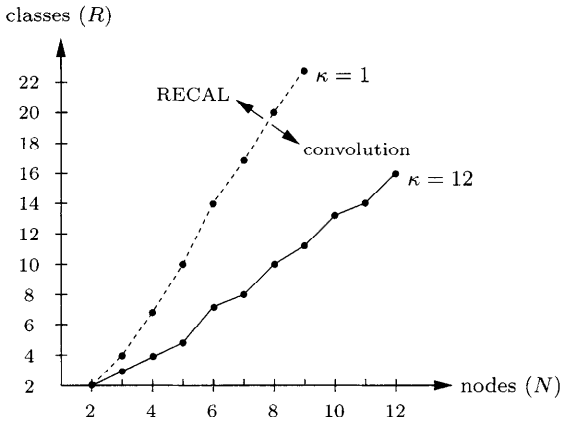


Fig. 8.10 Regions in the $(N \times R)$ space in which the *storage requirement* of RECAL is less than that of convolution, for $\kappa = 1$ and 12 (on or above the curves the storage requirement of RECAL is less than that of convolution).

8.4 FLOW EQUIVALENT SERVER METHOD

The last method for analyzing product-form queueing networks that we consider is called *flow equivalent server (FES) method.* The method is based on *Norton's theorem* from electric circuit theory [CHW75b]. We encounter this method again in Chapter 9 when we deal with approximate solution of non-product-form networks. Here we consider this technique in the context of exact analysis of product-form networks. If we select one or more nodes in the given network and combine all the other nodes into an FES, then Norton's theorem says that the reduced system has the same behavior as the original network.

8.4.1 FES Method for a Single Node

To describe the FES method, we consider the central-server model given in Fig. 8.12 where we suppose that a product-form solution exists. For this network we can construct an equivalent network by choosing one node (e.g., node 1) and combining all other nodes to one *FES node c.* As a result we get a reduced net consisting of two nodes only. This network (shown in Fig. 8.13) is much easier to analyze then the original one, and all computed performance measures are the same as in the original network.

To determine the service rates $\mu_c(k), k = 1, \ldots, K$ of the FES node, the chosen node 1 in the original network is short circuited, i.e., the mean service time of this node is set to zero, as shown in Fig. 8.14. The throughput in the short circuit path with $k = 1, \ldots, K$ jobs in the network is then equated to

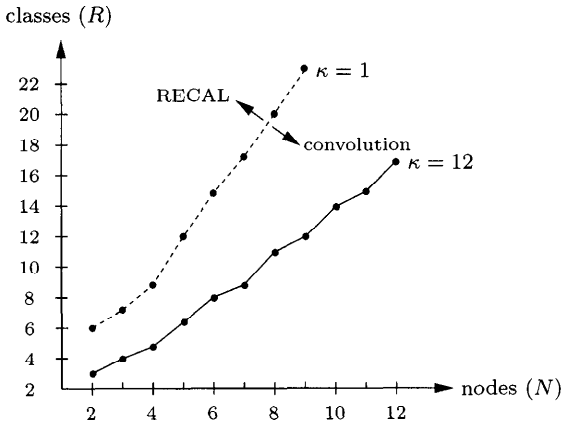


Fig. 8.11 Regions in the $(N \times R)$ space in which the *time requirement* of RECAL is less than that of convolution, for $\kappa = 1$ and 12 (on or above the curves the time requirement of RECAL is less than that of convolution).

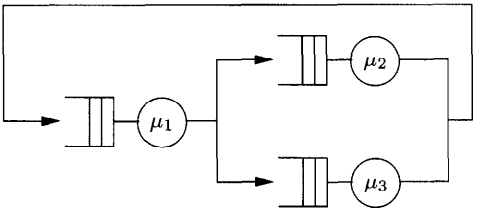


Fig. 8.12 The central-server network.

the load-dependent service rate $\mu_c(k), k = 1, \ldots, K$ of the FES node in the reduced queueing network.

The FES algorithm can now be summarized in the following three steps:

STEP 1 In the given network, choose a node i and short-circuit it by setting the mean service time in that node to zero. Compute the throughputs $\lambda_i^{sc}(k)$ along the short circuit, as a function of the number of jobs $k = 1, \ldots, K$ in the network. For this computation, any of the earlier solution algorithms for product-form queueing networks can be used.

STEP 2 From the given network, construct an equivalent reduced network consisting only of the chosen node i and the FES node c. The visit ratios in both nodes are e_i. The load-dependent service rate of the FES node is the throughput along the short-circuit path when there are k jobs in the network, that is: $\mu_c(k) = \lambda_i^{sc}(k)$ for $k = 1, \ldots, K$.

STEP 3 Compute the performance measures in the reduced network with any suitable algorithm for product-form networks (e.g., convolution or MVA).

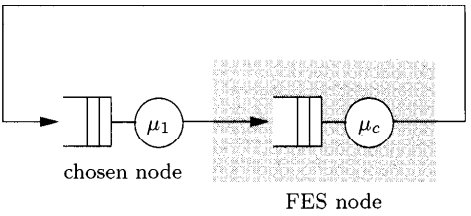


Fig. 8.13 Reduced queueing network.

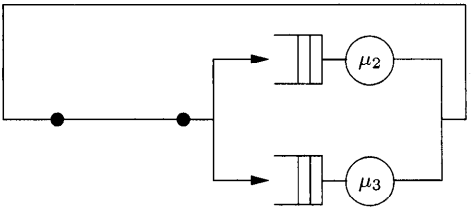


Fig. 8.14 The short-circuited model.

The technique just described is very well suited for examining the influence of changes in the parameters of a single node while the rest of the system parameters are kept fixed. To clarify the usage of the FES method, we give the following simple example:

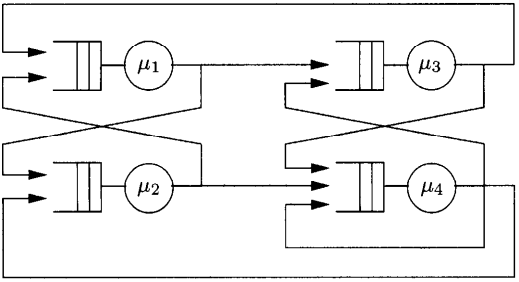


Fig. 8.15 Yet another closed queueing network.

Example 8.10 Consider the closed single class product-form network shown in Fig. 8.15. It consists of $N = 4$ nodes and $K = 2$ jobs. The service times are exponentially distributed with parameters:

$$\mu_1 = 1, \quad \mu_2 = 2, \quad \mu_3 = 3, \quad \mu_4 = 4.$$

The routing probabilities are given as:

$$p_{12} = 0.5, \quad p_{21} = 0.5, \quad p_{31} = 0.5, \quad p_{42} = 0.4,$$
$$p_{13} = 0.5, \quad p_{24} = 0.5, \quad p_{34} = 0.5, \quad p_{43} = 0.4, \quad p_{44} = 0.2.$$

With Eq. (7.5) we compute the visit ratios:

$$e_1 = 1, \quad e_2 = 1, \quad e_3 = 1, \quad e_4 = 1.25.$$

Now, the analysis of the network is carried out in the following three steps:

STEP 1 Choose a node, e.g., node 4 of the network shown in Fig. 8.16, and short circuit it. The throughput of the short circuit is computed for $k = 1, 2$

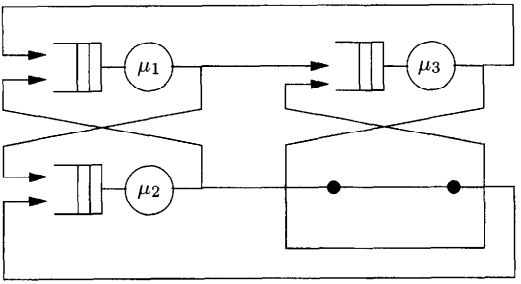


Fig. 8.16 Network after node 4 is short circuit.

by using the MVA, for example. Then we get the following results:

$$\lambda(1) = \underline{0.545}, \qquad \lambda(2) = \underline{0.776}.$$

and therefore:

$$\lambda_4^{sc}(1) = \underline{0.682}, \quad \lambda_4^{sc}(2) = \underline{0.971}.$$

STEP 2 Construct the reduced network that consists of the selected node 4 and the FES node c as shown in Fig. 8.17. For the service rates of the FES

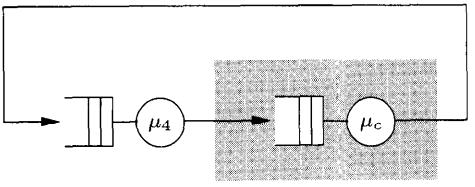


Fig. 8.17 Reduced network.

node we have:

$$\mu_c(1) = \lambda_4^{sc}(1) = 0.682, \quad \mu_c(2) = \lambda_4^{sc}(2) = 0.971.$$

The visit ratios are:

$$e_4 = e_c = \underline{1.25}.$$

STEP 3 Analyze the reduced network shown in Fig. 8.17 using, for example, the MVA for load-dependent nodes. The load-dependent service rates of node 4 can be computed by Eq. (7.49) as $\mu_4(1) = \mu_4(2) = 4$.

MVA-Iteration for $k = 1$:

$$\overline{T}_4(1) = \frac{1}{\mu_4(1)}\pi_4(0 \mid 0) = \underline{0.25}, \quad \overline{T}_c(1) = \frac{1}{\mu_c(1)}\pi_c(0 \mid 0) = \underline{1.467},$$

$$\lambda(1) = \underline{0.466}, \quad G(1) = \frac{1}{\lambda(1)} = \underline{2.146},$$

$$\pi_4(1 \mid 1) = \underline{0.146}, \quad \pi_c(1|1) = \underline{0.854}, \quad \pi_4(0 \mid 1) = \underline{0.854}, \quad \pi_c(0|1) = \underline{0.146}.$$

MVA-Iteration for $k = 2$:

$$\overline{T}_4(2) = \frac{1}{\mu_4(1)}\pi_4(0 \mid 1) + \frac{2}{\mu_4(2)}\pi_4(1 \mid 1) = \underline{0.286},$$

$$\overline{T}_c(2) = \frac{1}{\mu_c(1)}\pi_c(0|1) + \frac{2}{\mu_c(2)}\pi_c(1|1) = \underline{1.974},$$

$$\lambda(2) = \underline{0.708}, \quad G(2) = \frac{G(1)}{\lambda(2)} = \underline{3.032},$$

$$\pi_4(1|2) = \underline{0.189}, \quad \pi_4(2|2) = \underline{0.032}, \quad \pi_4(0|2) = \underline{0.779},$$

$$\pi_c(1|2) = \underline{0.189}, \quad \pi_c(2|2) = \underline{0.779}, \quad \pi_c(0|2) = \underline{0.032}.$$

Now all other performance measures of the network can be computed:

- Mean number of jobs:
  For node 4 we get from Eq. (7.26):

$$\overline{K}_4 = \sum_{k=1}^{2} k \cdot \pi_4(k \mid 2) = \underline{0.253}.$$

  For the other nodes we have by Eq. (8.16):

$$\overline{K}_1 = \sum_{k=1}^{2} \left(\frac{e_1}{\mu_1}\right)^k \cdot \frac{G(2-k)}{G(2)} = \underline{1.038}, \quad \overline{K}_2 = \underline{0.436}, \quad \overline{K}_3 = \underline{0.273}.$$

- Throughputs, Eq. (8.14):

$$\lambda_1 = e_1 \frac{G(1)}{G(2)} = \underline{0.708}, \quad \lambda_2 = \underline{0.708}, \quad \lambda_3 = \underline{0.708}, \quad \lambda_4 = \underline{0.885}.$$

- Mean response times, Eq. (7.43):

$$\overline{T}_1 = \frac{\overline{K}_1}{\lambda_1} = \underline{1.466}, \quad \overline{T}_2 = \underline{0.617}, \quad \overline{T}_3 = \underline{0.385}.$$

8.4.2 FES Method for Multiple Nodes

The usage of the FES method, when only one node is short-circuited, has nearly no advantage in reducing the computation time. Therefore [ABP85] suggest an extension of the concept of [CHW75b]. For this extension, the closed product-form network is partitioned in a number of subnetworks. Each of these subnetworks is analyzed independently from the others, i.e., the whole network is analyzed by short-circuiting the nodes that do not belong to the subnetwork that is to be examined. The computed throughputs of the short-circuited network form the load-dependent service rates of FES node j, which represents the jth subnetwork in the reduced network. When analyzing this reduced network, we get the normalizing constant of the whole network. The normalizing constant can then be used to compute the performance measures of interest. The FES method for this case can be described in the following five steps.

STEP 1 Partition the original network into M disjoint subnetworks $SN-j$ $(1 \leq j \leq M)$. It is even allowed to combine nodes in the subnetwork that do not have a direct connection to this subnetwork.

STEP 2 Analyze each of these subnetworks $j = 1, \ldots, M$ by short-circuiting all nodes that do not belong to the considered subnetwork. The visit ratios of the nodes in the subnetworks are taken from the original network. For analyzing the subnetworks, any product-form algorithm can be used. Determine the throughputs $\lambda_{SN-j}(k)$ and the normalizing constants $G_j(k)$, for $k = 1, \ldots, K$.

STEP 3 Combine the nodes of each subnetwork into one FES node and construct an equivalent reduced network out of the original network by putting the FES nodes (with visit ratio 1) together in a tandem connection. The load-dependent service rates of the FES node j are identical with the throughputs in the corresponding jth subnetwork:

$$\mu_{cj}(k) = \lambda_{SN-j}(k), \quad \text{for } j = 1, \ldots, M \text{ and } k = 1, \ldots, K.$$

STEP 4 The normalizing constant of the reduced network can be computed by convoluting the M normalizing vectors

$$G_j = \begin{pmatrix} G_j(0) \\ \vdots \\ G_j(K) \end{pmatrix}$$

of the subnetworks [ABP85]:

$$G = G_1 \otimes G_2 \otimes \ldots \otimes G_M, \tag{8.79}$$

where the convolution operator $\otimes$ is defined as in Eq. (8.1). To determine the normalizing constant, the MVA for networks with load-dependent nodes (see Section 8.2.4) can be used instead.

STEP 5 Compute the performance measures of the original network and the performance measures of the subnetworks with the help of the normalizing constants and the formulae given in Section 8.1.

The FES method is very well suited if the system behavior for different input parameters needs to be examined, because in this case only one new normalizing constant for the corresponding subnetwork needs to be recomputed. The normalizing constants of the other subnetworks remain the same.

Example 8.11 The queueing network from Example 8.10 is now analyzed again.

STEP 1 Partition the whole network into $M = 2$ subnetworks where Subnetwork 1 contains nodes 1 and 2 and Subnetwork 2, nodes 3 and 4 of the original network.

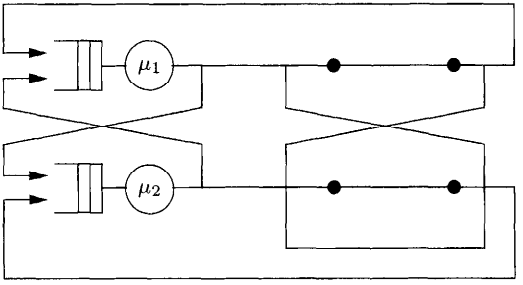


Fig. 8.18 Subnetwork 1 with subnetwork 2 short-circuited.

STEP 2 Analyze the two subnetworks: To analyze Subnetwork 1, nodes 3 and 4 are short-circuited (see Fig. 8.18). We use the MVA to compute for $k = 1, 2$ the load-dependent throughputs and normalizing constants of this subnetwork:

$$\lambda_{SN-1}(1) = \underline{0.667}, \quad \lambda_{SN-1}(2) = \underline{0.857},$$
$$G_1(1) = \underline{1.5}, \qquad G_1(2) = \underline{1.75}.$$

To analyze Subnetwork 2, nodes 1 and 2 are short-circuited (see Fig. 8.19).

Analogously, we get the following results for Subnetwork 2:

$$\lambda_{SN-2}(1) = \underline{1.548}, \quad \lambda_{SN-2}(2) = \underline{2.064},$$
$$G_2(1) = \underline{0.646}, \qquad G_2(2) = \underline{0.313}.$$

STEP 3 Construct the reduced network shown in Fig. 8.20.

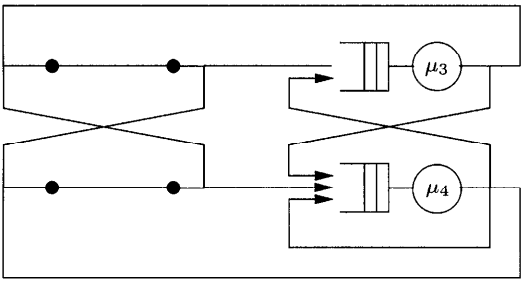


Fig. 8.19 Short-circuit of Subnetwork 1.

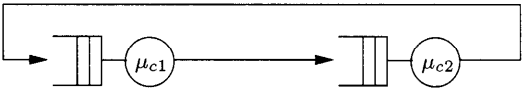


Fig. 8.20 Reduced network.

The first FES node describes the nodes of Subnetwork 1 with service rates $\mu_{c1}(k) = \lambda_{SN-1}(k)$ for $k = 1, 2$, and the second FES node describes the nodes of Subnetwork 2 with service rates $\mu_{c2}(k) = \lambda_{SN-2}(k)$.

STEP 4 The normalization constants of the reduced network are computed using Eq. (8.79):

$$G = G_1 \otimes G_2 = \begin{pmatrix} 1 \\ 1.5 \\ 1.75 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0.646 \\ 0.313 \end{pmatrix} = \begin{pmatrix} 1 \\ 2.146 \\ 3.032 \end{pmatrix}.$$

By using Eq. (8.1) we get:

$$G(1) = G_1(0) \cdot G_2(1) + G_1(1) \cdot G_2(0) = \underline{2.146},$$
$$G(2) = G_1(0) \cdot G_2(2) + G_1(1) \cdot G_2(1) + G_1(2) \cdot G_2(0) = \underline{3.032}.$$

STEP 5 Compute the performance measures:

Mean number of jobs at each node is computed using, Eq. (8.16):

$$\overline{K}_1 = \sum_{k=1}^{2} \left(\frac{e_1}{\mu_1}\right)^k \cdot \frac{G(2-k)}{G(2)} = \underline{1.038}, \quad \overline{K}_2 = \underline{0.436}, \quad \overline{K}_3 = \underline{0.273},$$
$$\overline{K}_4 = \underline{0.253}.$$

As we can see, the same results are obtained as in Example 8.10.

The FES method can also be applied to the case of multiple class queueing networks. The method is very flexible in the need for resources and lies between the two extremes: MVA and convolution. This flexibility is based on how the subnetworks are chosen.

Table 8.5 Storage requirement in number of storage elements and time requirement in number of operations [CoGe86]

Method	Storage Requirement	Time Requirement
Convolution	$O(\prod_{r=1}^{R}(K_r+1))$	$O(2 \cdot R(N-1) \cdot \prod_{r=1}^{R}(K_r+1))$
MVA	$O(N \cdot \prod_{r=1}^{R}(K_r+1))$	$\approx O(2 \cdot R(N-1) \cdot \prod_{r=1}^{R}(K_r+1))$
RECAL*	$O(\frac{\kappa^{N+1}+1}{(N-1)!})$	$O(\frac{4N-1}{(N+1)!}R(N+1))$
FES	$> O(3 \cdot \prod_{r=1}^{R}(K_r+1))$	$\approx O(2 \cdot R(N-1) \cdot \prod_{r=1}^{R}(K_r+1))$

* $K_r = \kappa$ for $r = 1, \ldots, R$

8.5 SUMMARY

As shown in Table 8.5, the time requirement for convolution, MVA and FES is approximately the same and differs for RECAL in dependency of the number of nodes and number of classes (see also Fig. 8.11). Also, the storage requirement depends greatly on the number of nodes and classes for RECAL compared to the other methods (see Fig. 8.10). For larger numbers of nodes, MVA is much worse than FES and convolution. For FES, the storage requirement depends also on the number of subnetworks. The larger the number of subnetworks, the smaller is the storage requirement. In Table 8.6 the main advantages and disadvantages of the convolution algorithm, the MVA, the RECAL method, and the FES method are summarized.

Problem 8.1 For the closed queueing network of Problems 7.3 and 7.4, compute the performance measures using the convolution method.

Problem 8.2 For a closed queueing network with $N = 2$ nodes and $R = 2$ job classes, compute the performance measures for each class and node using the convolution method. Node 1 is of Type-3 and node 2 is of Type-2. Each class contains two jobs $K_1 = K_2 = 2$ and class switching is not allowed. The service rates are given as follows:

$$\mu_{11} = 0.4\,\text{sec}^{-1}, \quad \mu_{21} = 0.3\,\text{sec}^{-1}, \quad \mu_{12} = 0.2\,\text{sec}^{-1}, \quad \mu_{22} = 0.4\,\text{sec}^{-1},$$

The routing probabilities are:

$$p_{11,11} = p_{11,21} = 0.5, \quad p_{21,11} = p_{22,12} = p_{12,22} = 1.$$

Problem 8.3 Compute performance measures of the network in Problems 7.3 and 7.4 again, with the MVA.

Table 8.6 Comparison of the four solution methods for product-form networks

Method	Advantages	Disadvantages
MVA	Mean values can be computed directly without computing the normalizing constant. But if required, the normalization constant and hence state probabilities can also be computed.	High storage requirement (for multiple classes and multiple servers) Overflow and underflow problems when computing the marginal probabilities for -/M/m nodes
Convolution	Less storage requirement than MVA	Normalizing constant (NC) has to be computed Overflow or underflow problems when computing the NC
RECAL	Less storage and time requirement for a large number of job classes and nodes than MVA and convolution	More storage and time requirement for small number of job classes and nodes than MVA and convolution
FES	The time and storage requirement is reduced considerably when examining the influence of changing the parameters of an individual or a few nodes while the rest of the system parameters remain unchanged Basis for solution techniques for non-product-form networks	Multiple application of convolution or MVA Network needs to be transformed

Problem 8.4 For a mixed queueing network with $N = 2$ Type-2 nodes and $R = 4$ classes (see Fig. 8.21), compute all performance measures per node and class. The system parameters are given as follows:

$$\lambda_1 = 0.2\,\text{sec}^{-1}, \quad \lambda_2 = 0.1\,\text{sec}^{-1}, \quad K_2 = K_3 = 2,$$
$$\mu_{11} = 1.5\,\text{sec}^{-1}, \quad \mu_{12} = 1\,\text{sec}^{-1}, \quad \mu_{13} = 3\,\text{sec}^{-1}, \quad \mu_{14} = 2\,\text{sec}^{-1},$$
$$\mu_{21} = 2\,\text{sec}^{-1}, \quad \mu_{22} = 0.5\,\text{sec}^{-1}, \quad \mu_{23} = 2\,\text{sec}^{-1}, \quad \mu_{24} = 1\,\text{sec}^{-1},$$
$$p_{21,11} = p_{22,12} = p_{21,0} = p_{22,0} = 0.5.$$
$$p_{0,11} = p_{0,12} = p_{11,21} = p_{12,22} = p_{13,23} = p_{14,24} = p_{23,13} = p_{24,14} = 1.$$

Problem 8.5 Consider a closed queueing network, consisting of a Type-3 node and a Type-1 node ($m_2 = 2$). This model represents a simple two-processor system with terminals. The system parameters are given as:

$$K = 3, \quad p_{12} = p_{21} = 1, \quad \frac{1}{\mu_{1.}} = 2\,\text{sec}, \quad \frac{1}{\mu_2} = 1\,\text{sec}.$$

(a) Apply the mean value analysis to determine the performance measures of each node, especially throughput and mean response time of the terminals.

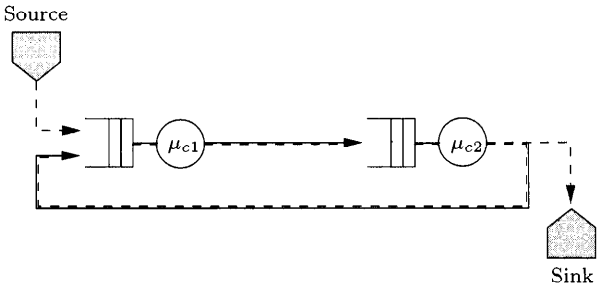


Fig. 8.21 Mixed queueing network.

(b) Determine for both nodes the load-dependent service rates and analyze the network again, using the load-dependent MVA.

Problem 8.6 Compute the performance measures for the network given in Problem 8.2 using the RECAL algorithm.

Problem 8.7 Consider a simple central-server model with $N = 3$ Type-1 nodes ($m_i = 1$) and $K = 3$ jobs. The service rates are given as follows:

$$\mu_1 = 1\,\text{sec}^{-1}, \quad \mu_2 = 0.65\,\text{sec}^{-1}, \mu_3 = 0.75\,\text{sec}^{-1},$$

and the routing probabilities are:

$$p_{12} = 0.6, \quad p_{21} = p_{31} = 1, \quad p_{13} = 0.4.$$

Combine nodes 2 and 3 into a FES node. Use the FES method and compare the results with the results obtained without the use of FES method.

Problem 8.8 Consider a closed queueing network with $N = 4$ Type-1 nodes and $K = 3$ jobs. The service rates are:

$$\mu_1 = 4\,\text{sec}^{-1}, \quad \mu_2 = 3\,\text{sec}^{-1}, \quad \mu_3 = 2\,\text{sec}^{-1}, \quad \mu_4 = 1\,\text{sec}^{-1},$$

and the routing probabilities are:

$$p_{23} = p_{24} = 0.5, \quad p_{12} = p_{31} = p_{41} = 1.$$

For computing the performance measures, use the extended FES method by combining nodes 1 and 2 in subsystem 1, and nodes 3 and 4 into subsystem 2.

9

Approximation Algorithms for Product-Form Networks

In Chapter 8, several efficient algorithms for the exact solution of queueing networks are introduced. However, the memory requirements and computation time of these algorithms grows exponentially with the number of job classes in the system. For computationally difficult problems of networks with a large number of job classes, we resort to approximation methods. In Sections 9.1, 9.2, and 9.3 we introduce methods for obtaining such approximate results.

The first group of methods is based on the MVA. The approximate methods that we present need much less memory and computation time than the exact MVA and yet give very accurate results.

The second approach is based on the idea that the mean number of jobs at a network node can be computed approximately if the utilization ρ_i of this node is known. The sum of all the equations for the single nodes leads to the so-called *system equation*. By solving this equation we get approximate performance measures for the whole queueing network.

In some cases it is enough to have upper and lower bounds to make some statements about the network. Therefore, the third approach (Section 9.4) deals with methods on how to get upper and lower bounds on performance measures of a queueing network. In Section 9.5 we introduce a method that allows intervals as input parameters.

For other techniques that have been developed to analyze very large networks see [ChYu83, SLM86, HsLa89].

9.1 APPROXIMATIONS BASED ON THE MVA

The fundamental equation of the MVA (8.31) describes the relation between the mean response time of a node when there are k jobs at the node and the mean number of jobs in that node with one job less in the network. Therefore, to solve a queueing network it is necessary to iterate over all jobs in the network, starting from 0 to the maximum number K. For this reason, an implementation of the MVA requires a lot of computation time and main memory. For multiclass queueing networks with a large number of jobs in the system, we will very quickly reach the point at which the system runs out of memory, especially if we have multiserver nodes.

An alternative approach is to approximate the fundamental equation of the MVA in a way that the mean response time depends only on K and not on $K-1$. Then it is not necessary to iterate over the whole population of jobs in the system but we have to improve only the performance measures iteratively, starting with an initial vector. This approach leads to a substantial reduction in computation time and memory requirement. In the following, two techniques based on this idea, are introduced.

9.1.1 Bard-Schweitzer Approximation

Bard and Schweitzer [Bard79, Schw79] suggested an approximation of the MVA for single server queueing networks that is based on the following idea: Starting with an initial value $\overline{K}_{ir}(\mathbf{K})$ for the mean number of class-r jobs at node i for a given population vector $\mathbf{K}$, make an estimate of the mean number of jobs for a population vector $(\mathbf{K}-\mathbf{1}_s)$. This estimating needs to be done for all classes s. These $\overline{K}_{ir}(\mathbf{K}-\mathbf{1}_s)$ estimates are then used in the MVA to compute the mean response times, and only one iteration step is now needed to evaluate the performance measures for a given population vector $\mathbf{K}$. In particular, new values for the $\overline{K}_{ir}(\mathbf{K})$ are computed. The values we get are used to estimate the $\overline{K}_{ir}(\mathbf{K}-\mathbf{1}_s)$ again. This iteration stops if the values of the $\overline{K}_{ir}(\mathbf{K})$ in two iteration steps differ by less than the chosen error criterion ϵ.

Obviously, the problem with this MVA approximation is how to estimate the mean value $\overline{K}_{ir}(\mathbf{K}-\mathbf{1}_s)$, given $\overline{K}_{ir}(\mathbf{K})$. An approximate formula for the case of a very large network population was suggested by [Schw79] since in this case a reasonable explanation can be given as to why the mean number of jobs of class r at node i remains nearly the same if the number of jobs in the network is reduced by one. For the mean number of class-r jobs at node i given a population vector $(\mathbf{K}-\mathbf{1}_s)$, they suggest the approximation:

$$\overline{K}_{ir}(\mathbf{K}-\mathbf{1}_s)=\frac{(\mathbf{K}-\mathbf{1}_s)_r}{K_r}\overline{K}_{ir}(\mathbf{K}), \tag{9.1}$$

where:

$$(\mathbf{K}-\mathbf{1}_s)_r = \begin{cases} K_r, & r \neq s, \\ K_r - 1, & r = s, \end{cases} \tag{9.2}$$

gives the number of class-r jobs when the network population $\mathbf{K}$ is reduced by one class-s job. If we assume that the jobs are initially equally distributed over the whole network, then the Bard-Schweitzer approximation can be described in the following steps:

STEP 1 Initialization. For $i = 1, \ldots, N$ and $s = 1, \ldots, R$:

$$\overline{K}_{is}(\mathbf{K}) = \frac{K_s}{N}.$$

STEP 2 For all $i = 1, \ldots, N$ and all $r, s = 1, \ldots, R$, compute the estimated values for the mean number of jobs for the population vector $(\mathbf{K} - \mathbf{1}_r)$:

$$\overline{K}_{is}(\mathbf{K}-\mathbf{1}_r) = \frac{(\mathbf{K}-\mathbf{1}_r)_s}{K_s} \cdot \overline{K}_{is}(\mathbf{K}).$$

STEP 3 Analyze the queueing network for a population vector $\mathbf{K}$ by using one iteration step of the MVA.

STEP 3.1 For $i = 1, \ldots, N$ and $r = 1, \ldots, R$, compute the mean response times:

$$\overline{T}_{ir}(\mathbf{K}) = \begin{cases} \dfrac{1}{\mu_{ir}} \left[1 + \displaystyle\sum_{s=1}^{R} \overline{K}_{is}(\mathbf{K}-\mathbf{1}_r) \right], & \text{Type-1,2,4 } (m_i = 1), \\ \dfrac{1}{\mu_{ir}}, & \text{Type-3.} \end{cases}$$

STEP 3.2 For $r = 1, \ldots, R$, compute the throughputs:

$$\lambda_r(\mathbf{K}) = \frac{K_r}{\displaystyle\sum_{i=1}^{N} \overline{T}_{ir}(\mathbf{K}) e_{ir}}.$$

STEP 3.3 For $i = 1, \ldots, N$ and $r = 1, \ldots, R$, compute the mean number of jobs:

$$\overline{K}_{ir}(\mathbf{K}) = \overline{T}_{ir}(\mathbf{K}) \lambda_r(\mathbf{K}) e_{ir}.$$

STEP 4 Check the stopping condition:
If there are no significant changes in the $\overline{K}_{ir}(\mathbf{K})$ values between the nth and $(n-1)$th iteration step, that is, when:

$$\max_{i,r} \left| \overline{K}_{ir}^{(n)}(\mathbf{K}) - \overline{K}_{ir}^{(n-1)}(\mathbf{K}) \right| < \varepsilon,$$

for a suitable ε (here the superscript $^{(n)}$ denotes the values for the nth iteration step), then the iteration stops and all other performance values are computed. If the stopping criterion is not fulfilled, then return to Step 2.

This approximation is very easy to program and faster than the exact MVA. The memory requirements are proportional to the product of N and R. Therefore, this approximation needs considerably less memory than the exact MVA or the convolution method, especially for networks with a large number of job classes. A disadvantage of this approximation is that networks with multiple server nodes cannot be solved using it. We give a simple example of the use of the algorithm:

Example 9.1 We revisit the single class network of Example 8.2 (Fig. 8.4) with network parameters as shown in Table 9.1. For ε we choose the value 0.06. The analysis of the network ($K = 6$) is carried out in the following

Table 9.1 Input parameters for Example 9.1.

i	e_i	$1/\mu_i$	m_i
1	1	0.02	1
2	0.4	0.2	1
3	0.2	0.4	1
4	0.1	0.6	1

steps:

STEP 1 Initialization:

$$\overline{K}_1(K) = \overline{K}_2(K) = \overline{K}_3(K) = \overline{K}_4(K) = \frac{K}{N} = \underline{1.5}.$$

1. Iteration:

STEP 2 Estimated values, Eq. (9.1):

$$\overline{K}_1(K-1) = \frac{K-1}{K} \cdot \overline{K}_1(K) = \underline{1.25},$$
$$\overline{K}_2(K-1) = \overline{K}_3(K-1) = \overline{K}_4(K-1) = \underline{1.25}.$$

STEP 3 One MVA iteration:

STEP 3.1 Mean response times:

$$\overline{T}_1(K) = \frac{1}{\mu_1}\left[1 + \overline{K}_1(K-1)\right] = \underline{0.045}, \quad \overline{T}2(K) = \frac{1}{\mu_2}\left[1 + \overline{K}_2(K-1)\right] = \underline{0.45},$$
$$\overline{T}_3(K) = \frac{1}{\mu_3}\left[1 + \overline{K}_3(K-1)\right] = \underline{0.9}, \quad \overline{T}_4(K) = \frac{1}{\mu_4}\left[1 + \overline{K}_4(K-1)\right] = \underline{1.35}.$$

STEP 3.2 Throughput:

$$\lambda(K) = \frac{K}{\sum_{i=1}^{4} e_i \overline{T}_i(K)} = \underline{11.111}.$$

STEP 3.3 Mean number of jobs:

$$\overline{K}_1(K) = \overline{T}_1(K)\lambda(K)e_1 = \underline{0.5}, \quad \overline{K}_2(K) = \overline{T}_2(K)\lambda(K)e_2 = \underline{2},$$
$$\overline{K}_3(K) = \overline{T}_3(K)\lambda(K)e_3 - \underline{2}, \quad \overline{K}_4(K) = \overline{T}_4(K)\lambda(K)e_4 = \underline{1.5}.$$

STEP 4 Check the stopping condition:

$$\max_i \left| \overline{K}_i^{(1)}(K) - \overline{K}_i^{(0)}(K) \right| = \underline{1} \quad > 0.06.$$

2. Iteration:

STEP 2 Estimated values:

$$\overline{K}_1(K-1) = \frac{K-1}{K} \cdot 0.5 = \underline{0.417},$$
$$\overline{K}_2(K-1) - \underline{1.667}, \quad \overline{K}_3(K-1) = \underline{1.667}, \quad \overline{K}_4(K-1) = \underline{1.25}.$$

STEP 3 One MVA Iteration:

$$\overline{T}_1(K) = \frac{1}{\mu_1}\left[1 + \overline{K}_1(K-1)\right] = \underline{0.028}$$
$$\overline{T}_2(K) = \underline{0.533}, \quad \overline{T}_3(K) = \underline{1.067}, \quad \overline{T}_4(K) = \underline{1.35},$$
$$\lambda(K) = \frac{K}{\sum_{i=1}^{4} e_i \overline{T}_i(K)} = \underline{10.169},$$
$$\overline{K}_1(K) = \overline{T}_1(K)\lambda(K)e_1 = \underline{0.288},$$
$$\overline{K}_2(K) = \underline{2.169}, \quad \overline{K}_3(K) = \underline{2.169}, \quad \overline{K}_4(K) = \underline{1.373}.$$

STEP 4 Stopping condition:

$$\max_i \left| \overline{K}_i^{(2)}(K) - \overline{K}_i^{(1)}(K) \right| = \underline{0.212} \quad > 0.06.$$

3. Iteration:

STEP 2 Estimated values:

$$\overline{K}_1(K-1) = \frac{K-1}{K} \cdot 0.288 = \underline{0.240},$$
$$\overline{K}_2(K-1) = \underline{1.808}, \quad \overline{K}_3(K-1) = \underline{1.808}, \quad \overline{K}_4(K-1) = \underline{1.144}.$$

STEP 3 MVA Iteration:

$$\overline{T}_1(K) = \underline{0.025}, \quad \overline{T}_2(K) = \underline{0.562}, \quad \overline{T}_3(K) = \underline{1.123}, \quad \overline{T}_4(K) = \underline{1.286},$$
$$\lambda(K) = \underline{9.955},$$
$$\overline{K}_1(K) = \underline{0.247}, \quad \overline{K}_2(K) = \underline{2.236}, \quad \overline{K}_3(K) = \underline{2.236}, \quad \overline{K}_4(K) = \underline{1.281}.$$

STEP 4 Stopping condition:

$$\max_i \left| \overline{K}_i^{(3)}(K) - \overline{K}_i^{(2)}(K) \right| = \underline{0.092} \quad > 0.06.$$

4. Iteration:

STEP 2

$$\overline{K}_1(K-1) = \underline{0.206}, \quad \overline{K}_2(K-1) = \underline{1.864},$$
$$\overline{K}_3(K-1) = \underline{1.864}, \quad \overline{K}_4(K-1) = \underline{1.067}.$$

STEP 3

$$\overline{T}_1(K) = \underline{0.024}, \quad \overline{T}_2(K) = \underline{0.573}, \quad \overline{T}_3(K) = \underline{1.145}, \quad \overline{T}_4(K) = \underline{1.240},$$
$$\lambda(K) = \underline{9.896},$$
$$\overline{K}_1(K) = \underline{0.239}, \quad \overline{K}_2(K) = \underline{2.267}, \quad \overline{K}_3(K) = \underline{2.267}, \quad \overline{K}_4(K) = \underline{1.227}.$$

STEP 4

$$\max_i \left| \overline{K}_i^{(4)}(K) - \overline{K}_i^{(3)}(K) \right| = \underline{0.053} \quad < 0.06.$$

Now, the iteration stops and the other performance measures are computed. For the throughputs of the nodes we use Eq. (8.40):

$$\lambda_1 = \lambda(K) \cdot e_1 = \underline{9.896}, \quad \lambda_2 = \underline{3.958}, \quad \lambda_3 = \underline{1.979}, \quad \lambda_4 = \underline{0.986}.$$

For the utilization of the nodes we get:

$$\rho_1 = \frac{\lambda_1}{\mu_1} = \underline{0.198}, \quad \rho_2 = \underline{0.729}, \quad \rho_3 = \underline{0.729}, \quad \rho_4 = \underline{0.594}.$$

The final results are summarized in Table 9.2.

A comparison with the values from Example 8.2 shows that the Bard-Schweitzer approximation yields results that in this case are very close to the exact ones. The storage requirement of the Bard-Schweitzer approximation depends only on the number of nodes N and the number of classes R ($= O(N \times R)$) and is independent of the number of jobs. This is much smaller than those for convolution and MVA.[1] The accuracy of the Bard-

[1] Recall that the storage requirements for convolution and MVA are $O\left(\prod_{r=1}^{R}(K_r+1)\right)$ and $O\left(N \cdot \prod_{r=1}^{R}(K_r+1)\right)$, respectively.

Table 9.2 Bard-Schweitzer approximation for Example 9.1

Node:	1	2	3	4
Mean response time $\overline{T}_i$	0.024	0.573	1.145	1.240
Throughput λ_i	9.896	3.958	1.979	0.986
Mean number of jobs $\overline{K}_i$	0.239	2.267	2.267	1.240
Utilization ρ_i	0.198	0.729	0.729	0.594

Schweitzer approximation is not very good (the mean deviation from exact results is approximately 6%) but is sufficient for most applications. An essential assumption for the Bard-Schweitzer method is that the removal of a job from a class influences only the mean number of jobs in that class and not the mean number of jobs in other classes (see Eq. (9.1)). This assumption is not very good and we can construct examples for which the computed results deviate considerably from the exact ones. In Section 9.1.2, the *self-correcting approximation technique (SCAT)* is introduced, which does not have this disadvantage. Of course, the algorithm is more complex to understand and implement than the Bard-Schweitzer approximation.

9.1.2 Self-Correcting Approximation Technique

Based on the basic idea of Bard and Schweitzer, Neuse and Chandy [NeCh81, ChNe82] developed techniques that are an improvement over the Bard-Schweitzer approximation [ZES88]. The main idea behind their approach is demonstrated on queueing networks consisting only of single server nodes.

9.1.2.1 Single Server Nodes The SCAT algorithm for single server nodes [NeCh81] gives better results than the Bard-Schweitzer approximation, especially for networks with small populations, because in the SCAT approximation we not only estimate the mean number of jobs but also the change in the mean number of jobs from one iteration step to the next. This estimate is used in the approximate formula for the $\overline{K}_{ir}(\mathbf{K}-\mathbf{1}_s)$. We define $F_{ir}(\mathbf{K})$ as the contribution of class-r jobs at the ith node for a given population vector $\mathbf{K}$ as:

$$F_{ir}(\mathbf{K}) = \frac{\overline{K}_{ir}(\mathbf{K})}{K_r}. \tag{9.3}$$

If a class-s job is removed from the network, $D_{irs}(\mathbf{K})$ gives the change in this contribution:

$$\begin{aligned} D_{irs}(\mathbf{K}) &= F_{ir}(\mathbf{K}-\mathbf{1}_s) - F_{ir}(\mathbf{K}) \\ &= \frac{\overline{K}_{ir}(\mathbf{K}-\mathbf{1}_s)}{(\mathbf{K}-\mathbf{1}_s)_r} - \frac{\overline{K}_{ir}(\mathbf{K})}{K_r}, \end{aligned} \tag{9.4}$$

where $(\mathbf{K}-\mathbf{1}_s)_r$ is defined as in Eq. (9.2). Because the values $\overline{K}_{ir}(\mathbf{K}-\mathbf{1}_s)$ are unknown, we are not able to compute the values $D_{irs}(\mathbf{K})$ as well. Therefore we estimate values for the difference $D_{irs}(\mathbf{K})$ and then we approximate the $\overline{K}_{ir}(\mathbf{K}-\mathbf{1}_s)$ values by the following formula based on Eq. (9.4):

$$\overline{K}_{ir}(\mathbf{K}-\mathbf{1}_s) = (\mathbf{K}-\mathbf{1}_s)_r \cdot [F_{ir}(\mathbf{K}) + D_{irs}(\mathbf{K})]\,. \tag{9.5}$$

Note that if $D_{irs}(\mathbf{K}) = 0$ we get the Bard Schweitzer approximation.

The *core algorithm* of SCAT approximation needs the estimated values for $D_{irs}(\mathbf{K})$ as input and, in addition, estimates for the mean values $\overline{K}_{ir}(\mathbf{K})$. The core algorithm can be described in the following three steps:

STEP C1 For $i = 1,\ldots,N$ and $r,s = 1,\ldots,R$, compute with Eq. (9.5) the estimated values for the mean number $\overline{K}_{ir}$ of jobs for the population $(\mathbf{K}-\mathbf{1}_s)$.

STEP C2 Analyze the queueing network by using one iteration step of the MVA as in Step 3 of the Bard- Schweitzer algorithm.

STEP C3 Check the stopping condition. If:

$$\max_{i,r} = \frac{\left|\overline{K}_{ir}^{(n)}(\mathbf{K}) - \overline{K}_{ir}^{(n-1)}(\mathbf{K})\right|}{K_r} < \varepsilon,$$

then stop the iteration, otherwise return to Step C1 (here the superscript $^{(n)}$ denotes the results of the nth iteration step). The use of $\varepsilon = (4000+16|K|)^{-1}$ as a suitable value is suggested by [NeCh81].

The SCAT algorithm produces estimates of the differences D_{irs} needed as input parameters of the core algorithm. So the core algorithm is run several times, and with the results of each run the differences $D_{irs}(\mathbf{K})$ are estimated. The estimated differences $D_{irs}(\mathbf{K})$ are computed by the SCAT algorithm, hence the name *self-correcting approximation technique*. To initialize the method, the jobs are distributed equally over all nodes of the network and the differences $D_{irs}(\mathbf{K})$ are set to zero. The SCAT algorithm can be described by the following four steps:

STEP 1 Use the core algorithm for the population vector $\mathbf{K}$, and input values $\overline{K}_{ir}(\mathbf{K}) = K_r/N$ and $D_{irs}(\mathbf{K}) = 0$ for all i,r,s. The auxiliary values of the core algorithm are not changed by the SCAT algorithm.

STEP 2 Use the core algorithm for each population vector $(\mathbf{K}-\mathbf{1_j})$ for $j = 1,\ldots,R$, with the input values $\overline{K}_{ir}(\mathbf{K}-\mathbf{1}_j) = (\mathbf{K}-\mathbf{1}_j)_r/N$ and $D_{irs}(\mathbf{K}-\mathbf{1}_j) = 0$ for all i,r,s.

STEP 3 For all $i = 1,\ldots,N$ and $r,s = 1,\ldots,R$, compute the estimated values for the $F_{ir}(\mathbf{K})$ and $F_{ir}(\mathbf{K}-\mathbf{1}_s)$, respectively, from Eq. (9.3) and the estimated values for the $D_{irs}(\mathbf{K})$ from Eq. (9.4).

STEP 4 Use the core algorithm for the population vector $\mathbf{K}$ with values $\overline{K}_{ir}(\mathbf{K})$ computed in Step 1, and values $D_{irs}(\mathbf{K})$, computed in Step 3. Finally, compute all other performance measures using Eqs. (7.21)-(7.31).

The memory requirement and computation time of the SCAT algorithm grows as $O(N \cdot R^2)$ and produces, in general, very good results. Now, the algorithm is used on an example:

Example 9.2 Consider the central-server model of Example 8.3 again, but now we use the SCAT algorithm to solve the network.

STEP 1 Use the core algorithm with $K = 6$ jobs in the network and the input data $\overline{K}_i(K) = K/N = 1.5$ and $D_i(K) = 0$, for $i = 1, \ldots, 4$. With these assumptions and $\varepsilon = 0.01$, the usage of the core algorithm will produce the same results as the Bard-Schweitzer algorithm as in Example 9.1. Therefore, we use those results directly:

$$\begin{aligned} &\overline{T}_1(6) = \underline{0.024}, \quad \overline{T}_2(6) = \underline{0.573}, \quad \overline{T}_3(6) = \underline{1.145}, \quad \overline{T}_4(6) = \underline{1.240}, \\ &\lambda(6) = \underline{9.896}, \\ &\overline{K}_1(6) = \underline{0.239}, \quad \overline{K}_2(6) = \underline{2.267}, \quad \overline{K}_3(6) = \underline{2.267}, \quad \overline{K}_4(6) = \underline{1.227}. \end{aligned}$$

STEP 2 Use the core algorithm for $(K-1) = 5$ jobs and the inputs $\overline{K}_i(5) = 5/N = 1.25$ and $D_i(5) = 0$, for $i = 1, \ldots, 4$.

STEP C1 Compute the estimated values for the $\overline{K}_i(K-1)$ from Eq. (9.5):

$$\overline{K}_i(4) = 4 \cdot \frac{\overline{K}_i(5)}{5} = \underline{1} \quad \text{for } i = 1, \ldots, 4.$$

STEP C2 One step of MVA:

$$\overline{T}_1(5) = \frac{1}{\mu_1}\left[1 + \overline{K}_1(4)\right] = \underline{0.04}, \quad \overline{T}_2(5) = \underline{0.4}, \quad \overline{T}_3(5) = \underline{0.8}, \quad \overline{T}_4(5) = \underline{1.2},$$

$$\lambda(5) = \frac{5}{\sum_{i=1}^{4} e_i \overline{T}_i(5)} = \underline{10.417}, \quad \overline{K}_1(5) = \overline{T}_1(5)\lambda(5)e_1 = \underline{0.417},$$

$$\overline{K}_2(5) = \underline{1.667}, \quad \overline{K}_3(5) = \underline{1.667}, \quad \overline{K}_4(5) = \underline{1.25}.$$

STEP C3 Check the stopping condition:

$$\max_i \frac{\left|\overline{K}_i^{(1)}(5) - \overline{K}_i^{(0)}(5)\right|}{5} = \underline{0.166} \quad > \varepsilon.$$

The iteration must be started again with Step C1.

For the sake of brevity, we do not present all the other iterations but give the results after the last (fourth) iteration:

$$\overline{T}_1(5) = \underline{0.024}, \quad \overline{T}_2(5) = \underline{0.495}, \quad \overline{T}_3(5) = \underline{0.989}, \quad \overline{T}_4(5) = \underline{1.123},$$
$$\lambda(5) = \underline{9.405},$$
$$\overline{K}_1(5) = \underline{0.222}, \quad \overline{K}_2(5) = \underline{1.861}, \quad \overline{K}_3(5) = \underline{1.861}, \quad \overline{K}_4(5) = \underline{1.056}.$$

STEP 3 Compute with Eq. (9.4) the estimated values for the differences $D_i(K)$:

$$D_1(6) = \frac{\overline{K}_1(5)}{5} - \frac{\overline{K}_1(6)}{6} = \frac{0.222}{5} - \frac{0.239}{6} = \underline{4.7 \cdot 10^{-3}},$$
$$D_2(6) = \frac{\overline{K}_2(5)}{5} - \frac{\overline{K}_2(6)}{6} = \underline{-5.7 \cdot 10^{-3}},$$
$$D_3(6) = \frac{\overline{K}_3(5)}{5} - \frac{\overline{K}_3(6)}{6} = \underline{-5.7 \cdot 10^{-3}},$$
$$D_4(6) = \frac{\overline{K}_4(5)}{5} - \frac{\overline{K}_4(6)}{6} = \underline{6.72 \cdot 10^{-3}}.$$

STEP 4 Use the core algorithm for $K = 6$ jobs in the network with the $\overline{K}_i(K)$ values from Step 1 and the $D_i(K)$ values from Step 3 as inputs.

STEP C1 Estimated values:

$$\overline{K}_1(5) = 5 \cdot \left[\frac{\overline{K}_1(6)}{6} + D_1(6)\right] = \underline{0.222}, \quad \overline{K}_2(5) = \underline{1.861},$$
$$\overline{K}_3(5) = \underline{1.861}, \quad \overline{K}_4(5) = \underline{1.056}.$$

STEP C2 One step of MVA:

$$\overline{T}_1(6) = \frac{1}{\mu_1}\left[1 + \overline{K}_1(5)\right] = \underline{0.024}, \quad \overline{T}_2(6) = \underline{0.572}, \quad \overline{T}_3(6) = \underline{1.144},$$
$$\overline{T}_4(6) = \underline{1.234},$$
$$\lambda(6) = \underline{9.908},$$
$$\overline{K}_1(6) = \overline{T}_1(6)\lambda(6)e_1 = \underline{0.242}, \quad \overline{K}_2(6) = \underline{2.267}, \quad \overline{K}_3(6) = \underline{2.267},$$
$$\overline{K}_4(6) = \underline{1.223}.$$

STEP C3 Stopping condition:

$$\max_i \frac{\left|\overline{K}_i^{(1)}(6) - \overline{K}_i^{(0)}(6)\right|}{6} = \underline{8.128 \cdot 10^{-3}} < \varepsilon,$$

is fulfilled and, therefore the iteration stops and the preceding results are the final results from the SCAT algorithm.

The exact results (MVA) are (see Example 8.3):

$$\overline{K_1}(6) = \underline{0.244}, \quad \overline{K_2}(6) = \underline{2.261}, \quad \overline{K_3}(6) = \underline{2.261}, \quad \overline{K_4}(6) = \underline{1.234}.$$

9.1.2.2 Multiple Server Nodes If we wish to analyze networks with multiple server nodes, then from the MVA analysis, Eq. (8.31), we need not only the $\overline{K}_{ir}$ values but also the conditional marginal probabilities $\pi_i(j \mid \mathbf{K} - \mathbf{1}_r)$. Similarly, for the SCAT algorithm, we need estimates not only for the values $\overline{K}_{ir}(\mathbf{K} - \mathbf{1}_s)$ but also for the probabilities $\pi_i(j \mid (\mathbf{K} - \mathbf{1}_r)$. In [NeCh81] a technique is suggested to estimate the marginal probabilities by putting the probability mass as close as possible to the mean number of jobs in the network. Thus, for example, if $\overline{K}_i(K-1) = 2.6$, then the probability that there are two jobs at node i is estimated to be $\pi_i(2 \mid K-1) = 0.4$, and the probability that there are three jobs at node i is estimated to be $\pi_i(3 \mid K-1) = 0.6$. This condition can be described by the following formulae. We define:

$$\text{ceiling}_{ir} = \left\lceil \overline{K}_i(\mathbf{K} - \mathbf{1}_r) \right\rceil, \tag{9.6}$$

$$\text{floor}_{ir} = \left\lfloor \overline{K}_i(\mathbf{K} - \mathbf{1}_r) \right\rfloor, \tag{9.7}$$

with:

$$\overline{K}_i = \sum_{s=1}^{R} \overline{K}_{is},$$

and estimate:

$$\pi_i(\text{floor}_{ir} \mid \mathbf{K} - \mathbf{1}_r) = \text{ceiling}_{ir} - \overline{K}_i(\mathbf{K} - \mathbf{1}_r), \tag{9.8}$$

$$\pi_i(\text{ceiling}_{ir} \mid \mathbf{K} - \mathbf{1}_r) = 1 - \pi_i(\text{floor}_{ir} \mid \mathbf{K} - \mathbf{1}_r), \tag{9.9}$$

$$\pi_i(j \mid \mathbf{K} - \mathbf{1}_r) = 0 \quad \text{for } j < \text{floor}_{ir} \text{ and } j > \text{ceiling}_{ir}. \tag{9.10}$$

Now we need to modify the core algorithm of SCAT to analyze networks with multiple server nodes. This modified core algorithm, which takes suitable estimates of $D_{irs}(\mathbf{K})$ and $\overline{K}_{ir}$ as inputs, can be described in the following steps:

STEP C1 For $i = 1, \ldots, N$ and $r, s = 1, \ldots, R$, compute estimates for the mean values $\overline{K}_{ir}(\mathbf{K} - \mathbf{1}_s)$ using Eq. (9.5). Compute using Eqs. (9.7)-(9.10), for each multiple server node i, the estimated values of the marginal probabilities $\pi_i(j \mid \mathbf{K} - \mathbf{1}_r), j = 0, \ldots, (m_i - 2)$ and $r = 1, \ldots, R$.

STEP C2 Analyze the queueing network using one step of the MVA via Eqs. (8.38), (8.39), and (8.41).

STEP C3 Check the stopping condition:

$$\max_{i,r} \frac{\left| \overline{K}_{ir}^{(n)}(\mathbf{K}) - \overline{K}_{ir}^{(n-1)}(\mathbf{K}) \right|}{K_r} < \varepsilon.$$

If this condition is not fulfilled, then return to Step C1.

The SCAT algorithm, which computes the necessary estimated input values for the differences $D_{irs}(\mathbf{K})$ and mean values $\overline{K}_{ir}(\mathbf{K})$ for the core algorithm, differs from the algorithm for networks with single server nodes only in the usage of the modified core algorithm, as follows:

STEP 1 Use the modified core algorithm for the population vector $\mathbf{K}$, where all $D_{irs}(\mathbf{K})$ values are set to 0.

STEP 2 Use the modified core algorithm for each population vector $(\mathbf{K} - \mathbf{1}_j)$.

STEP 3 Compute the F_{irs} and D_{irs} values from the previous results (see Eqs. (9.3) and (9.4)).

STEP 4 Use the modified core algorithm with the computed F and D values for the population vector $\mathbf{K}$.

Example 9.3 The closed network from Example 8.4 (Fig. 8.6) is now analyzed with the SCAT algorithm. The network contains $N = 4$ nodes and $K = 3$ jobs. The network parameters are given in Table 9.3. The analysis of the network is carried in the following four steps:

Table 9.3 Input parameters for the network of Example 9.3

i	e_i	$1/\mu_i$	m_i
1	1	0.5	2
2	0.5	0.6	1
3	0.5	0.8	1
4	1	1	∞

STEP 1 The modified core algorithm for $K = 3$ jobs in the network with the input parameters $\overline{K}_i(K) = K/N = 0.75$ and $D_i(K) = 0$ for $i = 1, \ldots, 4$ is executed as follows:

STEP C1 Estimate values for the $\overline{K}_i(K-1)$ from Eq. (9.5):

$$\overline{K}_i(2) = 2 \cdot \frac{\overline{K}_i(3)}{3} = \underline{0.5} \quad \text{for } i = 1, \ldots, 4.$$

Estimate values for $\pi_1(0 \mid K-1)$. Because of $\text{floor}_1 = \lfloor \overline{K}_1(K-1) \rfloor = \underline{0}$ and $\text{ceiling}_1 = \text{floor}_1 + 1 = \underline{1}$, we get:

$$\pi_1(0 \mid 2) = \text{ceiling}_1 - \overline{K}_1(2) = \underline{0.5}.$$

STEP C2 One step of MVA:

$$\overline{T}_1(3) = \frac{1}{\mu_1 m_1}\left[1 + \overline{K}_1(2) + \sum_{j=0}^{m_1-2}(m_1 - j - 1)\cdot \pi_1(j \mid 2)\right]$$
$$= \frac{1}{\mu_1 m_1}\left[1 + \overline{K}_1(2) + (m_1 - 1)\cdot \pi_1(0 \mid 2)\right] = \underline{0.5},$$
$$\overline{T}_2(3) = \frac{1}{\mu_2}\left[1 + \overline{K}_2(2)\right] = \underline{0.9}, \quad \overline{T}_3(3) = \frac{1}{\mu_3}\left[1 + \overline{K}_3(2)\right] = \underline{1.2},$$
$$\overline{T}_4(3) = \frac{1}{\mu_4} = \underline{1}. \quad \lambda(3) = \frac{3}{\sum_{i=1}^{4} e_i \overline{T}_i(3)} = \underline{1.176},$$
$$\overline{K}_1(3) = \overline{T}_1(3)\lambda(3)e_1 = \underline{0.588}, \quad \overline{K}_2(3) = \underline{0.529},$$
$$\overline{K}_3(3) = \underline{0.706}, \quad \overline{K}_4(3) = \underline{1.176}.$$

STEP C3 Check the stopping condition:

$$\max_i \frac{\left|\overline{K}_i^{(1)}(3) - \overline{K}_i^{(0)}(3)\right|}{3} = \underline{0.142} > \varepsilon = 0.01.$$

STEP C1 Estimate values for the $\overline{K}_i(K-1)$:

$$\overline{K}_1(2) = 2 \cdot \left(\frac{\overline{K}_1(3)}{3}\right) = \underline{0.392}, \quad \overline{K}_2(2) = \underline{0.353},$$
$$\overline{K}_3(2) = \underline{0.471}, \qquad \overline{K}_4(2) = \underline{0.784}.$$

Estimate values for $\pi_1(0 \mid K-1)$. With $\text{floor}_1 = \lfloor \overline{K}_1(K-1) \rfloor = \underline{0}$ and $\text{ceiling}_1 = \text{floor}_1 + 1 = \underline{1}$, we get:

$$\pi_1(0 \mid 2) = \text{ceiling}_1 - \overline{K}_1(2) = \underline{0.608}.$$

STEP C2 One step of MVA:

$$\vdots$$

After three iteration steps, we get the following results for the mean number of jobs:

$$\overline{K}_1(3) = \underline{0.603}, \quad \overline{K}_2(3) = \underline{0.480}, \quad \overline{K}_3(3) = \underline{0.710}, \quad \overline{K}_4(3) = \underline{1.207}.$$

STEP 2 The modified core algorithm for $(K-1) = 2$ jobs in the network with the input parameters $\overline{K}_i(2) = 2/N = 0.5$ and $D_i(2) = 0$ for $i = 1, \ldots, 4$ is executed. After three iteration steps, for the mean number of jobs we have:

$$\overline{K}_1(2) = \underline{0.430}, \quad \overline{K}_2(2) = \underline{0.296}, \quad \overline{K}_3(2) = \underline{0.415}, \quad \overline{K}_4(2) = \underline{0.859}.$$

STEP 3 Determine the estimates for the F_i and D_i values using Eqs. (9.3) and (9.4), respectively:

$$F_1(3) = \frac{\overline{K}_1(3)}{3} = \underline{0.201}, \quad F_1(2) = \frac{\overline{K}_1(2)}{2} = \underline{0.215},$$
$$F_2(3) = \frac{\overline{K}_2(3)}{3} = \underline{0.160}, \quad F_2(2) = \frac{\overline{K}_2(2)}{2} = \underline{0.148},$$
$$F_3(3) = \frac{\overline{K}_3(3)}{3} = \underline{0.237}, \quad F_3(2) = \frac{\overline{K}_3(2)}{2} = \underline{0.208},$$
$$F_4(3) = \frac{\overline{K}_4(3)}{3} = \underline{0.402}, \quad F_4(2) = \frac{\overline{K}_4(2)}{2} = \underline{0.430}.$$

Therefore we get:

$$D_1(3) = F_1(2) - F_1(3) = \underline{0.014},$$
$$D_2(3) = F_2(2) - F_2(3) = \underline{-0.012},$$
$$D_3(3) = F_3(2) - F_3(3) = \underline{-0.029},$$
$$D_4(3) = F_4(2) - F_4(3) = \underline{0.027}.$$

STEP 4 Execute the modified core algorithm for $K = 3$ jobs in the network with values $\overline{K}_i(K)$ from Step 1 and values $D_i(K)$ from Step 3 as inputs.

STEP C1 Estimate values for the $\overline{K}_i(K-1)$:

$$\overline{K}_1(2) = 2\left[\frac{\overline{K}_1(3)}{3} + D_1(3)\right] = \underline{0.430}, \quad \overline{K}_2(2) = \underline{0.296},$$
$$\overline{K}_3(2) = \underline{0.415}, \qquad \overline{K}_4(2) = \underline{0.859}.$$

Estimate value for:

$$\text{floor}_1 = \lfloor \overline{K}_1(2) \rfloor = \underline{0}, \quad \text{ceiling}_1 = \text{floor}_1 + 1 = \underline{1},$$
$$\pi_1(0 \mid 2) = \text{ceiling}_1 - \overline{K}_1(2) = \underline{0.570}.$$
$$\vdots$$

After two iterations, the modified core algorithm yields the following final results:

$$\overline{T}_1(3) = \underline{0.5}, \quad \overline{T}_2(3) = \underline{0.776}, \quad \overline{T}_3(3) = \underline{1.122}, \quad \overline{T}_4(3) = \underline{1},$$
$$\lambda(3) = \underline{1.224},$$
$$\overline{K}_1(3) = \underline{0.612}, \quad \overline{K}_2(3) = \underline{0.475}, \quad \overline{K}_3(3) = \underline{0.687}, \quad \overline{K}_4(3) = \underline{1.224}.$$

The exact results are (see Example 8.4):

$$\overline{T}_1(3) = \underline{0.512}, \quad \overline{T}_2(3) = \underline{0.776}, \quad \overline{T}_3(3) = \underline{1.127}, \quad \overline{T}_4(3) = \underline{1},$$
$$\lambda(3) = \underline{1.217},$$
$$\overline{K}_1(3) = \underline{0.624}, \quad \overline{K}_2(3) = \underline{0.473}, \quad \overline{K}_3(3) = \underline{0.686}, \quad \overline{K}_4(3) = \underline{1.217}.$$

9.1.2.3 Extended SCAT Algorithm The accuracy of the SCAT algorithm for networks with multiple server nodes depends not only on the estimates of the $\overline{K}_{ir}(\mathbf{K}-\mathbf{1}_s)$ values, but also on the approximation of the conditional marginal probabilities $\pi_i(j \mid \mathbf{K}-\mathbf{1}_r)$. The suggestion of [NeCh81], to distribute the whole probability mass only between the two values adjacent to the mean number of jobs $\overline{K}_i(\mathbf{K}-\mathbf{1}_r)$, is not very accurate. If, for example, $\overline{K}_i(\mathbf{K}-\mathbf{1}_r) = 2.6$, then the probability of having 0, 1, 4, and 5 jobs at the node cannot be neglected. Especially in the case when values $\overline{K}_i(\mathbf{K}-\mathbf{1}_r)$ are close to m_i (number of identical service units at the ith node), the discrepancy can be large. In this case, all relevant probabilities are zero because the values $\pi_i(j \mid \mathbf{K}-\mathbf{1}_r)$ for computing the mean response time are determined for $j = 0,\ldots,(m_i-2)$ only. In [AkBo88a] an improvement over this approximation of the conditional probabilities is recommended. In Akyildiz and Bolch's SCAT scheme, the marginal probabilities are spread over the whole range $0,\ldots,\overline{K}_i(\mathbf{K}-\mathbf{1}_r),\ldots,2\lfloor\overline{K}_i(\mathbf{K})\rfloor+1$. They define the following two functions, the first function, PR, being a scaling function with:

$$
\begin{aligned}
&PR(1) = \alpha \\
&PR(n) = \beta \cdot PR(n-1) \quad \text{for } n = 2,\ldots,\max_r\{K_r\},
\end{aligned}
$$

where the authors give 45 and 0.7, respectively, as the optimal values for α and β. The second function W is a weighting function that is defined for all $i = 1,\ldots,\max(K_r)$ as follows:

$$
\begin{aligned}
&W(0,0) = 1, \\
&W(i,j) = W(i-1,j) - \frac{W(i-1,j)\cdot PR(i)}{100}, \quad \text{for } j = 1,\ldots,(i-1), \\
&W(i,i) = 1 - \sum_{j=0}^{i-1} W(i,j).
\end{aligned}
$$

The values of the weighting function $W(i,j)$, $j = 0,\ldots,5$ are given in Table 9.4.

Table 9.4 Values of the weighting function W

i	$W(i,0)$	$W(i,1)$	$W(i,2)$	$W(i,3)$	$W(i,4)$	$W(i,5)$
0	1.0					
1	0.55	0.45				
2	0.377	0.308	0.315			
3	0.294	0.240	0.245	0.221		
4	0.248	0.203	0.208	0.187	0.154	
5	0.222	0.181	0.185	0.166	0.138	0.108

In addition to these two functions, we need the values of floor, ceiling, and maxval, where floor and ceiling are defined as in Eqs. (9.7) and (9.6) and

maxval is defined as follows:

$$\text{maxval}_{ir} = \min(2\,\text{floor}_{ir} + 1, m_i). \tag{9.11}$$

The term maxval describes the maximum variance of the probability mass. To compute the conditional probabilities, we divide the probability mass into $(\text{floor}_{ir} + 1)$ pairs, where the sum of each of these pairs has the combined probability mass $W(i,j)$. Formally this computation can be described as follows:

- For $j = 0, \ldots, \text{floor}_{ir}$, compute the conditional probabilities that there are j jobs at the ith node:

$$\pi_i(\text{floor}_{ir} - j \mid \mathbf{K} - \mathbf{1}_r) = W(\text{floor}_{ir}, \text{l\_dist}) \frac{\text{upperval} - \overline{K}_i(\mathbf{K} - \mathbf{1}_r)}{\text{upperval} - \text{lowerval}}, \tag{9.12}$$

  with:

$$\begin{aligned} \text{l\_dist} &= \text{floor}_{ir} - j, \\ \text{upperval} &= \text{ceiling}_{ir} + \text{l\_dist}, \\ \text{lowerval} &= \text{floor}_{ir} - \text{l dist} = j. \end{aligned}$$

- For $j = \text{ceiling}_{ir}, \ldots, \text{maxval}_{ir}$, we have:

$$\pi_i(j \mid \mathbf{K} - \mathbf{1}_r) = W(\text{floor}_{ir}, \text{u\_dist}) - \pi_i(\text{floor}_{ir} - \text{u\_dist} \mid \mathbf{K} - \mathbf{1}_r), \tag{9.13}$$

  with $\text{u\_dist} = j - \text{ceiling}_{ir}$.

- For $j > \text{maxval}_{ir}$ we get:

$$\pi_i(j \mid \mathbf{K} - \mathbf{1}_r) = 0.$$

This approximation assumes that the $\overline{K}_i(\mathbf{K} - \mathbf{1}_r)$ values are smaller than $K_r/2$. If any of these values is greater than $(K_r - 1)/2$, then we must also make sure that the value of upperval is not outside the range $0, \ldots, (K_r - 1)$. In this case upperval is set to $(K_r - 1)$. More details of this SCAT technique can be found in [AkBo88a].

Example 9.4 Consider the network shown in Fig. 9.1. To compute the marginal probabilities we apply the improved technique of [AkBo88a]. The service time of a job at the ith node, $i = 1, 2$, is exponentially distributed with rates $\mu_1 = 0.5$ and $\mu_2 = 1$. The queueing disciplines are FCFS and IS, respectively. There are $K = 3$ jobs in the network and $e_1 = e_2 = 1$.

STEP 1 Execute the modified core algorithm for $K = 3$ jobs in the network and the input parameters $\overline{K}_i(K) = K/N = 1.5$ and $D_i(K) = 0$ for $i = 1, 2$.

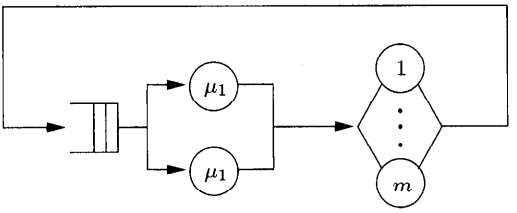


Fig. 9.1 A model of a multiprocessor system.

STEP C1 Estimate values for the $\overline{K}_i(2)$, Eq. (9.5):

$$\overline{K}_i(2) = 2 \cdot \left(\frac{\overline{K}_i(3)}{3} \right) = \underline{1}, \quad i = 1, 2.$$

Estimate value for $\pi_1(0 \mid K-1)$. Since:

$$\begin{aligned} \text{floor}_1 &= \lfloor \overline{K}_1(2) \rfloor = \underline{1}, \\ \text{ceiling}_1 &= \text{floor}_1 + 1 = \underline{2}, \\ \text{maxval}_1 &= \min(2 \cdot \text{floor}_1 + 1, m_1) = \underline{2}, \end{aligned}$$

$\pi_1(0 \mid 2)$ is determined with:

$$\begin{aligned} \text{l\_dist} &= \text{floor}_1 - 0 = \underline{1}, \quad \text{upperval} = \text{ceiling}_1 + 1 = \underline{3}, \\ \text{lowerval} &= \text{floor}_1 - 1 = \underline{0}, \end{aligned}$$

from Eq. (9.12):

$$\pi_1(0 \mid 2) = W(1,1) \cdot \frac{3 - \overline{K}_i(2)}{3} = \underline{0.3}.$$

STEP C2 One iteration step of the MVA:

$$\vdots$$

After four iterations, we get the following results:

$$\overline{K}_1(3) = \underline{2.187}, \quad \overline{K}_2(3) = \underline{0.813}.$$

STEP 2 Apply the modified core algorithm for $K = 2$ jobs with the input parameters $\overline{K}_i(2) = 1$ and $D_i(2) = 0$, for $i = 1, 2$.

STEP C1 Estimate values for the $\overline{K}_i(1)$:

$$\overline{K}_i(1) = \left(\frac{\overline{K}_i(2)}{2} \right) = \underline{0.5}, \quad i = 1, 2.$$

Estimate values for $\pi_1(0 \mid 1)$. With:

$$\text{floor}_1 = \lfloor \overline{K}_1(1) \rfloor = \underline{0}, \quad \text{ceiling}_1 = \text{floor}_1 + 1 = \underline{1},$$
$$\text{maxval}_1 = \min(2 \cdot \text{floor}_1 + 1, m_1) = \underline{1},$$

we determine:

$$\text{l\_dist} = \underline{0}, \quad \text{upperval} = \underline{1}, \quad \text{lowerval} = \underline{0},$$

and use these in Eq. (9.12) to compute:

$$\pi_1(0 \mid 1) = W(0,0) \cdot \frac{1 - \overline{K}_1(1)}{1} = \underline{0.5}.$$

STEP C2 One iteration step of the MVA:

$$\vdots$$

After two iterations, we get the following results:

$$\overline{K}_1(2) = \underline{1.333}, \quad \overline{K}_2(3) = \underline{0.667}.$$

STEP C3 Estimate the difference with Eq. (9.4):

$$D_1(3) = F_1(2) - F_1(3) = \underline{-0.062}, \quad D_2(3) = F_2(2) - F_2(3) = \underline{0.062}.$$

STEP 3 Apply the modified core algorithm for $K = 3$ jobs in the network:

STEP C1 Estimate values for the $\overline{K}_i(2)$:

$$\overline{K}_1(2) = \underline{1.333}, \quad \overline{K}_2(2) = \underline{0.667}.$$

Estimate value for $\pi_1(0 \mid 2)$. With:

$$\text{floor}_1 = \lfloor \overline{K}_1(2) \rfloor = \underline{1}, \quad \text{ceiling}_1 = \underline{2}, \quad \text{maxval}_1 = \underline{2},$$

we determine:

$$\text{l\_dist} = \underline{1}, \quad \text{upperval} = \underline{3}, \quad \text{lowerval} = \underline{0},$$

and, therefore:

$$\pi_1(0 \mid 2) = W(1,1) \cdot \frac{3 - \overline{K}_1(2)}{3} = \underline{0.25}.$$

STEP C2 One iteration step of the MVA:

$$\vdots$$

Table 9.5 Storage requirement for convolution, MVA, Bard-Schweitzer, and SCAT Algorithms

Methods	Storage Requirement
Convolution	$O(\prod_{r=1}^{R}(K_r+1))$
MVA	$O(N \cdot \prod_{r=1}^{R}(K_r+1))$
Bard-Schweitzer	$O(N \cdot R)$
SCAT	$O(N \cdot R^2)$

After two iterations, we get the final results:

$$\overline{T}_1(3) = \underline{2.57}, \quad \overline{T}_2(3) = \underline{1}, \quad \lambda(3) = \underline{0.84}, \quad \overline{K}_1(3) = \underline{2.16}, \quad \overline{K}_2(3) = \underline{0.84}.$$

A comparison with the exact values shows the accuracy of this technique for this example:

$$\overline{T}_1(3) = \underline{2.45}, \quad \overline{T}_2(3) = \underline{1}, \quad \lambda(3) = \underline{0.87}, \quad \overline{K}_1(3) = \underline{2.13}, \quad \overline{K}_2(3) = \underline{0.87}.$$

If we use the SCAT algorithm from Section 9.1.2.2 we get:

$$\overline{T}_1(3) = \underline{3}, \quad \overline{T}_2(3) = \underline{1}, \quad \lambda(3) = \underline{0.75}, \quad \overline{K}_1(3) = \underline{2.25}, \quad \overline{K}_2(3) = \underline{0.75}.$$

The storage requirement of SCAT is independent of the number of jobs ($= O(N \cdot R^2)$) but higher than for the Bard-Schweitzer approximation, especially for systems with many job classes and many jobs. The storage requirement for SCAT is much less than those of MVA or convolution (see Table 9.5). The accuracy is much better for the SCAT algorithm (the average deviation from exact results is approximately 3%) than for the Bard-Schweitzer approximation and, therefore, in practice SCAT is used more often than MVA, Bard-Schweitzer, or convolution to solve product-form queueing networks.

Problem 9.1 Solve Problem 7.3 with the Bard-Schweitzer and the SCAT algorithms.

Problem 9.2 Solve Problem 7.4 with the SCAT and the extended SCAT algorithms.

9.2 SUMMATION METHOD

The summation method (SUM) is based on the simple idea that for each node of a network, the mean number of jobs at the node is a function of the

throughput of this node:

$$\overline{K}_i = f_i(\lambda_i). \tag{9.14}$$

For nodes with load-independent service rates, function f_i has the following properties:

- λ_i must be in the range $0 \leq \lambda_i \leq m_i\mu_i$. For IS-nodes we have $0 < \lambda_i \leq K \cdot \mu_i$, because $\overline{K}_i = \lambda_i/\mu_i$ and $\overline{K}_i \leq K$.
- $f_i(\lambda_i) \leq f(\lambda_i + \Delta\lambda_i)$ for $\Delta\lambda_i > 0$, that is, f_i is a monotone non-decreasing function of λ_i.
- $f_i(0) = 0$.

Monotonicity does not hold for nodes with general load-dependent service rates and must therefore be examined for each individual case. For multiple server nodes the monotonicity is maintained.

To analyze product-form queueing networks, [BFS87] suggest the following formulae:

$$f_i(\lambda_i) = \overline{K}_i = \begin{cases} \dfrac{\rho_i}{1 - \dfrac{K-1}{K}\rho_i}, & \text{Type-1,2,4 } (m_i = 1), \\ m_i\rho_i + \dfrac{\rho_i}{1 - \dfrac{K-m_i-1}{K-m_i}\rho_i} \cdot P_{m_i}, & \text{Type-1 } (m_i > 1), \\ \dfrac{\lambda_i}{\mu_i}, & \text{Type-3.} \end{cases} \tag{9.15}$$

The utilization ρ_i is computed using Eq. (7.21) and the waiting probability P_{m_i} with Eq. (6.28) or approximately with Eq. (6.80). It should be noted that Eq. (9.15) gives exact values only for Type-3 nodes while the equations for the other node types are approximations. The approximate equations for Type-1 (-/M/1), Type-2, and Type-4 nodes can be derived from Eq. (6.13), $\overline{K}_i = \rho_i/(1-\rho_i)$, and the introduction of a correction factor $(K-1)/K$, respectively. The correction factor is motivated by the fact that in the case of $\rho_i = 1$, all K jobs are at node i, hence, for $\rho_i = 1$ we have $\overline{K}_i = K$. The idea of the correction factor can also be found in the equation for Type-1 (-/M/m) nodes by considering Eq. (6.29). If we assume that the functions f_i are given for all nodes i of a closed network, then the following equation for the whole network can be obtained by summing over all functions f_i of the nodes in the network:

$$\sum_{i=1}^{N} \overline{K}_i = \sum_{i=1}^{N} f_i(\lambda_i) = K. \tag{9.16}$$

With the relation $\lambda_i = \lambda \cdot e_i$, an equation to determine the overall throughput λ of the system can be given as:

$$\sum_{i=1}^{N} f_i(\lambda \cdot e_i) = g(\lambda) = K. \tag{9.17}$$

In the multiple class case, the Function (9.14) can be extended:

$$\overline{K}_{ir} = f_{ir}(\lambda_{ir}). \tag{9.18}$$

The function $f_{ir}(\lambda_{ir})$ has the following characteristics:

- f_{ir} is defined in the range $0 \leq \lambda_{ir} \leq \mu_{ir} \cdot m_i$ and, in case of an infinite server node, $0 \leq \lambda_{ir} \leq \mu_{ir} \cdot K_r$.
- f_{ir} is a monotone non decreasing function. $f_{ir}(\lambda_{ir}) \leq f_{ir}(\lambda_{ir} + \epsilon)$, $\epsilon > 0$.
- $f_{ir}(0) = 0$.

For *multiclass product-form queueing networks*, the following formulae, as extensions of Eq. (9.15), are given:

$$\overline{K}_{ir} = \begin{cases} \dfrac{\rho_{ir}}{1 - \dfrac{K-1}{K}\rho_i}, & \text{Type-1,2,4 } (m_i = 1) \\ m_i\rho_{ir} + \dfrac{\rho_{ir}}{1 - \dfrac{K-m_i-1}{K-m_i}\rho_i} \cdot P_{m_i}, & \text{Type-1 } (m_i > 1), \\ \dfrac{\lambda_{ir}}{\mu_{ir}}, & \text{Type-3,} \end{cases} \tag{9.19}$$

with:

$$\rho_i = \sum_{r=1}^{R} \rho_{ir}, \tag{9.20}$$

$$K = \sum_{r=1}^{R} K_r. \tag{9.21}$$

For the probability of waiting P_{m_i}, we can use the same formula as for the single class case (Eq. (6.28)) and Eq. (9.20) for the utilization ρ_i. Now we obtain a system of equations for the throughputs λ_r by summing over all functions f_{ir}:

$$\sum_{i=1}^{N} \overline{K}_{ir} = \sum_{i=1}^{N} f_{ir}(\lambda_{ir}) = K_r \quad (r = 1, \ldots, R). \tag{9.22}$$

With $\lambda_{ir} = \lambda_r e_{ir}$ we obtain the system of equations:

$$\sum_{i=1}^{N} f_{ir}(\lambda_r e_{ir}) = g_r(\lambda_r) = K_r \quad r = 1, \ldots, R. \tag{9.23}$$

In the next section, we introduce solution algorithms to determine the throughput λ in the single class case or the throughputs λ_r in the multiple class case using the preceding fixed-point equations.

9.2.1 Single Class Networks

An algorithm for single class queueing networks to determine λ with the help of Eq. (9.17) and the monotonicity of $f_i(\lambda_i)$ has the following form:

STEP 1 Initialization. Choose $\lambda_l = 0$ as a lower bound for the throughput and $\lambda_u = \min_i \left\{ \frac{\mu_i m_i}{e_i} \right\}$ as an upper bound. For -/G/∞-nodes, m_i must be replaced by K.

STEP 2 Use the bisection technique to determine λ:

STEP 2.1 Set $\lambda = \frac{\lambda_l + \lambda_u}{2}$.

STEP 2.2 Determine $g(\lambda) = \sum_{i=1}^{N} f_i(\lambda \cdot e_i)$, where the functions $f_i(\lambda \cdot e_i)$ are determined by using Eqs. (9.15) and (9.14).

STEP 2.3 If $g(\lambda) = K \pm \epsilon$, then stop the iteration and compute the performance measures of the network with the equations from Section 7.1.
If $g(\lambda) > K$, set $\lambda_u = \lambda$ and go back to Step 2.1.
If $g(\lambda) < K$, set $\lambda_l = \lambda$ and go also back to Step 2.1.

The advantage of this method is that the performance measures can be computed very easily. Furthermore, if we use suitable functions f_i, then the number K of jobs in the network and the number of servers m_i does not affect the computation time. In the case of monotonicity, the convergence is always guaranteed. By using suitable functions $f_i(\lambda_i)$, this method can also be used for an approximate analysis of closed non-product-form networks (see Section 10.1.4). In [BFS87] and [AkBo88b], it is shown that the summation method is also well suited for solving optimization problems in queueing networks (see Chapter 11). In the following example we show the use of the summation method:

Example 9.5 Consider Example 8.4 from Section 8.2.1. This network is examined again to have a comparison between the exact values and the approximate ones as computed with the summation method. The network

Table 9.6 Input parameters for the network of Example 9.5

i	e_i	$1/\mu_i$	m_i
1	1	0.5	2
2	0.5	0.6	1
3	0.5	0.8	1
4	1	1	∞

parameters are given in Table 9.6. The analysis of the network is carried out in the following steps:

STEP 1 Initialization:

$$\lambda_l = \underline{0} \quad \text{and} \quad \lambda_u = \min_i \left\{ \frac{m_i \mu_i}{e_i} \right\} = \underline{2.5}.$$

STEP 2 Bisection:

STEP 2.1 $\lambda = \dfrac{\lambda_l + \lambda_u}{2} = \underline{1.25}.$

STEP 2.2 Computation of the functions $f_i(\lambda_i)$, $i = 1, 2, 3, 4$ with Eq. (9.15). For the -/M/2 node 1 we have:

$$\rho_1 = \frac{\lambda \cdot e_1}{\mu_1 \cdot m_1} = \underline{0.3125} \quad \text{and} \quad P_{m_1} = \underline{0.149} \quad \text{Eq. (6.28)},$$

and, therefore:

$$f_1(\lambda_1) = 2\rho_1 + \rho_1 P_{m_1} = \underline{0.672}.$$

Furthermore:

$$f_2(\lambda_2) = \frac{\rho_2}{1 - \frac{2}{3}\rho_2} = \underline{0.5} \quad \text{with} \quad \rho_2 = \underline{0.375},$$

$$f_3(\lambda_3) = \frac{\rho_3}{1 - \frac{2}{3}\rho_3} = \underline{0.75} \quad \text{with} \quad \rho_3 = \underline{0.5},$$

$$f_4(\lambda_4) = \frac{\lambda_4}{\mu_4} = \underline{1.25}.$$

Thus, we get:

$$g(\lambda) = \sum_{i=1}^{N} f_i(\lambda_i) = \underline{3.172}.$$

STEP 2.3 Check the stopping condition. Because of $g(\lambda) > K$, we set $\lambda_u = \lambda = 1.25$ and go on with the iteration.

STEP 2.1

$$\lambda = \frac{\lambda_l + \lambda_u}{2} = \underline{0.625}.$$

STEP 2.2

$$\begin{aligned} f_1(\lambda_1) &= 2\rho_1 + \rho_1 P_{m_1} = \underline{0.319} && \text{with } \rho_1 = \underline{0.156} \text{ and } P_{m_1} = \underline{0.042}, \\ f_2(\lambda_2) &= \underline{0.214} && \text{with } \rho_2 = \underline{0.1875}, \\ f_3(\lambda_3) &= \underline{0.3} && \text{with } \rho_3 = \underline{0.25}, \\ f_4(\lambda_4) &= \underline{0.625}. \end{aligned}$$

It follows that:

$$g(\lambda) = \sum_{i=1}^{4} f_i(\lambda_i) = \underline{1.458}.$$

STEP 2.3 Because of $g(\lambda) < K$, we set $\lambda_l = \lambda = 0.625$.

STEP 2.1

$$\lambda = \frac{\lambda_l + \lambda_u}{2} = \frac{0.625 + 1.25}{2} = \underline{0.9375}.$$

$$\vdots$$

This iteration is continued until the value of $g(\lambda)$ equals (within $\epsilon = 0.001$) the number of jobs K in the system. To make the bisection clearer, we show the series of λ_l and λ_u values in Table 9.7.

Table 9.7 Intervals for Example 9.5

Step:	0	1	2	3	4	5	6	7	8	9	10
λ_l	0	0	0.625	0.9375	1.094	1.172	1.172	1.191	1.191	1.191	1.191
λ_u	2.5	1.25	1.25	1.25	1.25	1.25	1.211	1.211	1.201	1.196	1.194

The overall throughput of the network is $\lambda = \underline{1.193}$ as computed by this approximation method. Compare this with the exact value of the overall throughput is $\lambda = \underline{1.217}$ The approximate values of the throughputs and the mean number of jobs calculated using the SUM (Eq. (9.14)) and the exact values for this network calculated using MVA (see Example 8.4) are summarized in Table 9.8.

Table 9.8 Exact and approximate values for the throughputs λ_i and the number of jobs $\overline{K}_i$ for Example 9.5

i	1	2	3	4
$\lambda_{i_{\text{SUM}}}$	1.193	0.596	0.596	1.193
$\lambda_{i_{\text{MVA}}}$	1.218	0.609	0.609	1.218
$\overline{K}_{i_{\text{SUM}}}$	0.637	0.470	0.700	1.193
$\overline{K}_{i_{\text{MVA}}}$	0.624	0.473	0.686	1.217

9.2.2 Multiclass Networks

Now we assume R job classes and $\mathbf{K} = (K_1, K_2, \ldots, K_R)$ jobs in the network. Therefore, we get a system of R equations. In general, g_r are non-linear functions of λ_r: $g_r(\lambda_1, \lambda_2, \ldots, \lambda_R)$. This is a coupled system of non-linear equations in the throughput of the R job classes and, hence, it is not possible to solve each equation separately from the others. We first transform the system of equations into the fixed-point form:

$$\mathbf{x} = f(\mathbf{x}).$$

Under certain conditions f has only one fixed-point and can then be used to calculate $\mathbf{x}$ iteratively using successive substitution as follows:

$$\mathbf{x}^{(n+1)} = f(\mathbf{x}^{(n)}).$$

This method of solution is also called *fixed-point iteration.* In the case of the SUM, for the calculation of the throughputs λ_r, we get the following system of equations:

$$\sum_{i=1}^{N} \overline{K_{ir}} = \sum_{i=1}^{N} f_{ir}(\lambda_r \cdot e_{ir}) = K_r, \quad r = 1, \ldots, R. \tag{9.24}$$

To apply the fixed-point iteration, this equation can easily be transformed to:

$$\lambda_r \sum_{i=1}^{N} \text{fix}_{ir}(\lambda_r \cdot e_{ir}) = K_r,$$

or:

$$\lambda_r = \frac{K_r}{\sum_{i=1}^{N} \text{fix}_{ir}(\lambda_r \cdot e_{ir})} = f_r(\lambda_1, \ldots, \lambda_R) = f_r(\boldsymbol{\lambda}), \tag{9.25}$$

see [BaTh94].

For the functions fix_{ir}, we get from Eq. (9.25):

$$\text{fix}_{ir} = \begin{cases} \dfrac{\dfrac{e_{ir}}{\mu_{ir}}}{1 - \dfrac{K-1}{K} \cdot \rho_i}, & \text{Type-1,2,4 } m_i = 1, \\ \dfrac{e_{ir}}{\mu_{ir}} + \dfrac{\dfrac{e_{ir}}{m_i \cdot \mu_{ir}}}{1 - \dfrac{K-m_i-1}{K-m_i} \cdot \rho_i} \cdot P_{m_i}(\rho_i), & \text{Type-1 } m_i > 1, \\ \dfrac{e_{ir}}{\mu_{ir}}, & \text{Type-3}, \end{cases} \tag{9.26}$$

and the throughputs λ_r can be obtained by the following iteration:

STEP 1 Initialization:

$$\lambda_1 = \lambda_2 = \cdots = \lambda_R = 0.00001; \quad \epsilon = 0.00001.$$

STEP 2 Compute the throughput λ_r for all $r = 1, \ldots, R$:

$$\lambda_r = \frac{K_r}{\sum_{i=1}^{N} \text{fix}_{ir}(\lambda_1, \ldots, \lambda_R)}.$$

STEP 3 Determine error norm:

$$e = \sqrt{\sum_{r=1}^{R} (\lambda_{r_n} - \lambda_{r_{n+1}})^2}.$$

If $e > \epsilon$, go to Step 2.

STEP 4 Compute other performance measures.

The Newton-Raphson method can also be used to solve the non-linear system of equations for the SUM [Hahn90]. There is no significant difference between fixed-point iteration and the Newton-Raphson method in our experience. The computing time is tolerable for small networks for both methods (below 1 second), but it increases dramatically for the Newton-Raphson method (up to several hours) while it is still tolerable for the fixed-point iteration (several seconds) [BaTh94]. Therefore, the fixed-point iteration should preferably be used. The relative error can be computed with the formula:

$$\text{difference} = \frac{|\text{exact value} - \text{computed value}|}{\text{exact value}} \cdot 100,$$

and gives, for the previous example, an error of 2.6% for the overall throughput. Experiments have shown that the relative error for the SUM method is between 5% and 15%. It is clear that the results can be further improved if better functions f_i could be found.

Problem 9.3 Solve Problem 7.3 and Problem 7.4 using the SUM.

9.3 BOTTAPPROX METHOD

We have seen that, compared to the MVA, the SUM approximation needs much less memory and computation time. But if we take a closer look at it, we will see that the number of iterations to obtain reasonably accurate results is still very large for many networks. Now the question arises whether it is possible to choose the initial guess of throughput $\lambda^{(0)}$ so that the number of iterations can be reduced while still maintaining the accuracy of the results. A good initial value of throughput based on the indices of the bottleneck node is suggested by [BoFi93]. Recall that the node with the highest utilization ρ_i is called the bottleneck node. The performance measures of the bottleneck node are characterized by the index bott. The resulting approximation method is called *bottapprox (BOTT).*

9.3.1 Initial Value of λ

For the throughput λ we have the relation:

$$\lambda = \frac{\mu_i \cdot m_i}{e_i} \cdot \rho_i, \quad i = 1, \ldots, N,$$

with:

$$0 \leq \rho_i \leq 1, \quad i = 1, \ldots, N.$$

In contrast to the SUM in the BOTT we assume that the utilization ρ_{bott} of the bottleneck node is very close to 1, and therefore we can choose for the initial value of the throughput $\lambda^{(0)}$:

$$\lambda^{(0)} = \min_{i \in \{1,\ldots N\}} \left\{ \frac{\mu_i \cdot m_i}{e_i} \right\}, \quad \text{bott} = \text{argmin} \left\{ \frac{\mu_i \cdot m_i}{e_i} \right\},$$

where argmin is the function that returns the index of the node with the minimum value.

9.3.2 Single Class Networks

The BOTT is an iterative method where for each node i in each iteration step, the mean number of jobs $\overline{K}_i$ is determined as a function of the chosen

throughput $\lambda_i^{(k)}$:

$$\overline{K}_i = f_i(\lambda_i). \tag{9.27}$$

For the function f_i, the same approximations are used as in the SUM. By summing over all nodes, we get for the function $g(\lambda)$:

$$g(\lambda) = \sum_{i=1}^{N} \overline{K}_i. \tag{9.28}$$

This sum $g(\lambda)$ is compared to the number of jobs K in the network and a correction factor:

$$\text{corr} = \frac{K}{g(\lambda)}, \tag{9.29}$$

is determined. We assume that in each node i the error in computing the mean number of jobs $\overline{K}_i$ is the same, and therefore for each node the following relation holds:

$$\overline{K}_i^{(k+1)} = \overline{K}_i^{(k)} \cdot \frac{K}{g(\lambda)}. \tag{9.30}$$

For the bottleneck node the new mean number of jobs in the node is computed as follows:

$$\overline{K}_{\text{bott}}^{(k+1)} = \overline{K}_{\text{bott}}^{(k)} \cdot \frac{K}{g(\lambda)}, \tag{9.31}$$

and the new utilization $\rho_{\text{bott}}^{(k+1)}$ can then be determined as a function of the new mean number of jobs $\overline{K}_{\text{bott}}^{(k+1)}$ at the bottleneck node using Eq. (9.15), that is:

$$\rho_{\text{bott}}^{(k+1)} = h(\overline{K}_{\text{bott}}^{(k+1)}). \tag{9.32}$$

Now the new value for the throughput λ at the bottleneck node can be computed as a function of the new utilization (Eqs. (7.23) and (7.25)):

$$\lambda^{(k+1)} = \frac{\rho_{\text{bott}}^{(k+1)} \cdot m_{\text{bott}} \cdot \mu_{\text{bott}}}{e_{\text{bott}}}. \tag{9.33}$$

If none of the following stopping conditions is fulfilled then the iteration is repeated:

- If the correction factor corr is in the interval $[(1-\epsilon), \ldots, (1+\epsilon)]$, then stop the iteration. Tests have shown that for $\epsilon = 0.025$ we get good results. For smaller ϵ, only the number of iterations increased but the accuracy of the results did not improve significantly.

- If in the computation the value of $\rho_{\text{bott}} = h(\overline{K}_{\text{bott}})$ is greater than 1, then set $\rho_{\text{bott}} = 1$ and stop the iteration.
- To prevent endless loops, the procedure is stopped when the number of iterations reaches a predefined upper limit.

The function h previously mentioned for different node types is defined as follows (see Eq. (9.15)):

1. For node types -/M/1-FCFS, -/G/1-PS, and -/G/1-LCFS-PR:

$$h(\overline{K}_{\text{bott}}) = \frac{\overline{K}_{\text{bott}}}{1 + \frac{K-1}{K} \cdot \overline{K}_{\text{bott}}}.$$

2. For node type -/G/∞-IS:

$$h(\overline{K}_{\text{bott}}) = \frac{\overline{K}_{\text{bott}}}{K}.$$

3. For node type -/M/m-FCFS it is not possible to give an exact expression for f_i and therefore an appropriate approximation is chosen (see [BoFi93]):

$$h(\overline{K}_{\text{bott}}) = \frac{b - \sqrt{c}}{2 \cdot f \cdot m_{\text{bott}}},$$

with:

$$f = \frac{K - m_{\text{bott}} - 1}{K - m_{\text{bott}}},$$
$$b = \overline{K}_{\text{bott}} \cdot f + m_{\text{bott}} + P_{m_{\text{bott}}},$$
$$c = b^2 - 4 \cdot m_{\text{bott}} \cdot \overline{K}_{\text{bott}} \cdot f.$$

Now we give an algorithm for the BOTT:

STEP 1 Initialize:

$$\lambda = \min_{i \in \{1,\ldots,N\}} \left\{ \frac{\mu_i \cdot m_i}{e_i} \right\}, \quad \text{bott} = \operatorname{argmin} \left\{ \frac{\mu_i \cdot m_i}{e_i} \right\}.$$

STEP 2 Iteration:

STEP 2.1 Compute the following performance measures for $i = 1, \ldots, N$:

$$\lambda_i = \lambda \cdot e_i, \quad \rho_i = \frac{\lambda_i}{m_i \cdot \mu_i}, \quad \overline{K}_i = f_i(\lambda).$$

STEP 2.2 Compute the new throughput λ:

$$g(\lambda) = \sum_{i=1}^{N} \overline{K}_i, \qquad \overline{K}_{\text{bott}} = \overline{K}_{\text{bott}} \cdot \frac{K}{g(\lambda)},$$

$$\rho_{\text{bott}} = h(\overline{K}_{\text{bott}}), \qquad \lambda = \frac{\rho_{\text{bott}} \cdot m_{\text{bott}} \cdot \mu_{\text{bott}}}{e_{\text{bott}}}.$$

STEP 2.3 Check the stopping condition:

$$\left| \frac{K}{g(\lambda)} \right| \leq \epsilon.$$

If it is fulfilled, then stop the iteration, otherwise return to Step 2.1.

STEP 3 For $i = 1, \ldots, N$ compute:

$$\overline{K}_i = \overline{K}_i \cdot \frac{K}{g(\lambda)}.$$

To estimate the accuracy of the BOTT and to compare it with the SUM, we investigated 64 different product-form networks. The mean deviation from the exact solution for the overall throughput λ and for the mean response time $\overline{T}$ was 2%. The corresponding value for the summation method was about 3%. The number of iterations of the SUM were about three times the number of iterations of the BOTT. For details see Table 9.9.

Table 9.9 Deviation in percent (%) from the exact values (obtained by MVA)

	Error in λ		Error in $\overline{T}$		Number of Iterations	
	SUM	BOTT	SUM	BOTT	SUM	BOTT
min	0	0	0	0	1	1
mean	2.60	2.10	2.70	2.10	6.60	2.30
max	7.40	13.90	7.80	12.20	13	5

9.3.3 Multiclass Networks

One of the advantages of the BOTT method is that it can easily be extended to multiple class networks. The necessary function $h_r(\overline{K}_{ir})$ is given by (Eq. (9.19)):

1. For node types -/M/1-FCFS, -/G/1-PS and -/G/1-PR:

$$h_r(\overline{K}_{\text{bott},r}) = \overline{K}_{\text{bott},r} \cdot \left(1 - \frac{K-1}{K} \cdot \rho_{\text{bott}}\right). \tag{9.34}$$

2. For node type -/G/∞-IS, $h_r(\overline{K}_{i,r})$:

$$h_r(\overline{K}_{\text{bott},r}) = \frac{\overline{K}_{\text{bott},r}}{K}. \tag{9.35}$$

3. For the node type -/M/m-FCFS:

$$h_r(\overline{K}_{\text{bott},r}) = \frac{\overline{K}_{\text{bott},r}}{m_{\text{bott}} + \dfrac{1}{1 - \dfrac{K - m_{\text{bott}} - 1}{K - m_{\text{bott}}} \cdot \rho_{\text{bott}}} \cdot P_{m_{\text{bott}}}(\rho_{\text{bott}})} \tag{9.36}$$

Remark: Eqs. (9.34) and (9.36) are approximations because we used $\rho_{\text{bott},r}$ part of $\rho_{\text{bott}} = \sum_{i=1}^{R} \rho_{\text{bott},i}$ when we created the $h_r(\overline{K}_{i,r})$ function.

Now we give the algorithm for the multiple class BOTT:

STEP 1 Initialize:

$$\lambda_r = \min_{i \in \{1,\ldots,N\}} \left\{ \frac{\mu_{i,r} \cdot m_i}{R \cdot e_{i,r}} \right\}, \text{ for } r = 1, \ldots, R,$$

$$\text{bott}, r = \text{argmin} \left\{ \frac{\mu_{i,r} \cdot m_i}{e_{i,r}} \right\}, r.$$

STEP 2 Iteration:

STEP 2.1 Compute the performance measures:

$$\left.\begin{aligned} \lambda_{ir} &= \lambda_r e_{ir}, \\ \rho_{ir} &= \frac{\lambda_{ir}}{m_i \cdot \mu_{ir}}, \\ \overline{K}_{ir} &= f_{ir}(\lambda_1, \ldots, \lambda_R), \\ \rho_i &= \sum_{r=1}^{R} \rho_{ir}, \end{aligned}\right\} \quad \begin{array}{l} r = 1, \ldots, R, \\ i = 1, \ldots, N. \end{array}$$

STEP 2.2 Compute the new throughputs:

$$\left.\begin{aligned} g(\lambda_r) &= \sum_{i=1}^{N} \overline{K}_{ir}, \\ \overline{K}_{\text{bott},r} &= \overline{K}_{\text{bott},r} \frac{K_r}{g(\lambda_r)}, \\ \rho_{\text{bott},r} &= h_r(\overline{K}_{\text{bott},r}), \\ \lambda_r &= \frac{\rho_{\text{bott},r} \cdot m_{\text{bott}} \cdot \mu_{\text{bott},r}}{e_{\text{bott},r}}, \end{aligned}\right\} \quad r = 1, \ldots, R.$$

STEP 2.3 Stop the iteration for class r. If:

$$\left(\left|\frac{K_r}{g(\lambda_r)}\right| > \epsilon\right),$$

go to Step 2.1, else stop iteration for class r.

STEP 3 Final value of $\overline{K}_{ir}$:

$$\overline{K}_{ir} = \overline{K}_{ir} \cdot \frac{K_r}{g(\lambda_r)}, \quad r = 1, \ldots, R, \ i = 1, \ldots, N.$$

The deviation from the exact results is similar to the one in the single class case for the SUM and is slightly worse but tolerable for the BOTT (see Table 9.10). The average number of iterations for the examples of Table 9.10

Table 9.10 Deviation in percent (%) from the exact results for the throughput λ (average of 56 examples)

	BOTT	SUM
min	0	0
mean	6.7	4.68
max	38.62	40.02

used by the bottapprox algorithm is 4.5, and the maximal value is 20, which is much smaller in comparison with the SUM.

Problem 9.4 Solve Problem 7.3 and Problem 7.4 using the BOTT.

9.4 BOUNDS ANALYSIS

Another possibility for reducing the computation time for the analysis of product-form queueing networks is to derive the upper and lower bounds for the performance measures instead of the exact values. In this section we introduce two methods for computing upper and lower bounds, *asymptotic bounds analysis (ABA)* and *balanced-job-bounds analysis (BJB)*. These two methods use simple equations to determine the bounds. Bounds for the system throughput and for the mean number of jobs are calculated. The largest possible throughput and the lowest possible response time of the network are called *optimistic bounds*, and the lowest possible throughput and greatest possible mean response time are called *pessimistic bounds*. So we have:

$$\lambda_{\text{pes}} \leq \lambda \leq \lambda_{\text{opt}} \quad \text{and} \quad \overline{T}_{\text{opt}} \leq \overline{T} \leq \overline{T}_{\text{pes}}.$$

With these relations other bounds for the performance measures such as throughputs and utilizations of the nodes can be determined.

In the following we make a distinction between three different network types:

- Type A describes a closed network containing no infinite server nodes.
- Type B describes a closed network with IS nodes.
- Type C describes any kind of open network.

We restrict our consideration to networks with only one job class. Generalizations to networks with several job classes can be found in [EaSe86].

9.4.1 Asymptotic Bounds Analysis

The ABA [MuWo74a, Klei76, DeBu78] gives upper bounds for the system throughput and lower bounds for the mean response time of a queueing model (optimistic bounds). The only condition is that the service rates have to be independent of the number of jobs at the node or in the network. As inputs for the ABA method, the maximum relative utilization of all non-IS-nodes, $x_{\max}$, and the sum x_{sum} of the relative utilizations of these nodes are needed:

$$x_{\max} = \max_i(x_i) \quad \text{and} \quad x_{\text{sum}} = \sum_i x_i,$$

where $x_i = e_i/\mu_i$ is the relative utilization of node i (Eq. (7.6)). The relative utilization of an IS node (thinking time at the terminals) is called Z.

At first we consider the case of an open network (Type C): Since the utilization $\rho_i = \lambda_i/\mu_i$ with $\lambda_i = \lambda \cdot e_i$, the following relation holds:

$$\rho_i = \lambda \cdot x_i. \tag{9.37}$$

Due to the fact that no node can have a utilization greater than 1, the upper bound for the throughput of an open network cannot exceed $\lambda_{\text{sat}} = 1/x_{\max}$ if the network is to remain stable. Therefore, λ_{sat} is the lowest arrival rate where one of the nodes in the system is fully utilized. The node that is utilized most is the node with the highest relative utilization $x_{\max}$. For this *bottleneck* node we have $\rho_{\max} = \lambda \cdot x_{\max} \leq 1$, and the optimistic bound for the throughput is therefore:

$$\lambda \leq \frac{1}{x_{\max}}. \tag{9.38}$$

To determine the optimistic bounds for the mean response time, we assume that in the best case no job in the network is hindered by another job. With this assumption, the waiting time of a job is zero. Because the relative utilization $x_i = e_i/\mu_i$ of a node is the mean time a job spends being served at

this node, the mean system response time in the non-IS nodes is simply given as the sum of the relative utilizations. Therefore, for the optimistic bounds on mean response time we get:

$$\overline{T} \geq x_{\text{sum}}. \tag{9.39}$$

For closed networks we consider networks with IS nodes (Type B). From the results obtained for such networks we can easily derive those for networks of Type A by setting the mean think time Z at the terminal nodes to zero. The bounds are determined by examining the network behavior under both light and heavy loads. We start with the assumption of a heavily loaded system (many jobs are in the system). The greater the number of jobs K in the network, the higher the utilization of the individual nodes. Since:

$$\rho_i(K) = \lambda(K)x_i \leq 1,$$

the highest possible overall network throughput is restricted by each node in the network, especially by the bottleneck node. As in open models we have:

$$\lambda(K) \leq \frac{1}{x_{\max}}.$$

Now assume that there are only a few jobs in the network (lightly loaded case). In the extreme case $K = 1$, the network throughput is given by $1/(x_{\text{sum}} + Z)$. For $K = 2, 3, \ldots$ jobs in the network, the throughput reaches its maximum when each job in the system is not hindered by other jobs. In this case we have $\lambda(K) = K/(x_{\text{sum}} + Z)$.

These observations can be summarized in the following upper bound for the network throughput:

$$\lambda(K) \leq \min\left\{\frac{1}{x_{\max}}, \frac{K}{x_{\text{sum}} + Z}\right\}. \tag{9.40}$$

To determine the bound for the mean response time $\overline{T}(K)$, we transform Eq. (9.40) with the help of Little's theorem:

$$\frac{K}{\overline{T}(K) + Z} \leq \min\left\{\frac{1}{x_{\max}}, \frac{K}{x_{\text{sum}} + Z}\right\},$$

and get:

$$\begin{aligned} &\max\left\{x_{\max}, \frac{x_{\text{sum}} + Z}{K}\right\} \leq \frac{\overline{T}(K) + Z}{K}, \\ \text{or} \quad &\max\{x_{\text{sum}}, K \cdot x_{\max} - Z\} \leq \overline{T}(K). \end{aligned} \tag{9.41}$$

The asymptotic bounds for all three network types are listed in Table 9.11.

Table 9.11 Summary of the ABA bounds

	Network Type	ABA Bounds
λ	A	$\lambda(K) \leq \min\left\{\frac{K}{x_{\text{sum}}}, \frac{1}{x_{\max}}\right\}$
	B	$\lambda(K) \leq \min\left\{\frac{K}{x_{\text{sum}}+Z}, \frac{1}{x_{\max}}\right\}$
	C	$\lambda \leq \frac{1}{x_{\max}}$
$\overline{T}$	A	$\overline{T}(K) \geq \max\{x_{\text{sum}}, K \cdot x_{\max}\}$
	B	$\overline{T}(K) \geq \max\{x_{\text{sum}}, K \cdot x_{\max} - Z\}$
	C	$\overline{T} \geq x_{\text{sum}}$

Example 9.6 As an example for the use of ABA, consider the closed product-form queueing network given in Fig. 9.2. There are $K = 20$ jobs in the network and the mean service times and the visit ratios are given as follows:

$$1/\mu_1 = 4.6, \quad 1/\mu_2 = 8, \qquad 1/\mu_3 = 120 = Z,$$
$$e_1 = 2, \qquad e_2 = e_3 = 1.$$

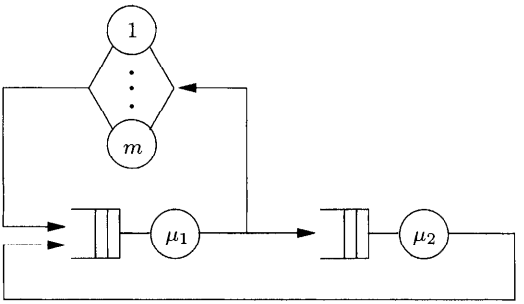


Fig. 9.2 Single class queueing network.

At first we determine:

$$x_{\max} = \max\left\{\frac{e_1}{\mu_1}, \frac{e_2}{\mu_2}\right\} = \underline{9.2}, \quad x_{\text{sum}} = \frac{e_1}{\mu_1} + \frac{e_2}{\mu_2} = \underline{17.2},$$
$$Z = \frac{e_3}{\mu_3} = \underline{120}.$$

With these values and the formulae given in Table 9.11, we can determine the asymptotic bounds immediately.

Throughput:

$$\lambda(K) \leq \min\left\{\frac{K}{x_{\text{sum}}+Z}, \frac{1}{x_{\max}}\right\} = \min\left\{\frac{20}{137.2}; \frac{1}{9.2}\right\} = \underline{0.109}.$$

Mean response time:

$$\overline{T}(K) \geq \max\{x_{\text{sum}}, K\cdot x_{\max} - Z\} = \max(17.2; 64) = \underline{64}.$$

Now the bounds for the other performance measures can be computed. With the formula $\rho_i = \lambda \cdot x_i$, we can, for example, compute the upper bounds for the utilizations $\rho_1 \leq 1$ and $\rho_2 \leq 0.87$. For this example, the exact values for throughput and response time are $\lambda(K) = \underline{0.100}$ and $\overline{T}(K) = \underline{80.28}$, respectively.

9.4.2 Balanced Job Bounds Analysis

With the BJB method [ZSEG82], upper as well as lower bounds can be obtained for the system throughput and mean response time of product-form networks. In general, using this analysis we get better results than with the ABA method. The derivation of bounds by the BJB method is demonstrated only for closed networks of Type A. The derivation of the bounds for the other two network types is similar and is therefore left to the reader as an exercise. In the derivation of BJB, it is assumed that the network is *balanced*: that is, the relative utilizations of all nodes are the same: $x_1 = \ldots = x_N = x$. The throughput of a balanced network can be computed by applying Little's theorem:

$$\lambda(K) = \frac{K}{\sum_{i=1}^{N} \overline{T}_i(K)},$$

where $\overline{T}_i(K)$ is the mean response time in the ith node with K jobs in the network. For product-form networks, Reiser and Lavenberg [ReLa80] have shown that:

$$\overline{T}_i(K) = \left[1 + \overline{K}_i(K-1)\right] \cdot x.$$

If we combine the two preceding formulae, we get:

$$\lambda(K) = \frac{K}{\sum_{i=1}^{N} \left[1 + \overline{K}_i(K-1)\right] \cdot x} = \frac{K}{(N+K-1)\cdot x}. \tag{9.42}$$

Equation (9.42) can be used to determine the throughput bounds of any type of product-form queueing network. For this computation, let $x_{\min}$ and $x_{\max}$,

respectively, denote the minimum and maximum relative utilization of an individual node in the network. The basic idea of the BJB analysis is to enclose the considered network by two adjoining balanced networks:

1. An "optimistic" network, where the same relative utilization $x_{\min}$ is assumed for all nodes,
2. A "pessimistic" network, where the same relative utilization $x_{\max}$ is assumed for all nodes.

With Eq. (9.42), the BJB bounds for the throughputs can be determined as:

$$\frac{K}{(N+K-1)} \cdot \frac{1}{x_{\max}} \leq \lambda(K) \leq \frac{K}{(N+K-1)} \cdot \frac{1}{x_{\min}}. \tag{9.43}$$

Similarly, the bounds for the mean response time can be determined by using Little's theorem:

$$(N+K-1) \cdot x_{\min} \leq \overline{T}(K) \leq (N+K-1) \cdot x_{\max}. \tag{9.44}$$

The optimistic bounds can be improved by observing that among all the networks with relative overall utilization x_{sum}, the one with the relative utilization, $x_i = x_{\text{sum}}/N$ for all i, has the highest throughput. If $x_{\text{ave}} = x_{\text{sum}}/N$ denotes the average relative utilization of the individual nodes in the network, then the improved optimistic bounds are given by:

$$\lambda(K) \leq \frac{K}{K+N-1} \cdot \frac{1}{x_{\text{ave}}} = \frac{K}{x_{\text{sum}} + (K-1)x_{\text{ave}}} \tag{9.45}$$

and:

$$x_{\text{sum}} + (K-1)x_{\text{ave}} \leq \overline{T}(K), \tag{9.46}$$

respectively. The optimistic network, the network with the highest throughput, has, therefore, $N = x_{\text{sum}}/x_{\text{ave}}$ nodes with relative utilization x_{ave}. Analogously, among all networks with relative overall utilization x_{sum} and maximum relative utilization $x_{\max}$, the network with the lowest throughput has altogether $x_{\text{sum}}/x_{\max}$ nodes with relative overall utilization $x_{\max}$. Therefore the improved pessimistic bounds are given by:

$$\frac{K}{K + \frac{x_{\text{sum}}}{x_{\max}} - 1} \cdot \frac{1}{x_{\max}} = \frac{K}{x_{\text{sum}} + (K-1)x_{\max}} \leq \lambda(K) \tag{9.47}$$

and:

$$\overline{T}(K) \leq x_{\text{sum}} + (K-1)x_{\max}, \tag{9.48}$$

respectively.

Table 9.12 Summary of the BJB bounds [LZGS84]

	Network Type	BJB Bounds
λ	A	$\frac{K}{x_{\text{sum}} + (K-1)x_{\max}} \le \lambda(K) \le \frac{K}{x_{\text{sum}} + (K-1)x_{\text{ave}}}$
	B	$\frac{K}{x_{\text{sum}} + Z + \frac{(K-1)x_{\max}}{1+\frac{Z}{K \cdot x_{\text{sum}}}}} \le \lambda(K) \le \frac{K}{x_{\text{sum}} + Z + \frac{(K-1)x_{\text{ave}}}{1+\frac{Z}{x_{\text{sum}}}}}$
	C	$\lambda \le \frac{1}{x_{\max}}$
$\overline{T}$	A	$x_{\text{sum}} + (K-1)x_{\text{ave}} \le \overline{T}(K) \le x_{\text{sum}} + (K-1)x_{\max}$
	B	$x_{\text{sum}} + \frac{(K-1)x_{\text{ave}}}{1+\frac{Z}{x_{\text{sum}}}} \le \overline{T}(K) \le x_{\text{sum}} + \frac{(K-1)x_{\max}}{1+\frac{Z}{K \cdot x_{\text{sum}}}}$
	C	$\frac{x_{\text{sum}}}{1-\lambda x_{\text{ave}}} \le \overline{T} \le \frac{x_{\text{sum}}}{1-\lambda x_{\max}}$

The inequalities for open networks can be obtained in a similar way. In Table 9.12, the BJB bounds for all three network types are summarized. The equations for the two closed network types are identical, if the value of Z is set to zero.

Example 9.7 Consider the network of Example 9.6 again, but now we determine the bounds with the BJB method. With $x_{\text{ave}} = x_{\text{sum}}/N = 8.6$, we can obtain the bounds directly from Table 9.12.

Throughput:

$$\frac{20}{17.2 + 120 + 129.5} \le \lambda(K) \le \frac{20}{17.2 + 120 + 20.5},$$
$$\underline{0.075} \le \lambda(K) \le \underline{0.127}.$$

Response time:

$$17.2 + \frac{163.4}{7.977} \le \overline{T}(K) \le 17.2 + \frac{174.8}{1.349},$$
$$\underline{37.70} \le \overline{T}(K) \le \underline{146.8}.$$

A comparison of the results of both methods shows that for this example we get better results with the ABA method than with the BJB method. Therefore the conclusion is irresistible that using both methods in combination will reduce the area where the performance measures can lie. Figures 9.3 and 9.4 show the results of the ABA and BJB values for the throughput and mean response time as functions of the number of jobs in the network, respectively. The point of intersection for the two ABA bounds is given by:

$$K^* = \frac{x_{\text{sum}} + Z}{x_{\max}}.$$

From the point:

$$K^{+} = \frac{(x_{\text{sum}} + Z)^2 - x_{\text{sum}} x_{\text{ave}}}{(x_{\text{sum}} + Z) \cdot x_{\max} - x_{\text{sum}} x_{\text{ave}}}$$

upwards, the optimistic curve of the ABA method gives better results than the BJB one. These bounds say that the exact values for the throughput and the mean response time reside in the shaded area.

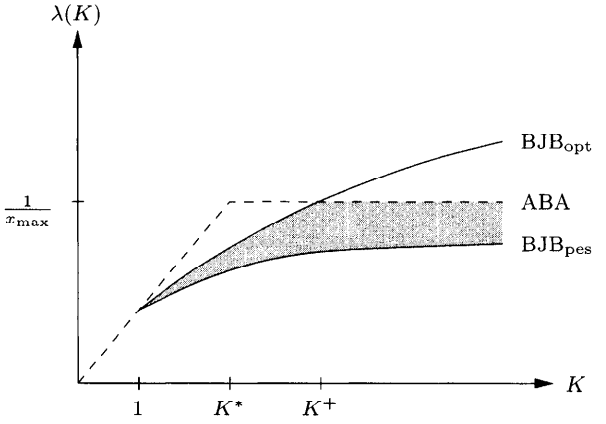


Fig. 9.3 ABA and BJB throughput bounds as a function of K.

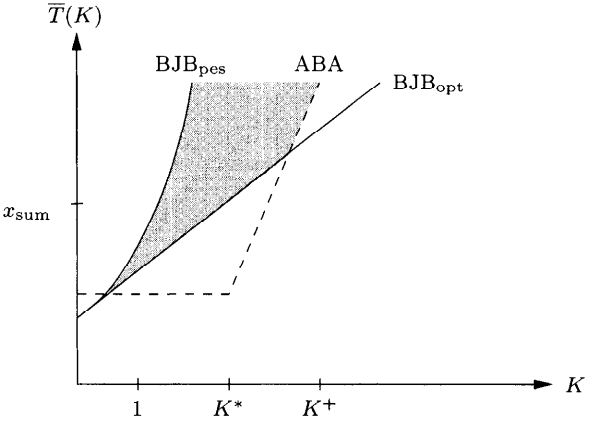


Fig. 9.4 ABA and BJB response time bounds as a function of K.

The *performance bound hierarchy method* for determing the bounds of the performance measures was developed by [EaSe83]. In this method we have a hierarchy of upper and lower bounds for the throughput. These bounds converge to the exact solution of the network. The method is based on the

MVA. Thus, there is a direct relation between the accuracy of the bounds and the cost of computation. If the number of iterations is high, then the cost is higher but the bounds are closer together. On the other hand, when the number of iterations is lower, the cost is lower but the bounds are farther apart. The extension of this method to multiclass networks can be found in [EaSe86]. Other versions can be found in [McMi84], [Kero86], [HsLa87], and [ShYa88].

Problem 9.5 Solve Problem 7.3 with the ABA and BJB methods and compare the results.

9.5 NETWORKS WITH VARIABILITIES IN WORKLOAD

Performance models need values of input parameters in order to predict values of performance measures. In practice, input parameters are estimated from measurements on real systems. Thus each input parameter will have a confidence interval in addition to a point estimate. Traditional algorithms such as MVA or SCAT accept only single values as input parameters and compute single values for the performance measures. In this section we describe a method that allows intervals as input parameters. As a result, the predicted performance measures are also given as intervals. The treatment here is adapted from [HLM96] and [MaRa95].

We can describe the workload variabilities, e.g., service rate variabilities, by intervals:

$$I = [i^-, i^+]. \tag{9.49}$$

If the performance measures are monotone with respect to the input parameters, given by intervals, then a simple interval based algorithm can be derived from a given single value algorithm $y = y(\mu_1, \mu_2, \ldots, \mu_N)$, where y is the considered performance measure and the service rates are given by intervals:

$$I_{\mu_n} = [\mu_n^-, \mu_n^+] \quad n = 1, \ldots, N. \tag{9.50}$$

Now the interval based algorithm can be described as follows:

STEP 1 Calculate the performance measures y for the lower and upper bounds of the service rates:

$$y^- = y(\mu_1^-, \ldots, \mu_N^-), \tag{9.51}$$

$$y^+ = y(\mu_1^+, \ldots, \mu_N^+). \tag{9.52}$$

STEP 2 Calculate the intervals for the performance measure y. If y is monotonically increasing in the parameter, then:

$$I_y = [y^-, y^+]. \tag{9.53}$$

If y is monotonically decreasing, then:

$$I_y = [y^+, y^-]. \tag{9.54}$$

Example 9.8 Consider a closed queueing network with two nodes and $e_1 = e_2 = 1$, $K = 5$, and the two intervals:

$$I_{\mu_1} = [1/\text{sec}, 2/\text{sec}] \quad I_{\mu_2} = [3/\text{sec}, 4/\text{sec}]. \tag{9.55}$$

The interesting performance measures are the throughput $\lambda_1 = \lambda_2 = \lambda$, which is monotonically increasing with service rates and the mean response time $\overline{T}$, which is monotonically decreasing with service rates.

STEP 1

$$\lambda^- = \lambda(1/\text{sec}, 3/\text{sec}) = 0.99/\text{sec}, \tag{9.56}$$

$$\lambda^+ = \lambda(2/\text{sec}, 4/\text{sec}) = 1.97/\text{sec}, \tag{9.57}$$

$$T^- = T(1/\text{sec}, 3/\text{sec}) = 5.01\,\text{sec}, \tag{9.58}$$

$$T^+ = T(2/\text{sec}, 4/\text{sec}) = 2.54\,\text{sec}. \tag{9.59}$$

STEP 2 Interval for the throughput λ:

$$I_\lambda = [\lambda^-, \lambda^+] = [0.99/\text{sec}, 1.91/\text{sec}]. \tag{9.60}$$

Interval for the mean response time $\overline{T}$:

$$I_{\overline{T}} = [\overline{T}^+, \overline{T}^-] = [2.54\,\text{sec}, 5.01\,\text{sec}]. \tag{9.61}$$

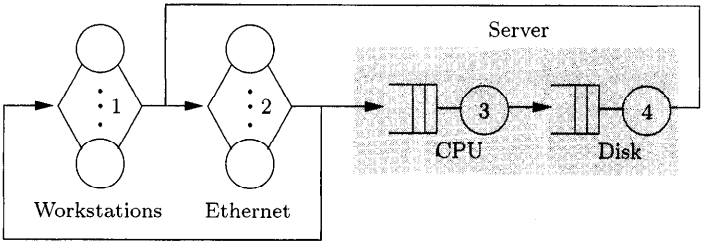


Fig. 9.5 Model of the client-server system.

Example 9.9 As another example, we consider a client-server system where the client workstations (labeled as node 1) submit SQL requests to a server, consisting of a CPU and n disk devices (labeled as nodes 3 and 4) [HLM96]. The workstations are modeled as -/G/1-IS station, while the CPU and disk device are modeled as -/M/1-FCFS stations. The ethernet (labeled as node 2), which connects the workstations with the server, can approximately be modeled by a -/G/1-IS server. The resulting network is shown in Fig. 9.5.

The parameters of the model are listed in Table 9.13. Note that workstation service rate is specified as three intervals each having a given probability. For further details see [HLM96].

Table 9.13 Parameters of the client-server system model

Node	Service Time	Visit Ratios
Workstation	$I_1 = [2.3, 2.7] : p(I_1) = 0.2$ $I_2 = [7.4, 8.4] : p(I_2) = 0.5$ $I_3 = [16, 21] : p(I_2) = 0.3$	1.0
Ethernet	0.00185 sec	2.0
CPU	0.12 sec	1.0
Disk	0.054 sec	1.0

9.6 SUMMARY

The main advantages and disadvantages of the approximation algorithms for product-form queueing networks described in this chapter are summarized in Table 9.14.

Table 9.14 Comparison of the approximation algorithms for product-form queueing networks.

Algorithms	Advantages	Disadvantages
Bard-Schweitzer (BS)	Very low storage and time requirement	No multiple server nodes Low accuracy
SCAT	Good accuracy Very low storage requirement compared with MVA or convolution	Needs more iterations than BS
SUM	Easy to understand and implement Low storage and time requirement Easy to extend to non-product-form networks	Accuracy is not very high (but sufficient for most applications)
BOTT	The same advantages as SUM Fewer iterations than SUM For multiple class networks, easier to implement than SUM Accuracy slightly better than SUM	Accuracy is not very high (but sufficient for most applications)
ABA, BJB	Well suited for a bottleneck analysis In the design phase, to obtain a rough prediction (insight, understanding) of the performance of a system Extremely low storage and time requirement	Only for single class networks Only upper and lower bounds

10

Algorithms for Non-Product-Form Networks

Although many algorithms are available for solving product-form queueing networks (see Chapters 8 and 9), most practical queueing problems lead to non-product-form networks. If the network is Markovian (or can be Markovized), automated generation and solution of the underlying CTMC via *stochastic Petri nets* (SPNs) is an option provided the number of states is fewer than a million. Instead of the costly alternative of a discrete-event simulation, approximate solution may be considered. Many approximation methods for non-product-form networks are discussed in this chapter. These algorithms and corresponding sections of this chapter are laid out as shown in the flowchart of Fig. 10.1. Networks with non-exponentially distributed service times are treated in the next section, while networks with FCFS nodes having different service times for different classes are treated in Section 10.2. Priority queueing networks and networks with simultaneous resource possession are treated in Sections 10.3 and 10.4, respectively. Network models of programs with internal concurrency are treated in Section 10.5. Fork-join systems and parallel processing are treated in Section 10.6. Networks with asymmetric server nodes and blocking networks are discussed in Section 10.7 and Section 10.8, respectively.

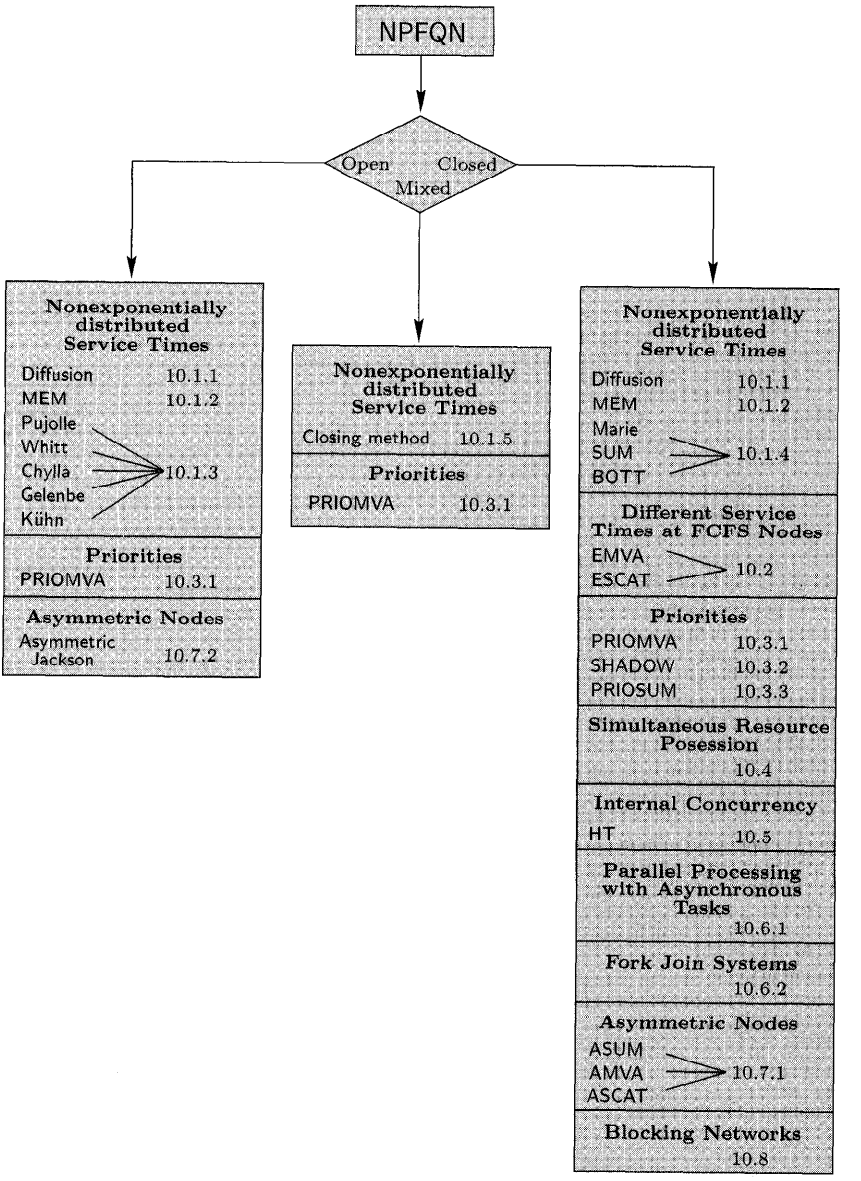


Fig. 10.1 Different algorithms for non-product-form networks and corresponding sections.

10.1 NON-EXPONENTIAL DISTRIBUTIONS

10.1.1 Diffusion Approximation

Although we present diffusion approximation as a method to deal with networks with non-exponential service time and interarrival time distributions, it is possible to apply this technique to Markovian networks as well. The diffusion approximation is a technique based on an approximate product-form solution. In this approximation, the discrete state stochastic process $\{K_i(t) \mid t \geq 0\}$ describing the number of jobs at the ith node at time t is approximated by a continuous state stochastic process (diffusion process) $\{X_i(t) \mid t \geq 0\}$. For the fluctuation of jobs in a time interval we assume a normal distribution. In the steady-state, the density function $f_i(x)$ of the continuous state process can be shown to satisfy the *Fokker-Planck equation* [Koba74] assuming certain boundary conditions. A discretization of the density function gives the approximated product-form-like state probabilities $\hat{\pi}_i(k_i)$ for node i (see Fig. 10.2).

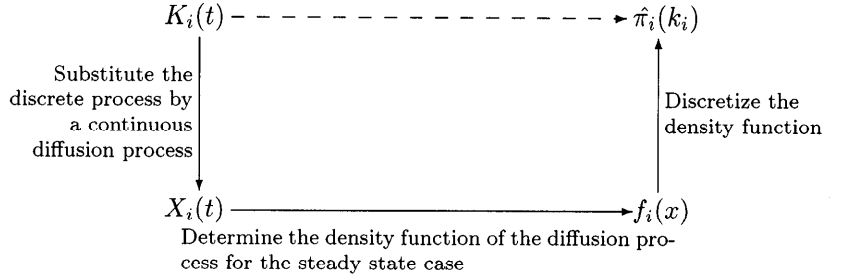


Fig. 10.2 The principle of the diffusion approximation.

Although the derivation of this method is very complex, the method itself is very simple to apply to a given problem. Steady-state behavior of networks with generally distributed service and interarrival times at nodes can be approximately solved with the restriction of a single single server at each node. At present no solutions are available for multiple class networks [Mitz97].

Consider a GI/G/1 queueing system with arrival rate λ, service rate μ, and the coefficients of variation c_A and c_B of the interarrival and service times, respectively. Using the diffusion approximation, the following approximated state probabilities can be obtained [Koba74, ReKo74]:

$$\hat{\pi}(k) = \begin{cases} 1-\rho, & k = 0, \\ \rho(1-\hat{\rho})\hat{\rho}^{k-1}, & k > 0, \end{cases} \tag{10.1}$$

with:

$$\hat{\rho} = \exp\left(\frac{-2(1-\rho)}{c_B^2 + \rho c_A^2}\right) = \exp(\gamma), \tag{10.2}$$

and $\rho = \lambda/\mu < 1$. Note that γ is defined in Eq. (10.2) for convenience so that $\hat{\rho} = \exp(\gamma)$. For the mean number of jobs we then have:

$$\overline{K} = \sum_{k=1}^{\infty} k \cdot \hat{\pi}(k) = \frac{\rho}{1-\hat{\rho}}. \tag{10.3}$$

Differential equations underlying the diffusion approximation need boundary conditions for their solution. Different assumptions regarding these boundary conditions lead to different expressions. The preceding expressions are based on the work of [Koba74] and [ReKo74]. Using different boundary conditions, [Gele75] and [Mitz97] get different results that are more accurate than those of [Koba74] and [ReKo74] for larger values of utilization ρ. They [Gele75], [Mitz97] derived the following expression for the approximated state probabilities:

$$\hat{\pi}(k) = \begin{cases} 1-\rho, & k = 0, \\ \rho\left[1 - \dfrac{1}{\gamma}(\hat{\rho}-1)\right], & k = 1, \\ -\dfrac{\rho}{\gamma\hat{\rho}^2}(1-\hat{\rho})^2\hat{\rho}^k, & k \geq 2, \end{cases} \tag{10.4}$$

with $\hat{\rho}$ and γ as in Eq. (10.2). The mean number of jobs is then:

$$\overline{K} = \rho\left[1 + \frac{\rho c_A^2 + c_b^2}{2(1-\rho)}\right]. \tag{10.5}$$

Next we show how the diffusion approximation can be applied to queueing networks.

10.1.1.1 Open Networks We can make use of the results for the GI/G/1 system for each node in the network, provided that we can approximate the coefficient of variation of interarrival times at each node. We assume the following:

- The external arrival process can be any renewal process with mean interarrival time $1/\lambda$ and coefficient of variation c_A.
- The service times at node i can have any distribution with mean service time $1/\mu_i$ and coefficient of variation c_{B_i}.
- All nodes in the network are single server with FCFS service strategy.

According to [ReKo74] the diffusion approximation for the approximated state probabilities of the network have a product-form solution:

$$\hat{\pi}(k_1, \ldots, k_N) = \prod_{i=1}^{N} \hat{\pi}_i(k_i), \tag{10.6}$$

with the approximate marginal probabilities as in Eq. (10.1):

$$\hat{\pi}_i(k_i) = \begin{cases} 1 - \rho_i, & k_i = 0, \\ \rho_i(1 - \hat{\rho}_i)\hat{\rho}_i^{k_i - 1}, & k \geq 1, \end{cases} \tag{10.7}$$

with

$$\rho_i = \frac{\lambda \cdot e_i}{\mu_i}, \tag{10.8}$$

$$\hat{\rho}_i = \exp\left(-\frac{2(1 - \rho_i)}{c_{Ai}^2 \cdot \rho_i + c_{Bi}^2}\right). \tag{10.9}$$

We approximate the squared coefficient of variation of interarrival times at node i using the following expression:

$$c_{Ai}^2 = 1 + \sum_{j=0}^{N} (c_{Bj}^2 - 1) \cdot p_{ji}^2 \cdot e_j \cdot e_i^{-1}, \tag{10.10}$$

where we set:

$$c_{B0}^2 = c_A^2. \tag{10.11}$$

For the mean number of jobs at node i we get:

$$\overline{K}_i = \sum_{k_i=1}^{\infty} k_i \cdot \hat{\pi}(k_i) = \frac{\rho_i}{1 - \hat{\rho}_i}. \tag{10.12}$$

Similar results are presented in [Gele75], [GeMi80], and [Koba74].

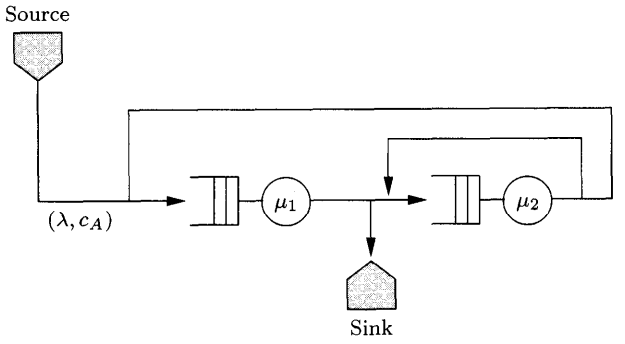


Fig. 10.3 A simple open queueing network.

Example 10.1 In this example (see Fig. 10.3), we show how to use the diffusion approximation for a simple open network with $N = 2$ stations. The

external arrival process has mean interarrival time $1/\lambda = 2.0$ and the squared coefficient of variation $c_A^2 = 0.94$. The service times at the two stations have the following parameters:

$$\mu_1 = 1.1, \quad \mu_2 = 1.2, \quad c_{B1}^2 = 0.5, \quad c_{B2}^2 = 0.8.$$

With routing probabilities $p_{10} = p_{12} = p_{21} = p_{22} = 0.5$ and $p_{01} = 1$, we compute the visit ratios, using Eq. (7.4):

$$e_1 = p_{01} + e_2 \cdot p_{21} = \underline{2}, \quad e_2 = e_1 \cdot p_{12} + e_2 \cdot p_{22} = \underline{2}.$$

Node utilizations are determined using Eq. (10.8):

$$\rho_1 = \frac{\lambda \cdot e_1}{\mu_1} = \underline{0.909}, \quad \rho_2 = \frac{\lambda \cdot e_2}{\mu_2} = \underline{0.833}.$$

We approximate the coefficients of variation of the interarrival time at both the nodes using Eq. (10.10):

$$\begin{aligned} c_{A1}^2 &= 1 + \sum_{j=0}^{2} (c_{Bj}^2 - 1) p_{j1}^2 \cdot \frac{e_j}{e_1} \\ &= 1 + (c_A^2 - 1) p_{01}^2 \cdot \frac{e_0}{e_1} + (c_{B1}^2 - 1) p_{11}^2 \cdot \frac{e_1}{e_1} + (c_{B2}^2 - 1) p_{21}^2 \cdot \frac{e_2}{e_1} \\ &= \underline{0.920}, \\ c_{A2}^2 &= 1 + \sum_{j=0}^{2} (c_{Bj}^2 - 1) p_{j2}^2 \cdot \frac{e_j}{e_2} = \underline{0.825}. \end{aligned}$$

With Eq. (10.9) we get:

$$\begin{aligned} \hat{\rho}_1 &= \exp\left(-\frac{2(1-\rho_1)}{c_{A1}^2 \cdot \rho_1 + c_{B1}^2} \right) = \underline{0.873}, \\ \hat{\rho}_2 &= \exp\left(-\frac{2(1-\rho_2)}{c_{A2}^2 \cdot \rho_2 + c_{B2}^2} \right) = \underline{0.799}. \end{aligned}$$

For the mean number of jobs $\overline{K}_i$, we use Eq. (10.12) and obtain:

$$\overline{K}_1 = \frac{\rho_1}{1 - \hat{\rho}_1} = \underline{7.147}, \quad \overline{K}_2 = \frac{\rho_2}{1 - \hat{\rho}_2} = \underline{4.151},$$

and the marginal probabilities are given by Eq. (10.7):

$$\begin{aligned} \hat{\pi}_1(0) = 1 - \rho_1 = \underline{0.091}, \quad \hat{\pi}_1(k) &= \rho_1 (1 - \hat{\rho}_1) \hat{\rho}_1^{k-1} \\ &= 0.116 \cdot 0.873^{k-1} \quad \text{for } k > 0, \\ \hat{\pi}_2(0) = 1 - \rho_2 = \underline{0.167}, \quad \hat{\pi}_2(k) &= \rho_2 (1 - \hat{\rho}_2) \hat{\rho}_2^{k-1} \\ &= 0.167 \cdot 0.799^{k-1} \quad \text{for } k > 0. \end{aligned}$$

10.1.1.2 Closed Networks To apply the diffusion approximation to closed queueing networks with arbitrary service time distributions, we can use Eq. (10.7) to approximate marginal probabilities $\hat{\pi}_i(k_i)$, provided that the throughputs λ_i and utilizations ρ_i are known. In [ReKo74] the following two suggestions are made to estimate λ_i:

1. For large values of K we use the bottleneck analysis: Search for the bottleneck node bott (node with the highest relative utilization), set its utilization to 1, and determine the overall throughput λ of the network using Eq. (10.8). With this throughput and the visit ratios e_i, compute the utilization of each node. The mean number of jobs is determined using Eq. (10.12).

2. If there is no bottleneck in the network and/or the number K of jobs in the network is small, replace the given network by a product-form one with the same service rates μ_i, routing probabilities p_{ij}, and the same K, and compute the utilizations ρ_i. The approximated marginal probabilities given by Eq. (10.7) are then used to determine the approximated state probabilities:

$$\hat{\pi}(k_1, k_2, \ldots, k_N) = \frac{1}{G} \prod_{i=1}^{N} \hat{\pi}(k_i), \tag{10.13}$$

where G is the normalizing constant of the network. Then for the marginal probabilities and the other performance measures improved values can be determined.

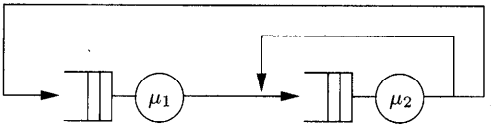


Fig. 10.4 A simple closed queueing network.

Example 10.2 Consider the closed network shown in Fig. 10.4 with the following parameters:

$$\mu_1 = 1.1, \quad \mu_2 = 1.2, \quad c_{B1}^2 = 0.5, \quad c_{B2}^2 = 0.8,$$

and the routing probabilities:

$$p_{12} = 1 \quad \text{and} \quad p_{21} = p_{22} = 0.5.$$

For the visit ratios we use Eq. (7.5) and get:

$$e_1 = 1, \quad e_2 = 2.$$

At first we choose $K = 6$ and use the bottleneck analysis to study the network.

10.1.1.2.1 Solution with Bottleneck Analysis Let node bott be the bottleneck with the highest relative utilization:

$$\rho_{\text{bott}} = \max_i \{\rho_i\} = \lambda \cdot \max_{1,2} \left\{ \frac{e_1}{\mu_1}, \frac{e_2}{\mu_2} \right\} = \lambda \cdot \max\{0.909, 1.667\},$$

which means that node 2 is the bottleneck and therefore its utilization is set to 1:

$$\rho_2 = 1.$$

Using Eq. (10.8) we determine the overall throughput λ of the network:

$$\lambda = \rho_2 \cdot \frac{\mu_2}{e_2} = \frac{1}{1.667} = \underline{0.6}.$$

For the utilization of node 1 we get:

$$\rho_1 = \frac{\lambda \cdot e_1}{\mu_1} = \underline{0.545}.$$

Now we use Eq. (10.10) to determine the coefficients of variation of the interarrival times:

$$c_{A1}^2 = 1 + (c_{B1}^2 - 1)p_{11}^2 \cdot e_1 \cdot e_1^{-1} + (c_{B2}^2 - 1)p_{21}^2 \cdot e_2 \cdot e_1^{-1} = \underline{0.9},$$
$$c_{A2}^2 = 1 + (c_{B1}^2 - 1)p_{12}^2 \cdot e_1 \cdot e_2^{-1} + (c_{B2}^2 - 1)p_{22}^2 \cdot e_2 \cdot e_2^{-1} = \underline{0.7}.$$

The $\hat{\rho}_i$ are given by Eq. (10.9):

$$\hat{\rho}_1 = \exp\left(-\frac{2(1-\rho_1)}{c_{A1}^2 \cdot \rho_1 + c_{B1}^2}\right) = \underline{0.3995}, \quad \hat{\rho}_2 = \exp\left(-\frac{2(1-\rho_2)}{c_{A2}^2 \cdot \rho_2 + c_{B2}^2}\right) = \underline{1}.$$

Now, by using Eq. (10.12) we can compute the approximate values for the mean number of jobs:

$$\overline{K}_1 = \frac{\rho_1}{1-\hat{\rho}_1} = \underline{0.908}, \quad \overline{K}_2 = K - \overline{K}_1 = \underline{5.092}.$$

Table 10.1 compares the values obtained by the preceding diffusion approximation (DA) with those obtained via DES. We can see that in this case the results match very well. Such close matching does not always occur, especially when one node in the network is highly utilized and the coefficients of variation are very small.

To show the second method, we assume that there are $K = 3$ jobs in the network.

Table 10.1 Comparison of results for the example ($K = 6$)

	ρ_i (DES)	$\overline{K}_i$ (DES)	ρ_i (DA)	$\overline{K}_i$ (DA)
Node 1	0.544	0.965	0.545	0.908
Node 2	0.996	5.036	1	5.092

10.1.1.2.2 Solution Using Product-Form Approximation Now we substitute the given network by a product-form one. To determine the utilizations of this network we can use the MVA and get:

$$\rho_1 = \underline{0.501} \quad \text{and} \quad \rho_2 = \underline{0.919}.$$

The coefficients of variation for the interarrival times remain the same as in the previous method:

$$c_{A1}^2 = \underline{0.9} \quad \text{and} \quad c_{A2}^2 = \underline{0.7}.$$

For the $\hat{\rho}_i$ we use Eq. (10.9) and obtain:

$$\hat{\rho}_1 = \exp\left(-\frac{2(1-\rho_1)}{c_{A_1}^2 \cdot \rho_1 + c_{B_1}^2}\right) = \underline{0.35}, \quad \hat{\rho}_2 = \underline{0.894}.$$

The approximated marginal probabilities can be computed using Eq. (10.7):

$$\begin{aligned}
\hat{\pi}_1(0) &= 1 - \rho_1 = \underline{0.499}, & \hat{\pi}_2(0) &= 1 - \rho_2 = \underline{0.081}, \\
\hat{\pi}_1(1) &= \rho_1(1 - \hat{\rho}_1) = \underline{0.326}, & \hat{\pi}_2(1) &= \rho_2(1 - \hat{\rho}_2) = \underline{0.0974}, \\
\hat{\pi}_1(2) &= \rho_1(1 - \hat{\rho}_1)\hat{\rho}_1 = \underline{0.114}, & \hat{\pi}_2(2) &= \rho_2(1 - \hat{\rho}_2)\hat{\rho}_2 = \underline{0.0871}, \\
\hat{\pi}_1(3) &= \rho_1(1 - \hat{\rho}_1)\hat{\rho}_1^2 = \underline{0.039}, & \hat{\pi}_2(3) &= \rho_2(1 - \hat{\rho}_2)\hat{\rho}_2^2 = \underline{0.0779}.
\end{aligned}$$

Now the following network states are possible:

$$(3,0), \quad (2,1), \quad (1,2), \quad (0,3),$$

whose probability is computed using Eq. (10.13):

$$\begin{aligned}
\hat{\pi}(3,0) &= \hat{\pi}_1(3) \cdot \hat{\pi}_2(0) \cdot \frac{1}{G} = 0.00316 \cdot \frac{1}{G}, \\
\hat{\pi}(2,1) &= 0.01111 \cdot \frac{1}{G}, \\
\hat{\pi}(1,2) &= 0.02839 \cdot \frac{1}{G}, \\
\hat{\pi}(0,3) &= 0.03887 \cdot \frac{1}{G}.
\end{aligned}$$

With the normalizing condition:

$$\hat{\pi}(3,0) + \hat{\pi}(2,1) + \hat{\pi}(1,2) + \hat{\pi}(0,3) = 1,$$

we determine the normalizing constant G:

$$G = 0.00316 + 0.01111 + 0.02839 + 0.03887 = \underline{0.08153},$$

which is used to compute the final values of the state probabilities:

$$\hat{\pi}(3,0) = \underline{0.039}, \quad \hat{\pi}(2,1) = \underline{0.136}, \quad \hat{\pi}(1,2) = \underline{0.348}, \quad \hat{\pi}(0,3) = \underline{0.477},$$

and with these state probabilities the improved values for the marginal probabilities are immediately derived:

$$\pi_1(3) = \pi_2(0) = \underline{0.039}, \quad \pi_1(2) = \pi_2(1) = \underline{0.136},$$
$$\pi_1(1) = \pi_2(2) = \underline{0.348}, \quad \pi_1(0) = \pi_2(3) = \underline{0.477}.$$

For the mean number of jobs we get

$$\overline{K}_1 = \sum_{k=1}^{3} k \cdot \pi_i(k) = \underline{0.737} \quad \text{and} \quad \overline{K}_2 = \underline{2.263}.$$

In Table 10.2 DES values for this example are compared to values from the preceding approximation.

Table 10.2 Comparison of results for the example ($K = 3$).

	ρ_i (DES)	$\overline{K}_i$ (DES)	ρ_i (DA)	$\overline{K}_i$ (DA)
Node 1	0.519	0.773	0.501	0.737
Node 2	0.944	2.229	0.919	2.263

A detailed investigation of the accuracy of the diffusion method can be found in [ReKo74]. The higher the utilization of the nodes and the closer the coefficient of variation is to 1, the more accurate are the results.

10.1.2 Maximum Entropy Method

An iterative method for the approximate analysis of open and closed queueing networks is based on the principle of the maximum entropy. The term *entropy* comes from information theory and is a measure of the uncertainty in the predictability of an event. To explain the principle of the maximum entropy, we consider a simple system that can take on a set of discrete states $\mathbf{S}$. The probabilities $\pi(\mathbf{S})$ for different states are unknown; the only information about the probability vector is the number of side conditions given as mean values of suitable functions. Because in general the number of side conditions is smaller than the number of possible states, in most cases there is an infinite number of probability vectors that satisfy the given side conditions. The question now is

which of these probability vectors shall be chosen as the one best suited for the information given by the side conditions and which is least prejudiced against the missing information. In this case the principle of maximum entropy says that the best-suited probability vector is the one with the largest entropy.

To solve a queueing network using the maximum entropy method (MEM) the steady-state probabilities $\pi(\mathbf{S})$ are determined so that the entropy function

$$II(\pi) = -\sum_{\mathbf{S}} \pi(\mathbf{S}) \ln \pi(\mathbf{S}) \tag{10.14}$$

is maximized subject to given side conditions. The normalizing condition –the sum of all steady-state probabilities is 1–provides another constraint. In [KGTA88], open and closed networks with one or several job classes and -/G/1, -/G/∞ nodes are analyzed using this approach. Queueing disciplines such as FCFS, LCFS, or PS as well as priorities are allowed. An extension to the case of multiple server nodes is given in [KoAl88]. For the sake of simplicity we only consider single class networks with -/G/1-FCFS nodes based on the treatment in [Kouv85] and [Wals85].

10.1.2.1 Open Networks If we maximize the entropy by considering the side conditions for the mean number of jobs $\overline{K}_i$, Eq. (7.18), and the utilization ρ_i, Eq. (7.26), then the following product-form approximation for the steady-state probabilities of open networks can be derived [Kouv85]:

$$\pi(k_1, k_2, \ldots, k_N) = \pi_1(k_1) \cdot \pi_2(k_2) \cdot \ldots \cdot \pi_N(k_N), \tag{10.15}$$

where the marginal probabilities are given by:

$$\pi_i(k_i) = \begin{cases} \dfrac{1}{G_i}, & k_i = 0, \\ \dfrac{1}{G_i} \cdot a_i b_i^{k_i}, & k_i \geq 1, \end{cases} \tag{10.16}$$

and where:

$$a_i = \frac{\rho_i^2}{(1-\rho_i)(\overline{K}_i - \rho_i)}, \tag{10.17}$$

$$b_i = \frac{\overline{K}_i - \rho_i}{\overline{K}_i}, \tag{10.18}$$

$$G_i = \frac{1}{1-\rho_i}. \tag{10.19}$$

To utilize the MEM, we thus need the utilizations ρ_i and the mean number of jobs $\overline{K}_i$ at each node. The equation $\rho_i = \lambda_i/\mu_i$ can be used to easily determine ρ_i from the given network parameters. The mean number of jobs

$\overline{K}_i$ can be approximated as a function of the utilization ρ_i and the squared coefficient of variation of the service and interarrival times [Kouv85]:

$$\overline{K}_i = \frac{\rho_i}{2}\left(1 + \frac{c_{Ai}^2 + \rho_i c_{Bi}^2}{1-\rho_i}\right), \tag{10.20}$$

if the condition $(1 - c_{Ai}^2)/(1 + c_{Bi}^2) \leq \rho_i < 1$ is fulfilled. The computation of the $\overline{K}_i$ is made possible by approximating the squared coefficient of variation c_{Ai}^2 of interarrival times. The following iterative expressions for computing the squared coefficients of variation is provided by [Kouv85]:

$$c_{Ai}^2 = -1 + \left(\sum_{j=0}^{N} \frac{\lambda_j p_{ji}}{\lambda_i \cdot (c_{ji}^2 + 1)}\right)^{-1}, \tag{10.21}$$

$$c_{ji}^2 = 1 + p_{ji}\left(c_{Dj}^2 - 1\right), \tag{10.22}$$

$$c_{Di}^2 = \rho_i(1-\rho_i) + (1-\rho_i)c_{Ai}^2 + \rho_i^2 c_{Bi}^2. \tag{10.23}$$

There are other approximations for the computation of the mean number of jobs $\overline{K}_i$ and for the estimation of the necessary squared coefficients of variation. For more information on these equations, see Section 10.1.3.

With this information, the MEM for the analysis of open queueing networks can be summarized in the following three steps:

STEP 1 Determine the arrival rates λ_i using the traffic equations (Eq. (7.1)) and the utilizations $\rho_i = \lambda_i/\mu_i$ for all nodes $i = 1, \ldots, N$.

STEP 2 Determine the squared coefficients of variation.

STEP 2.1 Initialize. The squared coefficients of variations c_{Ai}^2 of the interarrival times are initially set to one for $i = 1, \ldots, N$.

STEP 2.2 Compute the squared coefficients of variation c_{Di}^2 of the interdeparture times of node i for $i = 1, \ldots, N$ using Eq. (10.23). Compute the squared coefficients of variation c_{ij}^2 of node i for $i = 1, \ldots, N$ and $j = 0, \ldots, N$ using Eq. (10.22). From these the new values of the squared coefficients of variation c_{Ai}^2 of the interarrival times are computed using Eq. (10.21).

STEP 2.3 Check the halting condition. If the old and new values for the c_{Ai}^2 differ by more than ϵ, then go back to Step 2.2 with the new c_{Ai}^2 values.

STEP 3 Determine the performance measures of the network beginning with the mean number of jobs $\overline{K}_i$ for $i = 1, \ldots, N$ using Eq. (10.20) and then compute the maximum entropy solutions of the steady-state probabilities using Eq. (10.15).

The MEM normally approximates the coefficients of variation less accurately than the method of Kühn (see Section 10.1.3) and therefore the method of Kühn is generally preferred for open networks with a single job class. The MEM is mainly used for closed networks.

10.1.2.2 Closed Networks For closed networks that consist only of -/G/1-FCFS nodes, the following product-form formulation for the approximated steady-state probabilities can be given if we maximize the entropy under the side conditions on the mean number of jobs $\overline{K}_i$ and the utilization ρ_i:

$$\pi(k_1 \ldots, k_N) = \frac{1}{G(K)} \cdot F_1(k_1) \cdot \ldots \cdot F_N(k_N), \tag{10.24}$$

with:

$$F_i(k_i) = \begin{cases} 1, & k_i = 0, \\ a_i b_i^{k_i}, & 1 \leq k_i \leq K, \end{cases} \tag{10.25}$$

and:

$$G(K) = \sum_{\sum_{i=1}^{N} k_i = K} F_1(k_1) \cdot \ldots \cdot F_N(k_N). \tag{10.26}$$

As in the case of open networks, coefficients a_i and b_i are respectively given by Eqs. (10.17) and (10.18). The computation of the steady-state probabilities of closed networks is more difficult than that for open networks as the ρ_i and $\overline{K}_i$ values of the individual nodes cannot be determined directly. Therefore we use the following trick: We construct a *pseudo open network* with the same number of nodes and servers as well as identical service time distribution and routing probabilities as the original closed network. The external arrival rates of this open network are determined so that resulting average number of jobs in the pseudo open network equals the given number of jobs K in the closed network:

$$\sum_{i=1}^{N} \overline{K}_i^* = K, \tag{10.27}$$

where $\overline{K}_i^*$ denotes the mean number of jobs at the ith node in the pseudo open network. The pseudo open network arrival rate λ is determined iteratively by using Eqs. (10.20) and (10.27), assuming that the stability condition

$$\max_i \left\{ \frac{\lambda e_i}{\mu_i} \right\} < 1$$

is satisfied. Here e_i is the relative throughput of node i.

Then the performance measures $\overline{K}_i^*$ and ρ_i^* of the pseudo open network can be computed using the maximum entropy method for open networks as given in the last algorithm. These performance measures are then used to determine the coefficients a_i and b_i and steady-state probabilities of the pseudo open network. To compute the probability vector of the original closed network, we use the convolution method (see Section 8.1). By using the functions $F_i(k_i)$,

Eq. (10.25), the normalizing constants as well as the utilizations and mean number of jobs at each node of the closed network can be computed. This approximation uses values for the coefficients a_i and b_i, which are computed for the pseudo open network. To compensate the error in the approximation, we apply the work rate theorem (Eq. (7.7)), which gives us an iterative method for the computation of the coefficients a_i:

$$a_i^{(n+1)} = a_i^{(n)} \cdot K \cdot \rho_i^* \cdot \left(\rho_i \cdot \sum_{j=1}^{N} \frac{\overline{K}_j \cdot \rho_j^*}{\rho_j} \right)^{-1} \tag{10.28}$$

with the initial values:

$$a_i^{(0)} = \frac{\rho_i^*}{1-\rho_i^*} \cdot \frac{\rho_i^*}{\overline{K}_i^* - \rho_i^*}. \tag{10.29}$$

For the coefficients b_i we have:

$$b_i = \frac{\overline{K}_i^* - \rho_i^*}{\overline{K}_i^*}. \tag{10.30}$$

The asterisk denotes the performance measures of the pseudo open network.

The algorithm can now be given in the following six steps.

STEP 1 Compute the visit ratios e_i for $i = 1, \ldots, N$, using Eq. (7.2) and the relative utilizations $x_i = e_i/\mu_i$.

STEP 2 Construct and solve the pseudo open network.

STEP 2.1 Initialize. Initial value for the squared coefficients of the inter-arrival times at node $i = 1, \ldots, N$:

$$c_{Ai}^2 = 1.$$

Initial value for the arrival rate of the pseudo open network:

$$\lambda = \frac{0.99}{x_{\text{bott}}},$$

where $x_{\text{bott}} = \max_i \{x_i\}$ is the relative utilization of the bottleneck node.

STEP 2.2 Compute λ from Condition (10.27) by using Eq. (10.20). Thus, λ can be determined as solution of the equation:

$$\sum_{i=1}^{N} \frac{\lambda \cdot x_i}{2} \left(1 + \frac{c_{Ai}^2 + \lambda \cdot x_i \cdot c_{Bi}^2}{1 - \lambda \cdot x_i} \right) - K = 0, \tag{10.31}$$

if $(1 - c_{Ai}^2)/(1 + c_{Bi}^2) \leq \lambda \cdot x_i < 1$ for all i is fulfilled. To solve the non-linear equation, we use the Newton-Raphson method.

STEP 2.3 Compute for $i, j = 1, \ldots, N$ the squared coefficients of variation c_{Di}^2, c_{ij}^2 and c_{Ai}^2 using the Eqs. (10.21)-(10.23). For this computation we need the values $\rho_i = \lambda \cdot x_i$ and $\lambda_i = \lambda \cdot e_i$.

STEP 2.4 Check the halting condition. If the new values for the c_{Ai}^2 differ less than ϵ from the old values, substitute them by the new values and return to Step 2.2.

STEP 3 Determine for $i = 1, \ldots, N$ the utilizations $\rho_i^* = (\lambda \cdot e_i)/\mu_i$, and with Eq. (10.20) the mean number of jobs $\overline{K}_i^*$ in the pseudo open network.

STEP 4 Determine the coefficients a_i and b_i of the pseudo open network using Eqs. (10.29) and (10.30).

STEP 5 Solve the closed network.

STEP 5.1 Use the convolution method to determine the normalizing constant $G(K)$ of the network. The functions $F_i(k_i)$ are given by Eq. (10.25).

STEP 5.2 Determine the performance parameters ρ_i and $\overline{K}_i$ of the closed network. For this we use the normalizing constant $G_N^{(i)}(k)$ as defined in Eq. (8.8). From Eq. (8.7) we have $\pi_i(k) = F_i(k) \cdot G_N^{(i)}(K-k)/G(k)$ and therefore:

$$\rho_i = 1 - \pi_i(0) = 1 - \frac{G_N^{(i)}(K)}{G(K)}, \tag{10.32}$$

$$\overline{K}_i = \sum_{k=1}^{K} k \cdot \frac{F_i(k)}{G(K)} \cdot G_N^{(i)}(K-k). \tag{10.33}$$

STEP 5.3 Use Eq. (10.28) to determine a new value for the coefficients a_i. If there is a big difference between the old values and the new values, then return to Step 5.1.

STEP 6 Use Eq. (10.24) to determine the maximum entropy solution for the steady-state probabilities and other performance parameters of the network as, e.g., $\lambda_i = \mu_i \rho_i$ or $\overline{T}_i = \overline{K}_i/\lambda_i$.

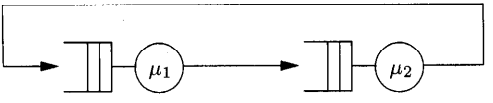


Fig. 10.5 Closed network.

Example 10.3 The maximum entropy method is now illustrated on a simple network with $N = 2$ nodes and $K = 3$ jobs (see Fig. 10.5). The queueing

discipline at all nodes is FCFS and the service rates and squared coefficients of variation are given as follows:

$$\mu_1 = 1, \quad \mu_2 = 2, \qquad \text{and} \qquad c_{B1}^2 = 5, \quad c_{B2}^2 = 0.5.$$

The analysis of the network is carried out in the following steps. As condition for the termination of iterations we choose $\epsilon = 0.001$.

STEP 1 Determine the visit ratios and relative utilizations:

$$e_1 = e_2 = \underline{1}, \qquad \text{and} \qquad x_1 = \underline{1}, \quad x_2 = \underline{0.5}.$$

STEP 2 Construct and solve the pseudo open network.

STEP 2.1 Initialize:

$$c_{A1}^2 = c_{A2}^2 = \underline{1}, \qquad \lambda = 0.99/x_1 = \underline{0.99}.$$

STEP 2.2 Determine λ. We use the Newton-Raphson iteration method to solve the following equation:

$$\sum_{i=1}^{2} \frac{\lambda \cdot x_i}{2} \left(1 + \frac{c_{Ai}^2 + \lambda \cdot x_i \cdot c_{Bi}^2}{1 - \lambda \cdot x_i}\right) - 3 = 0$$

and get $\lambda \approx 0.5558$.

STEP 2.3 Determine the squared coefficients of variation:

$$\begin{aligned}
c_{D1}^2 &= \rho_1(1-\rho_1) + (1-\rho_1)c_{A1}^2 + \rho_1^2 c_{B1}^2 = \underline{2.236}, \\
c_{D2}^2 &= \rho_2(1-\rho_2) + (1-\rho_2)c_{A2}^2 + \rho_2^2 c_{B2}^2 = \underline{0.961}, \\
c_{12}^2 &= 1 + p_{12} \cdot (c_{D1}^2 - 1) = \underline{2.236}, \\
c_{21}^2 &= 1 + p_{21} \cdot (c_{D2}^2 - 1) = \underline{0.961}, \\
c_{A1}^2 &= -1 + \left(\frac{\lambda_2 \cdot p_{21}}{\lambda_1 \cdot (c_{21}^2 + 1)}\right)^{-1} = \underline{0.961}, \\
c_{A2}^2 &= -1 + \left(\frac{\lambda_1 \cdot p_{12}}{\lambda_2 \cdot (c_{12}^2 + 1)}\right)^{-1} = \underline{2.236}.
\end{aligned}$$

STEP 2.4 Check the halting condition:

$$\begin{aligned}
|c_{A1}^{2(\text{new})} - c_{A1}^{2(\text{old})}| &= |0.961 - 1| = 0.039 > \epsilon, \\
|c_{A2}^{2(\text{new})} - c_{A2}^{2(\text{old})}| &= |2.236 - 1| = 1.236 > \epsilon.
\end{aligned}$$

As the halting condition is not fulfilled, we return to Step 2.2 and determine a new value for the constant λ.

After 14 iterations, the halting condition is fulfilled and we get the following results:

$$\lambda = \underline{0.490}, \qquad c_{A1}^2 = \underline{2.130}, \qquad c_{A2}^2 = \underline{2.538}.$$

STEP 3 Determine ρ_i^* and $\overline{K}_i^*$:

$$\rho_1^* = \frac{\lambda \cdot e_1}{\mu_1} = \underline{0.49}, \qquad \rho_2^* = \frac{\lambda \cdot e_2}{\mu_2} = \underline{0.245},$$

$$\overline{K}_1^* = \frac{\rho_1^*}{2}\left(1 + \frac{c_{A1}^2 + \rho_1^* \cdot c_{B1}^2}{1 - \rho_1^*}\right) = \underline{2.446}, \qquad \overline{K}_2^* = \frac{\rho_2^*}{2}\left(1 + \frac{c_{A2}^2 + \rho_2^* \cdot c_{B2}^2}{1 - \rho_2^*}\right) = \underline{0.554}.$$

STEP 4 Determine the coefficients:

$$a_1 = \frac{\rho_1^*}{1 - \rho_1^*} \cdot \frac{\rho_1^*}{\overline{K}_1^* - \rho_1^*} = \underline{0.241}, \qquad a_2 = \frac{\rho_2^*}{1 - \rho_2^*} \cdot \frac{\rho_2^*}{\overline{K}_2^* - \rho_2^*} = \underline{0.257},$$

$$b_1 = \frac{\overline{K}_1^* - \rho_1^*}{\overline{K}_1^*} = \underline{0.8}, \qquad b_2 = \frac{\overline{K}_2^* - \rho_2^*}{\overline{K}_2^*} = \underline{0.558}.$$

STEP 5 Solve the closed network.

STEP 5.1 Determine the normalizing constant using the convolution method (see Section 8.1). For the functions $F_i(k_i), i = 1, 2$, we use Eq. (10.25) and get:

$$F_1(0) = \underline{1}, \qquad F_2(0) = \underline{1},$$
$$F_1(1) = \underline{0.193}, \qquad F_2(1) = \underline{0.144},$$
$$F_1(2) = \underline{0.154}, \qquad F_2(2) = \underline{0.080},$$
$$F_1(3) = \underline{0.123}, \qquad F_2(3) = \underline{0.045}.$$

The procedure to determine the $G(K)$ is similar to the one in Table 8.2 and summarized in the following tabular form:

n \ k	1	2
0	1	1
1	0.193	0.3362
2	0.154	0.2618
3	0.123	0.2054

For the normalizing constant we get $G(K) = \underline{0.2054}$.

STEP 5.2 Determine ρ_i and $\overline{K}_i$. For this we need the normalizing constant $G_N^{(i)}(k)$ of the network with Node i short-circuited. We have only two nodes and therefore it follows immediately that:

$$G_N^{(1)}(k) = F_2(k), \qquad G_N^{(2)}(k) = F_1(k),$$

and, therefore:

$$\rho_1 = 1 - \frac{G_N^{(1)}(K)}{G(K)} = \underline{0.782}, \qquad \rho_2 = 1 - \frac{G_N^{(2)}(K)}{G(K)} = \underline{0.400},$$

$$\overline{K}_1 = \sum_{k=1}^{3} k \cdot \frac{F_1(k)}{G(K)} \cdot G_N^{(1)}(K-k) = \underline{2.090},$$

$$\overline{K}_2 = \sum_{k=1}^{3} k \cdot \frac{F_2(k)}{G(K)} \cdot G_N^{(2)}(K-k) = \underline{0.910}.$$

STEP 5.3 Determine the new coefficients a_i. With Eq. (10.28) we get:

$$a_1^{(\text{new})} = a_1 \cdot K \cdot \rho_1^* \cdot \left(\rho_1 \cdot \sum_{j=1}^{2} \frac{\overline{K}_j \cdot \rho_j^*}{\rho_j} \right)^{-1} = \underline{0.242},$$

$$a_2^{(\text{new})} = a_2 \cdot K \cdot \rho_2^* \cdot \left(\rho_2 \cdot \sum_{j=1}^{2} \frac{\overline{K}_j \cdot \rho_j^*}{\rho_j} \right)^{-1} = \underline{0.253}.$$

Because the old and new values of the a_i differ more than $\epsilon = 0.001$, the analysis of the closed network has to be started again, beginning with Step 5.1.

$$\vdots$$

After altogether three iteration steps the halting condition is fulfilled and we get the following final results:

$$\rho_1 = \underline{0.787}, \quad \rho_2 = \underline{0.394}, \qquad \text{and} \qquad \overline{K}_1 = \underline{2.105}, \quad \overline{K}_2 = \underline{0.895}.$$

STEP 6 Compute all other performance measures:

$$\lambda_1 = \mu_1 \rho_1 = \underline{0.787}, \qquad \lambda_2 = \mu_2 \rho_2 = \underline{0.787},$$
$$\overline{T}_1 = \overline{K}_1 / \lambda_1 = \underline{2.675}, \qquad \overline{T}_2 = \overline{K}_2 / \lambda_2 = \underline{1.137}.$$

To compare the results, we want to give the exact values for the mean number of jobs:

$$\overline{K}_1 = \underline{2.206}, \quad \overline{K}_2 = \underline{0.794}.$$

The method of Marie (see Section 10.1.4.2) gives the following results:

$$\overline{K}_1 = \underline{2.200}, \quad \overline{K}_2 = \underline{0.800}.$$

The closer the service time distribution is to an exponential distribution ($c_{Bi}^2 = 1$), the more accurate are the results of the MEM. For exponentially distributed service and arrival times the results are always exact. Another important characteristic of the MEM is the fact that the computation time is relatively independent of the number of jobs in the network.

10.1.3 Decomposition for Open Networks

This section deals with approximate performance analysis of open non-product-form queueing networks, based on the method of decomposition. The first step is the calculation of the arrival rates and the coefficients of variation of the interarrival times for each node. In the second step, performance measures such as the mean queue length and the mean waiting time are calculated using the GI/G/1 and GI/G/m formulae from Sections 6.12 and 6.14. The methods introduced here are those due to Kühn [Kühn79, Bolc89], Chylla [Chyl86], Pujolle [PuAi86], Whitt [Whit83a, Whit83b] and Gelenbe [GePu87].

Open networks to be analyzed by these techniques must have the following properties:

- The interarrival times and service times are arbitrarily distributed and given by the first and second moments.
- The queueing discipline is FCFS and there is no restriction on the length of the queue.
- The network can have several classes of customers (exception is the method of Kühn).
- The nodes of the network can be of single or multiple server type.
- Class switching is not allowed.

With these prerequisites the method works as follows:

STEP 1 Calculate the arrival rates and the utilizations of the individual nodes using Eqs. (10.35) and (10.36), and (10.37) and (10.38), respectively.

STEP 2 Compute the coefficient of variation of the interarrival times at each node, using Eqs. (10.41)-(10.44).

STEP 3 Compute the mean queue length and other performance measures.

We need the following *fundamental formulae* (Step 1); most of them are from Section 7.1.2:

Arrival rate $\lambda_{ij,r}$ from node i to node j of class r:

$$\lambda_{ij,r} = \lambda_{i,r} \cdot p_{ij,r}. \tag{10.34}$$

Arrival rate $\lambda_{i,r}$ to node i of class r:

$$\lambda_{i,r} = \lambda_{0i,r} + \sum_{j=1}^{N} \lambda_{j,r} \cdot p_{ji,r}. \tag{10.35}$$

Arrival rate λ_i to node i:

$$\lambda_i = \sum_{r=1}^{R} \lambda_{i,r}. \tag{10.36}$$

Utilization $\rho_{i,r}$ of node i due to customers of class r:

$$\rho_{i,r} = \frac{\lambda_{i,r}}{m_i \cdot \mu_{i,r}}. \tag{10.37}$$

Utilization ρ_i of node i:

$$\rho_i = \sum_{r=1}^{R} \rho_{i,r}. \tag{10.38}$$

The mean service rate μ_i of node i:

$$\mu_i = \frac{1}{\sum_{r=1}^{R} \frac{\lambda_{i,r}}{\lambda_i} \cdot \frac{1}{m_i \cdot \mu_{i,r}}}. \tag{10.39}$$

Coefficient of variation c_{Bi} of the service time of node i:

$$c_{Bi}^2 = -1 + \sum_{r=1}^{R} \frac{\lambda_{i,r}}{\lambda_i} \cdot \left(\frac{\mu_i}{m_i \cdot \mu_{i,r}} \right)^2 \cdot (c_{i,r}^2 + 1), \tag{10.40}$$

with coefficient of variation $c_{i,r}$ for the service time of customers of class r at node i.

The calculation of the *coefficient of variation* or interarrival times (Step 2) is done iteratively using the following three phases (see Fig. 10.6):

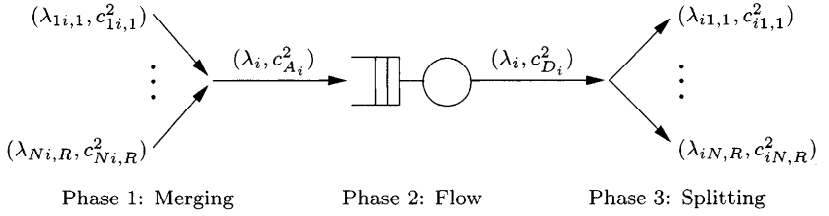


Fig. 10.6 Phases of the calculation of the coefficient of variation.

- Phase 1: Merging

  Several arrival processes to each node are merged into a single arrival process. The arrival rate λ is the sum of the arrival rates of the individual arrival processes. For the coefficient of variation of the interarrival time, the several authors suggest different approximate formulae.

- Phase 2: Flow

  The coefficient of variation c_D of the interdeparture times depends on the coefficients of variation of the interarrival times c_A and the service times c_B. Here again the different authors provide different approximate formulae.

- Phase 3: Splitting

  For the splitting of the departure process, all authors apply the formula:

$$c_{ij,r}^2 = 1 + p_{ij,r} \cdot (c_{Di}^2 - 1). \tag{10.41}$$

The iteration starts with phase 1 and the initial values are: $c_{ij,r} = 1$.

The corresponding formulae for phases 1 and 2 are introduced in the following:

- Decomposition of Pujolle [PuAi86]:

  Merging:

$$c_{Ai,r}^2 = \frac{1}{\lambda_{i,r}} \cdot \left(\sum_{j=1}^{N} c_{ji,r}^2 \cdot \lambda_{j,r} \cdot p_{ji,r} + c_{0i,r}^2 \cdot \lambda_{0,r} \cdot p_{0i,r} \right), \tag{10.42}$$

$$c_{Ai}^2 = \frac{1}{\lambda_i} \cdot \sum_{r=1}^{R} c_{Ai,r}^2 \cdot \lambda_{i,r}. \tag{10.43}$$

  Flow:

$$c_{Di}^2 = \rho_i^2 \cdot (c_{Bi}^2 + 1) + (1 - \rho_i) \cdot c_{Ai}^2 + \rho_i \cdot (1 - 2\rho_i). \tag{10.44}$$

- Decomposition of Whitt [Whit83b, Whit83a]:

  Merging: see Pujolle.

  Flow:

$$c_{Di}^2 = 1 + \frac{\rho_i^2 \cdot (c_{Bi}^2 - 1)}{\sqrt{m_i}} + (1 - \rho_i^2) \cdot (c_{Ai}^2 - 1). \tag{10.45}$$

- Decomposition of Gelenbe [GePu87]:

  Merging: see Pujolle.

  Flow:

$$c_{Di}^2 = -1 + \lambda_i \sum_{r=1}^{R} \frac{\rho_{i,r}}{\mu_{i,r}} \cdot (c_{Bi,r}^2 + 1) + (1 - \rho_i) \cdot (c_{Ai}^2 + 1 + 2 \cdot \rho_i). \tag{10.46}$$

- Decomposition of Chylla [Chyl86]:

  Merging:

$$c_{Ai,r}^2 = 1 + \sum_{j=0}^{N} \frac{\lambda_{j,r} \cdot p_{ji,r}}{\lambda_{i,r}} \cdot (c_{ji,r}^2 - 1), \tag{10.47}$$

$$c_{Ai}^2 = 1 + \sum_{r=1}^{R} \frac{\lambda_{i,r}}{\lambda_i} \cdot (c_{Ai,r}^2 - 1). \tag{10.48}$$

  Flow:

$$c_{Di}^2 = 1 + P_{m_i}^2(\rho_i) \cdot (c_{Bi}^2 - 1) + (1 - P_{m_i}^2(\rho_i)) \cdot (c_{Ai}^2 - 1). \tag{10.49}$$

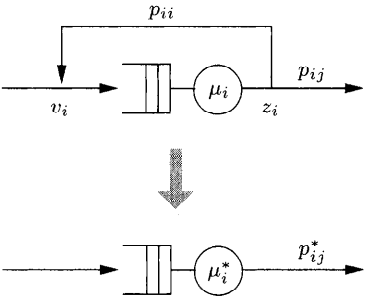


Fig. 10.7 Substitution of a feedback.

- Decomposition of Kühn [Kühn79]:

  Before starting with the iteration in the method of Kühn, nodes with direct feedback are replaced by nodes without feedback (see Fig. 10.7 and Eq. (10.50)).

$$\mu_i^* = \mu_i(1 - p_{ii}), \quad \lambda_i^* = \lambda_i(1 - p_{ii}), \quad c_{Bi}^{2*} = p_{ii} + (1 - p_{ii})c_{Bi}^2,$$
$$p_{ij}^* = \frac{p_{ij}}{1 - p_{ii}} \text{ for } j \neq i, \quad p_{ii}^* = 0. \tag{10.50}$$

  Merging:

$$c_{Ai}^2 = 2\lambda_{1i} \cdot \lambda_{2i} \cdot (\lambda_{1i} + \lambda_{2i}) \cdot (I_1 + I_2 + I_3 + I_4) - 1. \tag{10.51}$$

  Only two processes can be merged by this formula. If there are more processes to merge, then the formula must be used several times. The terms $I_l, l = 1, 2, 3, 4$ are functions of the λ_{ji} and the c_{ji}. For details see [Kühn79] or [Bolc89].

Flow:

$$c_{Di}^2 = c_{Ai}^2 + 2\rho_i^2 c_{Bi}^2 - \rho_i^2 (c_{Ai}^2 + c_{Bi}^2) \cdot G_{KLB}. \tag{10.52}$$

For G_{KLB} see Eqs. (6.72) and (6.92).

To calculate the *mean queue length* (Step 3) and the other performance measures, GI/G/1 and GI/G/m formulae from Sections 6.12 and 6.14 can be used in the case of a single class of customers. Here we give the extension of the most important formulae to the case of multiple classes of customers:

- M/M/m-FCFS:

$$\overline{Q}_{i,r\mathrm{M/M/m}} = \frac{\rho_{i,r}}{1-\rho_i} \cdot P_{mi}, \tag{10.53}$$

with the probability of waiting, P_{mi}, given by Eq. (6.28)

- Allen-Cunneen [Alle90]:

$$\overline{Q}_{i,r\mathrm{AC}} \approx \overline{Q}_{i,r\mathrm{M/M/m}} \cdot \frac{(c_{Ai,r}^2 + c_{Bi,r}^2)}{2}. \tag{10.54}$$

- Krämer/Langenbach-Belz [KrLa76]:

$$\overline{Q}_{i,r\mathrm{KLB}} \approx \overline{Q}_{i,r\mathrm{AC}} \cdot G_{\mathrm{KLB}}, \tag{10.55}$$

with the correction factor:

$$G_{\mathrm{KLB}} = \begin{cases} e^{\left(-\frac{2}{3} \cdot \frac{(1-\rho_i)}{\rho_i} \cdot \frac{(1-c_{Ai,r}^2)^2}{(c_{Ai,r}^2 + c_{Bi,r}^2)}\right)}, & c_{Ai,r}^2 \le 1, \\ e^{-\left((1-\rho_i) \cdot \frac{(c_{Ai,r}^2 - 1)}{(c_{Ai,r}^2 + c_{Bi,r}^2)}\right)}, & c_{Ai,r}^2 > 1. \end{cases} \tag{10.56}$$

- Kimura [Kimu85]:

$$\overline{Q}_{i,r\mathrm{KIM}} \approx \frac{\overline{Q}_{i,r\mathrm{M/M/m}}}{2} \cdot \frac{(c_{Ai,r}^2 + c_{Bi,r}^2)}{\frac{1-c_{Ai,r}^2}{1-4C(m_i,\rho_i)} \cdot e^{\left(-\frac{2}{3}\frac{(1-\rho_i)}{\rho_i}\right)} + \frac{1-c_{Bi,r}^2}{1+C(m_i,\rho_i)} + c_{Ai,r}^2 + c_{Bi,r}^2 - 1}, \tag{10.57}$$

with the coefficient of cooperation:

$$C(m_i,\rho_i) = (1-\rho_i)(m_i-1)\frac{\sqrt{4+5m_i}-2}{16 m_i \rho_i}. \tag{10.58}$$

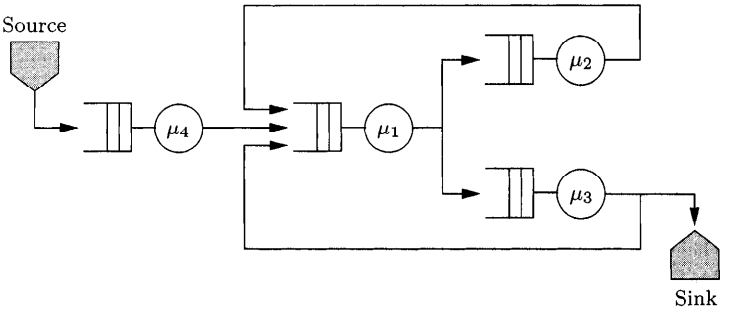


Fig. 10.8 Open non-product-form queueing network.

- Marchal [Marc78]:

$$\overline{Q}_{i,r\mathrm{MAR}} \approx \overline{Q}_{i,r\mathrm{M/M/m}} \frac{(1+c_{Bi,r}^2)\cdot(c_{Ai,r}^2+\rho_i^2 c_{Bi,r}^2)}{2(1+\rho_i^2 c_{Bi,r}^2)} \cdot G_{\mathrm{KLB}}. \quad (10.59)$$

Example 10.4 Consider the open non-product-form queueing network of Fig. 10.8. The routing probabilities of the network are given by:

$$p_{12}=0.5, \quad p_{13}=0.5, \quad p_{31}=0.6, \quad p_{21}=p_{41}=1.$$

Jobs are served at different nodes with rates:

$$\mu_1=25, \quad \mu_2=33.333 \quad \mu_3=16.666, \quad \mu_4=20,$$

and arrive at node 4 with rate $\lambda_{04}=4$. The coefficients of variation of the service times at different nodes as well as of the interarrival times are given by:

$$c_{B_1}^2=2.0, \quad c_{B_2}^2=6.0, \quad c_{B_3}^2=0.5, \quad c_{B_4}^2=0.2, \quad c_{04}^2=4.0.$$

With Eq. (10.35) we can determine the arrival rate at each node:

$$\lambda_1=\underline{20}, \quad \lambda_2=\underline{10}, \quad \lambda_3=\underline{10}, \quad \lambda_4=\underline{4},$$

and using Eq. (10.37) we determine the utilizations of different nodes:

$$\rho_1=\underline{0.8}, \quad \rho_2=\underline{0.3}, \quad \rho_3=\underline{0.6}, \quad \rho_4=\underline{0.2}.$$

The initial values for the coefficients of variation c_{ij} are:

$$c_{12}^2=c_{13}^2=c_{21}^2=c_{31}^2=c_{41}^2=1.$$

Now we start the iteration using the decomposition of Pujolle to obtain the different coefficients of variation of the interarrival times $c_{A_i}^2, i=1,2,3,4$.

Iteration 1:
Merging (Eq. (10.42)):

$$\begin{aligned} c_{A_1}^2 &= \frac{1}{\lambda_1}(c_{21}^2 \lambda_2 p_{21} + c_{31}^2 \lambda_3 p_{31} + c_{41} \lambda_4 p_{41}) \\ &= \frac{1}{20}(1 \cdot 10 \cdot 1 + 1 \cdot 10 \cdot 0.6 + 1 \cdot 4 \cdot 1) \\ &= \underline{1}. \end{aligned}$$

Similarly, we obtain:

$$c_{A_2}^2 = \underline{1}, \quad c_{A_3}^2 = \underline{1}, \quad c_{A_4}^2 = \underline{4}.$$

Flow (Eq. (10.44)):

$$\begin{aligned} c_{D_1}^2 &= \rho_1^2(c_{B_1}^2 + 1) + (1 - \rho_1)c_{A_1}^2 + \rho_1(1 - 2\rho_1) \\ &= 0.8^2(2 + 1) + (1 - 0.8) \cdot 1 + 0.8 \cdot (1 - 2 \cdot 0.8) \\ &= \underline{1.64}. \end{aligned}$$

Similarly, we get:

$$c_{D_2}^2 = \underline{1.45}, \quad c_{D_3}^2 = \underline{0.82}, \quad c_{D_4}^2 = \underline{3.368}.$$

Splitting (Eq. (10.41)):

$$\begin{aligned} c_{12}^2 &= 1 + p_{12} \cdot (c_{D_1}^2 - 1) \\ &= 1 + 0.5 \cdot (1.64 - 1) \\ &= \underline{1.32}, \\ c_{13}^2 &= \underline{1.32}, \quad c_{21}^2 = \underline{1.45}, \quad c_{31} = \underline{0.892}, \quad c_{41}^2 = \underline{3.368}. \end{aligned}$$

Iteration 2:
Merging (Eq. (10.42)):

$$c_{A_1}^2 = \underline{1.666}, \quad c_{A_2}^2 = \underline{1.32}, \quad c_{A_3}^2 = \underline{1.32}, \quad c_{A_4}^2 = \underline{4.0}.$$

Flow (Eq. (10.44)):

$$c_{D_1}^2 = \underline{1.773}, \quad c_{D_2}^2 = \underline{1.674}, \quad c_{D_3}^2 = \underline{0.948}, \quad c_{D_4}^2 = \underline{3.368}.$$

Splitting (Eq. (10.41)):

$$c_{12}^2 = \underline{1.387}, \quad c_{13}^2 = \underline{1.387}, \quad c_{21}^2 = \underline{1.674}, \quad c_{31}^2 = \underline{0.969}, \quad c_{41}^2 = \underline{3.368}.$$

$$\vdots$$

After six iteration steps we get the values for the coefficients of variation of the interarrival times $c_{A_i}^2$ at the nodes, shown in Table 10.3. In Table 10.4 the mean number of jobs at different nodes are given. We use the input parameters

Table 10.3 Coefficients of variation $c_{A_i}^2$, $i = 1, 2, 3, 4$ of the interarrival times at the nodes

Iteration	$c_{A_1}^2$	$c_{A_2}^2$	$c_{A_3}^2$	$c_{A_4}^2$
1	1.0	1.0	1.0	4.0
2	1.666	1.320	1.320	4.0
3	1.801	1.387	1.387	4.0
4	1.829	1.400	1.400	4.0
5	1.835	1.403	1.403	4.0
6	1.836	1.404	1.404	4.0

Table 10.4 Mean number of jobs $\overline{K}_i$ at node i using the formulae of Allen-Cunneen (AC), Krämer-Langenbach/Belz (KLB), and discrete-event simulation (DES)

Methods	$\overline{K}_1$	$\overline{K}_2$	$\overline{K}_3$	$\overline{K}_4$
AC	6.94	0.78	1.46	0.31
KLB	5.97	0.77	1.42	0.27
DES	4.36	0.58	1.42	0.23

and values for the coefficients of variation for the interarrival times at the nodes after six iterations (given in Table 10.3). As we can see in Table 10.4, we get better results if we use the Krämer-Langenbach/Belz formula instead of the Allen-Cunneen formula.

Example 10.5 In this second example we have two servers at node 1. The queueing network model is shown in Fig. 10.9.

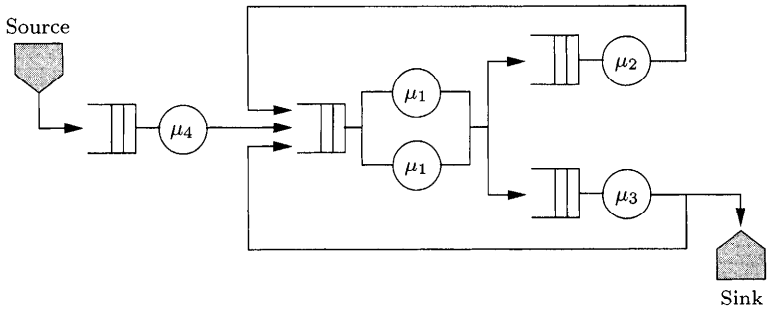


Fig. 10.9 Open non-product-form queueing network with two servers at node 1.

The service rate of each server at node 1 is $\mu_1 = 12.5$. All other input parameters are the same as in Example 10.4. We also have the same values for the arrival rates λ_i and the utilizations ρ_i. The initial values for the

coefficients of variation c_{ij} are again:

$$c_{12}^2 = c_{13}^2 = c_{21}^2 = c_{31}^2 = c_{41}^2 = 1.0.$$

This time we use the formula of Whitt to determine the coefficients of variation c_{A_i} iteratively.

Iteration 1:
Merging: Whitt uses the same merging formula as Pujolle (Eq. (10.42)) and we get:

$$c_{A_1}^2 = c_{A_2}^2 = c_{A_3}^2 = \underline{1}, \quad c_{A_4}^2 = \underline{4.0}.$$

Flow (Eq. (10.45)):

$$\begin{aligned} c_{D_1}^2 &= 1 + \frac{\rho_1^2(c_{B_1}^2 - 1)}{\sqrt{m_1}} + (1 - \rho_1^2)(c_{A_1}^2 - 1) \\ &= 1 + \frac{0.64(2-1)}{\sqrt{2}} + (1 - 0.64)(1 - 1) \\ &= \underline{1.453} \\ c_{D_2}^2 &= \underline{1.45}, \quad c_{D_3}^2 = \underline{0.82}, \quad c_{D_4}^2 = \underline{3.848}. \end{aligned}$$

Splitting (Eq. (10.41)):

$$c_{12}^2 = \underline{1.226}, \quad c_{13}^2 = \underline{1.226}, \quad c_{21}^2 = \underline{1.450}, \quad c_{31}^2 = \underline{0.892}, \quad c_{41}^2 = \underline{3.848}.$$

Iteration 2:
Merging (Eq. (10.42)):

$$c_{A_1}^2 = \underline{1.762}, \quad c_{A_2}^2 = \underline{1.226}, \quad c_{A_3}^2 = \underline{1.226}, \quad c_{A_4}^2 = \underline{4.0}.$$

Flow (Eq. (10.45)):

$$c_{D_1}^2 = \underline{1.727}, \quad c_{D_2}^2 = \underline{1.656}, \quad c_{D_3}^2 = \underline{0.965}, \quad c_{D_4}^2 = \underline{3.848}.$$

Splitting (Eq. (10.41)):

$$c_{12}^2 = \underline{1.363}, \quad c_{13}^2 = \underline{1.363}, \quad c_{21}^2 = \underline{1.656}, \quad c_{31}^2 = \underline{0.979}, \quad c_{41}^2 = \underline{3.848}.$$

$$\vdots$$

After seven iterations, we get the values for the coefficients of variation of the interarrival times $c_{A_i}^2$ at the nodes shown in Table 10.5. In Table 10.6, the mean number of jobs at the different nodes are given. We use the input parameters and values for the coefficients of variation for the interarrival times at the nodes after seven iterations (given in Table 10.5). These results are similar to the results of Example 10.4, and the differences with discrete-event simulation results are slightly smaller than in Example 10.4.

Table 10.5 Coefficients of variation $c_{A_i}^2$ of the interarrival times at the nodes

Iteration	$c_{A_1}^2$	$c_{A_2}^2$	$c_{A_3}^2$	$c_{A_4}^2$
1	1.0	1.0	1.0	4.0
2	1.762	1.226	1.226	4.0
3	1.891	1.363	1.363	4.0
4	1.969	1.387	1.387	4.0
5	1.983	1.401	1.401	4.0
6	1.991	1.403	1.403	4.0
7	1.992	1.405	1.405	4.0

Table 10.6 Mean number of jobs $\overline{K}_i$ at the nodes using the formula of Allen-Cunneen (AC), Krämer-Langenbach/Belz (KLB), and discrete-event simulation (DES)

Methods	$\overline{K}_1$	$\overline{K}_2$	$\overline{K}_3$	$\overline{K}_4$
AC	6.48	0.78	1.46	0.31
KLB	6.21	0.77	1.42	0.27
DES	4.62	0.57	1.38	0.23

10.1.4 Methods for Closed Networks

10.1.4.1 Robustness for Closed Networks In case of closed non-product-form queueing networks with -/G/1 and -/G/m FCFS nodes, the easiest way to analyze them is to replace the -/G/1 and -/G/m FCFS nodes by -/M/1 and -/M/m FCFS nodes and to use a product-form method such as MVA, convolution, or the SCAT algorithm. This substitution can be done due to the property of *robustness* of such closed non-product-form queueing networks. Robustness, in general, means that a major change in system parameters generates only minor changes in the calculated performance measures. In our case it means that if we replace the values of the coefficients of variation of the service times c_{Bi} $(i = 1, \ldots, N)$ by 1, i.e., we assume exponentially distributed service times instead of arbitrarily distributed service times, we obtain a tolerable deviation in the calculated performance measures.

We have investigated more than 100 very different closed non-product-form queueing networks with -/G/1 and -/G/m FCFS nodes. The maximum number of nodes was 10 and the coefficient of variation varied from 0.1 to 15. For a network with only -/G/1 FCFS nodes, the mean deviation is about 6% for both single and multiple class networks. In the case of only -/G/m FCFS nodes, the deviation decreases to about 2%. If we have only -/G/∞ nodes, the deviation is zero. The difference between single and multiple class networks is that the maximum deviation is greater for multiple class networks (40% instead of 20% for single class networks).

For open non-product-form queueing networks, the influence of the coefficients of variation is much greater, especially if the network is a tandem one without any feedback. So we cannot suggest to use the property of robustness to obtain approximate results for open non-product-form queueing networks. It is better to use one of the approximation methods introduced in Section 10.1.2.

Example 10.6 In this example we study the robustness property of several closed networks with different coefficients of variation. Consider the three network topologies given in Fig. 10.10 together with four combinations of the coefficients of variation (see Table 10.7). Note that the first combination (a) corresponds the case of all service times being exponentially distributed and the corresponding network being a product-form one. The analysis of the resulting models is done with $K = 5$ and $K = 10$ jobs in the system.

Table 10.7 Squared coefficients of variation for the networks in Fig. 10.10

Combinations	Node 1 $c^2_{B_1}$	Node 2 $c^2_{B_2}$	Node 3 $c^2_{B_3}$
a	1.0	1.0	1.0
b	0.2	0.4	0.8
c	4.0	1.0	0.2
d	2.0	4.0	8.0

We use two combinations of service rates. The first combination leads to a more balanced network, while the second combination results in a network with a bottleneck at node 1 (see Table 10.8).

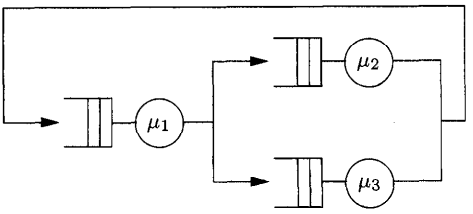
Table 10.8 Two combinations of service rates

	μ_1	μ_2	μ_3
Balanced network	0.5	0.333	0.666
Network with bottleneck	0.5	1.0	2.0

In Tables 10.9 and 10.10, the throughputs λ for the three network topologies together with the different combinations of the coefficient of variation are given. If we compare the product-form approximation results (case a, Table 10.7) with the results for non-product-form networks (cases b, c, and d, Table 10.7), given in Tables 10.9 and 10.10, we see that the corresponding product-form network is a good approximation to the non-product-form one provided:

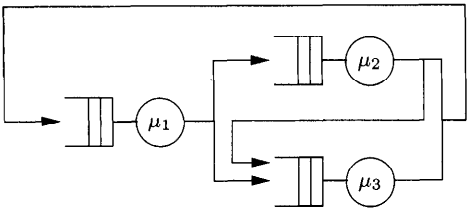
- The coefficients of variation are < 1 (case b) and/or not too big (case c).

Network 1



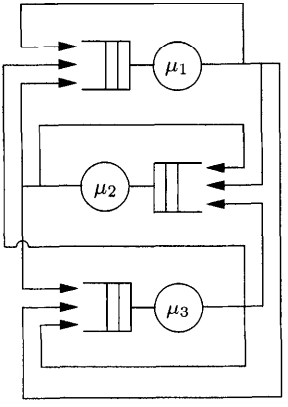
$p_{12} = 0.6;\ p_{13} = 0.4$

Network 2



$p_{12} = 0.6;\ p_{13} = 0.4$
$p_{21} = 0.3;\ p_{23} = 0.7$

Network 3



$p_{11} = 0.2;\ p_{12} = 0.4;\ p_{13} = 0.4;$
$p_{21} = 0.3;\ p_{22} = 0.2;\ p_{23} = 0.5;$
$p_{31} = 0.6;\ p_{32} = 0.2;\ p_{31} = 0.2;$

Fig. 10.10 Different closed networks.

Table 10.9 Throughputs λ for the networks in Fig. 10.10, for the balanced case

Squared Coefficient	Network 1		Network 2		Network 3	
of Variations	$K = 5$	$K = 10$	$K = 5$	$K = 10$	$K = 5$	$K = 10$
a	0.43	0.47	0.41	0.47	0.37	0.42
b	0.47	0.50	0.45	0.49	0.40	0.44
c	0.39	0.44	0.48	0.47	0.36	0.41
d	0.37	0.42	0.33	0.39	0.29	0.34

Table 10.10 Throughputs λ for the networks in Fig. 10.10, for the case with a bottleneck.

Squared Coefficient	Network 1		Network 2		Network 3	
of Variations	$K = 5$	$K = 10$	$K = 5$	$K = 10$	$K = 5$	$K = 10$
a	0.50	0.50	0.51	0.50	0.50	0.52
b	0.50	0.50	0.52	0.50	0.50	0.52
c	0.49	0.50	0.52	0.50	0.49	0.52
d	0.47	0.50	0.47	0.50	0.48	0.50

- The network has a bottleneck.
- The number of jobs in the network is high.

The larger the coefficients of variation ($c_{B_i}^2 > 1$), the larger are the deviations.

Example 10.7 In this example we consider in more detail the dependency of the approximation accuracy on the number of jobs K in the network. The examination is based on the first network given in Fig. 10.10 together with the same input parameters except that $\mu_i = 1$, for $i = 1, 2, 3$. The results for the throughputs λ for different numbers of jobs K are shown in Table 10.11. As we can see in Table 10.11, the larger the number of jobs K, the better is the approximation.

Table 10.11 Throughput λ for the Network 1 shown in Fig. 10.10, for different values of K and combinations of the $c_{B_i}^2$

K	3	4	5	10	20	50
a	0.842	0.907	0.940	0.996	1.00	1.00
b	–	0.970	0.991	1.00	1.00	1.00
c	–	0.856	0.894	0.972	0.998	1.00
d	0.716	0.766	0.805	0.917	0.984	1.00

10.1.4.2 Marie's Method If the network contains single or multiple server FCFS nodes with generally distributed service times, then an approximate iterative method due to Marie [Mari79, Mari80] can be used. In this method each node of the network is considered in isolation with a Poisson arrival stream having load-dependent arrival rates $\lambda_i(k)$. To determine the unknown arrival rates $\lambda_i(k)$, we consider a product-form network corresponding to the given non-product-form network. The product-form network is derived from the original network by simply substituting the FCFS nodes with generally distributed service times by FCFS nodes with exponentially distributed service times (and load-dependent service rates). Because the service rates $\mu_i(k)$ for the nodes of the substitute network are determined in the isolated analysis of each node, the method is iterative where the substitute network is initialized with the service rates of the original network. The load-dependent arrival rates $\lambda_i(k)$ of the nodes $1, \ldots, N$ can be computed by short-circuiting node i in the substitute network (see Fig. 10.11). All other nodes of the network are combined into the composite node c (see Section 8.4).

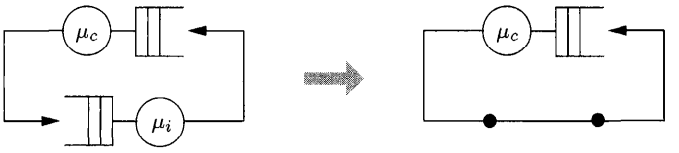


Fig. 10.11 Short-circuiting node i in the substitute network.

If there are k jobs at node i, then there are exactly $K - k$ jobs at the composite node c. The load-dependent service rate through the substitute network with $K - k$ jobs and short-circuited node i is, therefore, the arrival rate at node i with k jobs at node i, which is:

$$\lambda_i(k) = \lambda_c^{(i)}(K - k), \quad \text{for } k = 0, \ldots, (K - 1). \tag{10.60}$$

Here $\lambda_c^{(i)}(K - k)$ denotes the throughput of the network with node i short-circuited. This throughput can be computed using any algorithm for a product-form network.

The arrival rates $\lambda_i(k)$ can now be used for the isolated analysis of the single node of the network. The state probabilities $\pi_i(k)$ of the nodes in the substitute network and the values of the load-dependent service rates $\mu_i(k)$ for the next iteration step have to be computed. It is shown in [MaSt77] that in the considered network for each isolated node, a partition of the state space can be found so that the following condition holds:

$$\nu_i(k) \cdot \pi_i(k) = \lambda_i(k - 1) \cdot \pi_i(k - 1). \tag{10.61}$$

According to this equation the probability that a job leaves a node in state k is equal to the probability that a job arrives at the same node when this node is in state $k - 1$. The stochastic process, which can be assigned to the node

under consideration, behaves exactly like a birth-death process with birth rates $\lambda_i(k)$ and death rate $\nu_i(k)$ (see Section 3.1). The state probabilities, which we are looking for, can therefore be computed as follows:

$$\pi_i(k) = \pi_i(0) \prod_{j=0}^{k-1} \frac{\lambda_i(j)}{\nu_i(j+1)}, \quad k = 1, \ldots, K, \tag{10.62}$$

$$\pi_i(0) = \frac{1}{1 + \sum_{j=1}^{K} \prod_{k=0}^{j-1} \frac{\lambda_i(k)}{\nu_i(k+1)}}. \tag{10.63}$$

The new load-dependent service rates for the nodes in the substitute network for the next iteration step are then chosen as follows:

$$\mu_i(k) = \nu_i(k) \quad \text{for } i = 1, \ldots, N. \tag{10.64}$$

Now the main problem is to determine the rates $\nu_i(k)$ for the individual nodes in the network. The fact that in closed queueing networks the number of jobs is constant allows us to consider the individual nodes in the network as elementary $\lambda(k)$/G/m/K stations in the isolated analysis. The notation describes a more specific case of a GI/G/m node where the arrival process is Poisson with state-dependent arrival rates and for which an additional number K exists, so that $\lambda(k) > 0$ for all $k < K$ and $\lambda(k) = 0$ for all $k \geq K$.

If we do the isolated analysis, we have to differentiate between the following

- Product-form node
- Single server node with FCFS service strategy and generally distributed service time

In the first case, a comparison of the Eqs. (10.62) and (10.63) with the corresponding equations to compute the state probabilities of product-form nodes with load-dependent arrival rates [Lave83] shows that the rates $\nu_i(k)$ correspond to the service rates $\mu_i(k)$. For nodes with exponentially distributed service times and service discipline PS, IS, LCFS PR, respectively, it follows immediately from Eq. (10.64), that the service rates of the substitute network remain unchanged in each iteration step.

Next, consider FCFS nodes with generally distributed service times, which can be analyzed as elementary $\lambda(k)$/G/1/K-FCFS systems. Under the assumption that the distribution of the service time has a rational Laplace transform, we use a Cox distribution to approximate the general service times. Depending on the value of the squared coefficient of variation c_{Bi}^2 of the service time, different Cox models are used: if $c_{Bi}^2 = 1/m \pm \varepsilon$ for $m = 3, 4, \ldots$, where ε is a suitable tolerance area, then we use an Erlang model with m exponential phases. If $c_{Bi}^2 \geq 0.5$, then a Cox-2 model is chosen. For all other cases a special Cox model with m phases is chosen.

According to [Mari80], the parameters for the models are determined as follows:

- For a *Cox-2 model* (Fig. 10.12):

$$\hat{\mu}_{i1} = 2\mu_i, \quad \hat{\mu}_{i2} = \mu_i \cdot \frac{1}{c_{Bi}^2}, \quad a_i = \frac{1}{2c_{Bi}^2}.$$

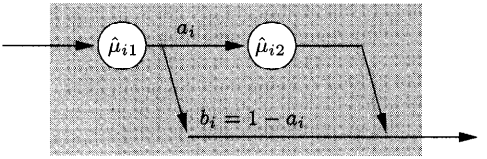


Fig. 10.12 Cox-2 model.

- The special *Cox-m model* is shown in Fig. 10.13. A job leaves this model with probability b_i after completing service at the first phase or runs through the rest of the $(m-1)$ exponential phases before it is completed. The number m of exponential phases is given by:

$$m = \left\lceil \frac{1}{c_{Bi}^2} \right\rceil.$$

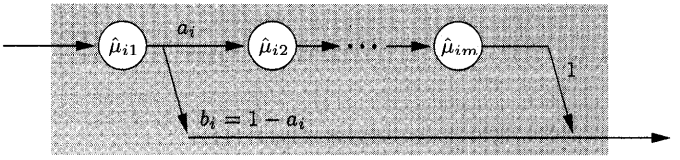


Fig. 10.13 Special Cox-m model.

From Section 1.2.1, we have:

$$b_i = \frac{2mc_{Bi}^2 + (m-2) - \sqrt{m^2 + 4 - 4mc_{Bi}^2}}{2(m-1)(c_{Bi}^2 + 1)},$$
$$\hat{\mu}_i = [m - b_i(m-1)]\,\mu_i.$$

Here b_i denotes the probability that a job leaves node i after the first phase.

- For the *Erlang-m model* we have:

$$\hat{\mu}_i = m \cdot \mu_i.$$

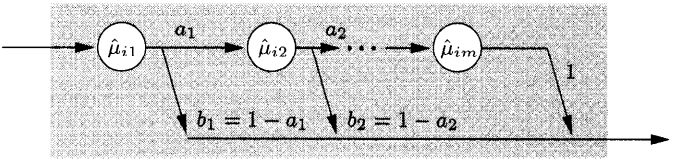


Fig. 10.14 General Cox-m model.

Next consider the general Cox-m model, shown in Fig. 10.14. Each of the elementary $\lambda(k)/C_m/1/K$ nodes can be assigned a CTMC whose states (k, j) indicate that the service process is in the jth phase and k jobs are at node i with corresponding steady-state probability $\pi(k, j)$. The underlying CTMC is shown in Fig. 10.15.

The corresponding balance equations for each node i are given by:

$$\lambda_i(0)\pi_i(0,0) = \sum_{j=1}^{m} b_{i,j}\hat{\mu}_{i,j}\pi_i(1,j), \tag{10.65}$$

$$\lambda_i(k-1)\pi_i(k-1,1) + \sum_{j=1}^{m} b_{i,j}\hat{\mu}_{i,j}\pi_i(k+1,j) = \lambda_i(k)\pi_i(k,1) + \hat{\mu}_{i,1}\pi_i(k,1), \tag{10.66}$$

$$\lambda_i(k-1)\pi_i(k-1,j) + a_{i,j-1}\hat{\mu}_{i,j-1}\pi_i(k,j-1) = \lambda_i(k)\pi_i(k,j) + \hat{\mu}_{i,j}\pi_i(k,j). \tag{10.67}$$

Let $\pi_i(k)$ denote the probability that there are k jobs at node i and $\nu_i(k)$ the conditional throughput of node i with k jobs:

$$\pi_i(k) = \sum_{j=1}^{m} \pi_i(k,j), \tag{10.68}$$

$$\nu_i(k) = \frac{\sum_{j=1}^{m} b_{i,j}\hat{\mu}_{ij}\pi_i(k,j)}{\sum_{j=1}^{m} \pi_i(k,j)}. \tag{10.69}$$

Here $b_{i,j}$ denotes the probability that a job at node i leaves the node after the jth phase. If we add Eq. (10.66) to Eq. (10.67) (for $j = 2, \ldots, m$), we obtain for node i:

$$\lambda_i(k-1) \cdot \sum_{j=1}^{m} \pi_i(k-1,j) + \sum_{j=1}^{m} b_{i,j}\hat{\mu}_{ij}\pi_i(k+1,j) = \lambda_i(k)\sum_{j=1}^{m} \pi_i(k,j) + \sum_{j=1}^{m} b_{i,j}\hat{\mu}_{ij}\pi_i(k,j). \tag{10.70}$$

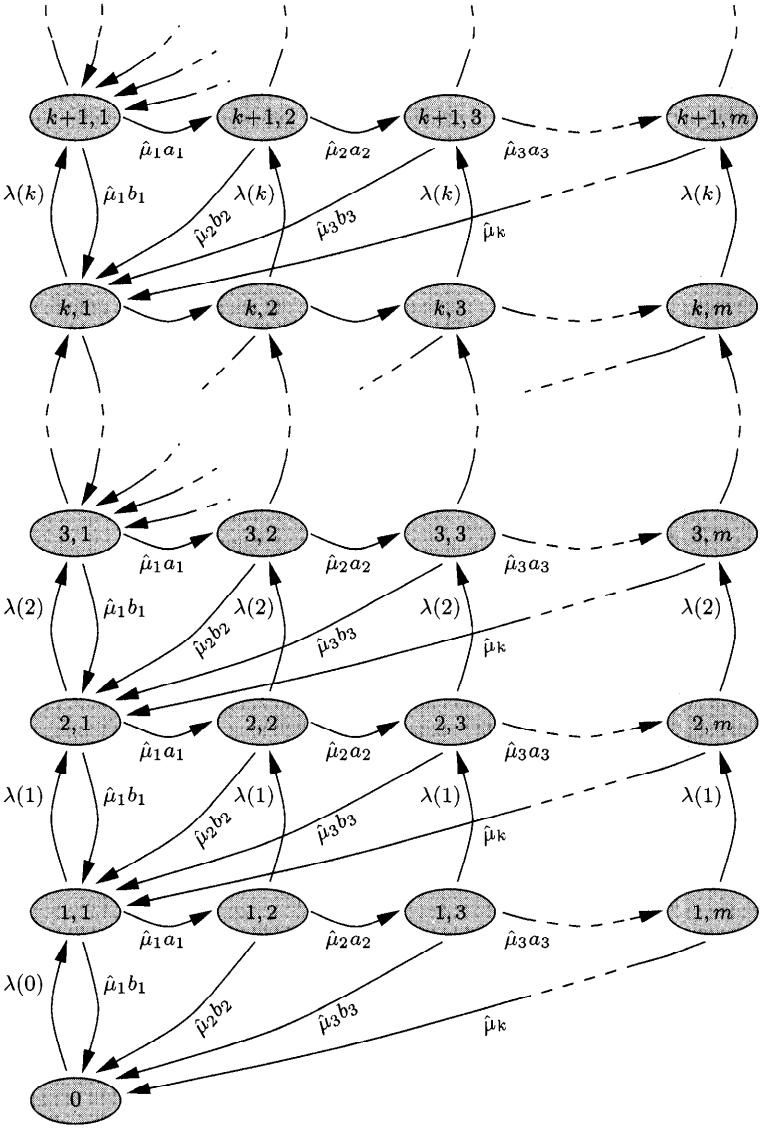


Fig. 10.15 CTMC for the general Cox-m model.

Now we can apply Eq. (10.69) to Eq. (10.70) and obtain:

$$\lambda_i(k-1)\cdot\sum_{j=1}^{m}\pi_i(k-1,j)+\nu_i(k+1)\cdot\sum_{j=1}^{m}\pi_i(k+1,j)$$
$$=\lambda_i(k)\cdot\sum_{j=1}^{m}\pi_i(k,j)+\nu_i(k)\sum_{j=1}^{m}\pi_i(k,j),$$
$$\lambda_i(k-1)\pi_i(k-1)+\nu_i(k+1)\pi_i(k+1)=\lambda_i(k)\pi_i(k)+\nu_i(k)\pi_i(k).$$

If we now consider Eq. (10.65), we finally get for the solution of an elementary $\lambda(k)/C_m/1/K$ system in steady-state [Mari78]:

$$\nu_i(k)\pi_i(k)=\lambda_i(k-1)\pi_i(k-1).$$

But this is exactly Eq. (10.61). For this reason the state probabilities of an FCFS node with generally distributed service time can be determined using Eqs. (10.62) and (10.63) where Eq. (10.69) is used to compute the conditional throughputs $\nu_i(k)$ of the single nodes.

To determine the conditional throughputs, efficient algorithms are given in [Mari80]. In the case of an exponential distribution, $\nu_k(k)$ is just given by:

$$\nu_i(k)=\mu_i. \tag{10.71}$$

In the case of an Erlang-m model, $\nu_i(k)$ is given by:

$$\nu_i(k)=\frac{\hat{\mu}_i}{E_{i,k}(0)+\sum_{j=1}^{y-1}(-1)^j E_{i,k}(j)\left(\prod_{l=1}^{j}\frac{\nu_i(k-l)}{\hat{\mu}_i}\right)}, \tag{10.72}$$

with:

$$E_{i,k}(j)=\sum_{n=j}^{y-1}T_{i_{k,n}}(j),\qquad T_{i_{k,n}}(j)=\sum_{\tau_{n,j}^{k}}\prod_{h=k-j}^{k}u_{i,h}^{V_h},$$
$$\tau_{n,j}^{k}=\Big\{(V_k,V_{k-1},\ldots,V_{k-j}):\forall k\in\{k,\ldots,k-j\},$$
$$0\le V_h\le n-j,\quad\sum_{h=k-j}^{k}V_h=n-j\Big\},$$
$$u_{i,h}=1+\phi_{i,h},\qquad\phi_{i,h}=\frac{\lambda_i(h)}{\hat{\mu}_i},\qquad y=\min\{m,k\}.$$

For the Cox-2 model ($c_{Bi}^2\ge 0.5$), an even simpler equation exists:

$$\nu_i(1)=\frac{\lambda_i(1)\cdot\hat{\mu}_{i1}\cdot b_{i,1}+\hat{\mu}_{i1}\cdot\hat{\mu}_{i2}}{\lambda_i(1)+\hat{\mu}_{i2}+a_{i1}\cdot\hat{\mu}_{i1}}, \tag{10.73}$$

$$\nu_i(k)=\frac{\lambda_i(k)\cdot\hat{\mu}_{i1}\cdot b_{i,1}+\hat{\mu}_{i1}\cdot\hat{\mu}_{i2}}{(\lambda_i(k)+\hat{\mu}_{i1}+\hat{\mu}_{i2})-\nu_i(k-1)},\quad\text{for } k>1. \tag{10.74}$$

We do not discuss the algorithms for the computation of the conditional throughputs of the other model type ($c_{Bi}^2 < 0.5$). The interested reader is referred to the original literature [Mari80].

The computed $\nu_i(k)$ are used in the next iteration step as load-dependent service rates for the nodes in the substitute network, Eq. (10.64). To stop the iteration, two conditions must be fulfilled. First, after each iteration we check whether the sum of the mean number of jobs equals the number of jobs in the network:

$$\left| \frac{K - \sum_{i=1}^{N} \overline{K}_i}{K} \right| < \varepsilon, \tag{10.75}$$

where ε is a suitable tolerance and

$$\overline{K}_i = \sum_{k=1}^{K} k \cdot \pi_i(k)$$

is the mean number of jobs at the ith node.

Second, we check whether the conditional throughputs of each node are consistent with the topology of the network:

$$\left| \frac{r_j - \frac{1}{N} \sum_{i=1}^{N} r_i}{\frac{1}{N} \sum_{i=1}^{N} r_i} \right| < \varepsilon \quad \text{for } j = 1, \ldots, N, \tag{10.76}$$

where:

$$r_j = \frac{\lambda_j}{e_j} = \frac{1}{e_j} \left(\sum_{k=1}^{K} \pi_j(k) \cdot \nu_j(k) \right)$$

is the normalized throughput of node j and e_j is the visit ratio. Usual values for ε are 10^{-3} or 10^{-4}.

Thus, the method of Marie can be summarized in the following four steps:

STEP 1 Initialize. Construct a substitute network with the same topology as the original one. The load-dependent service rates $\mu_i(k)$ of the substitute network for $i = 1, \ldots, N$ and $k = 0, \ldots, K$ are directly given by the original network. A multiple server node with service rate μ_i and m_i servers can be replaced by a single server node with the following load-dependent service rate (see also Section 6.5):

$$\mu_i(k) = \begin{cases} 0, & \text{for } k = 0, \\ \min\{k, m_i\} \cdot \mu_i, & \text{for } k > 0. \end{cases}$$

STEP 2 With Eq. (10.60), determine for $i = 1, \ldots, N$ and $k = 0, \ldots, (K-1)$ the load-dependent arrival rates $\lambda_i(k)$ by analyzing the substitute network. Here any product-form analysis algorithm from Chapter 8 can be used. For $k = K$ we have: $\lambda_i(K) = 0$.

STEP 3 Analyze each individual node of the non-product-form network with the load-dependent arrival rate $\lambda_i(k)$. Determine the values $\nu_i(k)$ and the state probabilities $\pi_i(k)$. The computation method to be used for the $\nu_i(k)$ depends on the type of node being considered (for example Eqs. (10.73) and (10.74) for Cox-2 nodes), while the $\pi_i(k)$ can all be determined using Eqs. (10.62) and (10.63).

STEP 4 Check the stopping Conditions (10.75) and (10.76). If the conditions are fulfilled, then the iteration stops and the performance measures of the network can be determined with the equations given in Section 7.2. Otherwise use Eq. (10.64), $\mu_i(k) = \nu_i(k)$, to compute the new service rates for the substitute network and return to Step 2.

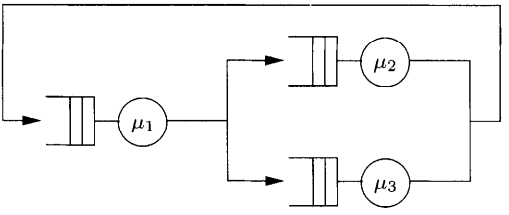


Fig. 10.16 Closed queueing network.

Example 10.8 In this example we show how to apply the method of Marie to a simple closed queueing network with $N = 3$ nodes and $K = 2$ jobs (see Fig. 10.16). The service strategy at all nodes is FCFS and the service times are generally distributed with the mean service times:

$$\frac{1}{\mu_1} = 2\,\text{sec}, \quad \frac{1}{\mu_2} = 4\,\text{sec}, \quad \frac{1}{\mu_3} = 2\,\text{sec},$$

and the squared coefficients of variation are:

$$c_{B1}^2 = 1, \quad c_{B2}^2 = 4, \quad c_{B3}^2 = 0.5.$$

The visit ratios are given as:

$$e_1 = 1, \quad e_2 = 0.5, \quad e_3 = 0.5.$$

We assume a tolerance of $\varepsilon = 0.001$. The analysis of the network can then be done in the given four steps.

STEP 1 Initialize. The service rates of the substitute network can directly be taken from the given network:

$$\mu_1(1) = \mu_1(2) = 0.5, \quad \mu_2(1) = \mu_2(2) = 0.25, \quad \mu_3(1) = \mu_3(2) = 0.5.$$

We obtain the substitute network from the original model by replacing the FCFS nodes with generally distributed service times by FCFS nodes with exponentially distributed service times.

STEP 2 Compute the load-dependent arrival rates $\lambda_i(k)$ for $i = 1, 2, 3$, and for $k = 0, 1, 2$: When using the convolution method (see Section 8.1), we first need to determine the normalizing constants of the substitute network:

$$G(0) = 1, \quad G(1) = 5, \quad G(2) = 17,$$

and the normalizing constants $G_N^{(i)}$ for the network with the short-circuited node i with Eq. (8.11) are:

$$\begin{aligned} G_N^{(1)}(0) &= \underline{1}, & G_N^{(2)}0) &= \underline{1}, & G_N^{(3)}0) &= \underline{1}, \\ G_N^{(1)}1) &= \underline{3}, & G_N^{(2)}1) &= \underline{3}, & G_N^{(3)}1) &= \underline{4}, \\ G_N^{(1)}2) &= \underline{7}, & G_N^{(2)}2) &= \underline{7}, & G_N^{(3)}2) &= \underline{12}. \end{aligned}$$

The throughput in the short-circuited node with $(K-k)$ jobs in the substitute network is then given by Eq. (8.14):

$$\lambda_c^{(i)}(K-k) = e_i \frac{G_N^{(i)}(K-k-1)}{G_N^{(i)}(K-k)}.$$

Now the load-dependent arrival rates can be determined using Eq. (10.60):

$$\lambda_1(0) = \lambda_c^{(1)}(2) = e_1 \frac{G_N^{(1)}(1)}{G_N^{(1)}(2)} = \underline{0.429}.$$

Similarly, the values for the other arrival rates can be determined:

$$\begin{aligned} & & \lambda_2(0) &= \underline{0.214}, & \lambda_3(0) &= \underline{0.167}, \\ \lambda_1(1) &= \underline{0.333}, & \lambda_2(1) &= \underline{0.167}, & \lambda_3(1) &= \underline{0.125}, \\ \lambda_1(2) &= \underline{0}, & \lambda_2(2) &= \underline{0}, & \lambda_3(2) &= \underline{0}. \end{aligned}$$

STEP 3 Perform the isolated analysis of the nodes of the given network. Because $c_{B1}^2 = 1$, the service time of node 1 is exponentially distributed (product-form node) and we have to use Eq. (10.64) $\nu_1(k) = \mu_1(k)$. The state

probabilities for node 1 can be determined using Eqs. (10.62) and (10.63):

$$\pi_1(0) = \frac{1}{1 + \sum_{j=1}^{2} \prod_{k=0}^{j-1} \frac{\lambda_1(k)}{\mu_1(k+1)}} = \underline{0.412},$$

$$\pi_1(1) = \pi_1(0) \cdot \frac{\lambda_1(0)}{\mu_1(1)} = \underline{0.353},$$

$$\pi_1(2) = \pi_1(0) \cdot \prod_{j=0}^{1} \frac{\lambda_1(j)}{\mu_1(j+1)} = \underline{0.235}.$$

Nodes 2 and 3 are non-product-form nodes. Because $c_{Bi}^2 \geq 0.5$, for $i = 2, 3$, we use the Cox-2 distribution to model the non-exponentially distributed service time. The corresponding parameters for node 2 are given as follows:

$$\hat{\mu}_{21} = 2\mu_2 = \underline{0.5}, \quad \hat{\mu}_{22} = \mu_2 \cdot \frac{1}{c_{B2}^2} = \underline{0.0625}, \quad a_2 = \frac{1}{2c_{B2}^2} = \underline{0.125}.$$

For node 3 we get:

$$\hat{\mu}_{31} = \hat{\mu}_{32} = a_3 = 1.$$

Using Eqs. (10.73) and (10.74), we can determine the conditional throughputs:

$$\nu_2(1) = \frac{\lambda_2(1)\hat{\mu}_{21}b_2 + \hat{\mu}_{21}\hat{\mu}_{22}}{\lambda_2(1) + \hat{\mu}_{22} + a_{21}\hat{\mu}_{21}} = \underline{0.357},$$

$$\nu_2(2) = \frac{\lambda_2(2)\hat{\mu}_{21}b_2 + \hat{\mu}_{21}\hat{\mu}_{22}}{(\lambda_2(2) + \hat{\mu}_{21} + \hat{\mu}_{22}) - \nu_2(1)} = \underline{0.152}.$$

Then:

$$\nu_3(1) = \underline{0.471}, \quad \nu_3(2) = \underline{0.654}.$$

For the state probabilities we use Eqs. (10.62) and (10.63) and get:

$$\pi_2(0) = \underline{0.443}, \quad \pi_2(1) = \underline{0.266}, \quad \pi_2(2) = \underline{0.291},$$
$$\pi_3(0) = \underline{0.703}, \quad \pi_3(1) = \underline{0.249}, \quad \pi_3(2) = \underline{0.048}.$$

STEP 4 Check the stopping condition, Eqs. (10.75) and (10.76):

$$\left| \frac{K - \sum_{i=1}^{N} \sum_{k=1}^{K} k \cdot \pi_i(k)}{K} \right| = \underline{7.976 \cdot 10^{-3}}.$$

Because the first stopping condition is not fulfilled, it is not necessary to check the second one.

Now we compute the new service rates of the substitute network using Eq. (10.64):

$$\mu_i(k) = \nu_i(k), \quad \text{for } i = 1, 2, 3,$$

and go back to Step 2.

STEP 2 Determine the load-dependent arrival rates using Eq. (10.60) and analyze the nodes of the network isolated from each other:

$$\vdots$$

After four iterations we get the following final results:

Marginal probabilities:

$$\begin{aligned} \pi_1(0) &= \underline{0.438}, \quad \pi_2(0) = \underline{0.438}, \quad \pi_3(0) = \underline{0.719}, \\ \pi_1(1) &= \underline{0.311}, \quad \pi_2(1) = \underline{0.271}, \quad \pi_3(1) = \underline{0.229}, \\ \pi_1(2) &= \underline{0.251}, \quad \pi_2(2) = \underline{0.291}, \quad \pi_3(2) = \underline{0.052}. \end{aligned}$$

Conditional throughputs:

$$\nu_2(1) = \underline{0.356}, \quad \nu_3(1) = \underline{0.466}, \quad \nu_2(2) = \underline{0.151}, \quad \nu_3(2) = \underline{0.652}.$$

Utilizations, Eq. (7.19):

$$\rho_1 = 1 - \pi_1(0) = \underline{0.562}, \quad \rho_2 = \underline{0.562}, \quad \rho_3 = \underline{0.281}.$$

Mean number of jobs, Eq. (7.26):

$$\overline{K}_1 = \pi_1(1) + 2 \cdot \pi_1(2) = \underline{0.813}, \quad \overline{K}_2 = \underline{0.853}, \quad \overline{K}_3 = \underline{0.333}.$$

In most cases the accuracy of Marie's method has proven to be very high. The predecessor of Marie's method was the iterative method of [CHW75a], which is a combination of the FES method and a recursive numerical method. A network to be analyzed is approximated by K substitute networks where the kth substitute network consists of node k of the original network and one composite node representing the rest of the network. The service rates of the composite nodes are determined using the FES method under the assumption that the network has a product-form solution. Then, the K substitute networks are analyzed using the recursive numerical method. The performance measures obtained from this analysis are approximate values for the whole network. If the results are not precise enough, the substitute networks are analyzed again with appropriate modifications of the service rates. These results are then the approximate values for the given network. The results thus obtained are normally less accurate than the ones derived by Marie's method.

10.1.4.3 Extended SUM and BOTT The summation method and the bottapprox method can easily be extended to non-product-form queueing networks with arbitrarily distributed service times. For the mean number of jobs $\overline{K}_i$ in a -/G/1-FCFS node, we use Eq. (6.54):

$$\overline{K}_i = \rho_i + \frac{\rho_i^2}{1-\rho_i} \cdot a_i, \quad \text{with } a_i = \frac{c_{B_i}^2 + 1}{2}. \tag{10.77}$$

Again we introduce a correction factor in Eq. (10.77) as we did in the product-form case, and we get:

$$\overline{K}_i = \rho_i + \frac{\rho_i^2 \cdot a_i}{1 - \frac{K-1-a_i}{K-1}\rho_i}. \tag{10.78}$$

For multiple server nodes the mean number of jobs $\overline{K}_i$ in a -/G/m-FCFS node is given by:

$$\overline{K}_i = m\rho_i + \frac{\rho_i}{1-\rho_i} \cdot a_i \cdot P_{m_i}. \tag{10.79}$$

The introduction of the correction factor yields:

$$\overline{K}_i = m_i\rho_i + \frac{\rho_i}{1 - \frac{K-m_i-a}{K-m_i} \cdot \rho_i} \cdot a_i \cdot P_{m_i}. \tag{10.80}$$

Now we can use the SUM method for networks with these two additional node types by introducing Eqs. (10.78) and/or (10.80) in the system equation (9.23) and calculate the throughput λ and other performance measures in the same way as in Section 9.2.

We can also use the BOTT method if we use the corresponding formulae for the function to $h(\overline{K}_i)$ for the bottleneck $\overline{K}_i = \overline{K}_{\text{bott}}$. For the node type -/G/1-FCFS, $h(\overline{K}_{\text{bott}})$ has the form:

$$h(\overline{K}_{\text{bott}}) = \frac{-(1 + b \cdot \overline{K}_{\text{bott}}) + \sqrt{(1 + b \cdot \overline{K}_{\text{bott}})^2 + 4 \cdot \overline{K}_{\text{bott}} \cdot (a-b)}}{2 \cdot (a-b)},$$

with $b = (K-1-a)/(K-1)$. For the node type -/G/m-FCFS, the function $h(\overline{K}_{\text{bott}})$ cannot be given in an exact form and therefore an approximation is used:

$$h(\overline{K}_{\text{bott}}) = \frac{b - \sqrt{c}}{2 \cdot f \cdot m_{\text{bott}}},$$

with:

$$f = \frac{K - m_{\text{bott}} - a}{K - m_{\text{bott}}}, \quad a = \frac{1 + c_B^2}{2},$$

$$b = \overline{K}_{\text{bott}} \cdot f + m_{\text{bott}} + P_{m_{\text{bott}}} \cdot a, \quad c = b^2 - (4 \cdot m_{\text{bott}} \cdot \overline{K}_{\text{bott}} \cdot f).$$

The SUM and BOTT methods for non-product-form queueing networks can also easily be extended to the multiclass case as it was done in Sections 9.2 and 9.3 for product-form queueing networks.

10.1.5 Closing Method for Mixed Networks

In [BGJ92], a technique is presented that allows every open queueing network to be replaced by a suitably constructed closed network. The principle of the *closing method* is quite simple: The external world of the open network is replaced by a -/G/1 node with the following characteristics:

1. The service rate μ_∞ of the new node is equal to the arrival rate λ_0 of the open network.
2. The coefficient of variation c_{B_∞} of service time at the new node is equal to the coefficient of variation of the interarrival time of the open network.
3. If the routing behavior of the open network is specified by visit ratios, then the visit ratio e_∞ of the new node is equal to 1. Otherwise the routing probabilities are assigned so that the external world is directly replaced by the new node.

The idea behind this technique is shown in Figs. 10.17 and 10.18.

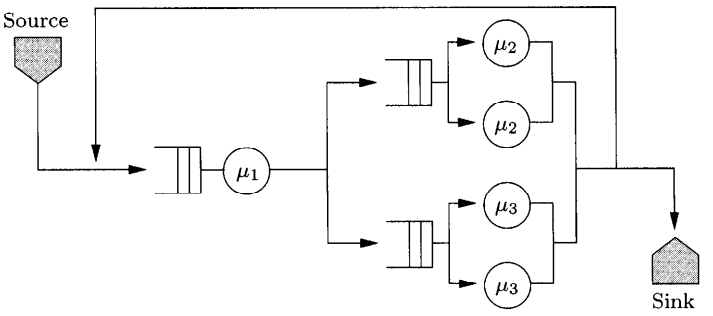


Fig. 10.17 Open network before using the closing method.

A very high utilization of the new node is necessary to reproduce the behavior of the open network with adequate accuracy. This utilization is achieved when there is a large number of jobs K in the closed network. The performance measures are sufficiently accurate after the number of jobs in the network has passed a certain threshold value K_i (see Fig. 10.19). Depending on the chosen solution algorithm, the following values for the number of jobs are proposed (see [BGJ92]):

$$\begin{aligned} K &= 100 \quad && \text{mean value analysis,} \\ K &= 5000 && \text{summation method,} \\ K &= 10000 && \text{SCAT algorithm.} \end{aligned}$$

To check whether the number of jobs K in the closed network is sufficiently large, we can repeat the closing method with a larger number of jobs K and compare the results.

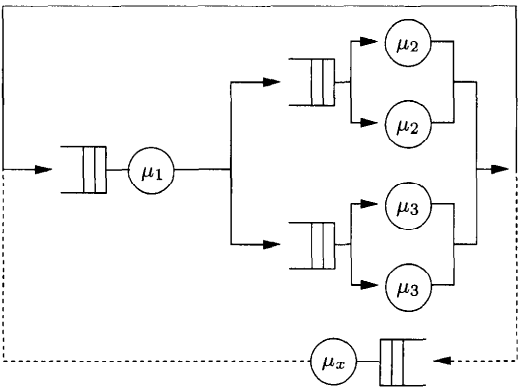


Fig. 10.18 Closed queueing network with the additional -/G/1 node for the closing method.

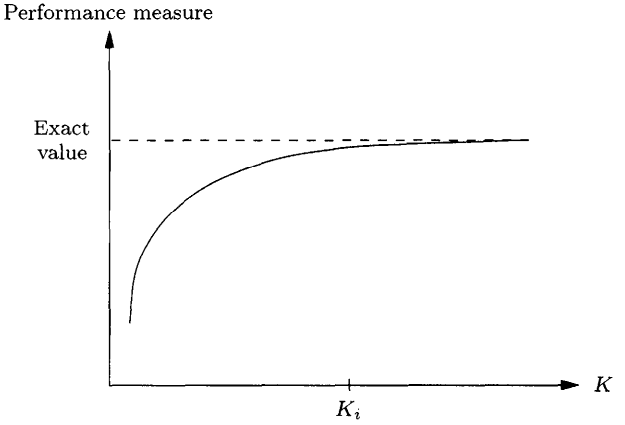


Fig. 10.19 Threshold in the closing method.

The closing method can be extended to multiple class queueing networks and mixed queueing networks. In the case of multiple class networks, the service rate $\mu_{\infty r}$ of the new node is given given by $R \cdot \lambda_{0r}$, with $r = 1, \ldots, R$, where R is the number of classes. In the case of mixed queueing networks, the new node is added to the open classes only and then we can use any method for closed queueing networks also for mixed queueing networks. Open non-product-form networks are easier to analyze using the methods introduced in Section 10.1.3. But the closing method in combination with the SCAT algorithm or the SUM method is an interesting alternative to the MVA (see Section 8.2) for mixed queueing networks in the product-form case. The main advantage of the closing method is the ability to solve mixed non-product-form-queueing networks.

Example 10.9 In this first example, the closing method is shown for an open product-form queueing network with $\lambda_0 = 5$, $p_{12} = 0.3$, $p_{13} = 0.7$, and $p_{21} = p_{31} = 0.1$ (see Fig. 10.17). The service rates are $\mu_1 = 7$, $\mu_2 = 4$, and $\mu_3 = 3$. Using Jackson's method for this network, we obtain the following values for the mean response time $\overline{T}$ and the mean number of jobs in the open network:

$$\overline{T} = \underline{1.30}, \quad \overline{K} = \underline{6.52}.$$

In Table 10.12 the values for these performance measures are shown as a function of K_{closed}, the number of jobs in the closed network. The results are obtained using MVA. From the table we see that we obtain exact results if

Table 10.12 Mean response time $\overline{T}$ and mean number of jobs $\overline{K}$ for several values of the number of jobs in the closed network

K_{closed}	$\overline{T}$	$\overline{K}$
10	0.95	4.76
20	1.24	6.17
30	1.29	6.47
50	1.30	6.52
60	1.30	6.52
100	1.30	6.52

K_{closed} is large enough and that, in this case, $K_{\text{closed}} = 50$ is the threshold.

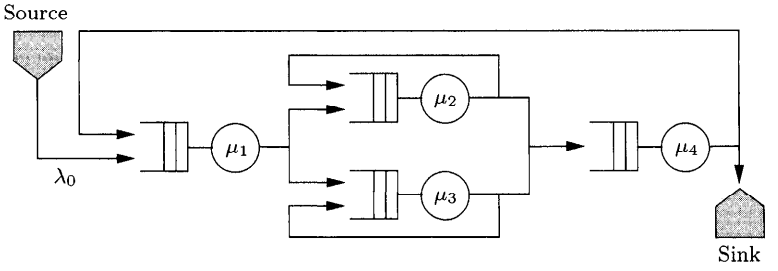


Fig. 10.20 Open queueing network.

Example 10.10 In this second example, the closing method is used for an open non-product-form network (see Fig. 10.20) with $\lambda_0 = 3$ and $c_0^2 = 1.5$. The routing probabilities are given by:

$$p_{12} = 0.3, \quad p_{13} = 0.7, \quad p_{22} = 0.3, \quad p_{24} = 0.7,$$
$$p_{33} = 0.1, \quad p_{34} = 0.9, \quad p_{41} = 0.4.$$

Table 10.13 Input parameters for the network of Fig. 10.20

Node	μ_i	c_{Bi}^2
1	9	0.5
2	10	0.8
3	12	2.4
4	4	4

The other parameters are listed in Table 10.13.

With the method of Pujolle (see Section 10.1.3), we can analyze the open network directly to obtain:

$$\overline{T} = \underline{4.43} \quad \text{and} \quad \overline{K} = \underline{13.47}.$$

The alternative method we use to solve the network of Fig. 10.20 is to first use the closing method and then use the SUM method (see Section 10.1.4.3) to approximately solve the resulting closed non-product-form network. The results are shown in Table 10.14. From this table we see that the threshold for

Table 10.14 Mean response time $\overline{T}$ and mean number of jobs $\overline{K}$ as a function of K_{closed}

K_{closed}	$\overline{T}$	$\overline{K}$
100	3.94	11.83
200	4.16	12.49
500	4.29	12.88
1000	4.34	13.02
5000	4.37	13.12
10000	4.37	13.12

K_{closed} is about 5000 and the results of the closing method and the method of Pujolle differ less than 3%.

Example 10.11 The open networks of Example 10.9 and Example 10.10 are much easier to analyze using the Jackson's method and the Pujolle method, respectively, rather than by the closing method. Mixed product-form networks are easy to analyze using the MVA for mixed networks but also using SCAT or SUM in combination with the closing method, especially for networks with multiple server nodes. But the main advantage of the closing method is the solution of mixed non-product-form queueing networks. As an example we use a mixed network with $N = 5$ nodes and $R = 2$ classes. Class 1 is closed and contains $K_1 = 9$ jobs (see Fig. 10.21). Class 2 (see Fig. 10.22) is open with arrival rate $\lambda_{01} = 5$ and the squared coefficient of variation $c_{01}^2 = 0.7$.

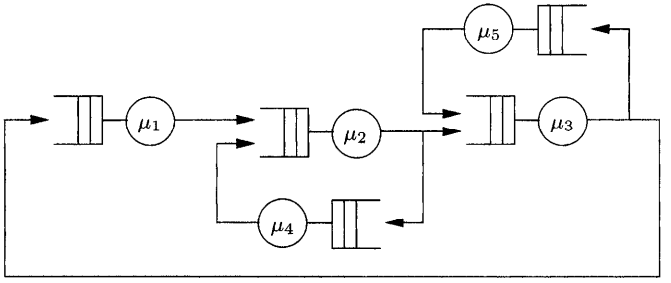


Fig. 10.21 Closed class 1 of the mixed network.

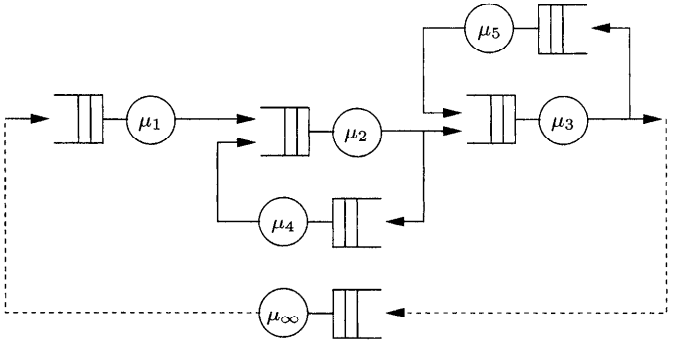


Fig. 10.22 Open class 2 of the mixed network with an additional node.

There is no class switching in this network. The routing probabilities of the two classes are as follows:

Class 1: $p_{12,1} = 1.0$,
$p_{23,1} = 0.7$, $p_{24,1} = 0.3$,
$p_{31,1} = 0.7$, $p_{35,1} = 0.3$,
$p_{42,1} = 1.0$,
$p_{53,1} = 1.0$,

Class 2: $p_{12,2} = 1.0$,
$p_{23,2} = 0.8$, $p_{24,2} = 0.2$,
$p_{30,2} = 0.7$, $p_{35,2} = 0.3$,
$p_{42,2} = 1.0$,
$p_{53,2} = 1.0$.

The other input parameters are listed in Table 10.15. Now we solve the equivalent closed queueing network with an additional node ($\mu_\infty = 5$, $c_\infty^2 = 0.7$) for class 2 using the SUM method (see Section 10.1.4.3) and $K_2 = 500$. The results for the utilization ρ_i and the mean number of jobs $\overline{K}_i$ are listed in Table 10.16. Since SUM is an approximation method, we obtain deviations from discrete-event simulation results, which are about 6% for the utilizations ρ_i and about 5% for the mean number of jobs $\overline{K}_i$.

Table 10.15 Input parameters for the mixed queueing network

Nodes	$\mu_{i,1} = \mu_{i,2}$	$c_{i1}^2 = c_{i2}^2$	m_i
1	4	0.4	3
2	4	0.3	4
3	6	0.3	3
4	4	0.4	2
5	5	0.5	2

Table 10.16 Utilizations ρ and mean number of jobs from discrete-event simulation (DES) and the closing method (CLS)

Nodes	$\rho_{i_{\text{DES}}}$	$\rho_{i_{\text{CLS}}}$	$\overline{K}_{i_{\text{DES}}}$	$\overline{K}_{i_{\text{sum}}}$
1	0.89	0.84	4.9	5.2
2	0.90	0.85	5.5	5.8
3	0.85	0.80	4.0	4.1
4	0.46	0.43	1.1	1.0
5	0.46	0.43	1.1	1.0

10.2 DIFFERENT SERVICE TIMES AT FCFS NODES

If the given non-product-form network is a "small-distance" away from a product-form network, we may be able to modify the MVA algorithm to approximately solve the network. Based on this idea [Bard79] extended the MVA to priority scheduling as well as to networks with FCFS nodes and with different service rates for different job classes. The key idea for treating the latter case is to replace the MVA Eq. (8.31) by:

$$\overline{T}_{ir}(\mathbf{k}) = \frac{1}{\mu_{ir}} + \sum_{s=1}^{R} \frac{1}{\mu_{is}} \cdot \overline{K}_{is}(\mathbf{k} - \mathbf{1}_r). \tag{10.81}$$

This equation can also be extended for multiple server nodes [Hahn88] as:

$$\overline{T}_{ir}(\mathbf{k}) = \frac{1}{m_i} \left(\frac{1}{\mu_{ir}} + \sum_{s=1}^{R} \frac{1}{\mu_{is}} \cdot \overline{K}_{is}(\mathbf{k} - \mathbf{1}_r) + \frac{1}{\mu_i} \sum_{j=0}^{m_i - 2} (m_i - 1 - j) \cdot \pi_i(j \mid \mathbf{k} - \mathbf{1}_r) \right), \tag{10.82}$$

with:

$$\frac{1}{\mu_i} = \sum_{s=1}^{R} \frac{\lambda_{is}}{\lambda_i} \cdot \frac{1}{\mu_{is}}.$$

In order to demonstrate the use of this method, consider a queueing network consisting of $N = 6$ nodes and $R = 2$ job classes. Nodes 1 to 5 are FCFS nodes with one server each and node 6 is an IS node. Class 1 contains $K_1 = 10$ jobs and class 2 has $K_2 = 6$ jobs. All service times are exponentially distributed. Table 10.17 shows all service rates and visit ratios for the network. Service

Table 10.17 Parameters of the network described in Fig. 10.22

i	μ_{i1}	μ_{i2}	e_{i1}	e_{i2}
1	*	0.5	8	20
2	1	1	2	2
3	1	1	2	4
4	1	1	2	6
5	1	1	2	8
6	0.1	10^7	1	1

rate μ_{11}, marked with an * in the table, is varied and the mean overall response time for all class-1 jobs is determined. We compare the results of the extended MVA with DES.

Table 10.18 Mean overall response time of a class-1 job for different service rates μ_{11}

μ_{11}	0.5	2	128
MVA-Ext.	260.1	143.1	105.4
DES	260.1	141.1	102.0

The values given in Table 10.18 show that the extension of the MVA by Eq. (10.81) gives good results for queueing networks with FCFS nodes with different service rates for different job classes. SCAT and Bard-Schweitzer algorithm can also be extended [Förs89] in a similar fashion.

10.3 PRIORITY NETWORKS

We will consider three different methods for networks with priority classes: Extended MVA, shadow server technique and extended SUM method.

10.3.1 Extended MVA (PRIOMVA)

If a queueing network contains nodes where some job classes have priority over other job classes, then the product-form condition is not fulfilled and the techniques presented in Chapters 8 and 9 cannot be used. For the approximate analysis of such priority networks [BKLC84] suggest an extension of the MVA.

The MVA for mixed queueing networks is thus extended to formulae for the following priority node types:

- -/M/1-FCFS PRS (preemptive resume):

  As per this strategy, as soon as a high priority job arrives at the node it will be serviced if no other job with higher priority is in service at the moment. If a job with lower priority is in service when the higher priority one arrives, then this low priority job is preempted. When all jobs with higher priority are serviced, then the preempted job is resumed starting at the point where it was stopped. It is assumed that the preemption does not cause any overhead. The service strategy within a priority class is FCFS.

- -/M/1-FCFS HOL (non-preemptive head-of-line):

  Here a job with higher priority must wait until the job that is processed at the moment is ready to leave the node even if its priority is lower than the priority of the arriving job. The service strategy within a class is again FCFS.

The priority classes are ordered linearly where class R has lowest priority and class 1 has the highest one.

At first we consider -/M/1-FCFS PRS nodes and derive a formula for the mean response time $\overline{T}_r$. In this case the mean response time of a class-r job consists of:

- The mean service time:

$$\frac{1}{\mu_r}.$$

- The time it has to wait until all jobs with higher or the same priority are serviced:

$$\sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s}.$$

- The time for serving all jobs with a higher priority that arrive during the response time $\overline{T}_r$:

$$\sum_{s=1}^{r-1} \frac{\overline{T}_r \cdot \lambda_s}{\mu_s}.$$

If we combine these three expressions, then for the mean response time of a priority class-r job, $1 \leq r \leq R$, we get the following implicit equation:

$$\overline{T}_r = \sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s} + \sum_{s=1}^{r-1} \frac{\overline{T}_r \cdot \lambda_s}{\mu_s} + \frac{1}{\mu_r},$$

where $\overline{K}_s$ denotes the mean number of class-s jobs at the node. With the relation $\rho_s = \lambda_s/\mu_s$ it follows that:

$$\overline{T}_r = \frac{\frac{1}{\mu_r} + \sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s}}{1 - \sum_{s=1}^{r-1} \rho_s}. \tag{10.83}$$

If, on the other hand, we consider an -/M/1-FCFS HOL node, then the considered job can not be preempted any more. In this case the mean response time consists of:

- The mean service time:

$$\frac{1}{\mu_r}.$$

- The time needed to serve higher priority jobs that arrive during the waiting time with the mean $\overline{W} = \left(\overline{T}_r - \frac{1}{\mu_r}\right)$:

$$\sum_{s=1}^{r-1} \left(\overline{T}_r - \frac{1}{\mu_r}\right) \cdot \frac{\lambda_s}{\mu_s}.$$

- The mean time needed for the job that is being served at the moment:

$$\sum_{s=1}^{R} \frac{\rho_s}{\mu_s}.$$

- The mean service time of jobs with the same or higher priority that are still in the queue:

$$\sum_{s=1}^{r} \frac{\overline{K}_s - \rho_s}{\mu_s}.$$

If we combine these three terms for the mean response time of a priority class-r job, $1 \leq r \leq R$, we get:

$$\overline{T}_r = \sum_{s=1}^{r} \frac{\overline{K}_s - \rho_s}{\mu_s} + \sum_{s=1}^{R} \frac{\rho_s}{\mu_s} + \sum_{s=1}^{r-1} \left(\overline{T}_r - \frac{1}{\mu_r}\right) \cdot \frac{\lambda_s}{\mu_s} + \frac{1}{\mu_r}.$$

If we use the relation $\rho_s = \lambda_s/\mu_s$, then we get:

$$\overline{T}_r(\mathbf{k}) = \frac{1}{\mu_r} + \frac{\sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s} + \sum_{s=r+1}^{R} \frac{\rho_s}{\mu_s}}{1 - \sum_{s=1}^{r-1} \rho_s}. \tag{10.84}$$

The equations just developed give the exact $\overline{T}_r$ values for M/M/1- priority queues. But in a priority queueing network the arrival process at a node is normally not Poisson. This condition means that if we apply the equations given to a priority network, we do not get exact results. Furthermore, these equations allow class-dependent service time distributions, while in product-form queueing networks a class-independent service time distribution is required at FCFS nodes. Nevertheless the approximation of [BKLC84] is based on these assumptions.

Let $\overline{K}_{is}^{(r)}$ denote the mean number of class-s jobs at node i that an arriving class-r job sees given the network population vector $\mathbf{k}$. If Eq. (10.83) is applied to node i of a priority network, then this equation can be generalized as follows:

$$\overline{T}_{ir}(\mathbf{k}) = \frac{\dfrac{1}{\mu_{ir}} + \displaystyle\sum_{s=1}^{r} \frac{\overline{K}_{is}^{(r)}(\mathbf{k})}{\mu_{is}}}{1 - \displaystyle\sum_{s=1}^{r-1} \rho_{is}}. \tag{10.85}$$

Similarly, Eq. (10.84) is of form:

$$\overline{T}_{ir}(\mathbf{k}) = \frac{1}{\mu_{ir}} + \frac{\displaystyle\sum_{s=1}^{r} \frac{\overline{K}_{is}^{(r)}(\mathbf{k})}{\mu_{is}} + \sum_{s=r+1}^{R} \frac{\rho_{is}}{\mu_{is}}}{1 - \displaystyle\sum_{s=1}^{r-1} \rho_{is}}. \tag{10.86}$$

For the computation of $\overline{K}_{is}^{(r)}(\mathbf{k})$, the arrival theorem (see [BKLC84]) is assumed to hold:

$$\overline{K}_{is}^{(r)}(\mathbf{k}) = \begin{cases} \overline{K}_{is}(\mathbf{k} - \mathbf{1}_r), & \text{for closed classes } r, \\ \overline{K}_{is}(\mathbf{k}), & \text{for open classes } r. \end{cases} \tag{10.87}$$

For open class s, the equation:

$$\overline{K}_{is}(\mathbf{k}) = \lambda_s \cdot e_{is} \cdot \overline{T}_{is}(\mathbf{k}) \tag{10.88}$$

can be used to eliminate the unknown $\overline{K}_{is}(\mathbf{k})$. Then in Eqs. (10.85) and (10.86) we only need to determine the utilizations ρ_{is}. For open classes we have $\rho_{is} = \lambda_{is}/\mu_{is}$, independent of the number of jobs in the closed classes. But for closed class s the utilization ρ_{is} depends on the population vector and it is not obvious which $\rho_{is}(\mathbf{k})$, $\mathbf{0} \leq \mathbf{k} \leq \mathbf{K}$, should be used. However, [ChLa83] suggest:

$$\rho_{is} = \rho_{is}(\mathbf{k} - \overline{K}_{is}), \tag{10.89}$$

where $(\mathbf{k} - \overline{K}_{is})$ is the population vector of the closed classes with $\overline{K}_{is}$ jobs fewer in class s. If $\overline{K}_{is}$ is not an integer, then linear interpolation is used to

get a convenient value for ρ_{is}. Equation (10.89) is based on the assumption that if there are already $\overline{K}_{is}$ jobs at node i, then the arrival rate of class-s jobs at node i is given by the remaining $\mathbf{k} - \overline{K}_{is}$ jobs in the network, and hence:

$$\rho_{is} = \frac{\lambda_{is}(\mathbf{k} - \overline{K}_{is})}{\mu_{is}} = \rho_{is}(\mathbf{k} - \overline{K}_{is}).$$

If we insert Eqs. (10.87)-(10.89) into Eqs. (10.85) and (10.86), respectively, then the mean response time of the *closed* class r at the priority node i when there are $\mathbf{k}$ jobs in the closed classes of the network:

$$\text{PRS:} \quad \overline{T}_{ir}(\mathbf{k}) = \frac{\dfrac{1}{\mu_{ir}} + \displaystyle\sum_{s=1}^{r} \frac{\overline{K}_{is}(\mathbf{k} - \mathbf{1}_r)}{\mu_{is}}}{1 - \displaystyle\sum_{s=1}^{r-1} \rho'_{is}}, \tag{10.90}$$

$$\text{HOL:} \quad \overline{T}_{ir}(\mathbf{k}) = \frac{1}{\mu_{ir}} + \frac{\displaystyle\sum_{s=1}^{r} \frac{\overline{K}_{is}(\mathbf{k} - \mathbf{1}_r)}{\mu_{is}} + \sum_{s=r+1}^{R} \frac{\rho_{is}(\mathbf{k} - \mathbf{1}_r)}{\mu_{is}}}{1 - \displaystyle\sum_{s=1}^{r-1} \rho'_{is}}, \tag{10.91}$$

where:

$$\rho'_{is} = \begin{cases} \rho_{is}(\mathbf{k} - \overline{K}_{is}), & \text{for closed class } s, \\ \rho_{is}, & \text{for open class } s. \end{cases}$$

Analogously, for the mean response time of the *open* class r at the priority node i we get:

$$\text{PRS:} \quad \overline{T}_{ir}(\mathbf{k}) = \frac{\dfrac{1}{\mu_{ir}} + \displaystyle\sum_{s=1}^{r} \frac{\overline{K}_{is}(\mathbf{k})}{\mu_{is}}}{1 - \displaystyle\sum_{s=1}^{r-1} \rho'_{is}}, \tag{10.92}$$

$$\text{HOL:} \quad \overline{T}_{ir}(\mathbf{k}) = \frac{\dfrac{1}{\mu_{ir}}\left(1 - \displaystyle\sum_{s=1}^{r-1} \rho'_{is}\right) + \displaystyle\sum_{s=1}^{r-1} \frac{\overline{K}_{is}(\mathbf{k})}{\mu_{is}} + \sum_{s=r+1}^{R} \frac{\rho_{is}(\mathbf{k})}{\mu_{is}}}{1 - \displaystyle\sum_{s=1}^{r-1} \rho'_{is}}. \tag{10.93}$$

Now the modified MVA with the additional priority node types can be given. The closed classes are denoted by $1, \ldots, CL$ and the open classes by $1, \ldots, OP$. Because of the fact that the values of $\overline{K}_{is}(\mathbf{k})$ are also needed for the open class s, it is necessary to determine the performance measures for the open classes in each iteration step of the algorithm.

STEP 1 Initialize. For each node $i = 1, \ldots, N$ determine the utilizations of the open classes $op = 1, \ldots, OP$ using Eq. (8.55), that is:

$$\rho_{i,op} = \frac{1}{\mu_{i,op}} \lambda_o p e_{i,op},$$

and check for stability ($\rho_{i,op} \leq 1$).

Set $\overline{K}_{i,cl}(\mathbf{0}) = 0$ for all nodes $i = 1, \ldots, N$ and all closed classes $cl = 1, \ldots, CL$.

STEP 2 Construct a closed model that contains only jobs of the closed classes and solve this model using the MVA.

Iterate for $\mathbf{k} = \mathbf{0}, \ldots, \mathbf{K}$:

STEP 2.1 Determine $\overline{T}_{ir}(\mathbf{k})$ for all nodes $i = 1, \ldots, N$ and all closed classes $r = 1, \ldots, CL$ using Eqs. (8.53), (8.54), and (10.90) and (10.91), respectively.

STEP 2.2 Determine $\lambda_r(\mathbf{k})$ for all closed classes $r = 1, \ldots, CL$ using Eq. (8.46).

STEP 2.3 Determine $\overline{K}_{ir}(\mathbf{k})$ for all nodes $i = 1, \ldots, N$ and for all closed classes $r = 1, \ldots, CL$ using Eq. (8.47).

STEP 2.4 Determine $\overline{K}_{ir}(\mathbf{k})$ for all nodes $i = 1, \ldots, N$ and all open classes $r = 1, \ldots, OP$ using Eqs. (8.51) and (8.52), and Eqs. (10.88), (10.92) and (10.93) respectively.

STEP 3 Determine all other performance measures using the computed results.

The computation sequenced from the highest priority down to the lowest one.

The algorithm is demonstrated in the following example:

Example 10.12 Consider the queueing network in Fig. 10.23. The network consists of $N = 2$ nodes and $R = 3$ job classes. Class 1 has highest priority and is open, while the classes 2 and 3 are closed. Node 1 is of type

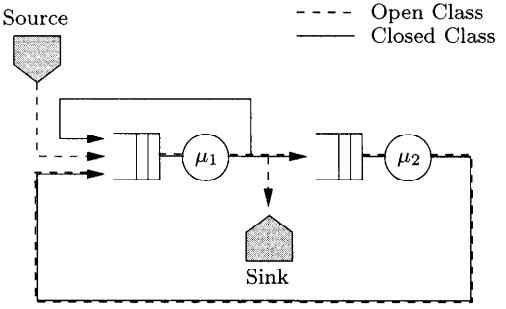


Fig. 10.23 Mixed network with priority nodes.

-/G/1-PS and node 2 is of type M/M/1-FCFS PRS. The arrival rate for class-1 jobs is $\lambda_1 = 1$. Each closed class contains one job ($K_2 = K_3 = 1$). The

mean service times are given as:

$$\frac{1}{\mu_{11}} = 0.4, \quad \frac{1}{\mu_{12}} = 0.3, \quad \frac{1}{\mu_{13}} = 0.5,$$
$$\frac{1}{\mu_{21}} = 0.6, \quad \frac{1}{\mu_{22}} = 0.5, \quad \frac{1}{\mu_{23}} = 0.8.$$

The visit ratios are:

$$e_{11} = 2, \quad e_{12} = 1, \quad e_{13} = 1, \quad e_{21} = 1, \quad e_{22} = 0.5, \quad e_{23} = 0.4.$$

STEP 1 Initialize. Determine the utilizations of the open class:

$$\rho_{11} = \lambda_1 e_{11} \cdot \frac{1}{\mu_{11}} = \underline{0.8}, \quad \rho_{21} = \lambda_1 e_{21} \cdot \frac{1}{\mu_{21}} = \underline{0.6}.$$

Set $\overline{K}_{12}(\mathbf{0}) = \overline{K}_{13}(\mathbf{0}) = \overline{K}_{22}(\mathbf{0}) = \overline{K}_{23}(\mathbf{0}) = 0$.

STEP 2 Analyze the model using the MVA.

MVA-iteration for $\mathbf{k} = (0,0)$:

STEP 2.4 Mean number of jobs in the open class 1:
Node 1, Eq. (8.51):

$$\overline{K}_{11}(0,0) = \frac{\rho_{11}\left[1 + \overline{K}_{12}(0,0) + \overline{K}_{13}(0,0)\right]}{1 - \rho_{11}} = \underline{4}.$$

Node 2, Eqs. (10.88) and (10.92):

$$\overline{K}_{21}(0,0) = \lambda_1 \cdot e_{21} \cdot \frac{1/\mu_{21}}{1 - \rho_{21}} = \underline{1.5}.$$

MVA-iteration for $\mathbf{k} = (0,1)$:

STEP 2.1 Mean response time for the closed class 3: We use Eqs. (8.53) and (10.90) to get:

$$\overline{T}_{13}(0,1) = \frac{1}{\mu_{13}} \cdot \frac{1}{1 - \rho_{11}} = \underline{2.5}, \quad \overline{T}_{23}(0,1) = \frac{\frac{1}{\mu_{23}} + \frac{\overline{K}_{21}(0,0)}{\mu_{21}}}{1 - \rho_{21}} = \underline{4.25}.$$

STEP 2.2 Throughput of class 3, from Eq. (8.46):

$$\lambda_3(0,1) = \underline{0.238}.$$

STEP 2.3 Mean number of jobs in class 3, from Eq. (8.47):

$$\overline{K}_{13}(0,1) = \underline{0.595}, \quad \overline{K}_{23}(0,1) = \underline{0.405}.$$

STEP 2.4 Mean number of jobs in the open class 1, using Eqs. (8.51), (10.88), and (10.92), respectively:

$$\overline{K}_{11}(0,1) = \underline{6.38}, \quad \overline{K}_{21}(0,1) = \underline{1.5}.$$

MVA-iteration for $\mathbf{k} = (1,0)$:

STEP 2.1 $\overline{T}_{12}(1,0) = \underline{1.5}, \quad \overline{T}_{22}(1,0) = \underline{3.5},$

STEP 2.2 $\lambda_2(1,0) = \underline{0.308},$

STEP 2.3 $\overline{K}_{12}(1,0) = \underline{0.462}, \quad \overline{K}_{22}(1,0) = \underline{0.538},$

STEP 2.4 $\overline{K}_{11}(1,0) = \underline{5.85}, \quad \overline{K}_{21}(1,0) = \underline{1.5}.$

MVA-iteration for $\mathbf{k} = (1,1)$:

STEP 2.1 $\overline{T}_{12}(1,1) = \underline{2.393}, \quad \overline{T}_{22}(1,1) = \underline{3.5},$

STEP 2.2 $\lambda_2(1,1) = \underline{0.241},$

STEP 2.3 $\overline{K}_{12}(1,1) = \underline{0.578}, \quad \overline{K}_{22}(1,1) = \underline{0.422},$

STEP 2.1 $\overline{T}_{13}(1,1) = \underline{3.654}, \quad \overline{T}_{23}(1,1) = \underline{4.923},$

STEP 2.2 $\lambda_3(1,1) = \underline{0.178},$

STEP 2.3 $\overline{K}_{13}(1,1) = \underline{0.65}, \quad \overline{K}_{23}(1,1) = \underline{0.35},$

STEP 2.4 $\overline{K}_{11}(1,1) = \underline{8.91}, \quad \overline{K}_{21}(1,1) = \underline{1.5}.$

These are the final results for the given mixed priority network.

Up to now we considered a pure HOL service strategy or a pure PRS service strategy at a node. In many cases we may need to combine these two priority strategies. Let us therefore consider a combination of PRS and HOL:

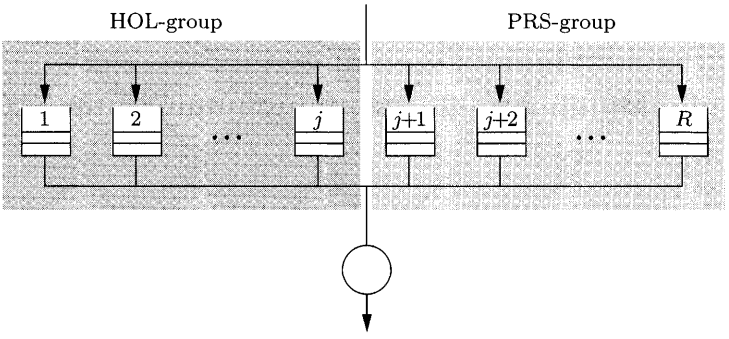


Fig. 10.24 Splitting into HOL and PRS classes.

In this combination, some job classes are processed by HOL and some job classes are processed by PRS. It is assumed that HOL jobs always have higher priority than PRS jobs (see Fig. 10.24). In this case:

- An arriving HOL job cannot preempt a HOL job that is still in progress, but it can preempt a PRS job that is still in progress. Therefore the response time of a HOL job is not influenced by the arrival of PRS jobs.

$$\overline{T}_r = \sum_{s=1}^{r} \frac{\overline{K}_s - \rho_s}{\mu_s} + \sum_{s=1}^{j} \frac{\rho_s}{\mu_s} + \sum_{s=1}^{r-1} \left(\overline{T}_r - \frac{1}{\mu_r} \right) \cdot \frac{\lambda_s}{\mu_s} + \frac{1}{\mu_r}.$$

By simplifying the above expression we get, in analogy to Eq. (10.91):

$$\overline{T}_{ir}(\mathbf{k}) = \frac{1}{\mu_{ir}} + \frac{\sum_{s=1}^{r} \frac{\overline{K}_{is}(\mathbf{k}-\mathbf{1}_r)}{\mu_{is}} + \sum_{s=r+1}^{j} \frac{\rho_{is}(\mathbf{k}-\mathbf{1}_r)}{\mu_{is}}}{1 - \sum_{s=1}^{r-1} \tilde{\rho}_{is}}, \tag{10.94}$$

$$\tilde{\rho}_{is} = \rho_{is}(\mathbf{k} - \overline{K}_{is}).$$

- A PRS job can always be preempted by a job with higher priority. For priority class r we therefore get:

$$\overline{T}_r = \sum_{s=1}^{r} \frac{\overline{K}_s}{\mu_s} + \sum_{s=1}^{r-1} \frac{\overline{T}_r \cdot \lambda_s}{\mu_s} + \frac{1}{\mu_r}.$$

In analogy to Eq. (10.90) we get after simplification:

$$\overline{T}_{ir}(\mathbf{k}) = \frac{\frac{1}{\mu_{ir}} + \sum_{s=1}^{r} \frac{\overline{K}_{is}(\mathbf{k}-\mathbf{1}_r)}{\mu_{is}}}{1 - \sum_{s=1}^{r-1} \tilde{\rho}_{is}}, \tag{10.95}$$

$$\tilde{\rho}_{is} = \rho_{is}(\mathbf{k} - \overline{K}_{is}).$$

If these formulae for $\overline{T}_{ir}(\mathbf{k})$ are used in the MVA instead of $\overline{T}_{ir}$, then we get an approximate MVA algorithm for closed queueing networks with a mixed priority strategy. In the next section we introduce the technique of shadow server that can also be used to analyze priority networks.

10.3.2 The Method of Shadow Server

10.3.2.1 The Original Shadow Technique This technique, shown in Fig 10.25, was introduced by [Sevc77] for networks with a pure PRS priority strategy. The idea behind this technique is to split each node in the network into R parallel nodes, one for each priority class. The problem then is that in this extended node, jobs can run in parallel whereas this is not possible in the original node. To deal with this problem and to simulate the effect of preemption, we iteratively increase the service time for each job class in the

shadow node. The lower the priority of a job, the higher is the service time for the job. After all iterations we have:

$$\tilde{s}_{i,r} \geq s_{i,r}.$$

Here $\tilde{s}_{i,r} = 1/\mu_{ir}$ denotes the approximate service time of class-r jobs at node i in the shadow node.

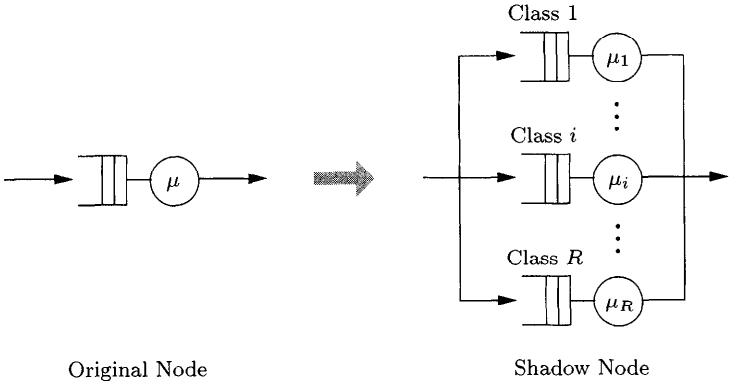


Fig. 10.25 The idea behind the shadow technique.

Shadow Algorithm

STEP 1 Transform the original model into the shadow model.

STEP 2 Set $\lambda_{i,r} = 0$.

STEP 3 Iterate:

STEP 3.1 Compute the utilization for each shadow node:

$$\tilde{\rho}_{i,r} = \lambda_{i,r} \cdot \tilde{s}_{i,r}. \tag{10.96}$$

STEP 3.2 Compute the shadow service times:

$$\tilde{s}_{i,r} = \frac{s_{i,r}}{1 - \sum_{s=1}^{r-1} \tilde{\rho}_{i,s}}. \tag{10.97}$$

Here $s_{i,r}$ denotes the original service time of a class-r job at node i in the original network while $\tilde{s}_{i,r}$ is the approximated service time in the shadow network.

STEP 3.3 Compute the new values of the throughput of class r at node i, $\lambda_{i,r}$, of the shadow model using any standard analysis technique from Chapters 8 and 9. If the $\lambda_{i,r}$ differ less than ε in two successive iterations, then stop the iteration. Otherwise go back to Step 3.1

10.3.2.2 Extensions of the Shadow Technique

10.3.2.2.1 Extension of Kaufmann Measurements on real systems and comparisons with discrete-event simulation results have shown that the technique of shadow server is not very accurate. Because of the lack of accuracy, [Kauf84] suggested an extension of the original shadow technique. This extended method differs from the original method in that an improved expression $\hat{\rho}_{i,r}$ is used instead of $\tilde{\rho}_{i,r}$. To derive this improved utilization we define:

$$\begin{aligned} c &= \text{class of the job under consideration,} \\ N_{i,r} &= \text{number of class-}r\text{ jobs at node } i \text{ in the shadow node,} \\ s_{i,r}^{pt} &= \text{approximated mean service position time,} \\ M_{i,r} &- \sum_{l=1}^{r-1} N_{i,l}. \end{aligned}$$

For a pure PR strategy the following relations hold:

- The improved utilization $\hat{\rho}_{ir}$ is expressed using $s_{i,r}^{pt}$:

$$\begin{aligned} \hat{\rho}_{i,r} &= \lambda_{i,r} \cdot s_{i,r}^{pt}, \\ \hat{\rho}_{i,r} &= P(N_{i,r} > 0), \\ \lambda_{i,r} &= P(N_{i,1} = 0 \wedge N_{i,2} = 0 \wedge \cdots \wedge N_{i,r-1} = 0 \\ &\quad \wedge N_{i,r} > 0) \cdot \mu_{i,r} \\ &= P(M_{i,r} = 0 \wedge N_{i,r} > 0) \cdot \mu_{i,r}, \\ P(N_{i,r} > 0) &= P(M_{i,r} = 0 \wedge N_{i,r} > 0) \cdot \mu_{i,r} \cdot s_{i,r}^{pt}. \end{aligned}$$

- The last equation can now be used to derive an expression for s_{ir}^{pt}:

$$\begin{aligned} s_{i,r}^{pt} &= \frac{P(N_{i,r} > 0)}{P(M_{i,r} = 0 \wedge N_{i,r} > 0)} \cdot \mu_{i,r}^{-1} \\ &= \frac{1}{\dfrac{P(M_{i,r} = 0 \wedge N_{i,r} > 0)}{P(N_{i,r} > 0)}} \cdot \mu_{i,r}^{-1}. \end{aligned}$$

- By using conditional probabilities the last expression can be rewritten into:

$$s_{i,r}^{pt} = \frac{1}{P(M_{i,r} = 0 \mid N_{i,r} > 0)} \cdot \mu_{i,r}^{-1}.$$

- Now we rewrite the numerator of the last expression and define the correction factor $\delta_{i,r}$ which we use to express $\hat{\rho}_{i,r}$:

$$s_{i,r}^{pt} = \frac{1}{1 - P(M_{i,r} > 0 \wedge c \neq r \mid N_{i,r} > 0)} \cdot \mu_{i,r}^{-1}.$$

With:

$$\delta_{i,r} = 1 - P(M_{i,r} > 0 \wedge c \neq r \mid N_{i,r} > 0)$$

and:

$$\tilde{\rho}_{i,r} = \lambda_{i,r} \cdot \tilde{s}_{i,r}, \quad \hat{\rho}_{i,r} = \lambda_{i,r} \cdot s_{i,r}^{pt},$$

we obtain:

$$\hat{\rho}_{i,r} = \lambda_{i,r} \cdot \tilde{s}_{i,r} \cdot \frac{1}{\delta_{i,r}} = \tilde{\rho}_{i,r} \cdot \frac{1}{\delta_{i,r}}. \tag{10.98}$$

As we see, the improved utilization $\hat{\rho}_{i,r}$ is just given by dividing the utilization $\tilde{\rho}_{i,r}$ by the correction factor $\delta_{i,r}$. In [Kauf84] it is shown that the following approximation holds:

$$\delta_{i,r} = \frac{\rho(r)[1-\rho(r)] + \alpha(r) \cdot \rho(r+1)}{\rho(r)[1-\rho(r)]^2 + \alpha(r) \cdot \rho_{i,r}} \quad \text{for } r = 2, 3, \ldots, R, \tag{10.99}$$

with:

$$\alpha(r) = \sum_{k=1}^{r-1} \frac{\tilde{\rho}_{i,r}}{\omega_{r,k}}, \quad \omega_{r,k} = \frac{s_{i,k}}{s_{i,r}}, \quad \rho(r) = \sum_{k=1}^{r-1} \tilde{\rho}_{i,k}.$$

This correction factor δ changes only Step 3.2 in the original shadow algorithm:

$$\tilde{s}_{i,r} = \frac{s_{i,r}}{1 - \sum_{s=1}^{r-1} \frac{1}{\delta_{i,s}} \cdot \tilde{\rho}_{i,s}}. \tag{10.100}$$

If the utilizations of the shadow nodes are very close to 1, then the expression:

$$1 - \sum_{s=1}^{r-1} \frac{1}{\delta_{i,s}} \cdot \tilde{\rho}_{i,s}$$

can be negative because of the approximate technique. In this case the iteration must be stopped or the extended shadow technique with bisection can be used.

In the original and extended shadow method the expression:

$$\sum_{s=1}^{r-1} \tilde{\rho}_{i,s}$$

can be very close to 1. Then the service time $s_{i,r}$ gets very large and the throughput through this node is very low. In addition, the value of:

$$\sum_{s=1}^{r-1} \tilde{\rho}_{i,s}$$

will be very small in the next iteration step and in some cases the shadow technique does not converge but the computed values move back and forth between the two values. Whenever such swinging occurs, we simply use the average of the last two values of throughput in Step 3.3. Thus we have used bisection.

10.3.2.2.2 Class Switching If class switching is allowed in the network then we use the shadow technique in combination with the concept of chains (see Section 7.3.6.1). The resulting algorithm can now be given in the following steps:

STEP 1 Determine the chains in the network.

STEP 2 Transform the original model into the shadow model.

STEP 3 Set $\lambda_{i,r} = 0$.

STEP 4 Iterate:

STEP 4.1

$$\tilde{\rho}_{i,r} = \lambda_{i,r} \cdot \tilde{s}_{i,r}.$$

STEP 4.2

$$\tilde{s}_{i,r} = \frac{s_{i,r}}{1 - \sum_{s=1}^{r-1} \frac{1}{\delta_{i,s}} \cdot \tilde{\rho}_{i,s}}.$$

Here $s_{i,r}$ is the original service time of a class-r job at node i in the original network.

STEP 4.3 Transform the class values into chain values (to differentiate between class values and chain values, the chain values are marked with *).

STEP 4.4 Compute the performance parameters of the transformed shadow model with the MVA. If the model parameters swing, then set

$$\overline{T}_{i,r} = \frac{\overline{T}_{i,r}^{\text{old}} + \overline{T}_{i,r}^{\text{new}}}{2}$$

and compute the performance measures with this new response time.

STEP 4.5 Transform chain values into class values. If the $\lambda_{i,r}^*$ (chain value) in two successive steps differs less than ε, stop the iteration. Otherwise go back to Step 4.1.

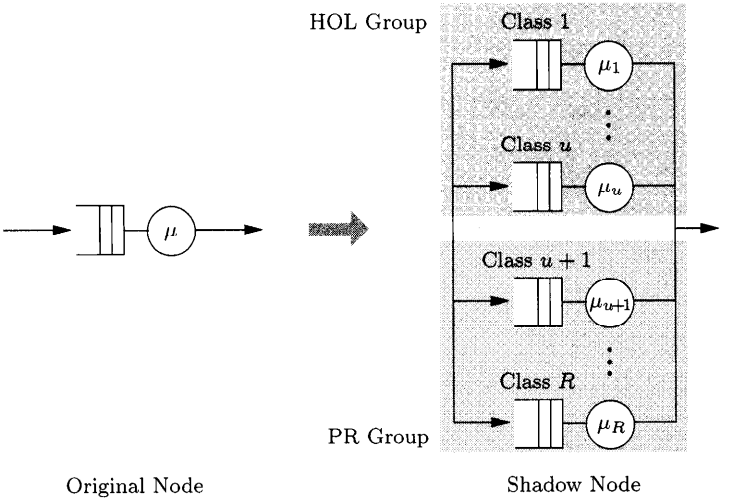


Fig. 10.26 Shadow technique and mixed strategy.

10.3.2.2.3 Class Switching and Mixed Priorities The original shadow technique [Sevc77] can only be used for networks with a pure PRS-strategy, but in the models of a UNIX-based operating system (see Section 13.1.4), for example, or in models of cellular mobile networks [GBB98], a mixed priority strategy is used. For this reason the original shadow technique is extended to deal with mixed priority strategies.

Consider the situation where the classes $1, 2, \ldots, u$ are pure HOL classes and $u+1, u+2, \ldots, R$ are pure PRS classes, as shown in Fig. 10.26. We define:

$$y_{i,r} = \sum_{l=1, l \neq r}^{u} N_{i,l},$$

and get the following relations:

- The utilizations can be expressed as:

$$\hat{\rho}_{i,r} = \lambda_{i,r} \cdot s_{i,r}^{pt},$$
$$\hat{\rho}_{i,r} = P(N_{i,r} > 0).$$

- A PRS job can only be served if no other job with higher priority is waiting:

$$\lambda_{i,r} = P(M_{i,r} = 0 \wedge N_{i,r} > 0) \cdot \mu_{i,r}.$$

- For HOL jobs, two cases have to be distinguished:

1. Class-r jobs as well as HOL jobs are at the node and a class-r job that is just served. Then we get:

$$N_{i,r} > 0 \quad \wedge \quad c = r \quad \wedge \quad y_{i,r} > 0,$$

or

2. Class-r jobs but no HOL jobs are waiting for service at a node. Then we get:

$$N_{i,r} > 0 \quad \wedge \quad y_{i,r} = 0.$$

For the HOL classes we therefore get:

- The utilization of node i by class-r HOL jobs is given by the probability that at least one class-r job is at node i (and is served) and other HOL jobs are waiting, or at least one class-r job is at node i and no other HOL jobs are waiting for service:

$$\begin{aligned}\lambda_{i,r} &= \rho_{i,r} \cdot \mu_{i,r} \\ &= [P(N_{i,r} > 0 \wedge c = r \wedge y_{i,r} > 0) + P(N_{i,r} > 0 \wedge y_{i,r} = 0)] \cdot \mu_{i,r} \\ &= [P(N_{i,r} > 0) - P(N_{i,r} > 0 \wedge y_{i,r} > 0 \wedge c \neq r)] \cdot \mu_{i,r}.\end{aligned}$$

- As in the case of a pure PRS strategy, we get analogously:

$$\begin{aligned}s_{i,r}^{pt} &= \frac{P(N_{i,r} > 0)}{P(N_{i,r} > 0) - P(N_{i,r} > 0 \wedge y_{i,r} > 0 \wedge c \neq r)} \cdot \mu_{i,r}^{-1} \\ &= \frac{1}{1 - \dfrac{P((y_{i,r} > 0 \wedge c \neq r) \wedge N_{i,r} > 0)}{P(N_{i,r} > 0)}} \cdot \mu_{i,r}^{-1}.\end{aligned}$$

- The fraction in the numerator is now rewritten using conditional probabilities:

$$s_{i,r}^{pt} = \frac{1}{1 - P(y_{i,r} > 0 \wedge c \neq r) \mid N_{i,r} > 0)} \cdot \mu_{i,r}^{-1}.$$

- As before, we now define the correction factor $\psi_{i,r}$ as:

$$\psi_{i,r} = 1 - P(y_{i,r} > 0 \wedge c \neq r \mid N_{i,r} > 0),$$

and get:

$$\hat{\rho}_{i,r} = \frac{1}{\psi_{i,r}} \cdot \tilde{\rho}_{i,r}. \tag{10.101}$$

As we see, the correction factor for the HOL case differs from the PRS case only in the summation index. We then get to the following equations:

$$\psi_{ij,r} = \frac{\varrho(r)[1-\varrho(r)] + \beta(r)\cdot\tilde{\varrho}(r)}{\varrho(r)[1-\varrho(r)]^2 + \beta(r)\cdot\varrho_{i,r}}, \tag{10.102}$$

$$\beta(r) = \begin{cases} \sum_{k=1,k\neq r}^{u} \frac{\tilde{\rho}_{i,r}}{\omega_{r,k}}, & \text{if } r \in \text{HOL}, \\ \sum_{k=1}^{r-1} \frac{\tilde{\rho}_{i,r}}{\omega_{r,k}}, & \text{if } r \in \text{PR}, \end{cases}$$

$$\omega_{r,k} = \frac{s_{i,k}}{s_{i,r}},$$

$$\varrho(r) = \begin{cases} \sum_{k=1,k\neq r}^{u} \tilde{\rho}_{i,k}, & \text{if } r \in \text{HOL}, \\ \sum_{k=1}^{r-1} \tilde{\rho}_{i,k}, & \text{if } r \in \text{PR}, \end{cases}$$

$$\tilde{\varrho}(r) = \varrho(r) + \tilde{\rho}_{i,r}.$$

10.3.2.2.4 Class Switching and Arbitrary Mixed Priorities Up to now we assumed that the classes $1,\ldots,u$ are pure HOL classes and the classes $u+1,\ldots,R$ are pure PRS classes. In the derived formulae we can see that in the HOL case, the summation in the correction factor is over all HOL classes excluding the actually considered class. In the PRS case, the summation is over all classes with higher priority. With this relation in mind it is possible to extend the formulae in such a way that every mixing of PRS and HOL classes is possible. Let ξ_r be the set of jobs that cannot be preempted by a class-r job, $r \notin \xi_r$. For the computation of the correction factor, this mixed priority strategy has the consequence that:

- For an HOL class r the summation is not from 1 to u but over all priority classes $s \in \xi_r$.
- For a PRS class r the summation is not from 1 to $r-1$ but over all priority classes $s \in \xi_r$.

For any mixed priority strategy we get the following formulae:

$$\tilde{s}_{i,r} = \frac{s_{i,r}}{1 - \sum_{s\in\xi_r} \frac{1}{\psi_{i,s}}\cdot\tilde{\rho}_{i,s}},$$

$$\psi_{i,r} = \frac{\varrho(r)[1-\varrho(r)] + \beta(r)\cdot\tilde{\varrho}(r)}{\varrho(r)[1-\varrho(r)]^2 + \beta(r)\cdot\varrho_{i,r}}, \tag{10.103}$$

$$\beta(r) = \sum_{k \in \xi_r} \frac{\tilde{\rho}_{ij,r}}{\omega_{r,k}}, \qquad \omega_{r,k} = \frac{s_{ik}}{s_{ir}},$$

$$\varrho(r) = \sum_{k \in \xi_r} \tilde{\rho}_{i,k}, \qquad \tilde{\varrho}(r) = \sum_{k \in \{\xi_r \cup r\}} \tilde{\rho}_{i,k}.$$

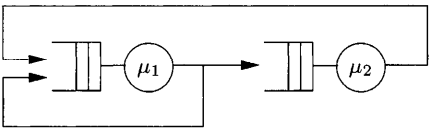


Fig. 10.27 Simple network for Example 10.13.

Example 10.13 Consider the network, given in Fig. 10.27. Node 1 is a -/M/1-PRIORITY node and node 2 is a -/M/1-FCFS node. The network contains two priority job classes, Class 1: PRS, and Class 2: HOL, and class switching is allowed. The routing probabilities are given by:

$$p_{11,11} = 0.5, \quad p_{11,12} = 0.4, \quad p_{11,21} = 0.1,$$
$$p_{12,11} = 0.5, \quad p_{12,12} = 0.4, \quad p_{12,22} = 0.1,$$
$$p_{21,11} = 1.0, \quad p_{22,11} = 1.0.$$

The other parameters are:

$$s_{11} = 0.1, \quad s_{12} = 0.1, \quad s_{21} = 1.0, \quad s_{22} = 1.0,$$
$$K = 3, \quad \epsilon = 0.001.$$

STEP 1 Compute the visit ratios e_{ir} in the network. For this we use the formula:

$$e_{ir} = \sum_{j=1}^{2} \sum_{s=1}^{2} e_{js} p_{js,ir}$$

and get $e_{11} = 1.5$, $e_{12} = 1.0$, $e_{21} = 0.15$, $e_{22} = 0.1$.

STEP 2 Determine the chains in the network and compute the number of jobs in each chain (refer to Section 7.3.6). In Fig. 10.28 the DTMC state diagram for the routing matrix for the chains is shown, and it can easily be seen that we can reach each state from every other state (in a state (i, r), i denotes the node number and r denotes the class number). Therefore we have only one chain in the network.

STEP 3 Transform the network into the shadow network, as it is shown in Fig. 10.29.

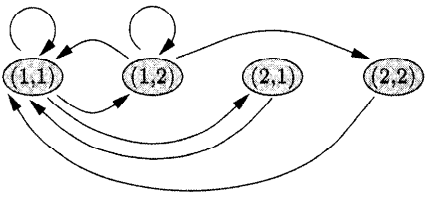


Fig. 10.28 DTMC state diagram of the routing matrix.

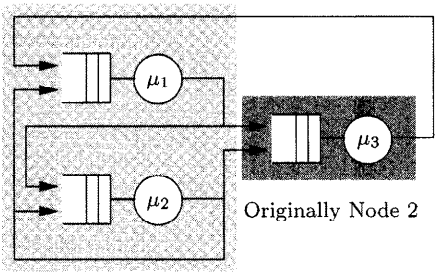


Fig. 10.29 The Shadow model for the network in Example 10.13.

The corresponding parameters are:

$$
\begin{aligned}
&s_{11} = 0.1, \quad e_{11} = 1.5, \quad e_{31} = 0.15,\\
&s_{12} = 0,\\
&s_{21} = 0,\\
&s_{22} = 0.1, \quad e_{12} = 0.0, \quad e_{32} = 0.1,\\
&s_{31} = 1.0, \quad e_{21} = 0.0,\\
&s_{32} = 1.0, \quad e_{22} = 1.0.
\end{aligned}
$$

STEP 4 Compute the visit ratios e^*_{iq} per chain and remember that we have only one chain:

$$e^*_{iq} = \frac{\sum\limits_{r \in q} e_{ir}}{\sum\limits_{r \in q} e_{1r}}.$$

Then we get:

$$e^*_{11} = \frac{e_{11} + e_{12}}{e_{11} + e_{12}} = 1, \quad e^*_{21} = \frac{e_{21} + e_{22}}{e_{11} + e_{12}} = \frac{2}{3}, \quad e^*_{31} = \frac{e_{31} + e_{32}}{e_{11} + e_{12}} = \frac{1}{6}.$$

STEP 5 Compute the scale factors α_{ir}. For this we use the equation:

$$\alpha_{ir} = \frac{e_{ir}}{\sum\limits_{s \in q} e_{is}},$$

and get:

$$\alpha_{11} = \frac{e_{11}}{e_{11}+e_{12}} = 1.0, \quad \alpha_{12} = \frac{e_{12}}{e_{11}+e_{12}} = 0,$$
$$\alpha_{21} = \frac{e_{21}}{e_{21}+e_{22}} = 0, \quad \alpha_{22} = \frac{e_{22}}{e_{21}+e_{22}} = 1.0,$$
$$\alpha_{31} = \frac{e_{31}}{e_{31}+e_{32}} = 0.6, \quad \alpha_{32} = \frac{e_{32}}{e_{31}+e_{32}} = 0.4.$$

STEP 6 Compute the service times per chain that are given by the equation:

$$s^*_{iq} = \frac{1}{\mu_{iq}} = \sum_{r \in q} s_{ir} \cdot \alpha_{ir},$$

which gives:

$$s^*_{11} = s_{11} \cdot \alpha_{11} + s_{12} \cdot \alpha_{12} = 0.1,$$
$$s^*_{21} = s_{21} \cdot \alpha_{21} + s_{22} \cdot \alpha_{22} = 0.1,$$
$$s^*_{31} = s_{31} \cdot \alpha_{31} + s_{32} \cdot \alpha_{32} = 1.0.$$

Now we can analyze the shadow network given in Fig. 10.29 using the following parameters:

$$e^*_{11} = 1.0, \quad e^*_{21} = \frac{2}{3}, \quad e^*_{31} = \frac{1}{6},$$
$$s^*_{11} = 0.1, \quad s^*_{21} = 0.1, \quad s^*_{31} = 1.0,$$
$$\alpha_{11} = 1.0, \quad \alpha_{21} = 0.0, \quad \alpha_{31} = 0.6,$$
$$\alpha_{12} = 0.0, \quad \alpha_{22} = 1.0 \quad \alpha_{32} = 0.4,$$
$$K = 3, \quad N = 3, \quad \epsilon = 0.001,$$
$$\xi_1 = \{2\}, \quad \xi_2 = \{1\}.$$

STEP 7 Start the shadow iterations.

1. Iteration:

$$\rho^*_{11} = 0.505, \quad \rho^*_{21} = 0.337, \quad \rho^*_{31} = 0.841,$$
$$\lambda^*_{11} = 5.049, \quad \lambda^*_{21} = 3.366, \quad \lambda^*_{31} = 0.841,$$
$$\tilde{s}^*_{11} = 0.117, \quad \tilde{s}^*_{21} = 0.145, \quad \tilde{s}^*_{31} = 1.000.$$

2. Iteration:

$$\rho^*_{11} = 0.541, \quad \rho^*_{21} = 0.447, \quad \rho^*_{31} = 0.766,$$
$$\lambda^*_{11} = 4.597, \quad \lambda^*_{21} = 3.065 \quad \lambda^*_{31} = 0.766,$$
$$\tilde{s}^*_{11} = 0.125, \quad \tilde{s}^*_{21} = 0.142, \quad \tilde{s}^*_{31} = 1.000.$$

3. Iteration:

$$\rho_{11}^* = 0.569, \quad \rho_{21}^* = 0.430, \quad \rho_{31}^* = 0.756,$$
$$\lambda_{11}^* = 4.533, \quad \lambda_{21}^* = 3.022, \quad \lambda_{31}^* = 0.756,$$
$$\tilde{s}_{11}^* = 0.122, \quad \tilde{s}_{21}^* = 0.147, \quad \tilde{s}_{31}^* = 1.000.$$

4. Iteration:

$$\rho_{11}^* = 0.556, \quad \rho_{21}^* = 0.447, \quad \rho_{31}^* = 0.755,$$
$$\lambda_{11}^* = 4.531, \quad \lambda_{21}^* = 3.012, \quad \lambda_{31}^* = 0.755,$$
$$\tilde{s}_{11}^* = 0.124, \quad \tilde{s}_{21}^* = 0.144, \quad \tilde{s}_{31}^* = 1.000.$$

5. Iteration:

$$\rho_{11}^* = 0.565, \quad \rho_{21}^* = 0.437, \quad \rho_{31}^* = 0.754,$$
$$\lambda_{11}^* = 4.528, \quad \lambda_{21}^* = 3.018, \quad \lambda_{31}^* = 0.754.$$

Now we stop the iteration because the difference between the λ^* in the last two iteration steps is smaller than ϵ.

STEP 8 Retransform the chain values into class values:

$$\lambda_{11} = \alpha_{11} * \lambda_{11}^* = 4.528,$$
$$\lambda_{22} = \alpha_{22} * \lambda_{21}^* = 3.018,$$
$$\lambda_{31} = \alpha_{31} * \lambda_{31}^* = 0.452,$$
$$\lambda_{32} = \alpha_{32} * \lambda_{31}^* = 0.301.$$

If we rewrite the results of the shadow network in terms of the original network, we get the following final results:

$$\lambda_{11}^{(\text{approx})} = \lambda_{11} = 4.528, \quad \lambda_{12}^{(\text{approx})} = \lambda_{22} = 3.018,$$
$$\lambda_{21}^{(\text{approx})} = \lambda_{31} = 0.452, \quad \lambda_{22}^{(\text{approx})} = \lambda_{32} = 0.301.$$

STEP 9 Verify the results. To verify these results we generate and solve the underlying CTMC via MOSES [BoHe96] (see Chapter 12). The exact results for this network are given by:

$$\lambda_{11}^{(\text{exact})} = 4.52, \quad \lambda_{12}^{(\text{exact})} = 3.01,$$
$$\lambda_{21}^{(\text{exact})} = 0.45, \quad \lambda_{22}^{(\text{exact})} = 0.30.$$

As we see, the differences between the exact MOSES results $\lambda_{ij}^{(\text{exact})}$ and the approximated results $\lambda_{ij}^{(\text{approx})}$ are very small.

10.3.2.2.5 Quota Nodes The idea of quota is to assign to each user in the system the percentage of CPU power he or she can use. If, for example, we have three job classes and the quota are given by $q_1 = 0.3$, $q_2 = 0.2$ and $q_3 = 0.5$, then class 1 customers get 30% of the overall CPU time, class 2 customers get 20% CPU time, and class 3 customers get 50% of the overall CPU time. In an attempt to enforce this quota, we give priority to jobs so that the priority of job class r, pri_r, is:

$$\text{pri}_r = \frac{q_r}{\rho_{i,r}}. \tag{10.104}$$

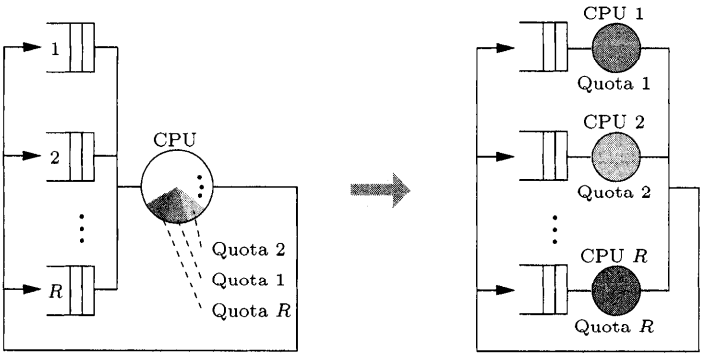


Fig. 10.30 Transformation of a quota node into the corresponding shadow node.

The higher the pri_r, the higher is the priority of class r. As we can see, the priority of class-r jobs depends on the CPU utilization at node i and the quota q_r of class r. A high priority job class is influenced only a little by lower priority job classes, while lower priority job classes are influenced much more by higher priority job classes. A measure for this influence is the weight $\tilde{\omega}_{r,j}$, which is defined as:

$$\tilde{\omega}_{r,j} = \begin{cases} \dfrac{\text{pri}_j^2}{\text{pri}_j^2 + \text{pri}_r^2}, & \text{if } r \neq j, \\ 1, & \text{if } r = j. \end{cases} \tag{10.105}$$

The weights $\tilde{\omega}_{r,j}$ are a measure for how much the rth job class is influenced by the jth job class. The closer these weights are to one, the higher is the influence. Since an active job influences itself directly ($r = j$), we have $\tilde{\omega}_{r,r} = 1$. The mean virtual service time of a class-r job can approximately be given as:

$$s_{(i,r)}^{\text{virt}} = \frac{s_{i,r}}{1 - \sum_{k=1}^{R} \rho_{i,r} \cdot \tilde{\omega}_{r,k}}. \tag{10.106}$$

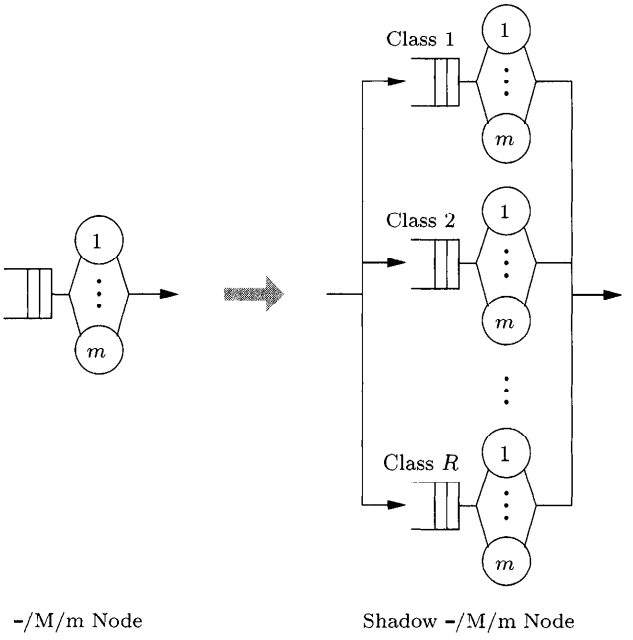


Fig. 10.31 Shadow transformation of a -/M/m node.

If we compare this equation with the shadow equation to compute the shadow service time, then the basic structure of the equations are the same. The idea, therefore, is to use the weights $\tilde{\omega}_{r,j}$ to include quota into the shadow technique. In order to transform a quota node with R job classes into a shadow node, we apply the shadow technique as shown in Fig. 10.30. The resulting equation for computing the service time at each shadow node can now be given as:

$$\tilde{s}_{i,r} = \begin{cases} \dfrac{s_{i,r}}{1 - \sum\limits_{\substack{k=1 \\ k \neq r}}^{R} \frac{\tilde{\omega}_{r,k}}{\psi_{i,k}} \cdot \tilde{\rho}_{i,k}} & \text{if } r \in \text{HOL}, \\ \dfrac{s_{i,r}}{1 - \sum\limits_{k=1}^{r-1} \frac{\tilde{\omega}_{r,k}}{\psi_{i,k}} \cdot \tilde{\rho}_{i,k}} & \text{if } r \in \text{PRS}, \end{cases} \tag{10.107}$$

$$\tilde{\omega}_{i,k} = \frac{\text{pri}_k^2}{\text{pri}_k^2 + \text{pri}_i^2}, \quad \text{pri}_k = \frac{q_k}{\rho_{i,k}}.$$

By using the weights $\tilde{\omega}_{i,k}$, we make sure that each class gets the amount of CPU time as specified by the CPU quota. In the case of an arbitrary mixed priority strategy, we use the summation index ξ as defined before.

10.3.2.2.6 Extended Nodes We now introduce a new node type, the so-called *extended -/M/m node.* It is very easy to analyze a regular -/M/m node since we only have to transform it into the corresponding shadow form and use the equations for -/M/m nodes when analyzing the node. The transformation of a -/M/m node into its corresponding shadow form is shown in Fig. 10.31.

The node shown in Fig. 10.32 is called *extended -/M/m node.* This node contains R priority classes that are priority ordered (job class 1 has highest priority, job class R has lowest priority). The CPU can process all job classes, while at the *APU* (associate processing unit), only the job classes $u, \ldots, R$ $(1 \leq u \leq R)$ can be processed. For a better overview, a queue

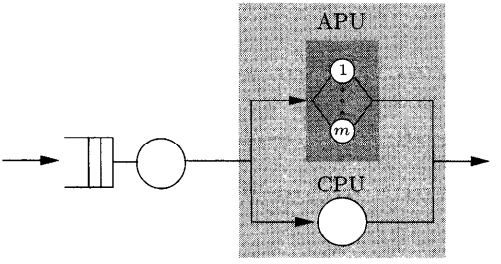


Fig. 10.32 Extended -/M/m node (original form).

is introduced for each job class. The result of this transformation is shown in Fig. 10.33. For example, let $u = R = 3$, then the job classes 1, 2, and 3 can

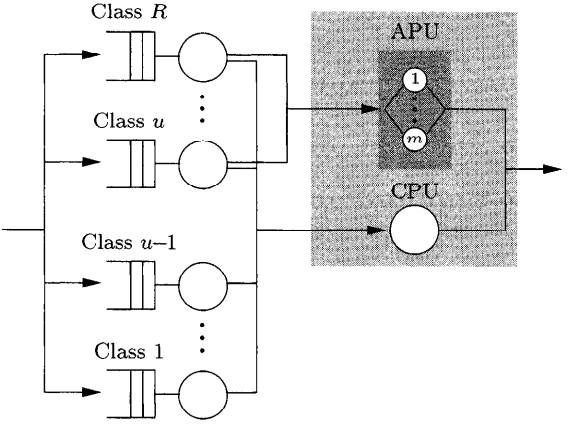


Fig. 10.33 Extended -/M/m node with priority queues.

enter the CPU, while the APU can only be entered by class-3 jobs. Priority class-3 jobs can always be preempted at the CPU, but in this case they cannot be preempted at the APU because this job class is the only one entering the APU. In general, the actual priority of a job also depends on the node it

enters. If it enters the CPU, the priority order remains unchanged (priority class 1 has highest priority and priority class R has lowest priority), while at the APU priority, class-u jobs have highest priority. Since preemption is also possible at the APUs, if we have more than one priority class, the shadow transformation can be applied to each node, from which a switch to the APU is possible. The result of this transformation is shown in Fig. 10.34.

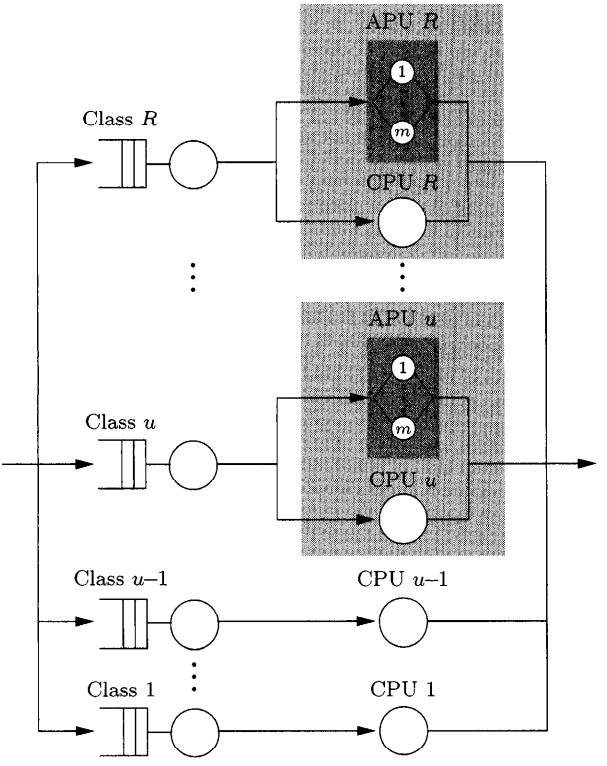


Fig. 10.34 Extended -/M/m node (complete shadow form).

The components $APU_u, \ldots, APU_R$ form the original APU in the network. Now it is possible to apply the shadow algorithm to the network transformed in this way. Because the priority order at the APU can change, we have to separate the APU iterations from the CPU iterations meaning that in one shadow step we first iterate over all CPUs and then over all APUs. Because of the separated iterations over CPU and APU, we usually get different final service times for CPU_i and APU_i, resulting in a so-called asymmetric node [BoRo94].

At the CPU, class-1 jobs have highest priority, while at the APUs, class-u jobs have highest priority. In the shadow iteration, only Step 3.2 is to be

changed as follows:

$$\forall c : \tilde{s}_{i,r} = \frac{s_{i,r}}{1 - \sum\limits_{s \geq c, s \in \xi_r} \frac{1}{\psi_{i,s}} \cdot \tilde{\rho}_{i,s}}, \quad c = 1, \ldots, R.$$

An application for this node type can be found in Section 13.1.4 where it plays an important role.

10.3.3 Extended SUM

The SUM method (see Section 9.2) can be applied to priority networks if we extend the formula for the mean number of jobs in the system, $\overline{K}_{ir}$ of an individual priority node as we did for an FCFS node with multiple job classes. The extended method is referred to as PRIOSUM. Here we use the formulae for single station queueing systems with priorities (see Section 6.15). In this context it is convenient to consider class 1 as the class with the highest priority and class R as the one with the lowest priority. Of course, this convention leads to minor changes in the formulae.

- -/M/1-FCFS-HOL node
  After we have introduced multiple classes and considered the reverse order of priorities, we derive the following formulae from Eqs. (6.106) and (6.107):

$$\overline{W}_{ir} = \frac{\overline{W}_{i0}}{(1 - \sigma_{ir})(1 - \sigma_{ir+1})}, \tag{10.108}$$

  with:

$$\sigma_{ir} = \begin{cases} \sum\limits_{j=1}^{r-1} \rho_{ij}, & r > 1, \\ 0, & \text{otherwise}, \end{cases} \tag{10.109}$$

  and the remaining service time:

$$\overline{W}_{i0} = \sum_{j=1}^{R} \frac{\rho_{ij}}{\mu_{ij}}. \tag{10.110}$$

  We obtain the final result for the mean number of jobs $\overline{K}_{ir}$ in node i with:

$$\overline{K}_{ir} = \rho_{ir} + \overline{Q}_{ir}, \tag{10.111}$$

  and the introduction of correction factors as in the FCFS case:

$$\overline{K}_{ir} = \rho_{ir} + \frac{\lambda_{ir} \sum\limits_{j=1}^{R} \frac{\rho_{ij}}{\mu_{ij}}}{\left(1 - \frac{K-2}{K-1}\sigma_{ir}\right)\left(1 - \frac{K-2}{K-1}\sigma_{ir+1}\right)}. \tag{10.112}$$

- -/M/m-FCFS-HOL node
  In this case we also can use Eqs. (10.108) and (10.109). Together with:

$$\overline{W}_{i0} = \frac{P_{m_i}(\rho_i)}{m_i \rho_i} \sum_{j=1}^{R} \frac{\rho_{ij}}{\mu_{ij}} \tag{10.113}$$

  and:

$$\overline{K}_{ir} = m_i \rho_{ir} + \overline{Q}_{ir}, \tag{10.114}$$

  we obtain:

$$\overline{K}_{ir} = m_i \rho_{ir} + \frac{\lambda_{ir} \sum_{j=1}^{R} \frac{\rho_{ij}}{\mu_{ij}}}{\left(1 - \frac{K-m_i-1}{K-m_i}\sigma_{ir}\right)\left(1 - \frac{K-m_i-1}{K-m_i}\sigma_{ir+1}\right)} \cdot \frac{P_{m_i}(\rho_i)}{m_i \rho_i}. \tag{10.115}$$

- -/G/1-FCFS-HOL node
  Again with Eqs. (10.108) and (10.109) and the remaining service time:

$$\overline{W}_{i0} = \frac{1}{2} \sum_{j=1}^{R} \lambda_{ij} \alpha_2(B_{ij}), \tag{10.116}$$

  we obtain:

$$\overline{K}_{ir} = \rho_{ir} + \frac{\frac{1}{2}\lambda_{ir} \sum_{j=1}^{R} \lambda_{ij} \alpha_2(B_{ij})}{\left(1 - \frac{K-1-a}{K-1}\sigma_{ir}\right)\left(1 - \frac{K-1-a}{K-1}\sigma_{ir+1}\right)}, \tag{10.117}$$

  with the second moment of the service time distribution:

$$\alpha_2(B_{ij}) = \frac{c_{ij}^2 + 1}{(\mu_{ij})^2}, \tag{10.118}$$

$$a = \frac{c_{ir}^2 + 1}{2}. \tag{10.119}$$

- -/G/m-FCFS-HOL node

$$\overline{W}_{i0} = \frac{P_{m_i}(\rho_i)}{2 m_i \rho_i} \sum_{j=1}^{R} \rho_{ij} \alpha_2(B_{ij}) \mu_{ij}, \tag{10.120}$$

$$\overline{K}_{ir} = m_i \rho_{ir} + \frac{\frac{1}{2}\lambda_{ir} \sum_{j=1}^{R} \rho_{ij} \alpha_2(B_{ij}) \mu_{ij}}{\left(1 - \frac{K-m_i-a}{K-m_i}\sigma_{ir}\right)\left(1 - \frac{K-m_i-a}{K-m_i}\sigma_{ir+1}\right)} \cdot \frac{P_{m_i}(\rho_i)}{m_i \rho_i}. \tag{10.121}$$

If *preemption* is allowed, we get:

$$\overline{W}_{ir} = \frac{1}{\rho_{ir}} \left(\sigma_{ir+1} w_{ir}^{(r)} - \sigma_{ir} w_{ir}^{(r-1)} \right). \tag{10.122}$$

Let $w_{ir}^{(r)}$ be the mean waiting time of a job of class r in node i if we consider only the classes with the r highest priorities. Then we get:

$$w_{ir}^{(r)} = \frac{\overline{W}_{i0}^{(r)}}{(1 - \sigma_{ir+1})}. \tag{10.123}$$

- -/M/1-FCFS-PRS node

$$\overline{W}_{i0}^{(r)} = \sum_{i=1}^{r} \frac{\rho_{ij}}{\mu_{ij}}. \tag{10.124}$$

Using Eq. (10.111) and Little's theorem we obtain:

$$\overline{K}_{ir} = \rho_{ir} + \lambda_{ir} \overline{W}_{ir}. \tag{10.125}$$

For $\overline{W}_{ir}$ we use Eqs. (10.122), (10.123), and (10.124) and introduce the correction factor:

$$\overline{K}_{ir} = \rho_{ir} + \frac{\lambda_{ir}}{\rho_{ir}} \left[\frac{\sigma_{ir+1} \sum_{j=1}^{r} \frac{\rho_{ij}}{\mu_{ij}}}{\left(1 - \frac{K-2}{K-1} \sigma_{ir+1}\right)} - \frac{\sigma_{ir} \sum_{j=1}^{r-1} \frac{\rho_{ir}}{\mu_{ir}}}{\left(1 - \frac{K-2}{K-1} \sigma_{ir}\right)} \right]. \tag{10.126}$$

- -/M/m-FCFS-PRS node

$$\overline{W}_{i0}^{(r)} = \frac{P_{mi}^{r}}{m_i \sigma_{ir+1}} \cdot \sum_{j=1}^{r} \frac{\rho_{ij}}{\mu_{ij}}, \tag{10.127}$$

with probability of waiting P_{mi}^{r}, which we get from the probability of waiting P_{m_i} by replacing ρ_i by σ_{ir+1}. With Eq. (10.114) and Little's theorem:

$$\overline{K}_{ir} = m_i \rho_{ir} + \lambda_{ir} \overline{W}_{ir}, \tag{10.128}$$

and using Eqs. (10.122), (10.123), and (10.127) we obtain:

$$\overline{K}_{ir} = m_i \rho_{ir} + \frac{\lambda_{ir}}{\rho_{ir}} \left[\frac{\frac{P_{m_i}^{r}}{m_i} \sum_{j=1}^{r} \frac{\rho_{ij}}{\mu_{ij}}}{\left(1 - \frac{K-m_i-1}{K-m_i} \sigma_{ir+1}\right)} - \frac{\frac{P_{m_i}^{r-1}}{m_i} \sum_{j=1}^{r-1} \frac{\rho_{ij}}{\mu_{ij}}}{\left(1 - \frac{K-m_i-1}{K-m_i} \sigma_{ir}\right)} \right]. \tag{10.129}$$

- -/G/1-FCFS-PRS node
  With the remaining service time:

$$\overline{W}_{i0}^{(r)} = \frac{1}{2}\sum_{j=1}^{r}\lambda_{ij}\alpha_2(B_{ij}), \tag{10.130}$$

and Eqs. (10.118), (10.119), (10.122), and (10.123) and the correction factors:

$$\overline{K}_{ir} = m_i\rho_{ir} + \frac{\lambda_{ir}}{\rho_{ir}}\left[\frac{\frac{\sigma_{ir+1}}{2}\sum_{j=1}^{r}\lambda_{ij}\alpha_2(B_{ij})}{\left(1-\frac{K-m_i-a}{K-m_i}\sigma_{ir+1}\right)} - \frac{\frac{\sigma_{ir}}{2}\sum_{j=1}^{r-1}\lambda_{ij}\alpha_2(B_{ij})}{\left(1-\frac{K-m_i-a}{K-m_i}\sigma_{ir}\right)}\right]. \tag{10.131}$$

- -/G/m-FCFS-PRS node
  Using the corresponding equations as before we get for this node type:

$$\overline{W}_{i0}^{(r)} = \frac{P_{m_i}^{r}}{2m_i\sigma_{ir+1}}\sum_{j=1}^{r}\rho_{ij}\alpha_2(B_{ij})\mu_{ij}, \tag{10.132}$$

$$\overline{K}_{ir} = m_i\rho_{ir} + \frac{\lambda_{ir}}{\rho_{ir}}\left[\frac{\frac{P_{m_i}^{r}}{2m_i}\sum_{j=1}^{r}\rho_{ij}\alpha_2(B_{ij})\mu_{ij}}{\left(1-\frac{K-m_i-a}{K-m_i}\sigma_{ir+1}\right)} - \frac{\frac{P_{m_i}^{r-1}}{2m_i}\sum_{j=1}^{r-1}\rho_{ij}\alpha_2(B_{ij})\mu_{ij}}{\left(1-\frac{K-m_i-a}{K-m_i}\sigma_{ir}\right)}\right]. \tag{10.133}$$

10.4 SIMULTANEOUS RESOURCE POSSESSION

In a product-form queueing network, a job can only occupy one service station at a time. But if we consider a computer system, then a job at times uses two or more resources at once. For instance, a computer program first reserves memory before the CPU or disks can be used to run the program. Thus memory (a passive resource) and CPU (or another active resource such as a disk) have to be reserved at the same time. To take the simultaneous resource possession into account in performance modeling, *extended queueing networks* were introduced [SMK82]. In addition to the normal (active) nodes these networks also contain *passive nodes* that consist of a set of *tokens* and a number of *allocate* queues that request these tokens. These passive nodes can also contain additional nodes for special actions such as the release or generation of tokens. The tokens of a passive node correspond to the service units of a passive node: A job that arrives at an allocate queue requests a number of tokens from the passive node. If it gets them, it can visit all other nodes of the network, otherwise the job waits at the passive node. When the job arrives

at the release node that corresponds to the passive node, the job releases all its tokens. These tokens are then available for other jobs. We consider two important cases where we have simultaneous resource possession. These two cases are queueing systems with memory constraints and I/O subsystems.

10.4.1 Memory Constraints

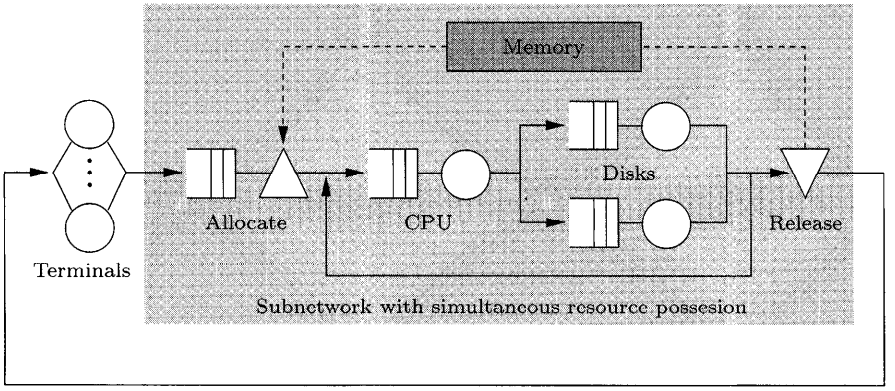


Fig. 10.35 Central-server model with memory constraint.

In Fig. 10.35 we see the typical case of a queueing network with memory constraints where a job first needs to reserve some memory (passive resource) before it can use an active resource such as the CPU or the disks. The memory queue is shown as an allocate node. The job occupies two resources simultaneously: the memory and the CPU or memory and disks. The maximum number of jobs that can compete for CPU and disks equals the number of tokens in the passive node, representing available memory units. The service strategy is assumed to be FCFS. Such models can be solved by generating and solving the underlying CTMC (via SPNP for example). To avoid the generation and solution of resulting large CTMC, we use an approximation technique.

The most common approximation technique is the flow-equivalent server method (FES) (see Section 8.4). We assume that the network without considering the simultaneous resource possession would have product-form solution. The subsystem that contains the simultaneous resource possession is replaced by an FES node and analyzed using product-form methods. For simple systems with only one job class, the load-dependent service rates $\mu_i(k)$ of the FES node can be determined by an isolated analysis of the active nodes of the subsystem (CPU, disks), where the rest of the network is replaced by a short circuit path, and determination of the throughputs $\lambda(k)$ for each possible job population along the short circuit. If K denotes the number of memory units,

then the service rates of the FES node are determined as follows [SaCh81]:

$$\mu(k) = \begin{cases} \lambda(k), & \text{for } k = 1, \ldots, K, \\ \lambda(K), & \text{for } k > K. \end{cases}$$

If there are no more than K jobs in the network, then there is no memory constraint, and the job never waits at the allocate node.

Example 10.14 To analyze the central-server network with memory constraint given in Fig. 10.35, we first replace the terminals with a short circuit and compute the throughputs $\lambda(k)$ of the resulting central-server network using any product-form method as a function of the number of jobs k in the network. Then we replace the subnetwork by an FES node (see Fig. 10.36). This reduced network can be analyzed by the MVA for queueing networks

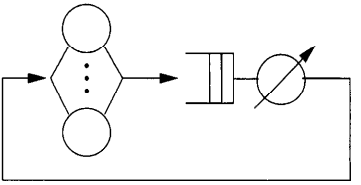


Fig. 10.36 Reduced network.

with load-dependent service rates or as we do it here using a birth-death type CTMC (see Section 3.1). With the mean think time $1/\mu$ at the terminals and the number of terminals M, we obtain for the birth rates:

$$\lambda_k = (M - k)\mu, \quad k = 0, 1, \ldots, K, \tag{10.134}$$

and for the death rates:

$$\nu_k = \begin{cases} \lambda(k), & i = 1, \ldots, K, \\ \lambda(K), & i > K. \end{cases} \tag{10.135}$$

Marginal probabilities of the FES node are (see Eq. (3.11)):

$$\pi(k) = \pi(0) \frac{M!}{(M-1)!} \cdot \frac{\mu^k}{\prod_{i=1}^{k} \nu_i}, \tag{10.136}$$

and with Eq. (3.12):

$$\pi(0) = \frac{1}{1 + \sum_{k=1}^{M} \frac{M!}{(M-1)!} \cdot \frac{\mu^k}{\prod_{i=1}^{k} \nu_i}}. \tag{10.137}$$

Now the throughput λ of the FES node that is also the throughput of terminal requests is obtained by:

$$\lambda = \sum_{i=1}^{M} \pi(i)\nu_i = \sum_{i=1}^{K} \pi(i)\nu_i + \nu_K \sum_{i=K+1}^{M} \pi(i). \tag{10.138}$$

As a numerical example, assume that the mean think time $1/\mu = 15\,\text{sec}$, the maximum number of programs in the subnetwork is $K = 4$ and the other parameters are as shown in Table 10.19. We calculate the mean response time $\overline{T}$ for terminal requests as a function of the number of terminals M and, in comparison, the mean response time $\hat{T}$ for the case that the main memory is large enough so that no waiting for memory space is necessary, i.e., $K \geq M$. In the case with ample memory, the overall network is of product-form type and, hence, $\hat{T}$ can be determined using algorithms from Chapter 8. From

Table 10.19 Parameters for the Central-server network

Node	CPU	Disk 1	Disk 2	Disk 3
μ_j	89.3	44.6	26.8	13.4
p_{1j}	0.05	0.5	0.3	0.15

Table 10.20 we see that $\hat{T}$ is only a good approximation if the number of terminals M is not too large.

Table 10.20 Response time for the central-server system with memory constraint

M	10	20	30	40	50
$\overline{T}$	1.023	1.23	1.64	2.62	7.03
$\hat{T}$	1.023	1.21	1.46	1.82	3.11

For other related examples see [Triv82], and for a SHARPE implementation of this technique see [STP96]. An extension of this technique to multiple class networks can be found in [Saue81], [Bran82], and [LaZa82]. This extension is simple in the case that jobs of different job classes reserve different amounts of memory. But if on the other side, jobs of different job classes use the same amount of memory, then the analysis is very costly and complex. Extensions to deal with complex memory strategies like paging or swapping also exist [BBC77, SaCh81, LZGS84]. For the solution of multiple class networks with memory constraints see also the ASPA algorithm (**a**verage subsystem **p**opulation **a**lgorithm) of [JaLa83]. This algorithm can also be extended to other cases of simultaneous resource possession.

Problem 10.1 Verify the results shown in Table 10.20 using PEPSY and SHARPE.

Problem 10.2 Solve the network of Fig. 10.35 by formulating it as a GSPN and solve it using SHARPE or SPNP. Compare the results obtained by means of FES approximation.

10.4.2 I/O Subsystems

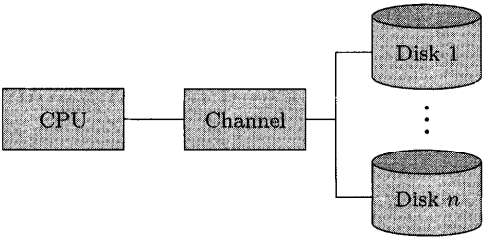


Fig. 10.37 I/O subsystem.

The other very important case of simultaneous resource possession occurs in modeling an I/O subsystem consisting of several disks that are connected by a common channel and a controller with the CPU. In the model of Fig. 10.37 channel and controller can be considered as one single service station because they are either simultaneously busy or simultaneously idle. An I/O operation from/to a certain disk can be subdivided into several phases: The first phase is the seek phase when the read/write head is moved to the right cylinder. The second phase is the latency phase, which is the disk rotation time needed for the read/write head to move to the right sector. Finally we have the data transfer phase. During the last two phases the job simultaneously needs the disk and the channel. In the special cases of rotational position sensing systems (RPS) the channel is needed only in the last phase. Delays due to simultaneous resource possession occur if a disk is blocked and no data can be transmitted because another disk is using the channel.

For the analysis of I/O subsystem models, different suggestions have been published. In the formulation of [Wilh77], an increased service time for the disks is computed approximately. This increased service time takes into account the additional delay for the competition of the channel. Each disk is then modeled as an simple M/G/1 queueing system. To determine the disk service time, another model for the probability that the channel is free when needed is used. Hereby it is assumed that only one access path for each disk exists, and a disk cannot be used by several processors simultaneously. This limitation is removed by [Bard80] and [Bard82] where the free path probability is determined using the maximum entropy principle.

We consider a method for analyzing disk I/O systems without RPS, introduced in [LZGS84]. The influence of the channel is considered by an additional

term in the mean service time s_i of the disk. The mean service time of the $disk_i$ is given then as:

$$s_i = s_{si} + s_{li} + s_{ti} + s_{ci}, \tag{10.139}$$

with:

$$\begin{aligned} s_{si} &= \text{ mean seek time}, & s_{li} &= \text{mean latency time}, \\ s_{ti} &= \text{mean transfer time}, & s_{ci} &= \text{mean contention time}. \end{aligned}$$

The contention time s_{ci} is calculated iteratively. To obtain an approximate formula for s_{ci} , we first use the formula for the mean number of customers in a -/M/1 FCFS node in an open queueing network (see Section 6.3):

$$\overline{K}_i = \frac{\rho_i}{1 - \rho_i}. \tag{10.140}$$

In the case of the channel, any requests ahead of a $disk_i$ request at the channel must be associated with some disk other than i, so the equation is slightly modified to:

$$\overline{K}_{ch} = \frac{\rho_{ch} - \rho_{ch}(i)}{1 - \rho_{ch}}, \tag{10.141}$$

where:

$$\rho_{ch} = \sum_{i=1}^{U} \rho_{ch}(i) \tag{10.142}$$

is the utilization of the channel and $\rho_{ch}(i)$ is the utilization of the channel due to requests from $disk_i$. Here we have assumed that the number of disks is equal to U. Because the mean channel service time is the sum of the mean latency time s_{si} and the mean transfer time s_{ti}, we obtain for the contention time:

$$s_{ci} = \frac{\rho_{ch} - \rho_{ch}(i)}{1 - \rho_{ch}} \cdot (s_{li} + s_{ti}), \tag{10.143}$$

and, hence, for the mean disk service time:

$$s_i = s_{si} + \frac{(s_{li} + s_{ti})(1 - \rho_{ch}(i))}{1 - \rho_{ch}}. \tag{10.144}$$

The measures $\rho_{ch}(i)$ and ρ_{ch}, and other interesting performance measures can be calculated using the following iterative algorithm. We assume that the overall model is a closed queueing network that will be of product-form type if we have no contention at the channel.

STEP 1 Initialization. System throughput $\lambda = 0$.

STEP 2 Iteration:

STEP 2.1 For each $disk_i$, the contribution of $disk_i$ to the utilization of the channel:

$$\rho_{ch}(i) = \lambda \cdot e_i(s_{li} + s_{ti}). \tag{10.145}$$

STEP 2.2 Channel utilization:

$$\rho_{ch} = \sum_{i=1}^{U} \rho_{ch}(i). \tag{10.146}$$

STEP 2.3 Compute the mean service time for each $disk_i$ using Eq. (10.144).

STEP 2.4 Use MVA or any other product-form method method to calculate system throughput λ. Return to Step 2.1 until successive iterates of λ are sufficiently close.

STEP 3 Obtain the performance measures after the final iteration.

Table 10.21 Iterative calculation of the throughput and the channel utilization.

	λ_{in}	$\rho_{ch}(i)$	ρ_{ch}	$s_i \cdot e_i$	λ_{out}
1	0	0	0	11.00	0.056
2	0.056	0.167	0.836	23.24	0.030
3	0.030	0.090	0.449	12.96	0.050
4	0.050	0.150	0.749	18.16	0.038
5	0.038	0.113	0.564	14.10	0.047
6	0.047	0.140	0.701	16.63	0.041
7	0.041	0.123	0.611	14.77	0.045
8	0.045	0.135	0.674	15.96	0.042
9	0.042	0.127	0.635	15.18	0.044
10	0.044	0.132	0.659	15.63	0.043
11	0.043	0.129	0.645	15.36	0.043
12	0.043	0.130	0.651	15.48	0.043

Example 10.15 We consider a batch computer system with multiprogramming level $K = 10$, the mean CPU service time of 15 sec, one channel, and five disks [LZGS84], each with the mean seek time $s_{si} = 8$ sec, the mean latency time $s_{li} = 1$ sec, and the mean transfer time $s_{ti} = 2$ sec. We assume $e_i = 1$ for each node. With a queueing network having six nodes, we can solve the problem iteratively using the preceding method. The first 12 steps of the iteration are given in Table 10.21. This iterative algorithm can be easily extended to disk I/O systems with:

- Multiple classes
- RPS

- Additional channels
- Controllers

For details see [LZGS84]. For related examples of the use of SHARPE, see [STP96].

Problem 10.3 Verify the results shown in Table 10.21 using the iterative algorithm of this section and implement it in SHARPE.

Problem 10.4 Develop a stochastic reward net model for Example 10.15 and solve using SPNP.

10.4.3 Method of Surrogate Delays

Apart from special algorithms for the analysis of I/O subsystem models and queueing networks with memory constraints, more general methods for the analysis of simultaneous resource possession in queueing networks are also known. Two general methods with which a satisfactory accuracy can be achieved in most cases are given in [FrBe83] and [JaLa82]. Both methods are based on the idea that simultaneously occupied service stations can be subdivided into two classes: the *primary stations* that are occupied before an access to a *secondary station* can be made. In the case of I/O subsystem models, the disks constitute the primary stations and the channel the secondary station. In the case of models with memory constraints, the memory constitutes the primary stations and the CPU and disks constitute the *secondary stations.* The time while a job occupies a primary station without trying to get service from the secondary station is called *non-overlapping service time.* This non-overlapping service time can also be zero. The time while a job occupies both the primary and the secondary station is called *overlapping service time.* Characteristic for the method of surrogates [JaLa82] is that the additional delay time can be subdivided into:

- The delay that is caused when the secondary station is heavily loaded. Heavy load means that more jobs arrive at a station than can immediately be served.
- The delays due to an overload at the primary station. This is exactly the time one has to wait for a primary station while the secondary station is not heavily loaded.

To estimate the delay times, two models are generated, where each model provides the parameters for the other model. In this way we have an iterative procedure for the approximate determination of the performance measures of the considered queueing model. This method can be applied to single class closed queueing networks that would otherwise have product-form solution if we neglect the simultaneous resource possession. Studies have shown that the method of surrogate delays tends to overestimate the delay and therefore

underestimate the throughput. In all examples we considered, the deviation compared to the exact values was found to be less than 5%.

10.4.4 Serialization

Up to now we only modeled contention for the hardware resources of a computer system. But delays can also be caused by the competition for software resources. Delay that is caused because a process has to wait for software resources and that makes a serialization in the system necessary is called *serialization delay.* Examples for such software resources are critical areas that have to be controlled by semaphores or non-reentrant subprograms. Serialization can be considered as a special case of simultaneous resource possession because jobs simultaneously occupy a special (passive) serialization node and an active resource such as CPU or disk.

The phrase *serialization phase* is used here for the processing phase in which at most one job can be active at any time. A job that wishes to enter the serialization phase or that is in the serialization phase is called a *serialized job.* The processing phase in which a job is not serialized is called the *non-serialized phase*, and a job that is in a non-serialization phase is called a *non-serialized job.* There can always be more than one job in the non-serialization phase. All nodes that can be visited by serialized jobs are called *serialized nodes*, all other nodes are called *non-serialized nodes.* Each serialized node can be visited by serialized jobs as well as non-serialized jobs.

An iterative algorithm for the analysis of queueing networks with serialization delay is suggested by [JaLa83]. This algorithm can also be used for multiple class networks where the jobs can be at most in one serialization phase at one time. The starting point of this method is the idea that the entrance of a job in a serialization phase shall lead to a class switch of that job. Therefore a class-r job that intends to enter the serialization phase s is denoted as a class-(s, r) job. When the job leaves the serialization phase, it goes back to its former class. In each serialization phase, there can be at most one job at a time. Non-serialization phases are not limited in this way. The method uses a two layer model and the iteration takes place between these two layers. For more information see [JaLa83].

There are also other formulations for the analysis of queueing networks with serialization delays, but these methods are all restricted to single class queueing networks. In the aggregate server method [AgBu83], an additional node for each serialization phase is introduced and the mean service time of the node is set equal to the mean time that a job spends in the serialization phase. The mean service time of the original node and the additional node have to be increased in a proper way to take into consideration the fact that at the original node the service of non-serialized jobs can be hindered by the service of serialized jobs and vice versa. Therefore, iteration is needed to solve the model.

Two different techniques to model the deleterious effect of the serialization delays to the performance of a computer system are proposed by [Thom83]: an iterative technique and a decomposition technique. The decomposition technique also uses two different model layers. At the lower layer the mean system throughputs for each possible system state is computed. These throughputs are then used in the higher level model to determine the transition rates between the states from which the steady-state probabilities can be determined by solving the global balance equations of the underlying CTMC. More details and solution techniques can be found in [AgTr82], [MüRo87], and [SmBr80]. A SHARPEimplementation can be found in [STP96].

Another complex extension of the models with serialization delays are models for the examination of synchronization mechanism in database systems (concurrency control). These mechanisms make sure that the data in a database are consistent when requests and changes to these data can be done concurrently. These models can be found in [BeGo81], [CGM83], [MeNa82], [Tay87], or [ThRy85].

10.5 PROGRAMS WITH INTERNAL CONCURRENCY

A two-level hierarchical model for the performance prediction of programs with internal concurrency was developed by Heidelberger and Trivedi (see [HeTr83]). Consider a system consisting of M servers (processors and I/O devices) and a pseudo server labeled 0. A job consists of a primary task and a set of secondary tasks. Each primary task spawns a fixed number of secondary tasks whenever it arrives at node 0. The primary task is then suspended until all the secondary tasks it has spawned have finished execution. Upon spawning, each of the secondary tasks is serviced by network nodes and finally on completion return to node 0. Each secondary task may have to wait for its siblings to complete execution and return to node 0. When all its siblings complete execution, the primary task is reactivated. It should be clear that a product-form queueing model cannot be directly used here as the waiting of the siblings at node 0 violates the assumptions. Of course, under the assumption of service times being exponentially distributed, a CTMC can be used. In such a CTMC, each state will have to include the number of tasks of each type (primary, secondary1, secondary2, etc.) at each node. The resulting large state space makes it infeasible for us to use the one-level CTMC model even if it can be automatically generated starting from an SPN model.

The two-level decomposition we use here is based on the following idea: Assume that the service requirements of the primary and secondary tasks are substantial so that on the average, queue lengths in the network will have time to reach steady state before any spawned task actually completes. We could then use a product-form queueing network model to compute throughputs and other measures for a fixed number of tasks of each type. This is assumed to be a closed product-form network and, hence, efficient algorithms such as MVA

can be used to compute steady-state measures of this lower level queueing network.

A CTMC is constructed as the outer model where the states are described just by a vector containing the numbers or each type of tasks currently in the system. To describe the generator matrix of the CTMC, we introduce the following notation. For simplicity, we assume that the parent task spawns exactly two children after arriving at node 0. The parent tasks are labeled 0, while the children tasks are labeled 1 and 2. Let a_i denote the number of tasks of type i. Then a generic CTMC state is $\mathbf{a} = (a_0, a_1, a_2)$. The state space of the CTMC will then be:

$$S = \{\mathbf{a} : 0 \leq a_0 \leq N, 0 \leq a_1 \leq N - a_0, 0 \leq a_2 \leq N - a_0, N \leq a_0 + a_1 + a_2\}$$

Let w_i be the number of children of type i waiting for their siblings at node 0. Let $r_i(\mathbf{a})$ be the probability that a task of type i, on arrival at node 0, finds its sibling waiting for it there. This probability is w_2/a_1 for a Type-1 task and w_1/a_2 for a Type-2 task. Denote the throughputs obtained from the solution of the lower level PFQN by $\lambda_i(\mathbf{a})$ as a function of the task vector $\mathbf{a}$. Then the generator matrix $\mathbf{Q}$ of the CTMC can be derived as shown in Table 10.22. The entries listed are the off-diagonal entries $q(\mathbf{a}, \mathbf{b})$. For vectors $\mathbf{b}$ not in the table, the convention is that $q(\mathbf{a}, \mathbf{b}) = 0$. For a detailed SHARPE

Table 10.22 Generator matrix

$\mathbf{b}$	$q(\mathbf{a}, \mathbf{b})$	Transition Explanation
$(a_0 - 1, a_1 + 1, a_2 + 1)$	$\lambda_0(\mathbf{a})$	Task 0 completion, tasks 1 and 2 spawned
$(a_0, a_1 - 1, a_2)$	$\lambda_1(\mathbf{a})(1 - r_1(\mathbf{a}))$	Task 1 completion, sibling active
$(a_0, a_1, a_2 - 1)$	$\lambda_2(\mathbf{a})(1 - r_2(\mathbf{a}))$	Task 2 completion, sibling active
$(a_0 + 1, a_1 - 1, a_2)$	$\lambda_1(\mathbf{a})(r_1(\mathbf{a}))$	Task 1 completion, sibling waiting
$(a_0 + 1, a_1, a_2 - 1)$	$\lambda_2(\mathbf{a})(r_2(\mathbf{a}))$	Task 2 completion, sibling waiting

implementation of this algorithm see [STP96].

10.6 PARALLEL PROCESSING

Two different types of models of networks with parallel processing of jobs will be considered in this section. The first subsection deals with a network in which jobs can spawn tasks that do not need any synchronization with the spawning task. In the second subsection we consider fork-join networks in which synchronization is required.

10.6.1 Asynchronous Tasks

In this section we consider a system that consists of m active resources (e.g., processors and channels). The workload model is given by a set of statistically

independent tasks. Each job again consists of one primary and zero or more statistically identical secondary tasks. The primary task is labeled 1 and the secondary tasks are labeled 2. A secondary task is created by a primary task during the execution time and proceeds concurrently with it, competing for system resources. It is assumed that a secondary task always runs independently from the primary task. This condition in particular means that we do not account for any synchronization between the tasks. Our treatment here is based on [HeTr82].

The creation of a secondary task takes place whenever the primary task enters a specially designated node, labeled 0; the service time at this node is assumed to be 0. Let $e_{i1}, i = 1, 2, \ldots, m$ denote the average number of visits to node i per visit to node 0 by a primary task. Furthermore $e_{i2}, i = 1, 2, \ldots, m$ denotes the average number of visits to node i by a secondary task. After completing execution, a secondary task leaves the network and $e_{i1}, e_{i2} < \infty$. Let $1/\mu_{ij}$ denote the average service time of a type j task at node i. Each task type does not hold more than one resource at any time. The number of primary jobs in the system is assumed to be constant K and concurrency within a job is allowed only through multitasking, while several independent jobs are allowed to execute concurrently and share system resources.

In the case that the scheduling strategy at a node is FCFS, we require each task to have an exponentially distributed service time with common mean. In the case of PS or LCFS-PRS scheduling, or when the node is an IS node, an arbitrary differentiable service time distribution is allowed and each task can have a distinct service time distribution. Since concurrency within a job is allowed, the conditions for a product-form network are not fulfilled (see [BCMP75]). Therefore we use an iterative technique where in each step a product-form queueing network is solved (the product-form solution of each network in the sequence is guaranteed due to the assumptions on the service time distributions and the queueing disciplines).

The queueing network model of the system consists of two job classes, an open class and a closed class. The open classes are used to model the behavior of the secondary tasks and, therefore, the arrival rate of the open class is equal to the throughput of the primary task at node 0. The closed class models the behavior of the primary tasks. Because there are K primary tasks in the network, the closed class population is K. Notice that the approximation assumes the arrival process at node 0 is a Poisson process that is independent of the network state. Due to these two assumptions (Poisson process and independence of the network state), our solution method is approximate. A closed form solution is not possible because the throughput of the closed chain itself is a non-linear function of the arrival rate of the open chain. For the solution of this non-linear function, a simple algorithm can be used. In [HeTr82] *regula falsi* was used and on the average only a small number of iteration steps was necessary to obtain an accurate solution of the non-linear function.

Let λ_{02} denote the arrival rate and λ_2 the throughput of the open class, respectively. If any of the queues is saturated, then $\lambda_2 \leq \lambda_{02}$. Furthermore let λ_1 denote the throughput of the closed class at node 0. We require $\lambda_2 = \lambda_1$ and for the stability of the network, we must have $\lambda_2 = \lambda_{02}$. Therefore the following non-linear fixed-point equation needs to be solved:

$$\lambda_1(\lambda_{02}) = \lambda_{02}. \tag{10.147}$$

Here $\lambda_1(\lambda_{02})$ is the throughput of the primary task at Node 0, given that the throughput of the secondary task is λ_{02}. Equation (10.147) is a non-linear function in λ_{02} and can be evaluated by solving the product-form queueing network with one open and one closed job class for any fixed value of λ_{02}. Let λ_1^* and λ_{02}^* denote the solution of Eq. (10.147), i.e., $\lambda_1^* = \lambda_{02}^* = \lambda_1(\lambda_{02}^*)$. Here λ_{02}^* is the approximated arrival rate of a secondary task. At this point, the arrival rate of a secondary task is equal to the throughput of a primary task at node 0. To get the approximate arrival rate λ_{02}, we can use any algorithm for the solution of a non-linear function in a single variable. Since the two classes share system resources, an increase of λ_{02} does not imply an increase in λ_1; we conclude that λ_1 is a monotone non-increasing function of λ_{02}. If $\lambda_2^{\max}$ denotes the maximum throughput of the open class, then the stability condition is given by:

$$\lambda_{02} < \lambda_2^{\max}. \tag{10.148}$$

The condition for the existence of a stable solution, i.e., a solution where no queue is saturated, for Eq. (10.147) is given by:

$$\lim_{\lambda_{02} \to \lambda_2^{\max}} \lambda_1(\lambda_{02}) < \lambda_2^{\max}. \tag{10.149}$$

If a stable solution exists, it is also a unique one (monotonicity property). If the condition is not fulfilled, primary tasks can generate secondary tasks at a rate that exceeds the system capacity. The node that presents a bottleneck determines the maximum possible throughput of the open class. The index of this node is given by:

$$\text{bott} = \text{argmax}\left\{\frac{e_{i2}}{\mu_{i2}}\right\}. \tag{10.150}$$

Here argmax is the function that determines the index of the largest element in a set. Therefore the maximum throughput is given by:

$$\lambda_2^{\max} = \frac{\mu_{\text{bott},2}}{e_{\text{bott},2}}. \tag{10.151}$$

In Fig. 10.38 the three types of possible behaviors, depending on the system parameters, are shown. In case (a), node bott is utilized by the primary task, i.e.:

$$\frac{e_{\text{bott},1}}{\mu_{\text{bott},1}} > 0. \tag{10.152}$$

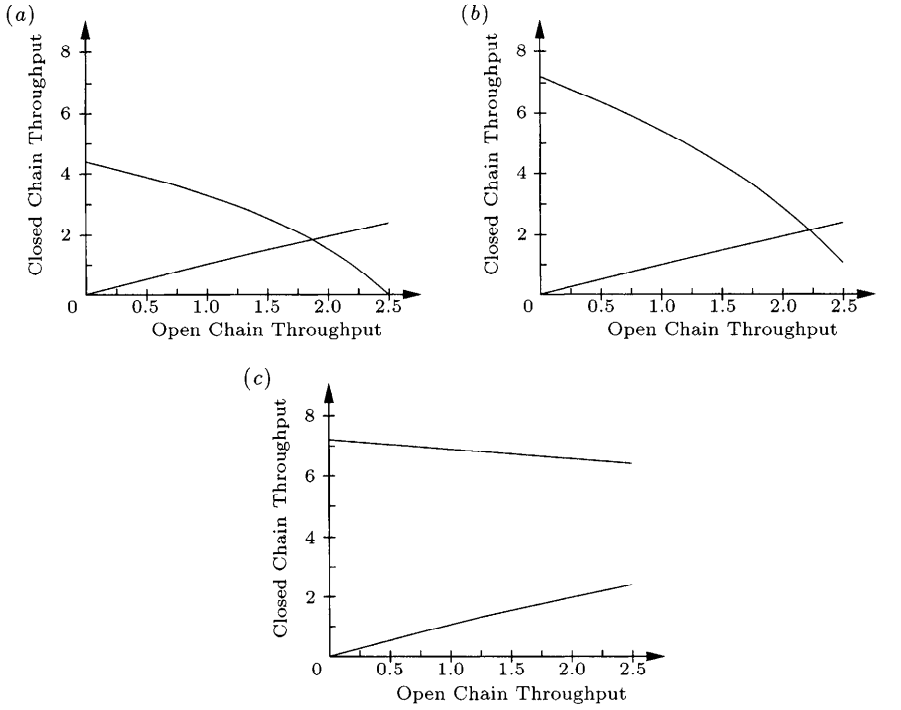


Fig. 10.38 Three types of possible behavior of the approximation method (*a*) Contention at bottleneck (*b*) No contention at bottleneck, moderate contention at other devices (*c*) No contention at bottleneck, little contention at other devices.

When the queue of node bott grows without bound due to an excessive arrival of jobs of the open job class, the throughput of the closed job class will approach zero. If Condition (10.152) is not satisfied, but some other network nodes are shared by two job classes, then either case (*b*) or case (*c*) results depending on the degree of sharing. Because case (*b*) yields a unique solution to (10.147), we can conclude that (10.152) is a sufficient but not necessary condition for convergence, while (10.150) is both necessary and sufficient. Compared to Condition (10.150), Condition (10.152) has the advantage that it is testable prior to the solution of the network.

We can also extend this technique to include cases where more than one type of secondary task is created. Assume, whenever a primary task passes node 0, a secondary task of type $k, (k = 2, \ldots, C)$ is created with probability p_k. To model the secondary tasks, $C - 1$ open job classes are used with arrival rate λ_{0k} each. Let $\lambda_0 = \sum_{k=2}^{C} \lambda_{0k}$ denote the total arrival rate of the secondary tasks. The individual arrival rates are constrained so that $\lambda_{0k} = p_k \lambda_0, k \geq 2$. The total throughput of secondary tasks is set equal to the throughput of the primary tasks at node 0. This condition again defines a non-linear equation in the single variable λ. This equation is similar

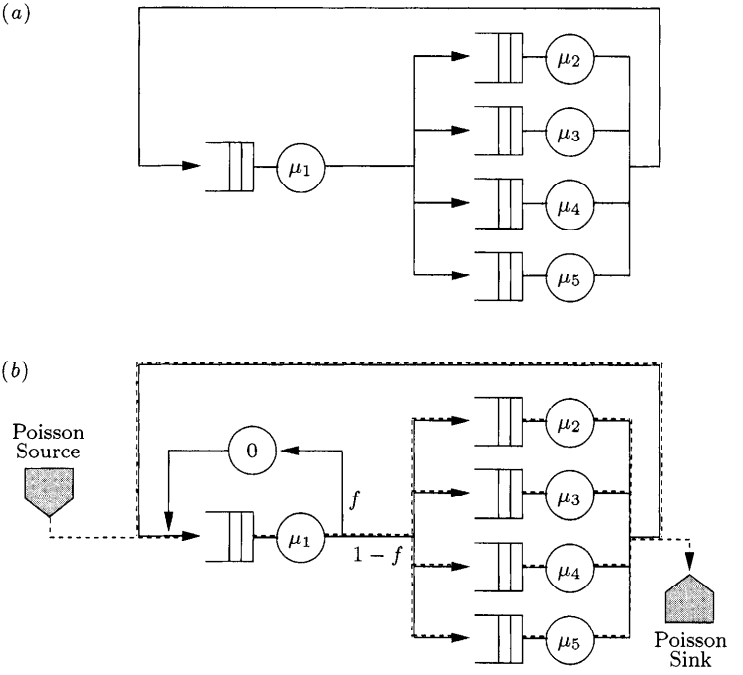


Fig. 10.39 The central-server model without overlapped I/O (a) and with overlapped I/O (b).

to Eq. (10.147) and must be solved. As a concrete example we consider the central-server model. The standard model without overlap is shown in Fig. 10.39a.

Node 1 represents the CPU and nodes 2, 3, 4, and 5 represent I/O devices such as disks. In the case without overlap, the number of primary tasks is constant C. The processing time of the primary tasks at the CPU is assumed to be random with mean $1/\mu_p$. As soon as a primary task finishes processing, a secondary task is created with probability f. Once a secondary task is generated, the primary task returns to the CPU for processing (with mean $1/\mu_1$), while the secondary task starts at the CPU queue (with mean service time $1/\mu_{12}$). With probability p_{12}, the secondary task moves to an I/O device. The secondary tasks leaves the system with probability $1 - p_2$ and returns to the CPU with probability p_2. In the case that a primary task completes CPU processing and does not create a new secondary task (with probability $1 - f$), it begins an overhead period of CPU processing with mean $1/\mu_0$. On completion of this overhead processing, the primary task moves to I/O device i with probability p_i and then returns to the CPU. As scheduling strategies, we assume PS at the CPU and FCFS at all I/O devices with mean service times $1/\mu_i$.

A primary task can issue I/O requests that are processed by the secondary task. Then the task continues processing without waiting for these I/O requests to complete. If we assume $1/\mu_0 = 1/\mu_{12}$, then this is the average time to initiate an I/O request. In the notation of Section 10.5 we have $1/\mu_{11} = 1/\mu_p + (1-f)1/\mu_0$. On the average, each secondary task generates $1/(1-p_2)$ I/O requests and thus the fraction of all I/Os that overlap with a primary task, is given by:

$$f_{ol} = \frac{f}{(1-p_2)} \cdot \frac{1}{(1-f) + \frac{f}{(1-p_2)}}. \tag{10.153}$$

In the case of $p_2 = 0$, we get $f_{ol} = f$.

In Fig. 10.39*b*, the central-server model with overlap is shown. Here the additional node 0 with service time 0 is introduced. Once a primary task leaves the CPU, it moves to node 0 with probability f and to I/O device i with probability $(1-f)p_{i1}$. The rate at which secondary tasks are generated is equal to the throughput of primary tasks at node 0. Secondary tasks are modeled by an open class of customers where all arrivals are routed to the CPU. We will assume that the arrival process is Poisson. The arrival rate of the open class customers λ_{02} is set equal to the throughput of the closed class (primary tasks) at node 0. The routing of the secondary task is as described earlier. A secondary task leaving the CPU visits the I/O device labeled i with probability p_{i2} and returns then to the CPU with probability p_2 or leaves the system with $1-p_2$, respectively. For this mixed queueing network, the stability condition is $e_{i2}\lambda_{02} < \mu_{i2}$ for all i where $e_{12} = 1/(1-p_2)$ and $e_{i2} = p_{i2}/(1-p_2)$, $i > 1$.

Table 10.23 Input parameters for the central-server model

Model Set	$\frac{1}{\mu_2} = \frac{1}{\mu_3} = \frac{1}{\mu_4}$	$\frac{1}{\mu_5}$	$p_{21} = p_{22}$ $p_{31} = p_{32}$ $p_{41} = p_{42}$	$p_{51} = p_{52}$	p_{02}
I	0.04	0.04	0.25	0.25	0.00
II	0.04	0.04	0.25	0.25	0.50
III	0.04	0.04	0.30	0.10	0.00
IV	0.04	0.10	0.30	0.10	0.00
V	0.04	0.20	0.30	0.10	0.00
VI	0.04	0.40	0.30	0.10	0.00

Six sets of central-server models with 60 models for each set were analyzed by [HeTr82]. Each set contains models with a moderate utilization of the devices as well as heavily CPU and/or I/O bound cases. Each set also contains a range of values of multiprogramming levels, K, the mean service time at the CPU, $1/\mu_1$, and an overlapping factor f. A model within a set is then characterized by the triple $(K, 1/\mu_1, f)$ where $K = 1, 3, 5$, $1/\mu_1 = 0.002, 0.01, 0.02, 0.10, 0.2$, and $f = 0.1, 0.25, 0.5, 0.75$. The differences

between the sets are the I/O service times and the branching probabilities p_{ij} and p_{02}, respectively. Furthermore they assume for all models that the overhead processing times are equal, $1/\mu_0 = 1/\mu_{12} = 0.0008$. The mean service times of the least active I/O device is varied in an interval around the mean service time of the other I/O devices. In addition the I/O access pattern ranges from an even distribution to a highly skewed distribution. The input parameters for each set of 60 central-server models are shown in Table 10.23.

As a second example, we consider the terminal-oriented timesharing systems model shown in Fig. 10.40*a*. Basically it is the same model as the central-server model except that it has an extra IS node (node 6) that represents a finite number of terminals. The mean service time at node 6, $1/\mu_{61}$, can be considered as the mean thinking time. The number of terminals submitting primary tasks to the computer system is denoted by K. Each primary task first uses the CPU and then moves to I/O device i with probability p_{i1}. As soon as a primary task completes service at the I/O, it returns to the CPU with probability p_1 or enters the terminals with probability $1 - p_1$. We assume that during the execution of a primary task a secondary task is generated which executes independently from the primary task (except for queueing effects). In particular, the secondary task can execute during the thinking time of its corresponding primary task. Routing and service times are the same as in the central-server model. The approximate model of this overlap situation is shown in Fig. 10.40*b*. Between the I/O devices and the terminals, node 0 (with service time 0) is inserted. Therefore the throughputs at node 0 and the terminals are equal. The secondary tasks are modeled as open job classes and arrive at the CPU. The corresponding arrival rate λ_{02} is set equal to the throughput of primary tasks at node 0. The stability condition for this model is the same as in the model before: $\lambda_{02}/(\mu_{12} \cdot (1 - p_2)) < 1$ and $\lambda_{02} \cdot p_{i2}/(\mu_{i2} \cdot (1 - p_2)) < 1$ for all $i > 2$.

Table 10.24 Input parameters for the terminal system

$\frac{1}{\mu_{61}}$	$\frac{1}{\mu_{11}} = \frac{1}{\mu_{12}}$	$\frac{1}{\mu_i}$	p_{ij}	$p_1 = p_2$	K (no. of terminals)
10	0.010	0.04	0.25	0.10	1, 5, 10, 20, 30, 40, 50,60, 70, 80, 90, 100
10	0.015	0.04	0.25	0.10	1, 5, 10, 20, 30, 40, 50,60, 70, 80, 90, 100
15	0.002	0.04	0.25	0.10	25, 50, 100
15	0.010	0.04	0.25	0.10	25, 50, 100
15	0.020	0.04	0.25	0.10	25, 50, 100
15	0.020	0.08	0.25	0.02	1, 5, 10, 20, 30, 40, 50
15	0.030	0.08	0.25	0.02	1, 5, 10, 20, 30, 40, 50

This model can be interpreted as follows: Each terminal interaction can be split into two tasks. A terminal user has to wait for completion of a primary task but not for completion of a secondary task. In [HeTr82] 47 models of this type were considered. The corresponding parameter settings for this model are described in Table 10.24. Each of the models was simulated using the IBM

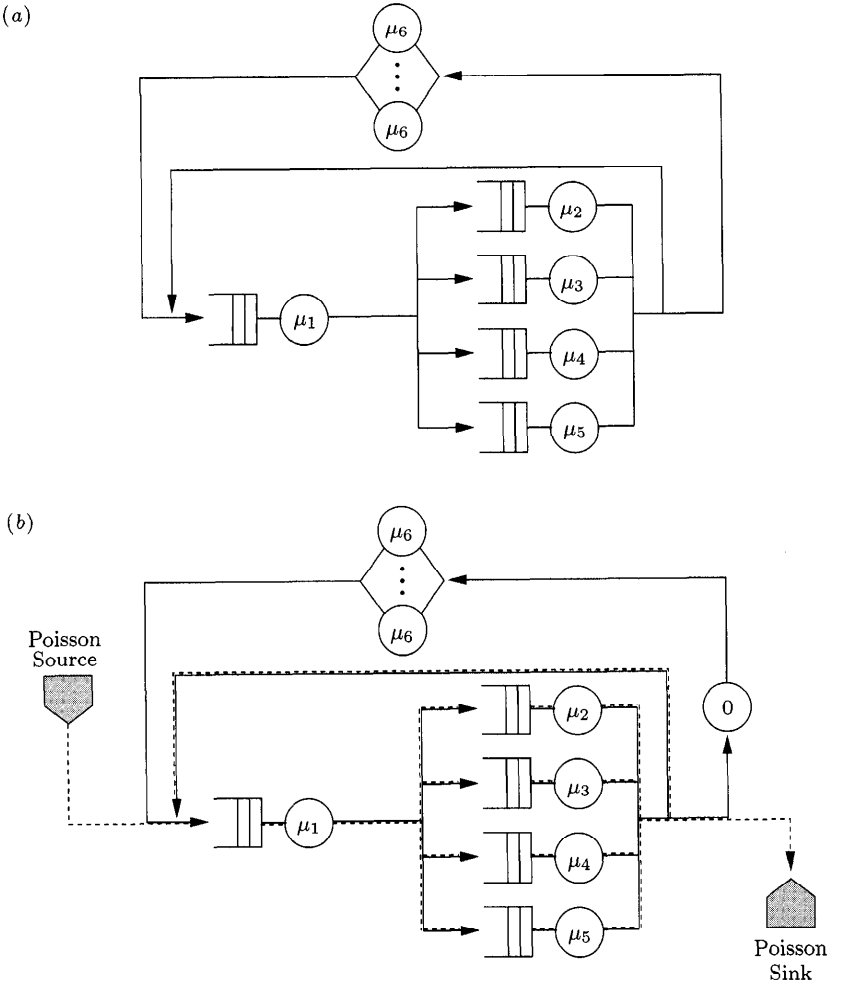


Fig. 10.40 Terminal model without overlapped tasks (a) and with overlapped tasks (b).

Table 10.25 Comparison of DES and analytic approximations for the central-server models with balanced I/O subsystem

Models		CPU ρ	Disk ρ	CPU $\overline{Q}$	Disk $\overline{Q}$	λ
Set I				*Absolute error*		
	Mean	0.003	0.006	0.011	0.044	0.056
	Maximum	0.026	0.071	0.110	0.457	0.710
				Relative error		
	Mean	1.2	1.5	2.8	3.2	1.3
	Maximum	15.9	15.4	26.2	16.2	15.5
Set II				*Absolute error*		
	Mean	0.003	0.005	0.009	0.072	0.034
	Maximum	0.020	0.054	0.073	0.670	0.317
				Relative error		
	Mean	1.0	1.1	2.8	3.1	1.0
	Maximum	9.2	8.7	25.0	15.1	9.0

Table 10.26 Comparison of simulation and analytic approximations for the terminal models with balanced I/O subsystem

Models		CPU ρ	Disk ρ	CPU $\overline{Q}$	Disk $\overline{Q}$	Terminal $\overline{Q}$	λ	$\overline{T}$
Terminal					*Absolute error*			
Models	Mean	0.002	0.002	0.429	0.056	0.115	0.010	0.228
	Maximum	0.010	0.008	2.261	0.421	0.674	0.036	1.141
					Relative error			
	Mean	0.3	0.4	3.0	1.4	0.6	0.7	1.7
	Maximum	1.4	1.5	13.4	4.5	2.7	2.2	5.4

Table 10.27 Comparison of simulation and analytic approximations for the central server models with imbalance I/O

Models		CPU ρ	Disk 2 ρ	Disk 3-5 ρ	CPU $\overline{Q}$	Disk 2 $\overline{Q}$	Disk 3-5 $\overline{Q}$	λ
Set III				*Absolute error*				
	Mean	0.003	0.007	0.003	0.010	0.060	0.004	0.050
	Maximum	0.025	0.076	0.025	0.101	0.799	0.065	0.631
				Relative error				
	Mean	1.2	1.4	1.9	2.9	3.8	2.0	1.3
	Maximum	14.7	14.4	13.9	27.7	18.3	13.1	14.3
Set IV				*Absolute error*				
	Mean	0.003	0.007	0.007	0.010	0.044	0.030	0.055
	Maximum	0.029	0.078	0.069	0.095	0.555	0.323	0.636
				Relative error				
	Mean	1.4	1.6	2.1	2.9	3.1	4.2	1.4
	Maximum	15.8	16.2	17.4	27.0	13.6	29.2	15.9
Set V				*Absolute error*				
	Mean	0.003	0.005	0.008	0.009	0.043	0.394	0.039
	Maximum	0.024	0.059	0.102	0.065	0.878	5.499	0.490
				Relative error				
	Mean	1.4	1.5	1.6	3.1	3.2	11.4	1.4
	Maximum	15.6	15.3	15.9	28.7	37.4	73.2	15.3
Set VI				*Absolute error*				
	Mean	0.002	0.002	0.007	0.008	0.041	0.592	0.019
	Maximum	0.011	0.180	0.067	0.032	0.394	4.224	0.149
				Relative error				
	Mean	1.0	1.2	1.3	4.0	7.4	13.7	1.1
	Maximum	7.2	7.2	8.1	28.9	47.9	89.0	7.1

Research Queueing package RESQ, which offers a so-called *split node* that splits a job into two tasks that proceed independently through the network. This node type was used to model the overlap of jobs. In Table 10.25, 10.27, and 10.26, the results of the comparison of discrete-event simulation (DES) with the approximate analytic method are shown.

Both absolute errors as well as relative errors are given. For each set of models, the mean and maximum absolute and relative errors are listed for utilizations, queue lengths, throughputs, and response times (for the terminal models). As we can see from the results, the approximation is particularly accurate for estimating utilizations and throughputs. For these quantities, the relative error is about 1.3%, and the maximum relative error is 17.4%. We can also see that the estimates for the mean queue length are somewhat less accurate, the mean relative error is 4.2%. The maximum relative error is very high (up to 89%). These high errors always occur in very imbalanced systems such as the central-server model sets V and VI (in these sets the overlap factor is very high). If, on the other hand, the model is better balanced (lower values of f_{ol}), we get quite accurate approximations. In this case the mean relative error for all performance measures is 1.9%, and the maximum relative error is 2.9%.

10.6.2 Fork-Join Systems

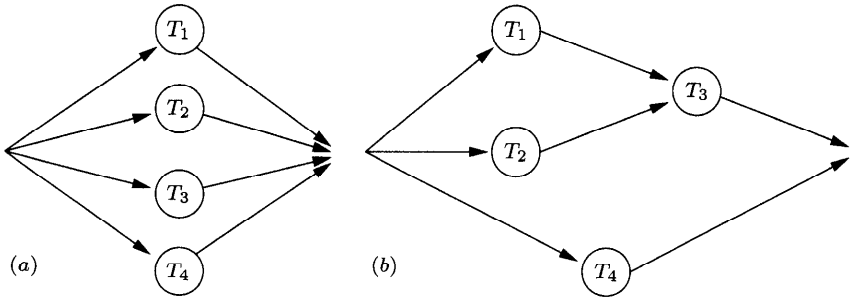


Fig. 10.41 Task precedence graph with (a) four parallel tasks and (b) four tasks.

In this section we consider computer systems that are able to run programs in parallel by using the *fork-join* or *parbegin-parend* constructs. The parallel programs (jobs) consist of *tasks* that have to be processed in a certain order. This order can be given by a *task precedence graph.* In Fig. 10.41a the task precedence graph for a program consisting of four parallel tasks T_1, T_2, T_3, T_4 is shown, and in Fig. 10.41b another task precedence graph for a program consisting of four tasks T_1, T_2, T_3, T_4 is shown. In Case b, task T_3 can start after both tasks T_1 and T_2 are finished while task T_4 can be executed in parallel with task T_1, T_2, and T_3. The corresponding *parbegin-parend* constructs look as shown in Fig.10.42.

10.6.2.1 Modeling Consider a model of a system in which a series of external parallel program execution requests arrive at an open queueing network where nodes indicate the processors and external arrivals indicate the jobs [Duda87]. It is assumed that each processor can execute exactly one special task. Thus task T_i always needs to be served by processor P_i. Furthermore it is assumed that processors are always available when needed. The interarrival times of jobs are exponentially distributed with mean value $1/\lambda$, and the service times of the tasks are generally distributed with mean value $1/\mu_i$, $i = 1, \ldots, N$. Figure 10.43 shows the queueing network model for jobs with the structure shown in Fig. 10.41*a*. Figure 10.44 shows the queueing network model of a parallel system, where jobs have the structure shown in Fig. 10.41*b*. The fork and join operators are shown as triangles.

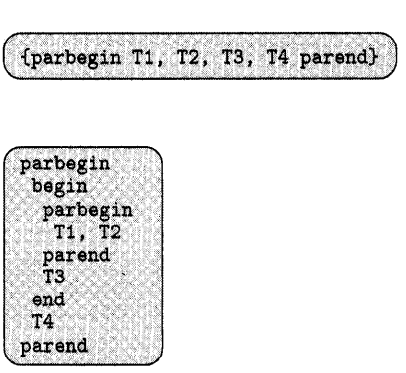


Fig. 10.42 *Parbegin-parend* construct for (*a*) four parallel tasks (Fig. 10.41*a*), (*b*) four parallel-sequential tasks (Fig. 10.41*b*)

The basic structure of an *elementary fork-join system* is shown in Fig. 10.43. The fork-join operation splits an arriving job into N tasks that arrive simultaneously at the N parallel processors $P_1, \ldots, P_N$. As soon as job i is served, it enters the join queue Q_i and waits until all N tasks are done. Then the job can leave the fork-join system.

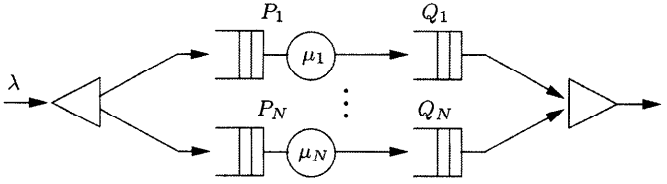


Fig. 10.43 Fork-join system.

A special case of the elementary fork-join system is the *fission-fusion system* (see Fig. 10.45), where all tasks are considered identical. A job can leave the

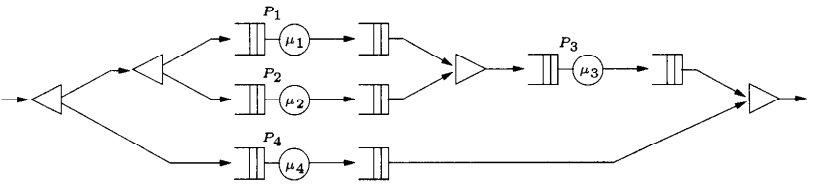


Fig. 10.44 Queueing network model for the execution of a job with the task precedence graph shown in Fig. 10.41*b*.

system, as soon as any N tasks are finished. These tasks do not necessarily have to belong to the same job.

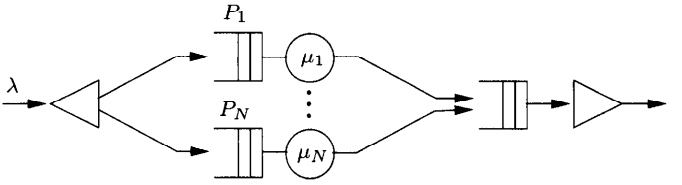


Fig. 10.45 Fission-fusion system.

Another special case is the *split-merge system* (see Fig. 10.46) where the N tasks of a job occupy all N processors. Only when all the N tasks have completed, can a new job occupy the processors. In this case there is a queue in front of the split station and there are no processor queues. This variant corresponds to a blocking situation: A completed task blocks the processor until all other tasks are finished.

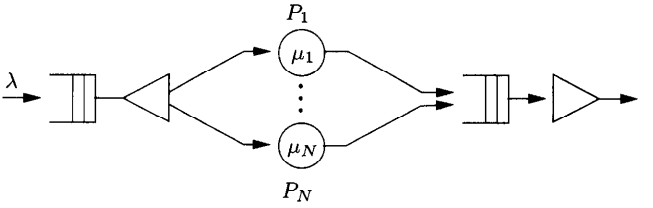


Fig. 10.46 Split-merge system.

10.6.2.2 Performance Measures In this section some performance measures of fork-join systems are defined. They are based on some basic measures of queueing systems that are defined as follows (see Fig. 10.47 and Section 7.2.1):

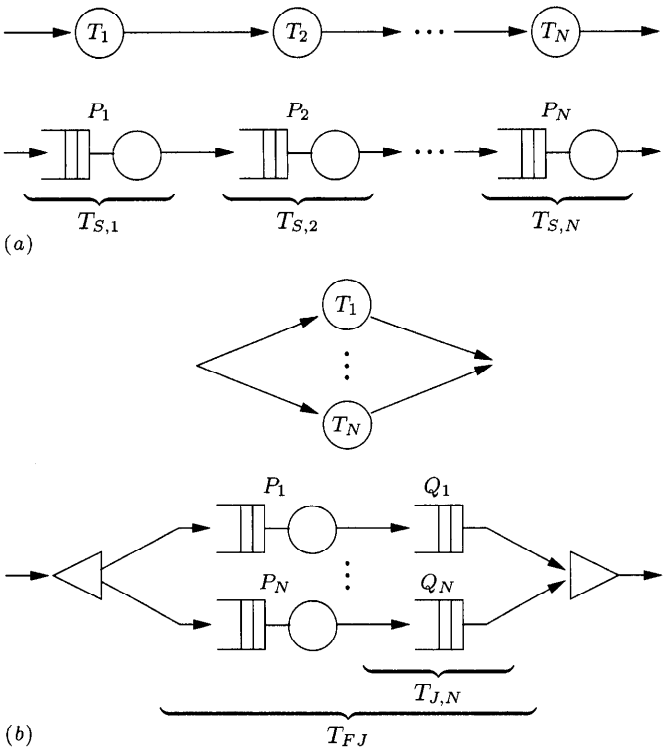


Fig. 10.47 System used for performance comparison: (*a*) sequential system, (*b*) parallel system.

$\overline{T}_{S,i}$ = The mean response time of task T_i belonging to a purely sequential job

$\overline{T}_S$ = The mean response time of a purely sequential job, equal to the sum of all $\overline{T}_{S,i}$

$\overline{T}_{\mathrm{FJ}}$ = The mean response time of parallel-sequential jobs with fork-join synchronization

$\overline{T}_{J,i}$ = The mean time spent by task T_i waiting for the join synchronization in queue Q_i

$\overline{Q}_i$ = The mean number of tasks waiting for the join synchronization in queue Q_i

$\overline{T}_J$ = The mean join time

$\overline{K}_{\mathrm{FJ}}$ = The mean number of parallel-sequential jobs with fork-join synchronization

In addition to these measures, the following other measures are of interest for parallel systems:

Speedup: Ratio of the mean response time in the system with N sequential tasks to the mean response time in a fork-join system with N parallel tasks:

$$G_N = \frac{\overline{T}_S}{\overline{T}_{\mathrm{FJ}}}. \tag{10.154}$$

Normalized Speedup:

$$G = \frac{G_N}{N}, \quad G \in \left[\tfrac{1}{N}, 1\right]. \tag{10.155}$$

Synchronization Overhead: Ratio of the mean time that tasks have to wait altogether in the join queues until all tasks are finished to the mean response time of the fork-join system:

$$S_N = \frac{\sum_{i=1}^{N} \overline{T}_{J,i}}{\overline{T}_{\mathrm{FJ}}}. \tag{10.156}$$

Normalized Synchronization Overhead:

$$S = \frac{S_N}{N}, S \in [0, 1]. \tag{10.157}$$

Blocking Factor: The blocking factor is the average total number of blocked tasks in the join queues:

$$B_N = \sum_{i=1}^{N} \overline{Q}_i. \tag{10.158}$$

Normalized Blocking Factor: The normalized blocking factor is given by:

$$B = \frac{B_N}{N}. \tag{10.159}$$

10.6.2.3 Analysis The method of [DuCz87] for the approximate analysis of fork-join systems is based on an application of the decomposition principle: We consider fork-join systems as *open* networks and replace the subnetwork that contains the fork-join construct by a composite load-dependent node. The service rates of the composite node are determined by analyzing the isolated (closed and) short-circuited subnetwork. This analysis is done for every number of tasks $(1 \ldots \infty)$. For practical reasons this number is, of course, limited. During the analysis, the mean number of tasks in the join queues is determined by considering the CTMC and computing the corresponding probabilities using a numerical solution method.

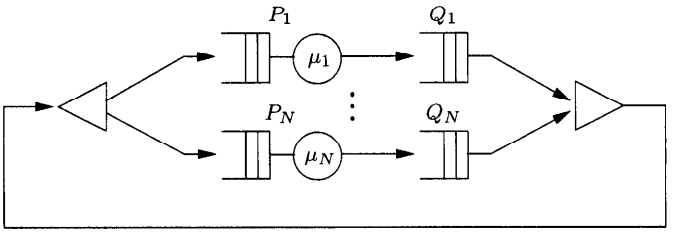


Fig. 10.48 Closed network with $N \cdot k$ tasks.

Consider an elementary fork-join system with N processors as shown in Fig. 10.43. To analyze this system, we consider the closed queueing network shown in Fig. 10.48. This network contains $N \cdot k$ tasks, $k = 1, 2, \ldots$.

The analysis of the network, via the numerical solution of the underlying CTMC, gives the throughput rates $\lambda(j)$, which are used as the load-dependent service rates $\mu(j)$ of a FES node that replaces the fork-join system (see Fig. 10.49). The analysis of the FES node finally gives the performance measures of the overall system.

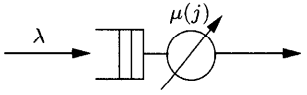


Fig. 10.49 Composite service station.

In the similar fashion, the performance measures of more complex fork-join systems can be computed because a job can have task precedence graphs with several nested fork and join constructs. The model for the service of a job has then the form of a series-parallel network with subnetworks that again contain fork-join systems. In Fig. 10.44 we have already given an example of such a model. The described decomposition principle is applied in this case several times in order to solve the fork-join subnetworks numerically so that they can be replaced by FES service stations.

Example 10.16 Consider a two-server fork-join queue. Here we assume that the service times at both servers are exponentially distributed with mean $1/\mu$. This system is shown in Fig. 10.43 (with $N = 2$). In order to solve the fork-join queue, we consider the FES queue with state-dependent service rate, shown in Fig. 10.49. The decomposition step consists of analyzing the closed subsystem with $2k$ tasks, $k = 1, 2, \ldots$ (Fig. 10.48). The service rate of the FES in Fig. 10.49 is set equal to the throughput of the closed subsystem. The solution of the FES gives overall performance measures. At the level of the closed subsystem we deal with forked tasks and delays caused by the join primitive, while at the FES level we obtain the performance measures for jobs.

Let $X(t)$ denote the total number of tasks in collectively both the join queues at time t. The stochastic process is a CTMC with the state space $\{i = 1, 2, \ldots, k\}$.

The state diagrams of the CTMCs for various values of k are shown in Fig. 10.50. Each of these CTMCs is a finite birth-death process, and thus the

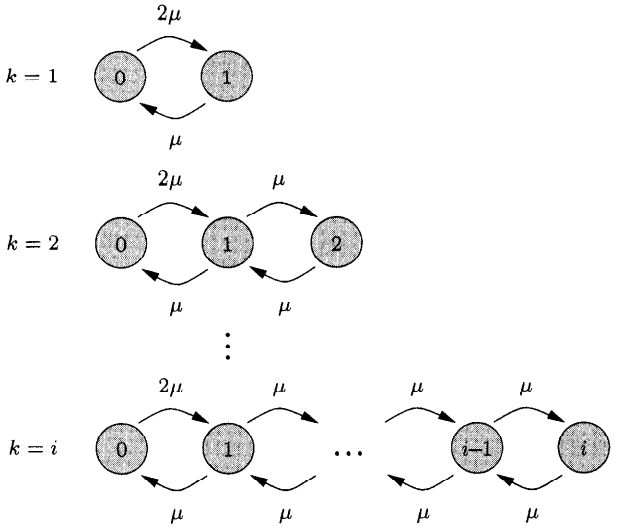


Fig. 10.50 CTMC for the total number of tasks waiting in the join queues.

steady-state probabilities can be derived using Eqs. (3.11) and (3.12):

$$\pi_0 = \frac{1}{1+2k}, \qquad \pi_i = 2\pi_0, \quad i = 1, 2, \ldots, k, \quad k = 1, 2, \ldots. \tag{10.160}$$

In order to calculate the throughput of jobs $\lambda(k)$, we observe that a successful join corresponds to a transition at rate μ from state i to state $(i-1)$ in the CTMCs of Fig. 10.50. Thus assigning reward rate μ to each of the states $i = 1, 2, \ldots, k$, we obtain the needed throughput as the expected reward rate in the steady states.

$$\lambda(k) = \mu \sum_{i=1}^{k} \pi_i = \frac{2k}{2k+1}\mu, \quad k = 1, 2, \ldots. \tag{10.161}$$

State-dependent service rates of the FES are then set equal to the throughput of the closed subsystem: $\mu(k) = \lambda(k), \quad k = 1, 2, \ldots$. The solution of the FES queue then gives the steady-state probabilities of the number of jobs in the

fork-join queue and is given by:

$$\nu_k = \nu_0 \rho^k \frac{\prod_{i=1}^{k} 2i+1}{\prod_{i=1}^{k} 2i}, \quad k = 1, 2, \ldots,$$

and with Eq. (3.12):

$$\begin{aligned} \nu_0 &= \frac{1}{1 + \sum_{k=1}^{\infty} \rho^k \frac{\prod_{i=1}^{k} 2i+1}{\prod_{i=1}^{k} 2i}} \\ &= (1-\rho)^{\frac{3}{2}}, \quad \text{(see [BrSe91], p.32)}, \end{aligned} \tag{10.162}$$

where $\rho = \lambda/\mu$. These results hold for $\rho < 1$, which is just the ergodicity condition for fork-join queues. Knowing the probabilities of Eq. (10.162), the expected number of jobs in the system is:

$$\overline{K}_{\mathrm{FJ}} = \sum_{k=1}^{\infty} k\nu_k = \frac{3}{2} \cdot \frac{\rho}{1-\rho}, \tag{10.163}$$

and, using Little's theorem:

$$\overline{T}_{\mathrm{FJ}} = \frac{\overline{K}_{\mathrm{FJ}}}{\lambda} = \frac{3}{2} \cdot \frac{1}{\mu(1-\rho)}. \tag{10.164}$$

If these values are compared to the corresponding ones for the M/M/1 queue, it can be seen that the mean number of jobs and the mean response time of the fork-join queue are both 3/2 times larger than those of the M/M/1 queue.

The performance indices of the synchronization primitives are derived in the following: With the mean response time $\overline{T}_{S,i}$ of task i of a purely sequential job (= mean response time of a M/M/1 queue, Eq. (6.15))

$$\overline{T}_{S,i} = \frac{1}{\mu(1-\rho)}, \quad i = 1, 2,$$

we obtain the mean response time of a purely sequential job:

$$\overline{T}_S = \sum_{i=1}^{2} \overline{T}_{S,i} = \frac{2}{\mu(1-\rho)}.$$

Equations (10.154) and (10.164) are used to derive the speedup:

$$G_N = \frac{\overline{T}_S}{\overline{T}_{\mathrm{FJ}}} = \frac{4}{3}.$$

The normalized speedup is computed using Eq. (10.155). With $N = 2$ we get:

$$G = \frac{G_N}{N} = \frac{2}{3}.$$

To compute the synchronization overhead S_N, we need the average time a task spends in the join queues. This time is just given by the difference between $\overline{T}_{\mathrm{FJ}}$ and $\overline{T}_{\mathrm{M/M/1}}$ (where $\overline{T}_{\mathrm{M/M/1}}$ is the time a task spends in the servers). If:

$$\overline{T}_{J,i} = \overline{T}_{\mathrm{FJ}} - \overline{T}_{\mathrm{M/M/1}}, \quad i = 1, 2, \tag{10.165}$$

then the synchronized overhead is given by (Eq. (10.156)):

$$S_N = \frac{\sum_{i=1}^{2} \overline{T}_{J,i}}{\overline{T}_{\mathrm{FJ}}} = \frac{3}{2},$$

and the normalized synchronization overhead (Eq. (10.157)):

$$S = \frac{S_N}{N} = \frac{1}{3}.$$

In order to obtain the blocking factor B_N, we need the mean number of tasks in the join queues. This number can be determined from Eq. (10.165), using Little's theorem:

$$\begin{aligned} B_N &= \sum_{i=1}^{2} \overline{Q}_{J,i} = \sum_{i=1}^{2} \lambda \overline{T}_{J,i} = \lambda \sum_{i=1}^{2} \left(\frac{3}{2} \frac{1}{\mu(1-\rho)} - \frac{1}{\mu(1-\rho)} \right) \\ &= \frac{\rho}{1-\rho}, \end{aligned}$$

and the normalized blocking factor is given by:

$$B = \frac{1}{2} \frac{\rho}{1-\rho}.$$

Note that the speedup as well as the synchronization overhead do not depend on the utilization ρ, and that the speedup is rather poor. It can be shown that the speedup G_2 increases from 4/3 to 2 if the coefficient of variation of the service time decreases from 1 (exponential distribution) to 0 (deterministic distribution). In Tables 10.28, 10.29, and 10.30, the speedup is given for different distribution functions of the service times. It is assumed that jobs arrive from outside with arrival rate $\lambda = 0.1$. The service rate is $\mu = 10$.

For the case $c_X < 1$, we use an Erlang-k distribution and get the results shown in Table 10.28. For $c_X > 1$, we use a hyperexponential distribution and get the results shown in Table 10.29. Because of the phase-type service time distribution, the method discussed so far can easily be extended. Underlying

Table 10.28 System speedup with Erlang-k distributed service times

	Exact	Erlang-k							
var(X)	0.0	$\frac{1}{1000}$	$\frac{1}{800}$	$\frac{1}{600}$	$\frac{1}{500}$	$\frac{1}{400}$	$\frac{1}{300}$	$\frac{1}{200}$	$\frac{1}{100}$
c_X	0.0	$\frac{1}{\sqrt{10}}$	$\frac{1}{\sqrt{8}}$	$\frac{1}{\sqrt{6}}$	$\frac{1}{\sqrt{5}}$	$\frac{1}{\sqrt{4}}$	$\frac{1}{\sqrt{3}}$	$\frac{1}{\sqrt{2}}$	1.0
Speedup	2.0	1.760	1.671	1.630	1.605	1.569	1.524	1.453	1.333

Table 10.29 System speedup with hyperexponentially distributed service times.

var(X)	2.0	4.0	6.0	8.0	10.0	20.0
Speedup	1.060	1.043	1.029	1.028	1.027	1.026

CTMCs will now be more complex than those shown in Fig. 10.50. We chose to generate and solve these CTMCs using the MOSES tool (see Chapter 12).

Finally we show in Table 10.30 the system speedup if the service times are normally distributed with standard deviation σ_X and mean value $\mu = 10$. The results in this case were obtained using DES. Using a similar approach,

Table 10.30 System speedup with normally distributed service times

var(X)	0.001	0.005	0.01	0.02	0.08	0.1	0.2	0.4
Speedup	1.697	1.525	1.487	1.463	1.439	1.437	1.433	1.431

performance measures for the fission fusion system (see Fig. 10.45) and the split merge system (see Fig. 10.46) can be computed [DuCz87].

For fork-join systems with a large number of processors, [Duda87] developed an approximate method for the analysis of the subnetworks. To demonstrate this method we consider a series-parallel closed fork-join network as shown in Fig. 10.51. In this method it is assumed that all service times of the tasks are exponentially distributed. The method is based on construction of a product-form network with the same topology as the original fork-join subnetwork and approximately the same number of states in the corresponding CTMC. The main difficulty when following this procedure is to determine the number of states in the fork-join subnetwork. If N denotes the number of parallel branches where the ith branch contains exactly n_i nodes and k tasks, then the number of ways to distribute k tasks to $n_i + 1$ nodes (including the join queue Q_i) is given by:

$$z_i(k) = \binom{n_i + k}{n_i}. \tag{10.166}$$

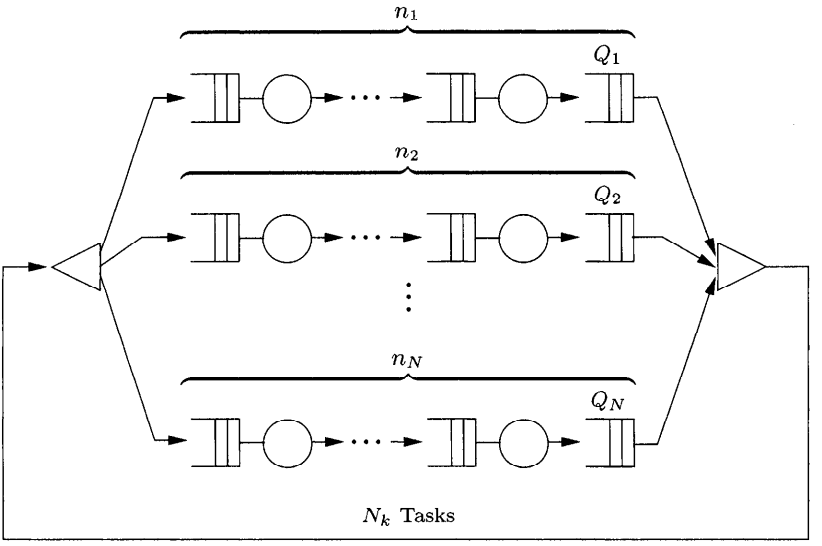


Fig. 10.51 Series-parallel closed fork-join network.

Then the number of ways to distribute $N \cdot k$ tasks to all nodes is given by the following expression:

$$\prod_{i=1}^{N} z_i(k). \tag{10.167}$$

This number can be limited further. If there is exactly one job in each join queue Q_i, then the jobs are passed immediately to the first station. Thus $\prod_{i=1}^{N} z_i(k-1)$ of the given combinations are not possible because this is the number of possible ways to distribute $k-1$ tasks over n_i stations, given that there is one task in each join queue Q_i. The number of states in the fork-join subnetwork is therefore given by:

$$Z_{\mathrm{FJ}}(k) = \prod_{i=1}^{N} z_i(k) - \prod_{i=1}^{N} z_i(k-1). \tag{10.168}$$

In the following steps, the algorithm for the analysis of fork-join networks with possibly several nested fork-join subnetworks is given. The replacement of these fork-join subsystems by FES nodes is done iteratively.

STEP 1 Choose one subsystem that does not contain any other fork-join structures and construct a closed network with $N \cdot k$ tasks by using a short circuit.

STEP 2 Compute the number of states $Z_{\mathrm{FJ}}(k)$ of this closed subnetwork for $k = 1, 2, \ldots$ (Eq. (10.168)).

STEP 3 Compute the throughput of the subnetwork.

STEP 3.1 Construct a product-form network with exactly the same $M = \sum_{i=1}^{N} n_i$ number of nodes as the subnetwork. The number of jobs K in the product-form network is chosen so that the number of states in the product-form network is approximately equal to the number of states in the fork-join subnetwork. Since the state space structures of both networks are almost identical, the throughput of the product-form network $\lambda_{\mathrm{PF}}(K)$ will be approximatively equal to the throughput of the fork-join network $\lambda_{\mathrm{FJ}}(k)$. Thus we need to determine K by using Eqs. (10.166) and (10.168) so that for $k = 1, 2, \ldots$:

$$K : |Z_{\mathrm{FJ}}(k) - Z_{\mathrm{PF}}(K)| = \min_{l} |Z_{\mathrm{FJ}}(k) - Z_{\mathrm{PF}}(l)|, \tag{10.169}$$

with $Z_{\mathrm{PF}}(K) = z_{M-1}(K)$ holding.

STEP 3.2 Solve the product-form network for all possible numbers of jobs K using an algorithm such as MVA and determine the throughput as a function of K. This throughput $\lambda_{\mathrm{PF}}(K)$ of the product-form network is approximately equal to the throughput $\lambda_{\mathrm{FJ}}(k)$ of the fork-join subnetwork for the value of K corresponding to k.

STEP 4 Replace the subnetwork in the given system by a composite node whose load-dependent service rates are equal to the determined throughput rates:

$$\mu(k) = \lambda_{\mathrm{FJ}}(k) \quad \text{for } k = 1, 2 \ldots .$$

STEP 5 If the network under consideration contains only one FES node, then the analysis of this node gives the performance measures for the whole network. Otherwise go back to Step 1.

STEP 6 From these performance measures, special characteristic values of parallel systems such as the speedup G_N and the synchronization overhead S_N can be computed.

The algorithm is now demonstrated on a simple example.

Example 10.17 Consider a parallel program with the precedence graph depicted in Fig. 10.41b. The model in the form of a series parallel fork-join network is presented in Fig. 10.44. The arrival rate of jobs is $\lambda = 0.04$. All service times of the tasks are exponentially distributed with the mean values:

$$\frac{1}{\mu_1} = 10, \quad \frac{1}{\mu_2} = 5, \quad \frac{1}{\mu_3} = 2, \quad \frac{1}{\mu_4} = 10.$$

The analysis of the model is carried out in the steps that follow:

STEP 1 At first the inner fork-join system, consisting of the processors P_1 and P_2, is chosen and short-circuited. This short circuited model consists of $2 \cdot k$ tasks, $k = 1, 2, \ldots$ (Fig. 10.52).

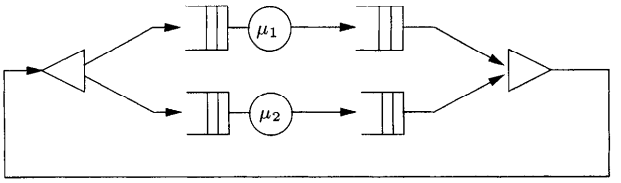


Fig. 10.52 Short-circuited model.

Table 10.31 Number of states in the subsystem.

k	1	2	3	4	5	6	7	8	9	10
Z_{FJ}	3	5	7	9	11	13	15	17	19	21
K	3	5	7	9	11	13	15	17	19	21
Z_{PF}	3	5	7	9	11	13	15	17	19	21

STEP 2 The number of states Z_{FJ} of this subsystem is determined using Eq. (10.168). For $k = 1, \ldots, 10$, the results are shown in Table 10.31.

STEP 3 Determine the throughput of the subnetwork.

STEP 3.1 Construct a product-form network with $N = 2$ nodes. The number of jobs K in this network are determined using Eq. (10.169). For $k = 1, 2 \ldots, 10$ the values for K and the number of states $Z_{\mathrm{PF}}(K)$ are given in Table 10.31.

STEP 3.2 Using MVA we solve the product-form network, constructed in Step 3.1, for all possible number of jobs K. The throughputs are given by:

$$\lambda_{\mathrm{PF}}(3) = \underline{0.093}, \quad \lambda_{\mathrm{PF}}(5) = \underline{0.098},$$
$$\lambda_{\mathrm{PF}}(7) = \underline{0.100}, \quad \lambda_{\mathrm{PF}}(9) = \underline{0.100},$$
$$\vdots$$

These values approximately correspond to the throughputs $\lambda_{\mathrm{FJ}}(k)$ of the fork-join subnetwork:

$$\lambda_{\mathrm{FJ}}(1) = \lambda_{\mathrm{PF}}(3) = \underline{0.093},$$
$$\lambda_{\mathrm{FJ}}(2) = \lambda_{\mathrm{PF}}(5) = \underline{0.098},$$
$$\lambda_{\mathrm{FJ}}(3) = \lambda_{\mathrm{PF}}(7) = \underline{0.100},$$
$$\lambda_{\mathrm{FJ}}(4) = \lambda_{\mathrm{PF}}(9) = \underline{0.100},$$
$$\vdots$$

STEP 4 The analyzed fork-join subnetwork, consisting of the processors P_1 and P_2, is replaced by an FES node (see Fig. 10.53) with the load-dependent service rates $\mu(k) = \lambda_{\mathrm{FJ}}(k)$ for $k = 1, 2, \ldots$.

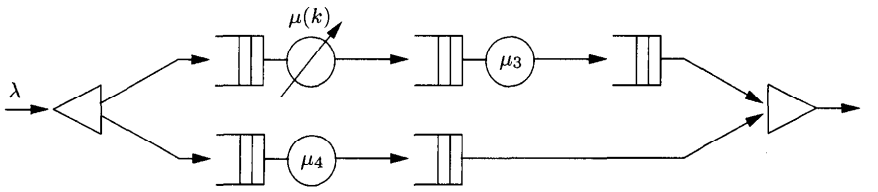


Fig. 10.53 Fork-join subnetwork with one FES node.

STEP 5 This network contains more than one node and therefore we return to Step 1.

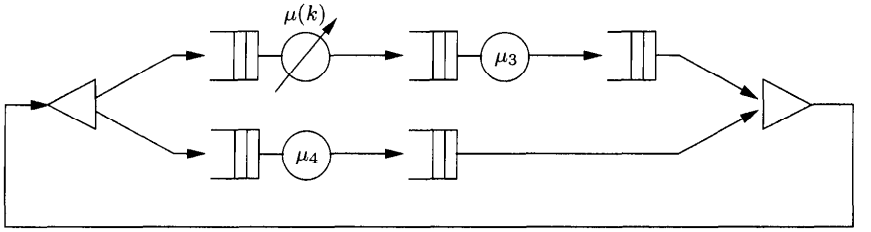


Fig. 10.54 Short-circuited network.

STEP 1 The short circuit of the network shown in Fig. 10.53 results in a closed network with $2 \cdot k$ tasks, $k = 1, 2, \ldots$ (Fig. 10.54).

STEP 2 The possible number of states Z_{FJ} in this network is given by Eq. (10.168). For $k = 1, \ldots, 10$ the values are shown in the Table 10.32.

Table 10.32 The possible number of states Z_{FJ} as a function of $k = 1, \ldots, 10$

k	1	2	3	4	5	6	7	8	9	10
Z_{FJ}	5	12	22	35	51	70	92	117	145	176
K	2	3	5	7	9	10	12	14	16	17
Z_{PF}	6	10	21	36	55	66	91	120	153	171

STEP 3 Determine the throughputs.

STEP 3.1 Construct a product-form network consisting of $N = 3$ nodes and determine the number of jobs K in the network using Eq. (10.169). For $k = 1, \ldots, 10$ the values of K and $Z_{\mathrm{PF}}(K)$ are given in Table 10.32.

STEP 3.2 Solve the network for all number of jobs K. For the computed throughputs, the following relation approximately holds: $\lambda_{\mathrm{PF}}(K) = \lambda_{\mathrm{FJ}}(k)$.

Using the MVA we get:

$$\lambda_{FJ}(1) \approx \lambda_{PF}(2) = \underline{0.062}, \quad \lambda_{FJ}(2) \approx \lambda_{PF}(3) = \underline{0.072},$$
$$\lambda_{FJ}(3) \approx \lambda_{PF}(5) = \underline{0.082}, \quad \lambda_{FJ}(4) \approx \lambda_{PF}(7) = \underline{0.087},$$
$$\lambda_{FJ}(5) \approx \lambda_{PF}(9) = \underline{0.089}, \quad \lambda_{FJ}(6) \approx \lambda_{PF}(10) = \underline{0.090},$$
$$\lambda_{FJ}(7) \approx \lambda_{PF}(12) = \underline{0.092}, \quad \lambda_{FJ}(8) \approx \lambda_{PF}(14) = \underline{0.093},$$
$$\lambda_{FJ}(9) \approx \lambda_{PF}(16) = \underline{0.094}, \quad \lambda_{FJ}(10) \approx \lambda_{PF}(17) = \underline{0.094}.$$

STEP 4 The fork-join subnetwork is now an FES node (see Fig. 10.55) with the load-dependent service rates $\mu'(k) = \lambda_{FJ}(k), \quad k = 1, 2, \ldots$.

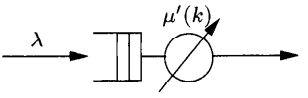


Fig. 10.55 FES queue for Example 10.17.

STEP 5 The fork-join network is now reduced to only one node and therefore the iteration stops and the performance measures of the system can be determined by analyzing the FES node of Fig. 10.55. For this analysis we determine at first the steady-state probabilities of the node using Eqs. (3.11) and (3.12):

$$\pi_0 = \underline{0.428}, \quad \pi_1 = \underline{0.287}, \quad \pi_2 = \underline{0.155},$$
$$\pi_3 = \underline{0.076}, \quad \pi_4 = \underline{0.034}, \quad \pi_5 = \underline{0.015},$$
$$\pi_6 = \underline{0.007}, \quad \pi_7 = \underline{0.003}, \quad \pi_8 = \underline{0.001},$$
$$\pi_9 = \underline{0.001}, \quad \pi_{10} = \underline{0.000}.$$

These probabilities are then used to determine the mean number of jobs in the fork-join model, Eq. (6.8):

$$\overline{K}_{FJ} = \underline{1.116},$$

and the mean response time, Eq. (6.9):

$$\overline{T}_{FJ} = \underline{27.89}.$$

With Eq. (6.15) and:

$$\overline{T}_S = \sum_{i=1}^{4} \overline{T}_{S,i} = \sum_{i=1}^{4} \frac{1}{\mu_1(1-\rho_i)},$$

we obtain:

$$\overline{T}_S = \underline{41.76}.$$

To obtain the speedup, we use Eq. (10.154) and get:

$$G_N = \underline{1.497}.$$

From this result we can compute the normalized speedup (Eq. (10.155)):

$$G = \frac{G_N}{N} = \underline{0.374}.$$

To obtain the synchronization overhead S_N, we need the mean waiting time $\overline{T}_{J,i}$, $(i = 1, 2, 3, 4)$ in synchronization queues. First we consider the fork-join queue of node 1 and node 2:

$$\overline{T}_{J,i} = \overline{T}_{FJ_{12}} - \overline{T}_i, \tag{10.170}$$

where $\overline{T}_{FJ_{12}}$ is the mean response time in the fork-join construct consisting of node 1 and node 2. It is obtained from the state-dependent service rate $\mu(k)$ for the composite node of the inner fork-join construct. At first we need the state probabilities, using Eqs. (3.11) and (3.12):

$$\begin{aligned} \pi_0 &= \underline{0.580}, \quad \pi_1 = \underline{0.249}, \quad \pi_2 = \underline{0.102}, \\ \pi_3 &= \underline{0.041}, \quad \pi_4 = \underline{0.016}, \quad \pi_5 = \underline{0.007}, \\ \pi_6 &= \underline{0.003}, \quad \pi_7 = \underline{0.001}, \quad \pi_8 = \underline{0.000}, \end{aligned}$$

and the mean number of jobs:

$$\overline{K}_{FJ_{12}} = \sum_{k=1}^{\infty} k\pi_k = \underline{0.701}.$$

If we apply Little's theorem, we obtain for the mean response time of a parallel-sequential job with fork-join synchronizations:

$$\overline{T}_{FJ_{12}} = \frac{\overline{K}_{FJ_{12}}}{\lambda} = \underline{17.514}.$$

In order to obtain the mean response time, we apply Eq. (6.15) and get:

$$\overline{T}_1 = \frac{1}{\mu_1(1-\rho_1)} = \underline{16.667}, \quad \overline{T}_2 = \underline{6.25}.$$

Using Eq. (10.170) we finally get the the results:

$$\overline{T}_{J,1} = \overline{T}_{FJ_{12}} - \overline{T}_1 = \underline{0.874}, \quad \overline{T}_{J,2} = \underline{11.264}.$$

From Fig. 10.53 we have:

$$\overline{T}_{J,3} = \overline{T}_{\mathrm{FJ}} - \overline{T}_{FJ_{12}} - \overline{T}_3, \qquad \overline{T}_{J,4} = \overline{T}_{\mathrm{FJ}} - \overline{T}_4.$$

Applying Eq. (6.15) again:

$$\overline{T}_3 = \underline{2.174}, \qquad \overline{T}_4 = \underline{16.667}.$$

Together with $\overline{T}_{\mathrm{FJ}}$ and $\overline{T}_{FJ_{12}}$ from the preceding we get:

$$\overline{T}_{J,3} = \underline{8.202}, \qquad \overline{T}_{J,4} = \underline{11.233},$$

which finally gives us the synchronization overhead (see Eq. (10.156)):

$$S_N = \frac{\sum_{i=1}^{4} \overline{T}_{J,i}}{\overline{T}_{\mathrm{FJ}}} = \underline{1.132}.$$

The normalized synchronization overhead is given by (Eq. (10.157)):

$$S = \frac{S_N}{N} = \underline{0.292}.$$

Now it is easy to calculate the blocking factor. Using Little's theorem, we obtain, via Eq. (10.158):

$$B_N = \sum_{i=1}^{4} \overline{Q}_i = 1.263,$$

and with Eq. (10.159) the normalized blocking factor is given by:

$$B = \frac{B_N}{4} = 0.316.$$

A comparison of the mean response time computed with the preceding approximation with DES shows that the differences between the DES results ($\overline{T} = 26.27$) and the approximation results are small in our example. For fork-join systems with higher arrival rates, the differences become larger. It can be shown in general that the power of the fork-join operations is reduced considerably by join synchronization operations. Also the speedup is reduced considerably by using synchronization operations. Other approaches to the analysis of parallel programs can be found in [Flei89], [MüRo87], [NeTa88], [SaTr87], and [ThBa86].

Problem 10.5 Consider a parallel program with the precedence graph shown in Fig. 10.56. The arrival rate is $\lambda = 1$ and the service rates of the tasks are $\mu_1 = 2$, $\mu_2 = 4$, $\mu_3 = 4$, $\mu_4 = 5$. All service times are exponentially distributed. Determine the mean response time and the speedup.

10.7 NETWORKS WITH ASYMMETRIC NODES

Whenever we have considered multiserver nodes in a network, we have always assumed that all the servers at the node are statistically identical. We now

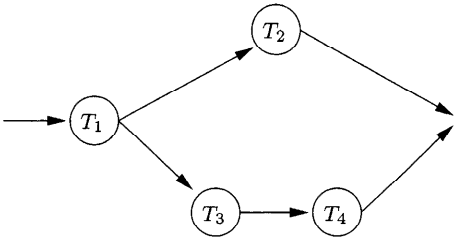


Fig. 10.56 Precedence graph for Problem 10.5.

consider the case of networks where servers at a multiserver node have different service times. We first consider closed networks with asymmetric nodes followed by open networks with asymmetric nodes. Note that in each of these cases we will only obtain an approximative solution.

10.7.1 Closed Networks

In this section we show how to modify the summation method, the mean value analysis, and the SCAT algorithm to deal with asymmetric nodes[BoRo94] as introduced in Section 9.1.2.

10.7.1.1 Asymmetric SUM (ASUM) To extend the SUM (see Section 9.2) to queueing networks with asymmetric nodes, we need to adapt the formula for computing the mean number of jobs (see Eqs. (9.15) and (6.28)):

$$\overline{K}_i = m_i \cdot \rho_i + \frac{\rho_i}{1 - \frac{K - m_i - 1}{K - m_i} \cdot \rho_i} \cdot P_{m_i}, \quad \text{Type-1},$$

with:

$$P_{m_i}(\rho_i) = \frac{\frac{(m_i \cdot \rho_i)^{m_i}}{m_i! \cdot (1 - \rho_i)}}{\sum_{k=0}^{m_i - 1} \frac{(m_i \cdot \rho_i)^k}{k!} + \frac{(m_i \cdot \rho_i)^{m_i}}{m_i \cdot (1 - \rho_i)}}.$$

For symmetric nodes, ρ_i is given by:

$$\rho_i = \frac{\lambda_i}{m_i \cdot \mu_i},$$

and for asymmetric nodes we can use Eq. (6.136) or Eqs. (6.150) and (6.146), respectively. Since the SUM is an approximate method, so is the extended SUM. By replacing ρ_i and P_{m_i} by the corresponding formulae, the BOTT can be extended to queueing networks with asymmetric nodes in the same way.

10.7.1.2 Asymmetric MVA (AMVA) The MVA was developed by Reiser and Lavenberg [ReLa80] for the exact analysis of product-form networks and is based on Little's theorem and the arrival theorem of Reiser, Lavenberg, Sevcik, and Mitrani (see Section 8.2) [Litt61, ReLa80, SeMi81]. To extend the MVA to networks with asymmetric nodes, the equation for the calculation of the mean response time for symmetric -/M/m nodes:

$$\overline{T}_i(k) = \frac{1}{\mu_i \cdot m_i} \left(1 + \overline{K}_i(k-1) + \sum_{j=0}^{m_i-2} (m_i - j - 1) \cdot \pi_i(j \mid k-1) \right), \tag{10.171}$$

needs to be extended. Here $\pi_i(j \mid k)$ is the probability that there are j jobs at the ith node, given that there are k jobs in the network. This condition probability $\pi_i(j \mid k)$ is given by:

$$\pi_i(j \mid k) = \frac{e_i \cdot \lambda(k)}{\mu_i \cdot j} \cdot \pi_i(j-1 \mid k-1), \quad j = 1, \ldots, m_i - 1, \tag{10.172}$$

and $\pi_i(0 \mid k)$ is obtained by:

$$\pi_i(0 \mid k) = 1 - \frac{1}{m_i} \left[\frac{e_i}{\mu_i} \cdot \lambda(k) + \sum_{j=1}^{m_i-1} (m_i - j) \cdot \pi_i(j \mid k) \right]. \tag{10.173}$$

In Eq. (10.173), the factor $\mu_i \cdot m_i$ is the overall service rate of the symmetric -/M/m node. For asymmetric nodes $m_i \cdot \mu_i$ is replaced by:

$$\sum_{j=1}^{m_i} \mu_{ij}.$$

Thus the mean response time of an asymmetric node can be computed as follows:

$$\overline{T}_i(k) = \frac{1}{\sum_{j=1}^{m_i} \mu_{ij}} \left(1 + \overline{K}_i(k-1) + \sum_{j=0}^{m_i-2} (m_i - j - 1) \cdot \pi_i(j \mid k-1) \right). \tag{10.174}$$

Because of the different service rates, it is necessary to take into account which servers are occupied as well as the number of servers when calculating the marginal probabilities $\pi_i(j \mid k)$. If we assume that the free server with the highest service rate is selected and the service rates are arranged in descending

order, then the marginal probabilities are given by:

$$\pi_i(j \mid k) = \frac{e_i \cdot \lambda(k)}{\sum_{l=1}^{j} \mu_{il}} \cdot \pi_i(j-1 \mid k-1), \tag{10.175}$$

$$\pi_i(0 \mid k) = 1 - \frac{e_i}{\sum_{l=1}^{m_i} \mu_{il}} \cdot \lambda(k) - \frac{1}{m_i} \cdot \sum_{j=1}^{m_i-1} (m_i - j) \cdot \pi_i(j \mid k). \tag{10.176}$$

If we consider Eq. (10.173) in more detail, we see that the factor:

$$\frac{e_i \cdot \lambda(k)}{\sum_{l=1}^{m_i} \mu_{il}}$$

is the utilization ρ_i and, therefore, Eq. (10.176) can be rewritten as:

$$\pi_i(0 \mid k) = 1 - \rho_i(k) - \frac{1}{m_i} \cdot \sum_{j=1}^{m_i-1} (m_i - j) \cdot \pi_i(j \mid k).$$

10.7.1.3 Asymmetric SCAT (ASCAT) The core of the SCAT algorithm is a single step of the MVA in which the formula for the mean response time, Eq. (10.174), is approximated so that it depends only on K and not on $(K-1)$. However, it is necessary to estimate the marginal probabilities $\pi_i(j \mid K-1)$ that need a lot of computation time. For this reason, another formula for the mean response time is derived from Eq. (6.29). This derivation depends only on $\overline{K}_i$ and μ_i. For -/M/1 nodes we apply Little's theorem to the following equation:

$$\overline{K}_i = \frac{\rho_i}{1-\rho_i},$$

and after reorganizing we get:

$$\overline{T}_i = \frac{1}{\mu_i}(1 + \overline{K}_i),$$

which is the basic equation of the core of the SCAT algorithm for -/M/1 nodes. Similarly, from Eq. (6.29) we obtain:

$$\overline{K}_i = m_i\rho_i + \frac{\rho_i}{(1-\rho_i)} \cdot P_{m_i}(\rho_i),$$

and for -/M/m nodes we have:

$$\overline{T}_i = \frac{1}{\mu_i \cdot m_i} \cdot (m_i + \overline{K}_i + P_{m_i}(\rho_i) - m_i \cdot \rho_i).$$

This formula can easily be extended to asymmetric nodes as follows:

$$\overline{T}_i = \frac{1}{\sum_{j=1}^{m_i} \mu_{ij}} \cdot \left(m_i + \overline{K}_i + P_{m_i}(\rho_i) - m_i \cdot \rho_i\right).$$

At this point we proceed analogously to the SUM by using the equations belonging to the selected method to calculate ρ_i and $P_{m_i}(\rho_i)$. Our experience with these methods suggests that the mean deviation for ASUM and AMVA is in the order of 2-3% for the random selection of a server as well as for the selection of the fastest free server. For ASCAT, the situation is somewhat different. For random selection the deviation is considerably high (7%), while for the selection of the fastest free server the mean deviation is only of the order of 1%. Therefore, this method can be recommended. ASUM, AMVA, and ASCAT can easily be extended to multiclass queueing networks [AbBo97].

10.7.2 Open Networks

Jackson's method for analyzing open queueing networks with symmetric nodes (see Section 7.3.4) can also be extended to analyze open queueing networks with asymmetric nodes. Note that this will result in an approximate solution. Jackson's theorem can be applied to open single class queueing networks with symmetric -/M/m-FCFS nodes.

The analysis of an open Jackson network is done in two steps:

STEP 1 Calculate the arrival rates λ_i using traffic Eq. (7.1) for all nodes $i = 1 \ldots N$.

STEP 2 Consider each node i as an elementary -/M/m-FCFS queueing system. Check whether the ergodicity ($\rho < 1$) is fulfilled and calculate the performance measures using the known formulae.

Step 1 remains unchanged when applied to asymmetric open networks. In Step 2, formulae given in Section 6.16 for asymmetric -/M/m-FCFS queueing networks should be used instead of formulae for symmetric -/M/m-FCFS queueing systems. Similarly, the BCMP theorem for open networks (see Section 7.3.6) can be extended to the approximate analysis of asymmetric queueing networks. Using the closing method (see Section 10.1.5) AMVA, ASCAT and ASUM can also be applied to open asymmetric queueing networks. Furthermore, it is also possible to apply the closing method to open and mixed queueing networks with asymmetric nodes.

Example 10.18 To demonstrate different methods for networks with asymmetric nodes we use the example network shown in Fig. 10.57. Node 1 is an asymmetric -/M/2, nodes 2 and 3 are of -/M/1 type, and node 4 is -/G/∞. The routing probabilities are:

$$p_{12} = 0.5, \quad p_{13} = 0.5, \quad p_{24} = p_{34} = p_{41} = 1,$$

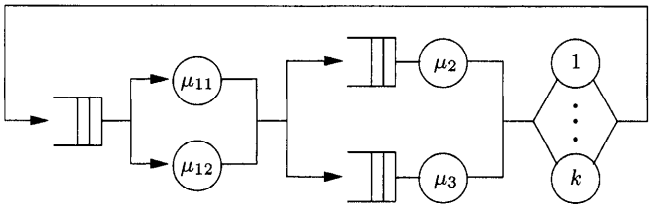


Fig. 10.57 Closed network with one asymmetric node.

and the service rates are:

$$\mu_{11} = 3.333, \quad \mu_{12} = 0.666, \quad \mu_2 = 1.666, \quad \mu_3 = 1.25, \quad \mu_4 = 1.$$

The total number of jobs in the network is $K = 3$. From the routing probabilities we obtain the following visit ratios:

$$e_1 = 1, \quad e_2 = 0.5, \quad e_3 = 0.5, \quad e_4 = 1.$$

We start with the ASUM method, which consists of the same steps as the SUM algorithm (see Section 9.2), but with different formulae for the utilization ρ_1 and the probability of waiting P_{m_1}. For the stopping condition we use $\epsilon = 0.001$.

STEP 1 Initialization:

$$\lambda_l = 0 \quad \text{and} \quad \lambda_u = \min_i \left\{ \frac{\sum_{l=1}^{m_i} \mu_{il}}{e_i} \right\} = \underline{2.5}.$$

STEP 2 Bisection:

STEP 2.1 $\lambda = \dfrac{\lambda_l + \lambda_u}{2} = \underline{1.25}.$

STEP 2.2 Calculation of the values $f_i(\lambda_i)$, $i = 1, \ldots, 4$. In order to get the result for node 1, Eqs. (6.146) and (6.150) for the exact analysis of networks with asymmetric nodes are used (selection of the fastest free server). With:

$$\rho_1 = \frac{e_1 \cdot \lambda}{\mu_{11} + \mu_{12}} = \frac{1.25}{4} = \underline{0.3125}$$

and:

$$P_{m_1} = \frac{1}{1-c} \cdot \frac{P_m^{(L)}}{N} = \frac{1}{1 - 0.3125} \cdot \frac{0.111}{1.050} = \underline{0.153},$$

we get:

$$\overline{K}_1(\lambda_1) = 2\rho_1 + \rho_1 \cdot P_{m_1} = \underline{0.673}.$$

Accordingly we obtain:

$$\overline{K}_2(\lambda_2) = \frac{\rho_2}{1 - \frac{2}{3}\rho_2} = 0.500 \quad \text{with } \rho_2 = \underline{0.375},$$

$$\overline{K}_3(\lambda_3) = \frac{\rho_3}{1 - \frac{2}{3}\rho_3} = 0.750 \quad \text{with } \rho_3 = \underline{0.500},$$

$$\overline{K}_4(\lambda_4) = \frac{\lambda_4}{\mu_4} = \underline{1.250},$$

which yields:

$$g(\lambda) = \sum_{i=1}^{N} \overline{K}_i(\lambda_i) = \underline{3.173}.$$

STEP 2.3 Check the stopping condition. Because of $g(\lambda) > K$, we set $\lambda_u = \lambda = 1.25$.

STEP 2.1 $\lambda = \dfrac{\lambda_l + \lambda_u}{2} = \underline{0.625}$.

STEP 2.2 Calculation of the values $f_i(\lambda_i)$, $i = 1, \ldots, 4$. With:

$$\overline{K}_1(\lambda_1) = 2\rho_1 + \rho_1 \cdot P_{m_1} = \underline{0.319}. \quad \text{with } \rho_1 = \underline{0.156} \text{ and } P_{m_1} = \underline{0.0386},$$

$$\overline{K}_2(\lambda_2) = \frac{\rho_2}{1 - \frac{2}{3}\rho_2} = \underline{0.214} \quad \text{with } \rho_2 = \underline{0.188},$$

$$\overline{K}_3(\lambda_3) = \frac{\rho_3}{1 - \frac{2}{3}\rho_3} = \underline{0.300} \quad \text{with } \rho_3 = 0.250,$$

$$\overline{K}_4(\lambda_4) = \frac{\lambda_4}{\mu_4} = \underline{0.625},$$

we get:

$$g(\lambda) = \sum_{i=1}^{N} \overline{K}_i(\lambda_i) = \underline{1.458}.$$

STEP 2.3 Check the stopping condition. Because of $g(\lambda) < K$, we set $\lambda_l = \lambda = 0.625$.

STEP 2.1 $\lambda = \dfrac{\lambda_l + \lambda_u}{2} = \underline{0.938}$.

STEP 2.2 Calculation of the values $f_i(\lambda_i)$, $i = 1, \ldots, 4$. With

$$\overline{K}_1(\lambda_1) = 2\rho_1 + \rho_1 \cdot P_{m_1} = \underline{0.489}. \quad \text{with } \rho_1 = \underline{0.234} \text{ and } P_{m_1} = \underline{0.0881},$$

$$\overline{K}_2(\lambda_2) = \frac{\rho_2}{1 - \frac{2}{3}\rho_2} = \underline{0.346} \quad \text{with } \rho_2 = \underline{0.281},$$

$$\overline{K}_3(\lambda_3) = \frac{\rho_3}{1 - \frac{2}{3}\rho_3} = \underline{0.500} \quad \text{with } \rho_3 = \underline{0.375},$$

$$\overline{K}_4(\lambda_4) = \frac{\lambda_4}{\mu_4} = \underline{0.938},$$

we get:

$$g(\lambda) = \sum_{i=1}^{N} \overline{K}_i(\lambda_i) = \underline{2.273}.$$

STEP 2.3 Check the stopping condition. Because of $g(\lambda) < K$, we set $\lambda_l = \lambda = 0.9375$.

$\vdots$

After 11 iterations we have $g(\lambda) = 3.000$ and the stopping condition is fulfilled. For the overall throughput we obtain $\lambda = 1.193$, which yields:

$$\lambda_1 = \underline{1.193}, \quad \lambda_2 = \underline{0.596}, \quad \lambda_3 = \underline{0.596}, \quad \lambda_4 = \underline{1.193}.$$

For the utilizations of the nodes we get:

$$\rho_{11} = \underline{0.294}, \quad \rho_{12} = \underline{0.317}, \quad \rho_1 = \underline{0.298}, \quad \rho_2 = \underline{0.358}, \quad \rho_3 = \underline{0.477},$$

and for the mean number of jobs in a node:

$$\overline{K}_1 = \underline{0.638}, \quad \overline{K}_2 = \underline{0.470}, \quad \overline{K}_3 = \underline{0.700}, \quad \overline{K}_4 = \underline{1.193}.$$

Now we analyze the network using the AMVA algorithm.

STEP 1 Initialization:

$$\overline{K}_1(0) = \overline{K}_2(0) = \overline{K}_3(0) = 0, \quad p_1(0 \mid 0) = 1, \quad p_1(1 \mid 0) = 0.$$

STEP 2 Iteration over the number of jobs in the network starting with $k = 1$.

STEP 2.1 Mean response times:

$$\overline{T}_1(1) = \frac{1}{\mu_{11} + \mu_{12}} \left(1 + \overline{K}_1(0) + p_1(0 \mid 0)\right) = \underline{0.5},$$

$$\overline{T}_2(1) = \frac{1}{\mu_2} \left(1 + \overline{K}_2(0)\right) = \underline{0.6},$$

$$\overline{T}_3(1) = \frac{1}{\mu_3} \left(1 + \overline{K}_3(0)\right) = \underline{0.8},$$

$$\overline{T}_4(1) = \frac{1}{\mu_4} = \underline{1}.$$

STEP 2.2 Throughput:

$$\lambda(1) = \frac{1}{\sum_{i=1}^{4} e_i \overline{T}_i(1)} = \underline{0.455}.$$

STEP 2.3 Mean number of jobs:

$$\overline{K}_1(1) = \lambda(1)\overline{T}_1(1)e_1 = \underline{0.227}, \quad \overline{K}_2(1) = \lambda(1)\overline{T}_2(1)e_2 = \underline{0.136},$$
$$\overline{K}_3(1) = \lambda(1)\overline{T}_3(1)e_3 = \underline{0.182}, \quad \overline{K}_4(1) = \lambda(1)\overline{T}_4(1)e_4 = \underline{0.455}.$$

Iteration for $k = 2$:

STEP 2.1 Mean response times:

$$\overline{T}_1(2) = \frac{1}{\mu_{11} + \mu_{12}} \left(1 + \overline{K}_1(1) + p_1(0 \mid 1)\right) = \underline{0.511},$$

where:

$$p_1(0 \mid 1) = 1 - \frac{e_1\lambda(1)}{\mu_{11} + \mu_{12}} - \frac{1}{m_1} p_1(1 \mid 1) = \underline{0.818} \quad \text{and}$$
$$p_1(1 \mid 1) = \frac{e_1\lambda(1)}{\mu_{11}} p_1(0 \mid 0) = \underline{0.136},$$
$$\overline{T}_2(2) = \frac{1}{\mu_2} \left(1 + \overline{K}_2(1)\right) = \underline{0.682},$$
$$\overline{T}_3(2) = \frac{1}{\mu_3} \left(1 + \overline{K}_3(1)\right) = \underline{0.945},$$
$$\overline{T}_4(2) = \frac{1}{\mu_4} = \underline{1}.$$

STEP 2.2 Throughput:

$$\lambda(2) = \frac{2}{\sum_{i=1}^{4} e_i \overline{T}_i(2)} = \underline{0.860}.$$

STEP 2.3 Mean number of jobs:

$$\overline{K}_1(2) = \lambda(2)\overline{T}_1(2)e_1 = \underline{0.440}, \quad \overline{K}_2(2) = \lambda(2)\overline{T}_2(2)e_2 = \underline{0.293},$$
$$\overline{K}_3(2) = \lambda(2)\overline{T}_3(2)e_3 = \underline{0.407}, \quad \overline{K}_4(2) = \lambda(2)\overline{T}_4(2)e_4 = \underline{0.860}.$$

Iteration for $k = 3$:

STEP 2.1 Mean response times:

$$\overline{T}_1(3) = \frac{1}{\mu_{11} + \mu_{12}} \left(1 + \overline{K}_1(2) + p_1(0 \mid 2)\right) = \underline{0.530},$$

where:

$$p_1(0 \mid 2) = 1 - \frac{e_1\lambda(2)}{\mu_{11}+\mu_{12}} - \frac{1}{m_1}p_1(1 \mid 2) = \underline{0.679} \quad \text{and}$$

$$p_1(1 \mid 2) = \frac{e_1\lambda(2)}{\mu_{11}}p_1(0 \mid 1) = \underline{0.211},$$

$$\overline{T}_2(3) = \frac{1}{\mu_2}\left(1 + \overline{K}_2(2)\right) = \underline{0.776},$$

$$\overline{T}_3(3) = \frac{1}{\mu_3}\left(1 + \overline{K}_3(2)\right) = \underline{1.125},$$

$$\overline{T}_4(3) = \frac{1}{\mu_4} = \underline{1}.$$

STEP 2.2 Throughput:

$$\lambda(3) = \frac{3}{\sum_{i=1}^{4} e_i\overline{T}_i(3)} = \underline{1.209}.$$

STEP 2.3 Mean number of jobs:

$$\overline{K}_1(3) = \lambda(3)\overline{T}_1(3)e_1 = \underline{0.641}, \quad \overline{K}_2(3) = \lambda(3)\overline{T}_2(3)e_2 = \underline{0.469},$$
$$\overline{K}_3(3) = \lambda(3)\overline{T}_3(3)e_3 = \underline{0.681}, \quad \overline{K}_4(3) = \lambda(3)\overline{T}_4(3)e_4 = \underline{1.209}.$$

The algorithm stops after three iteration steps. For the throughput we get:

$$\lambda_1 = \underline{1.209}, \quad \lambda_2 = \underline{0.605}, \quad \lambda_3 = \underline{0.605}, \quad \lambda_4 = \underline{1.209}.$$

For the utilizations of the nodes we obtain:

$$\rho_1 = \underline{0.302}, \quad \rho_2 = \underline{0.363}, \quad \rho_3 = \underline{0.484},$$

and for the mean number of jobs in a node:

$$\overline{K}_1 = \underline{0.641}, \quad \overline{K}_2 = \underline{0.469}, \quad \overline{K}_3 = \underline{0.681}, \quad \overline{K}_4 = \underline{1.209}.$$

Finally, the analysis of the network is performed using the ASCAT algorithm, where we use the approximate Eqs. (6.135) and (6.136) and Eqs. (6.27) and (6.28) for asymmetric multiple server nodes.

STEP 1 Modified core algorithm for $K = 3$ jobs in the network with the input parameters:

$$\overline{K}_i(K) = \frac{K}{N} = \underline{0.75} \quad \text{and} \quad D_i(K) = 0 \quad \text{for } i = 1, \ldots, 4.$$

STEP C1 Estimate values for the $\overline{K}_i(K-1)$ from Eq. (9.5):

$$\overline{K}_i(2) = 2\frac{\overline{K}_i(3)}{3} = 0.5 \quad \text{for } i = 1, \ldots, 4.$$

For the next step we need an initial value for the throughput:

$$\lambda(2) = \mu_4 \cdot \overline{K}_4(2) = \underline{0.5}.$$

STEP C2 One step of MVA:

$$\overline{T}_1(3) = \frac{1}{\mu_{11} + \mu_{12}} \cdot \left(m_1 + \overline{K}_1(2) + P_{m_1}(\rho_1) - m_1\rho_1\right) = \underline{0.569},$$

with:

$$\rho_1 = \frac{e_1\lambda(2)}{\mu_{11} + \mu_{12}} = \frac{0.5}{4} = \underline{0.125} \quad \text{and} \quad P_{m_1}(\rho_1) = \underline{0.0278},$$

$$\overline{T}_2(3) = \frac{1}{\mu_2}\left(1 + \overline{K}_2(2)\right) = \underline{0.9},$$

$$\overline{T}_3(3) = \frac{1}{\mu_3}\left(1 + \overline{K}_3(2)\right) = \underline{1.2},$$

$$\overline{T}_4(3) = \frac{1}{\mu_4} = \underline{1}.$$

Throughput:

$$\lambda(3) = \frac{3}{\sum_{i=1}^{4} e_i\overline{T}_i(3)} = \underline{1.145}.$$

Mean number of jobs:

$$\overline{K}_1(3) = \lambda(3)\overline{T}_1(3)e_1 = \underline{0.652}, \quad \overline{K}_2(3) = \lambda(3)\overline{T}_2(3)e_2 = \underline{0.515},$$
$$\overline{K}_3(3) = \lambda(3)\overline{T}_3(3)e_3 = \underline{0.687}, \quad \overline{K}_4(3) = \lambda(3)\overline{T}_4(3)e_4 = \underline{1.145}.$$

STEP C3 Check the stopping condition:

$$\max_i \left\{ \frac{\left|\overline{K}_i^{(1)}(3) - \overline{K}_i^{(0)}(3)\right|}{3} \right\} = \underline{0.132} > \epsilon = 0.01.$$

STEP C1 Estimate values for the $k_i(K-1)$ from Eq. (9.5):

$$\overline{K}_1(2) = \underline{0.435}, \quad \overline{K}_2(2) = \underline{0.344}, \quad \overline{K}_3(2) = \underline{0.458}, \quad \overline{K}_4(2) = \underline{0.764}.$$

For the next step we need an initial value for the throughput:

$$\lambda(2) = \mu_4\overline{K}_4(2) = \underline{0.764}.$$

STEP C2 One step of MVA:

$$\vdots$$

After three iterations we get the following results for the mean number of jobs:

$$\overline{K}_1(3) = \underline{0.626}, \quad \overline{K}_2(3) = \underline{0.475}, \quad \overline{K}_3(3) = \underline{0.702}, \quad \overline{K}_4(3) = \underline{1.198}.$$

STEP 2 Modified core algorithm for $(K-1) = 2$ jobs in the network with the input parameters:

$$\overline{K}_i(2) = \frac{2}{N} = \underline{0.5} \quad \text{and} \quad D_i(2) = 0 \quad \text{for } i = 1, \ldots, 4.$$

STEP C1 Estimate values for the $\overline{K}_i(1)$ from Eq. (9.5):

$$\overline{K}_i(1) = 2\frac{\overline{K}_i(2)}{2} = \underline{0.25} \quad \text{for } i = 1, \ldots, 4.$$

For the next step we need an initial value for the throughput:

$$\lambda(1) = \mu_4 \cdot \overline{K}_4(1) = \underline{0.25}.$$

STEP C2 One step of MVA:

$$\overline{T}_1(2) = \frac{1}{\mu_{11} + \mu_{12}} \cdot \left(m_1 + \overline{K}_1(1) + P_{m_1}(\rho_1) - m_1\rho_1\right) = \underline{0.533},$$

with:

$$\rho_1 = \frac{e_1\lambda(1)}{\mu_{11} + \mu_{12}} = \frac{0.25}{4} = \underline{0.0625} \quad \text{and} \quad P_{m_1}(\rho_1) = \underline{0.00735},$$

$$\overline{T}_2(2) = \frac{1}{\mu_2}\left(1 + \overline{K}_2(1)\right) = \underline{0.75},$$

$$\overline{T}_3(2) = \frac{1}{\mu_3}\left(1 + \overline{K}_3(1)\right) = \underline{1.0},$$

$$\overline{T}_4(2) = \frac{1}{\mu_4} = \underline{1}.$$

Throughput:

$$\lambda(2) = \frac{2}{\sum_{i=1}^{4} e_i\overline{T}_i(2)} = \underline{0.831}.$$

Mean number of jobs:

$$\overline{K}_1(2) = \lambda(2)\overline{T}_1(2)e_1 = \underline{0.443}, \quad \overline{K}_2(2) = \lambda(2)\overline{T}_2(2)e_2 = \underline{0.311},$$
$$\overline{K}_3(2) = \lambda(2)\overline{T}_3(2)e_3 = \underline{0.415}, \quad \overline{K}_4(2) = \lambda(2)\overline{T}_4(2)e_4 = \underline{0.831}.$$

STEP C3 Check the stopping condition:

$$\max_i \left\{ \frac{\left|\overline{K}_i^{(1)}(2) - \overline{K}_i^{(0)}(2)\right|}{2} \right\} = \underline{0.165} > \epsilon = 0.01.$$

STEP C1 Estimate values for the $\overline{K}_i(1)$ from Eq. (9.5):

$$\overline{K}_1(1) = \underline{0.221}, \quad \overline{K}_2(1) = \underline{0.156}, \quad \overline{K}_3(1) = \underline{0.208}, \quad \overline{K}_4(1) = \underline{0.415}.$$

For the next step we need an initial value for the throughput:

$$\lambda(1) = \mu_4 \overline{K}_4(1) = \underline{0.415}.$$

STEP C2 One step of MVA:

$$\vdots$$

After three iterations we get the following results for the mean number of jobs:

$$\overline{K}_1(2) = \underline{0.434}, \quad \overline{K}_2(2) = \underline{0.295}, \quad \overline{K}_3(2) = \underline{0.414}, \quad \overline{K}_4(2) = \underline{0.857}.$$

STEP 3 Estimate the F_i and D_i values with Eqs. (9.3) and (9.4), respectively:

$$F_1(3) = \frac{\overline{K}_1(3)}{3} = \underline{0.209}, \quad F_1(2) = \frac{\overline{K}_1(2)}{2} = \underline{0.217},$$
$$F_2(3) = \frac{\overline{K}_2(3)}{3} = \underline{0.158}, \quad F_2(2) = \frac{\overline{K}_2(2)}{2} = \underline{0.148},$$
$$F_3(3) = \frac{\overline{K}_3(3)}{3} = \underline{0.234}, \quad F_3(2) = \frac{\overline{K}_3(2)}{2} = \underline{0.207},$$
$$F_4(3) = \frac{\overline{K}_4(3)}{3} = \underline{0.399}, \quad F_4(2) = \frac{\overline{K}_4(2)}{2} = \underline{0.429}.$$

Therefore we get:

$$D_1(3) = F_1(2) - F_1(3) = \underline{0.00835}, \quad D_2(3) = F_2(2) - F_2(3) = \underline{-0.0106},$$
$$D_3(3) = F_3(2) - F_3(3) = \underline{-0.0270}, \quad D_4(3) = F_4(2) - F_4(3) = \underline{0.0292}.$$

STEP 4 Modified core algorithm for $K = 3$ jobs in the network with the computed $\overline{K}_i(K)$ values from Step 1 and the $D_i(K)$ values from Step 3 as inputs.

STEP C1 Estimate values for the $\overline{K}_i(K-1)$ from Eq. (9.5):

$$\overline{K}_1(2) = 2\left(\frac{\overline{K}_1(3)}{3} + D_1(3)\right) = \underline{0.434},$$
$$\overline{K}_2(2) = \underline{0.295}, \quad \overline{K}_3(2) = \underline{0.414}, \quad \overline{K}_4(2) = \underline{0.857}.$$

For the next step we need an initial value for the throughput:

$$\lambda(2) = \mu_4 \cdot \overline{K}_4(2) = \underline{0.857}.$$

STEP C2 One step of MVA:

$$\overline{T}_1(3) = \frac{1}{\mu_{11}+\mu_{12}} \cdot \left(m_1 + \overline{K}_1(2) + P_{m_1}(\rho_1) - m_1\rho_1\right) = \underline{0.520},$$

with:

$$\rho_1 = \underline{0.24} \quad \text{and} \quad P_{m_1}(\rho_1) = \underline{0.0756},$$
$$\overline{T}_2(3) = \frac{1}{\mu_2}\left(1 + \overline{K}_2(2)\right) = \underline{0.777},$$
$$\overline{T}_3(3) = \frac{1}{\mu_3}\left(1 + \overline{K}_3(2)\right) = \underline{1.131},$$
$$\overline{T}_4(3) = \frac{1}{\mu_4} = \underline{1}.$$

Throughput:

$$\lambda(3) = \frac{3}{\sum_{i=1}^{4} e_i \overline{T}_i(3)} = \underline{1.212}.$$

Mean number of jobs:

$$\overline{K}_1(3) = \lambda(3)\overline{T}_1(3)e_1 = \underline{0.631}, \quad \overline{K}_2(3) = \lambda(3)\overline{T}_2(3)e_2 = \underline{0.471},$$
$$\overline{K}_3(3) = \lambda(3)\overline{T}_3(3)e_3 = \underline{0.686}, \quad \overline{K}_4(3) = \lambda(3)\overline{T}_4(3)e_4 = \underline{1.212}.$$

STEP C3 Check the stopping condition:

$$\max_i \left\{ \frac{\left|\overline{K}_i^{(1)}(3) - \overline{K}_i^{(0)}(3)\right|}{3} \right\} = \underline{0.00531} < \epsilon = 0.01.$$

The stopping condition is fulfilled. Therefore the algorithm stops and we get the following results:

Throughputs:

$$\lambda_1 = \underline{1.212}, \quad \lambda_2 = \underline{0.606}, \quad \lambda_3 = \underline{0.606}, \quad \lambda_4 = \underline{1.212}.$$

Mean number of jobs:

$$\overline{K}_1 = \underline{0.631}, \quad \overline{K}_2 = \underline{0.471}, \quad \overline{K}_3 = \underline{0.686}, \quad \overline{K}_4 = \underline{1.212}.$$

Utilizations:

$$\rho_1 = \underline{0.303}, \quad \rho_2 = \underline{0.364}, \quad \rho_3 = \underline{0.485}.$$

Table 10.33 Throughput λ_i and mean number of jobs $\overline{K}_i$ for different methods compared with the DES results

Node		1	2	3	4
λ_i	ASUM	1.19	0.60	0.60	1.19
	AMVA	1.21	0.61	0.61	1.21
	ASCAT	1.21	0.61	0.61	1.21
	DES	1.21	0.61	0.61	1.21
$\overline{K}_i$	ASUM	0.64	0.47	0.70	1.19
	AMVA	0.64	0.47	0.68	1.21
	ASCAT	0.63	0.47	0.69	1.21
	DES	0.64	0.47	0.68	1.21

In Table 10.33 we compare the results for the throughputs λ_i and the mean number of jobs $\overline{K}_i$. For this example, the results of all methods are close together and close to the DES results. As in the case of asymmetric single station queueing networks, the preceding approximations are accurate if the values of the service rates in a station do not differ too much. As a rule of thumb, we consider the approximations to be satisfactory if $\mu_{\min}/\mu_{\max} < 10$ at each asymmetric node.

Problem 10.6 Formulate the closed network of Fig. 10.57 as a GSPN and solve it exactly using SHARPE or SPNP. Compare the exact results with the approximative ones obtained in this section using ASUM, AMVA and ASCAT.

10.8 NETWORKS WITH BLOCKING

Up to now we have generally assumed that all nodes of a queueing network have queues with infinite buffers. Thus a job that leaves a node will always find an empty place in the queue of the next node. But in the case of finite capacity queues, blocking can occur and jobs can be rejected at a node if the queue is full. Queueing networks that consist of nodes with finite buffer capacity are called *blocking networks*. Blocking networks are not only important for modeling and performance evaluation of computer systems and computer networks, but also in the field of production line systems. Exact results for general blocking networks can only be obtained by using the generation and

numerical solution of the underlying CTMC (either directly or via SPNs) that is possible for medium-sized networks. For small networks, closed-form results may be derived. The most common techniques presented in the literature for the analysis of large blocking networks are, therefore, approximate techniques.

10.8.1 Different Blocking Types

In [Perr94] three different types of blocking are introduced:

1. *Blocking after service* (see Fig. 10.58): With this type of blocking, node i is blocked when a job completing service at node i wishes to join node j that is already full. This job then has to stay in the server of node i and block until a job leaves node j. This type of blocking is used to model production line systems or external I/O devices [Akyi87, Akyi88b, Akyi88a, Alti82, AlPe86, OnPe89, PeAl86, Perr81, PeSn89, SuDi86]. In the literature, this type of blocking has been referred to by a variety of different terms such as Type-1 blocking, transfer blocking, production blocking, and non-immediate blocking.

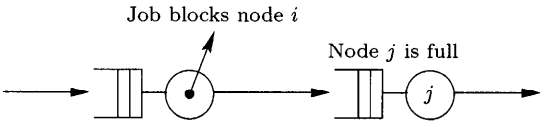


Fig. 10.58 Blocking after service.

2. *Blocking before service* (see Fig. 10.59): The job to be served next at node i determines its source node j before it enters service at node i. If node j is already full, the job cannot enter the server of node i and node i is therefore blocked. Only if another job leaves node j will node i be deblocked and the first job in the queue served [BoKo81, GoNe67b, KoRe76, KoRe78]. In the literature, this type of blocking has also been referred to as Type-2 blocking, service blocking, communication blocking, and immediate blocking.

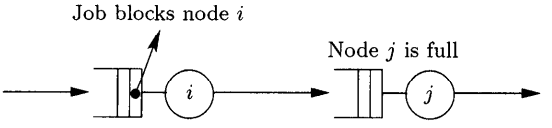


Fig. 10.59 Blocking before service.

3. *Repetitive blocking* (see Fig. 10.60): If a job that has been served at node i wishes to go to node j whose queue is full, it is rejected at node j and goes to the rear of node i queue. This procedure is repeated until a job leaves node j, thereby creating one empty slot in the queue of

node j. This type of blocking is used to model communications networks or flexible production line systems [Akvo89, BaIa83, Hova81, Pitt79]. Other terms for this type of blocking are Type-3 blocking or rejection blocking.

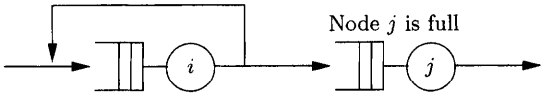


Fig. 10.60 Repetitive blocking.

In [OnPe86], different kinds of blocking are compared with each other. In [YaBu86], a queueing network model is presented that has finite buffer capacity and yet is not a blocking network. Here a job that would block node i because node j is full is buffered in a central buffer until there is an empty spot at node j. Such models are especially interesting for production line systems.

10.8.2 Product-Form Solution for Networks with Two Nodes

Many different techniques for the solution of blocking networks are available. We only consider a simple exact solution technique in detail that can be applied to closed blocking networks with only two nodes and blocking after service [Akyi87]. This method can also be extended to queueing networks with several nodes, but then it is an approximate technique that is very costly and it can only be used for throughput analysis [Akyi88b].

We consider queueing networks with one job class, exponentially distributed service times, and FCFS service strategy. Transitions back to the same node are not possible ($p_{ii} = 0$). Each node has a fixed capacity M_i that equals the capacity of the queue plus the number m_i of servers. Nodes with an infinity capacity can be taken into consideration by setting $M_i > K$ so that the capacity of the node is bigger than the number of jobs in the network. The overall capacity of the network must, of course, be bigger than the number of jobs in the network for it to be a blocking network:

$$K < M_1 + M_2. \tag{10.177}$$

In the case of equality in the preceding expression, deadlocks can occur [Akyi87]. The possible number of states in the underlying CTMC of a closed network with two nodes and unlimited storage capacity is:

$$Z = K + 1. \tag{10.178}$$

At first we consider the simple case where the number of servers at each node is $m_1 = m_2 = 1$. The state diagram of the CTMC for such a two-node network *without blocking* is shown in Fig. 10.61.

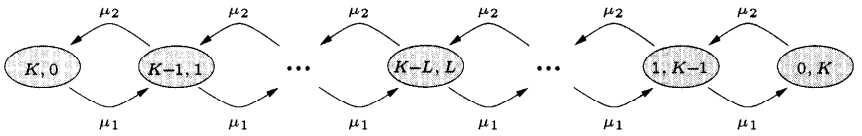


Fig. 10.61 CTMC for a two-node network without blocking ($m_1 = m_2 = 1$).

As can be seen, the minimum capacity at each node is $M_i = K$ (the queue must contain at least $K - 1$ spots) so that all states are possible. If the nodes have a finite capacity $M_i < K$, then not all $K + 1$ states of Fig. 10.61 are possible. Possible states of the blocking network are then given by the condition that the number of jobs at a node cannot be bigger than the capacity of that node. In the case of a transition to a state where the capacity of a node is exceeded, blocking occurs. The corresponding state is called a blocking state. Recall that due to Type-1 blocking, the job does not switch to the other node but stays in the original node. The state diagram of the blocking network is shown in Fig. 10.62.

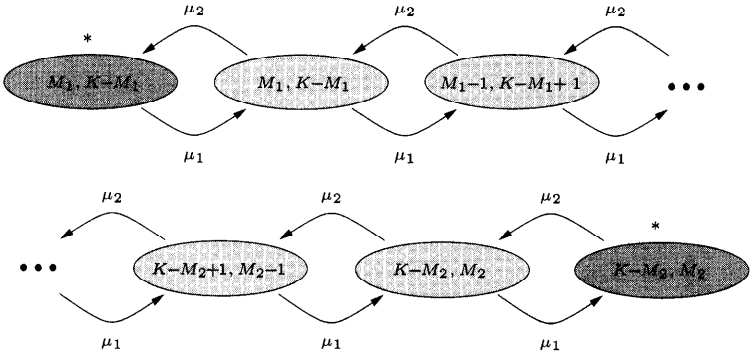


Fig. 10.62 CTMC for the two-node blocking network (states marked with $*$ are blocking states) $m_1 = m_2 = 1$ (blocking after service).

The state diagram of the blocking network shown in Fig. 10.62 differs from the state diagram shown in Fig. 10.61 in two aspects. First, all states where the capacity is exceeded are no longer necessary and, second, Fig. 10.62 contains the blocking states marked with an asterisk. The number of states $\hat{Z}$ in the blocking network is the sum of all possible non-blocking states plus all the blocking states:

$$\hat{Z} = \min\{K, M_1 + 1\} + \min\{K, M_2 + 1\} - K + 1. \tag{10.179}$$

It can be shown [Akyi87] that for this closed network with two nodes and blocking after service, an equivalent closed network with two nodes without blocking exists. Apart from the number of jobs $\hat{K}$ in the equivalent network, all other parameter values remain the same. The population $\hat{K}$ in the

equivalent network can be determined using Eqs. (10.178) and (10.179):

$$\hat{K} = \min\{K, M_1 + 1\} + \min\{K, M_2 + 1\} - K. \tag{10.180}$$

The CTMC of the equivalent network is shown in Fig. 10.63. The equivalent network is a product-form network with $\hat{k}_i$ jobs at node i, $0 \le \hat{k}_i \le \hat{K}$.

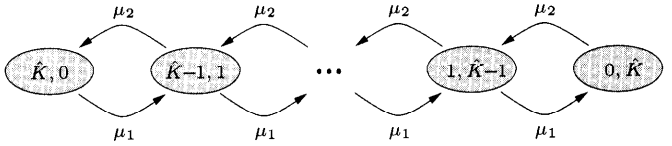


Fig. 10.63 CTMC of the equivalent network corresponding to the CTMC in Fig. 10.61.

The steady-state probabilities of the equivalent network can be determined using the following equation:

$$\pi(\hat{k}_1, \hat{k}_2) = \frac{1}{G(\hat{K})} \cdot \prod_{i=1}^{2} \left(\frac{1}{\mu_i}\right)^{\hat{k}_i}. \tag{10.181}$$

By equating the steady-state probabilities of the blocking network with the corresponding steady-state probabilities of the equivalent non-blocking network, the performance measures of the blocking network can be determined. To demonstrate how this technique works, the following simple example is used:

Example 10.19 Consider a closed queueing network with two nodes, $K = 10$ jobs, and the routing probabilities:

$$p_{12} = 1, \quad p_{21} = 1.$$

All other system parameters are given in Table 10.34.

Table 10.34 System parameters for Example 10.19

i	e_i	$1/\mu_i$	m_i	M_i
1	1	2	1	7
2	1	0.9	1	5

All service times are exponentially distributed and the service discipline at each node is FCFS. Using Eq. (10.178), the number of states in the network without blocking is determined to be $Z = K + 1 = 11$. The CTMC of the network without blocking is shown in Fig. 10.64, while the state diagram of the blocking network (see Fig. 10.65) contains all possible non-blocking states

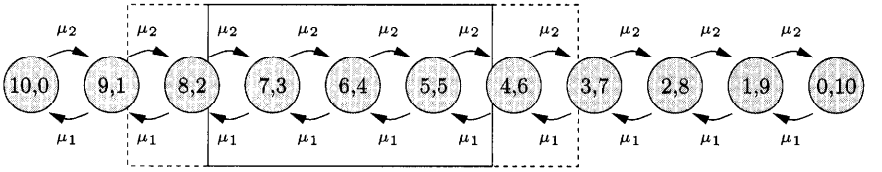


Fig. 10.64 CTMC of the Example 10.19 network without blocking.

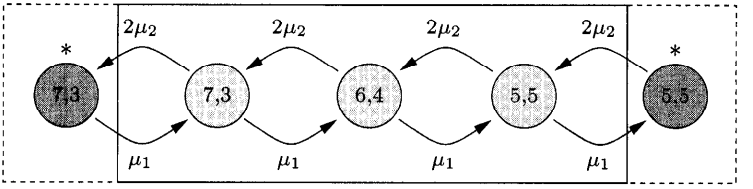


Fig. 10.65 CTMC of the Example 10.19 network with blocking.

from the original network as well as the blocking states (marked with asterisk). The number of states $\hat{Z}$ of this blocking network is given by Eq. (10.179) as the sum of all possible non-blocking states ($= 3$) and the blocking states ($= 2$): $\hat{Z} = 5$. The CTMC of the equivalent network (see Fig. 10.66) therefore contains $\hat{Z} = 5$ states and the number of jobs in the network is given by Eq. (10.180) as $\hat{K} = 4$.

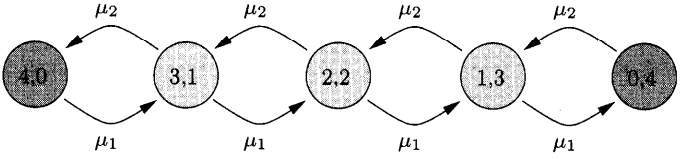


Fig. 10.66 CTMC of the equivalent network for Example 10.19.

The throughput $\lambda_{\mathrm{NB}}(4)$ of the equivalent network can be determined directly using the MVA and is equal to the throughput $\lambda_B(10)$ of the blocking network:

$$\lambda_{\mathrm{B}}(10) = \lambda_{\mathrm{NB}}(4) = \underline{0.500}.$$

The steady-state probabilities of the blocking network are obtained by equating with the corresponding steady-state probabilities of the equivalent non-blocking network:

$$\begin{aligned} \pi(6,4) &= \pi(2,2) = \underline{0.120}, \\ \pi(7,3)^* &= \pi(4,0) = \underline{0.550}, \quad \pi(5,5) = \pi(1,3) = \underline{0.054}, \\ \pi(7,3) &= \pi(3,1) = \underline{0.248}, \quad \pi(5,5)^* = \pi(0,4) = \underline{0.026}. \end{aligned}$$

The mean number of jobs at both nodes can be determined using the steady-state probabilities:

$$\overline{K}_1 = 7\,[\pi(7,3)^* + \pi(7,3)] + 6\pi(6,4) + 5\,[\pi(5,5) + \pi(5,5)^*] = \underline{6.73},$$
$$\overline{K}_2 = 3\,[\pi(7,3)^* + \pi(7,3)] + 4\pi(6,4) + 5\,[\pi(5,5) + \pi(5,5)^*] = \underline{3.27}.$$

With these results all other performance measures can be derived using the well-known formulae. Of special interest for blocking networks are the blocking probabilities:

$$P_{B_1} = \pi(5,5)^* = \underline{0.026},$$
$$P_{B_2} = \pi(7,3)^* = \underline{0.0550}.$$

Next consider the case where each node has multiple servers ($m_i > 1$). The state diagram of the CTMC underlying such a two-node network without blocking is shown in Fig. 10.67, while the state diagram of the blocking

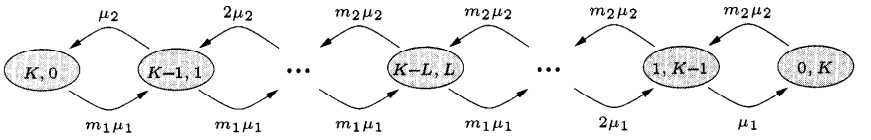


Fig. 10.67 CTMC of a two-node network without blocking ($m_i > 1$).

network (with Type-1 blocking) is shown in Fig. 10.68. Here a state marked

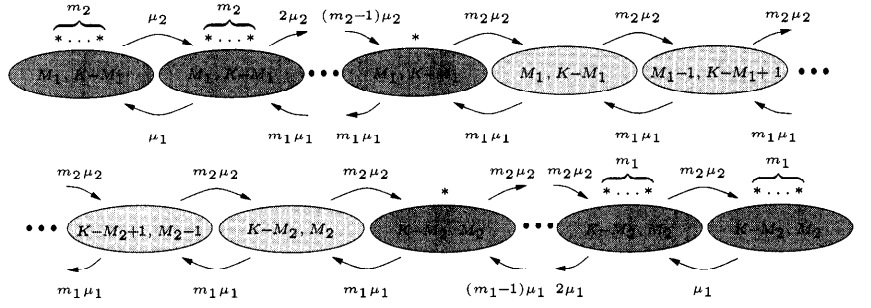


Fig. 10.68 CTMC for the two-node network with blocking. Blocking states are marked with $*$ ($m_i > 1$).

with asterisks shows the number of blocked servers in that state. Therefore we get for the number $\hat{Z}$ of states in the blocking network, the sum of all possible non-blocking states plus the blocking states ($m_i > 1$):

$$\hat{Z} = \min\{K, M_1 + m_2\} + \min\{K, M_2 + m_1\} - K + 1. \tag{10.182}$$

$\hat{K}$ is given by Eqs. (10.178) and (10.182):

$$\hat{K} = \min\{K, M_1 + m_2\} + \min\{K, M_2 + m_1\} - K. \tag{10.183}$$

The state diagram of the equivalent network is shown in Fig. 10.69. The equivalent network is again a product-form network with $\hat{k}_i$ jobs at node i, $0 \leq \hat{k}_i \leq \hat{K}$.

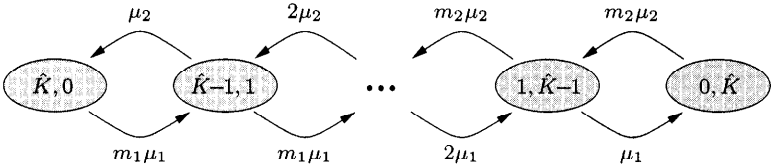


Fig. 10.69 CTMC of the equivalent product-form network ($m_i > 1$).

The steady-state probabilities of the equivalent network can be determined using Eq. (7.59):

$$\pi(\hat{k}_1, \hat{k}_2) = \frac{1}{G(\hat{K})} \cdot \prod_{i=1}^{2} \left(\frac{1}{\mu_i}\right)^{\hat{k}_i} \cdot \frac{1}{\beta_i(\hat{k}_i)}, \tag{10.184}$$

with $\beta_i(\hat{k}_i)$, Eq. (7.62).

The following example is used to demonstrate this technique:

Example 10.20 Consider again the network given in Example 10.19, but change the number of servers, $m_2 = 2$. All other network parameters remain the same. From Eq. (10.178), the number of states in the network without blocking is $Z = K + 1 = 11$. The corresponding state diagram is shown in Fig. 10.70. The state diagram of the blocking network (see Fig. 10.71)

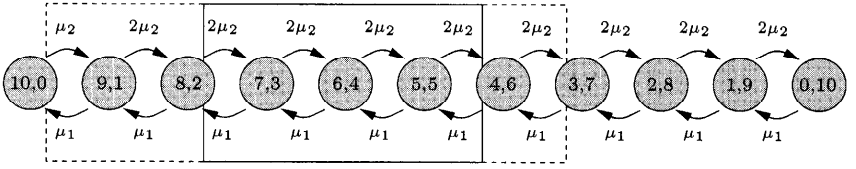


Fig. 10.70 CTMC of the Example 10.20 network without blocking.

contains all possible non-blocking states from the original network plus all the blocking states (marked with $*$). The number of states $\hat{Z}$ of the blocking network is given by Eq. (10.182) as the sum of all non-blocking states ($= 3$) and the blocking states ($= 3$): $\hat{Z} = 6$.

The state diagram of the equivalent network (see Fig. 10.72) contains, therefore, also $\hat{Z} = 6$ states, and for the number of jobs in the network, Eq. (10.180) is used: $\hat{K} = 5$. The throughput $\lambda_{\mathrm{NB}}(5)$ of the equivalent network can be determined using the MVA and, because of the equivalence of both networks, is identical with the throughput $\lambda_B(10)$ of the blocking network:

$$\lambda_{\mathrm{B}}(10) = \lambda_{\mathrm{NB}}(5) = \underline{0.499}.$$

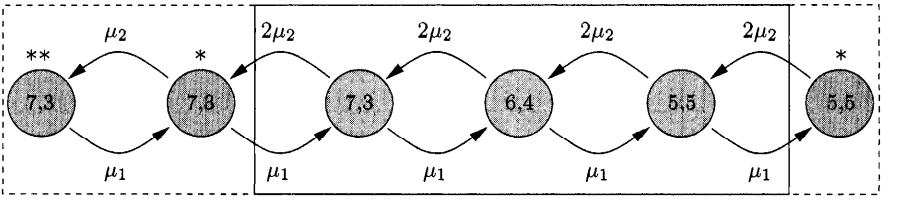


Fig. 10.71 CTMC of the Example 10.20 blocking network.

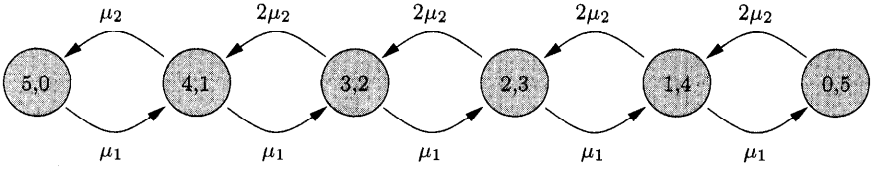


Fig. 10.72 CTMC of the Example 10.20 equivalent product-form network.

The state probabilities of the blocking network are given by equating the corresponding steady-state probabilities of the equivalent non-blocking network. The steady-state probabilities are determined using Eq. (10.181):

$$\begin{aligned} \pi(7,3)^{**} &= \pi(5,0) = \underline{0.633}, & \pi(6,4) &= \pi(2,3) = \underline{0.014}, \\ \pi(7,3)^{*} &= \pi(4,1) = \underline{0.284}, & \pi(5,5) &= \pi(1,4) = \underline{0.003}, \\ \pi(7,3) &= \pi(3,2) = \underline{0.064}, & \pi(5,5)^{*} &= \pi(0,5) = \underline{0.001}. \end{aligned}$$

The mean number of jobs at each node can be determined using the steady-state probabilities:

$$\begin{aligned} \overline{K}_1 &= 7\left[\pi(7,3)^{**} + \pi(7,3)^{*} + \pi(7,3)\right] + 6\pi(6,4) + 5\left[\pi(5,5) + \pi(5,5)^{*}\right] \\ &= \underline{6.978}, \\ \overline{K}_2 &= 3\left[\pi(7,3)^{**} + \pi(7,3)^{*} + \pi(7,3)\right] + 4\pi(6,4) + 5\left[\pi(5,5) + \pi(5,5)^{*}\right] \\ &= \underline{3.022}. \end{aligned}$$

All other performance measures can be determined using the well-known equations. Of special interest are blocking probabilities:

$$\begin{aligned} P_{B_1} &= \pi(5,5)^{*} = \underline{0.001}, \\ P_{B_2} &= \pi(7,3)^{**} + \frac{1}{2}\pi(7,3)^{*} = \underline{0.775}. \end{aligned}$$

For computing P_{B_2}, the steady-state probability $\pi(7,3)^{*}$ can only be counted half because in state $(7,3)^{*}$ only one of the two servers of node 2 is blocked.

For blocking networks with more than two nodes, no equivalent non-blocking network can be found. For these cases only approximate results exist, which

are often inaccurate or can only be used for very special types of networks such as tandem networks [SuDi84, SuDi86] or open networks [PeSn89].

Problem 10.7 Consider a closed blocking network with $N = 2$ nodes and $K = 9$ jobs. The service times are exponentially distributed and the service strategy at all nodes is FCFS. All other network parameters are given in the Table 10.35.

Table 10.35 Network input parameters for Problem 10.7

i	e_i	$1/\mu_i$	m_i	M_i
1	1	1	1	6
2	1	1	1	4

(a) Draw the state diagram for the network without blocking as well as the state diagram of the blocking network and the equivalent non-blocking network.

(b) Determine the throughput λ_B, the mean number of jobs $\overline{K}_i$, and the blocking probabilities $P_{B_i}(i = 1, 2)$ for the blocking network.

Problem 10.8 Directly solve the CTMC of the blocking network in Problem 10.7 by using SHARPE. Note that you can easily obtain transient probabilities and cumulative transients with SHARPE in addition to the steady-state probabilities.

Problem 10.9 For Problem 10.8, construct an SRN and solve it using SPNP. Compare the results obtained from SPNP with those obtained by hand computation. Note that SRNs enable computation of transients and cumulative transients. Furthermore, easy definition of rewards at the net levels enables straightforward computation of all desired performance measures.

11

Optimization

Analytic performance models are very well suited as kernels in optimization problems. Two major categories of optimization problems are static and dynamic optimization. In the former, performance measures are computed separately from an analytic queueing or CTMC model and treated simply as functions (generally complex and non-linear) of the control (decision) variables. In the latter class of problems, decision variables are integrated with the analytic performance model and hence optimization is intimately connected with performance evaluation. We limit our discussion to static optimization. For a treatment of dynamic optimization in the context of computer and communication systems see [PGTP96], [DrLa77], [MeDü97], [MTD94], [MeFi97], [MeSe97], and [Meer92].

Designs of early computer and communication systems made little use of mathematical optimization techniques. Because the systems were not so complex, the number of possible design alternatives was limited and a good design decision was often obvious. This state of affairs has changed dramatically since modern systems are very complex and requirements for high performance and dependability have evolved. Therefore an optimal design is not obvious any more and the number of possible design decisions can be enormous.

A system designer is called on to produce a design of an implementable system from a given design specification. The design specification may include cost, performance, and dependability specifications, in addition to functional specifications. The performance specification states how well the specified functions (as stated in the functional specification) should perform in terms of throughput, response time, etc. The dependability specification states how the system tolerates component/sub-system faults. The dependability spec-

ification also includes minimum acceptable reliability, availability, etc. Cost specifications may indicate maximization or minimization of total cost or provide a constraint in the optimization.

The system design methodology takes the system specification as an input and transforms it into a system configuration (characterized by a set of parameters) that fulfills the specification. These (hardware) parameters include the number of processors, the number of memories, the speed and capacity of resources, and other architectural parameters such as the system topology. Software configuration parameters describe the distribution of processes over system resources as well as resource scheduling policies. In addition to the hardware and software specification, maintenance policies might be described (e.g., preventive vs. corrective). It is thus a very difficult task to choose the configuration that best meets the system specification. Traditionally, dependability, performance, and economic analyses of computer systems have been studied separately and independently. Optimization methods provide a feasible avenue for combining performance, dependability, and cost for deriving optimal system parameters that meet the stated design specifications. Usually these questions are multiobjective optimization problems. Although it is possible to deal directly with multiobjective problems [IPM82], it is common to pick a single relevant system specification to be used as the objective function and to let the remaining design factors provide constraints. An alternative is to combine multiple criteria into a single utility function.

11.1 OPTIMIZATION PROBLEMS AND COST FUNCTIONS

To formulate the optimization problem it is important to determine the system parameters that can be varied and hence considered to be decision variables. Possible types of optimization are:

- Minimization of cost for given throughput.
- Maximization of throughput with cost constraint.
- Minimization of the mean response time with cost constraint.

The simplest cost functions are linear functions: If device (resource) service rates μ_i are decision variables, the linear cost function is given by:

$$C(\boldsymbol{\mu}) = \sum_{i=1}^{N} c_i \mu_i = \text{COST}, \tag{11.1}$$

with the vector of the service rates $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_N)$. COST is the total cost of the considered system, and c_i is the cost factor for service rate μ_i of resource i.

More realistic estimates are obtained by using non-linear cost functions:

$$C(\boldsymbol{\mu}) = \sum_{i=1}^{N} c_i \mu_i^{\alpha_i} = \text{COST}, \quad \alpha_i > 1. \tag{11.2}$$

The service rate μ_i of a server depends on the speed v_i of device i and the mean number of work units d_i per visit:

$$\mu_i = \frac{v_i}{d_i}.$$

For the CPU, the mean number of instructions per CPU burst is given by d_i and the CPU speed (number of instructions per time unit) by v_i. In the case of a storage device, d_i is the mean number of words in an I/O operation and v_i is the speed of the storage device in words per time unit [TrKi80]. Thus if the mean number of work units d_i of the jobs at station i is given, then the optimal value of the device speed v_i is immediately given using the optimal values of the service rates μ_i. We can extend the optimization problem if we also consider the cost for main memory. In the case of a multiprogramming computer system, this cost depends on the degree of multiprogramming K (the number of jobs in the system). The cost function is then given by:

$$C(\boldsymbol{\mu}, K) = C(K) + \sum_{i=1}^{N} c_i \mu_i^{\alpha_i} = \text{COST} \tag{11.3}$$

where the $C(K)$ is the cost of the main memory satisfying the storage requirement of K concurrent jobs. The next step is to choose the performance model. This can be a DES model or an analytic model. Among the latter we can choose a product-form queueing network model or a non-product-form queueing network model, a CTMC model, or a hierarchical model. The final step is to choose the optimization method.

There exist numerous optimization methods that can be used in combination with analytical methods for the performance evaluation introduced in this book. We show the use of a solution method based on Lagrange multipliers. We introduce an approximation method based on SUM (see Section 9.2) and a more complex but exact method, which is based on the convolution method (see Section 8.1). Finally, we show the use of geometric programming in the design of a memory hierarchy. For further study of performance optimization including the issue of convexity, see [TrKi80], and [SRT88].

11.2 OPTIMIZATION BASED ON THE SUMMATION METHOD

This method was introduced by Bolch, Fleischmann, and Schreppel [BFS87] (therefore called BFS method for short) and is now demonstrated using the following example:

11.2.1 Maximization of the Throughput

We assume a closed queueing network with N stations and K jobs of a single class. The visit ratios e_i and the functions f_i (see Eq. (9.15)) that describe the behavior of the queues are also given. The correction factor $(K-1)/K$ is neglected to obtain formulae that are as simple as possible. This approach is valid if the number of jobs is large, because then $(K-1)/K$ is close to 1. However, usable formulae are also obtained if the correction factor is included. These provide more accurate values for the optimum, especially for small values of K [AkBo88b]. For $\lambda_i = \lambda \cdot e_i$, $m_i = 1$ and $\rho_i = \frac{\lambda_i}{\mu_i}$, we obtain:

$$\overline{K}_i(\lambda, \mu_i) = f_i(\lambda, \mu_i) = \begin{cases} \dfrac{\lambda e_i}{\mu_i - \lambda e_i}, & \text{Type-1, 2, 4, and } m_i = 1, \\ \dfrac{\lambda e_i}{\mu_i}, & \text{Type-3 IS.} \end{cases}$$

We seek service rates μ_i that maximize the overall throughput λ and satisfy a linear cost constraint:

$$C(\boldsymbol{\mu}) = \sum_{i=1}^{N} c_i \mu_i = \text{COST}\,.$$

Here the cost $C(K)$ for the main memory is neglected but can easily be included by repeating the optimization for several values of K and choosing the one with the maximum throughput. The service rates must satisfy the system equation as well as the cost constraints. The system equation is as follows, using Eqs. (9.16) and (9.22):

$$\sum_{i=1}^{N} f_i(\lambda, \mu_i) = \sum_{\neq \text{IS}} \frac{\lambda e_i}{\mu_i - \lambda e_i} + \sum_{\text{IS}} \frac{\lambda e_i}{\mu_i} = K.$$

The optimization problem can be solved using Lagrange multipliers [BrSe91]. The Lagrange function $L(\lambda, \mu_1, \ldots, \mu_N, y_1, y_2)$ with objective function λ and two Lagrange multipliers y_1 and y_2 is given by:

$$L(\lambda, \mu_1, \ldots, \mu_N, y_1, y_2) = \lambda + y_1 \left(\sum_{i=1}^{N} c_i \mu_i - \text{COST} \right) + y_2 \left(\sum_{\neq \text{IS}} \frac{\lambda e_i}{\mu_i - \lambda e_i} + \sum_{\text{IS}} \frac{\lambda e_i}{\mu_i} - K \right).$$

A necessary condition for optimal service rates μ_i and maximum throughput λ is obtained by differentiating with respect to λ, μ_i, y_1, y_2:

$$\frac{\partial L}{\partial \lambda} = 0, \qquad \frac{\partial L}{\partial y_1} = 0, \tag{11.4}$$

$$\frac{\partial L}{\partial \mu_i} = 0, \quad i = 1, \ldots, N, \qquad \frac{\partial L}{\partial y_2} = 0. \tag{11.5}$$

The following optimal values are obtained by solving the preceding system of equations:[1]

$$\lambda^* = \frac{\text{COST} \cdot K}{\left(\sum_{i=1}^{N} \sqrt{c_i e_i}\right)^2 + K \sum_{\neq \text{IS}} c_i e_i}, \tag{11.6}$$

$$\mu_i^* = \begin{cases} \lambda^* \cdot e_i \left(\dfrac{\sum_{j=1}^{N} \sqrt{e_j c_j}}{K \sqrt{e_i c_i}} + 1 \right), & \text{type } i \neq \text{IS}, \\ \lambda^* \cdot e_i \left(\dfrac{\sum_{j=1}^{N} \sqrt{e_j c_j}}{K \sqrt{e_i c_i}} \right), & \text{type } i = \text{IS}. \end{cases} \tag{11.7}$$

It can be seen from Eq. (11.6) that the maximum throughput λ^* is directly proportional to the total cost COST. The optimal service rates μ_i^* cannot be explicitly stated in more complicated optimization problems with non-linear cost functions, several types of stations, and further auxiliary constraints. They can be obtained iteratively from the system of equations.

Further results for some of the optimization problems already mentioned are given in the following.

11.2.2 Minimization of Cost

Next we consider the minimization of cost subject to a minimum throughput requirement, λ. By the same method as in Section 11.2.1, we obtain a closed-form formula for the approximate optimal values of service rates:

$$\mu_i^* = \begin{cases} \lambda e_i \left(\dfrac{\sum_{j=1}^{N} \sqrt{e_j c_j}}{K \sqrt{e_i}} + 1 \right), & \text{type } i \neq \text{IS}, \\ \lambda e_i \left(\dfrac{\sum_{j=1}^{N} \sqrt{e_j c_j}}{K \sqrt{e_i}} \right), & \text{type } i = \text{IS}. \end{cases} \tag{11.8}$$

The minimal cost is given by:

$$C^*(\boldsymbol{\mu}) = \sum_{i=1}^{N} \mu_i^* c_i.$$

[1] Since the summation method is an approximation method for performance analysis, these values are approximate optimal values.

The following iteration method can be used for a *non-linear cost function*:

Initialize: $\mu_i = 1$, for $i = 1, \ldots, N$.

Iterate until the deviations are sufficiently small:

$$y = \lambda \cdot \left(\frac{1}{K} \sum_{i=1}^{N} \sqrt{\alpha_i c_i e_i \mu_i^{\alpha_i - 1}} \right)^2, \tag{11.9}$$

$$\mu_i = \begin{cases} \sqrt{\dfrac{\lambda y e_i}{\alpha_i c_i \mu_i^{\alpha_i - 1}}} + \lambda e_i, & \text{type } i \neq \text{IS}, \\ \left(\dfrac{\lambda y e_i}{\alpha_i c_i} \right)^{\frac{1}{\alpha_i - 1}} & \text{type } i = \text{IS}. \end{cases} \tag{11.10}$$

11.2.3 Minimization of the Response Time

The problem of minimizing the mean response time $\overline{T}$ is equivalent to the problem of maximizing the overall throughput. This result follows from Little's theorem, $\overline{T} = K/\lambda$, and the fact that the number of jobs K in a closed network is constant. Explicit formulae can be provided if the cost function is linear.

Example 11.1 In the following we wish to maximize the throughput with cost constraints. The service rates that produce maximum throughput are to be calculated. The example consists of a product-form queueing network with $N = 5$ stations, three of which are of type -/M/1 and two of type -/G/∞. The queueing network contains 20 jobs. The visit ratios are as follows:

$$e_1 = 1, \quad e_2 = 0.2, \quad e_3 = e_4 = 0.5, \quad e_5 = 0.3.$$

We assume a linear cost function with the following coefficients:

$$c_1 = 10, \quad c_2 = c_3 = 5, \quad c_4 = 2, \quad c_5 = 1.$$

The total cost is constrained to be COST = 100. Using the BFS method, the following values are obtained for approximate optimal service rates that achieve the throughput λ^* subject to the given cost constraints:

$$\mu_1^* = 6.189, \quad \mu_2^* = 1.689, \quad \mu_3^* = 3.812, \quad \mu_4^* = 1.096, \quad \mu_5^* = 1.236,$$

with the approximate optimal throughput:

$$\lambda^* = 6.189.$$

The exact value of the throughput $\lambda = 6.569$ for the approximate optimal service rates μ_i^* is obtained by means of MVA. In order to verify these approximate optimal values, we vary the service rates, subject to the constraint:

$$\sum_{i=1}^{N} c_i \mu_i = 100,$$

as before. The new throughput is then compared with the approximate optimal throughput. If the following values of the service rates are used:

$$\mu_1 = 7,\ \mu_2 = 1.7,\ \mu_3 = 3.8,\ \mu_4 = 1,\ \mu_5 = 0.5,$$

the value for the throughput obtained by MVA is $\lambda = 6.473$. This is less than $\lambda = 6.569$, the value of the throughput obtained using the approximate optimal optimal service rates μ_i^*. Similar results are obtained for other values of the service rates that satisfy the constraint given previously. If we increase the distance between the selected service rates and those that produce the approximate optimal throughput considerably:

$$\mu_1 = 8,\ \mu_2 = 2,\ \mu_3 = 1,\ \mu_4 = 2,\ \mu_5 = 1,$$

then the throughput decreases and approaches its minimal value $\lambda = 2$.

Table 11.1 Maximization of the throughput with cost constraints (COST = 100)

	Example 1		Example 2		Example 3	
	BFS Method	Complex Method	BFS Method	Complex Method	BFS Method	Complex Method
μ_1^*	6.90	6.81	8.07	7.97	3.04	3.00
μ_2^*	1.69	1.65	1.30	1.41	0.66	0.69
μ_3^*	3.81	3.99	2.83	2.95	1.27	0.80
μ_4^*	1.13	1.01	0.90	0.93	0.59	0.79
μ_5^*	1.24	1.50	1.10	1.13	0.67	0.84
λ_{BFS}^*	6.19	6.56	6.66	7.56	2.63	2.95
λ_{MVA}	6.57	6.55	7.54	7.57	3.00	2.97
COST	100.00	99.81	100.00	99.97	100.00	100.00

Table 11.1 contains additional examples of the maximization of the throughput of a network with linear cost constraints. It compares values obtained by the BFS method with values obtained by the complex method which requires much more computation. If we consider the simplicity of our method, it is surprising how close our values are to those obtained by the complex method. These examples show that optimization problems can be relatively easily solved using the system of equations. If the cost constraint is linear, then the optimal service rates μ_i^* can be explicitly stated. If the cost function is non-linear, then a simple iteration method converges well. Table 11.2 contains the results for the optimization of the throughput with a non-linear cost function. The service rates that produce optimal throughput were used as input values for the exact MVA. The BFS method is well suited for solving optimization problems because it provides a simple relation between the quantity to be optimized (e.g., throughput) and the decision variables (service rates). If the cost constraint is linear, then explicit formulae for optimal service rates and throughputs are provided. It is also possible to apply the BFS

Table 11.2 Optimization of throughput (α = exponent of the non-linear cost function)

α	1	1.25	1.5	2.0	2.5	3
μ_1^*	6.912	4.911	3.884	2.860	2.361	2.067
μ_2^*	1.689	1.242	1.017	0.806	0.716	0.674
μ_3^*	3.182	2.727	2.173	1.625	1.363	1.214
μ_4^*	1.096	0.921	0.821	0.735	0.706	0.704
μ_5^*	1.236	0.998	0.883	0.781	0.774	0.732
λ_{BFS}^*	6.189	4.438	3.532	2.625	2.180	1.918
λ_{MVA}	6.596	4.711	3.752	2.790	2.319	2.039

method to non-product-form queueing networks with arbitrarily distributed service times if we use the corresponding formulae in the system equation (see Section 10.1.4.3).

11.3 OPTIMIZATION BASED ON THE CONVOLUTION ALGORITHM

With the BFS method, combined with Lagrange multiplier, we obtained approximate results for the optimization problem. To obtain exact results, we can again use the concept of Lagrange multipliers together with the convolution algorithm [TrKi80, SRT88].

11.3.1 Maximization of the Throughput

In the convolution algorithm we use the following formula (see Eq. (8.14)) for the throughput:

$$\lambda(\boldsymbol{\mu}, K) = \frac{G(\boldsymbol{\mu}, K-1)}{G(\boldsymbol{\mu}, K)}, \tag{11.11}$$

with:

$$G(\boldsymbol{\mu}, K) = \sum_{\sum_{i=1}^{N} k_i = K} \prod_{i=1}^{N} F_i(k_i),$$

and:

$$F_i(k_i) = \left(\frac{e_i}{\mu_i}\right)^{k_i} \cdot \frac{1}{\beta_i(k_i)},$$

$$\beta_i(k_i) = \begin{cases} k_i!, & k_i \leq m_i, \\ m_i! m_1^{k_i - m_i}, & k_i \geq m_i, \\ 1, & m_i = 1. \end{cases} \tag{11.12}$$

As a cost function, we use the one suggested in Eq. (11.2) with additional main memory cost $C(K)$:

$$C(\boldsymbol{\mu}, K) = C(K) + \sum_{i=1}^{N} c_i \mu_i^{\alpha_i} = \text{COST}. \tag{11.13}$$

To maximize the throughput with a cost constraint, we can use the following Lagrange function, which is derived from Eqs. (11.11) and (11.13)):

$$L(\boldsymbol{\mu}, y, K) = \lambda(\boldsymbol{\mu}, K) + y \left(C(K) - \text{COST} + \sum_{i=1}^{N} c_i \mu_i^{\alpha_i} \right).$$

In order to obtain the optimal values for $\boldsymbol{\mu}$, we differentiate the Lagrange function. This operation results in the following non-linear system of equations:

$$\frac{\partial L}{\partial \mu_i} = 0, \qquad i = 1, \ldots, N, \tag{11.14}$$

$$\frac{\partial L}{\partial y} = 0. \tag{11.15}$$

We can use the computer algebra program MAPLE to obtain the derivatives and solve the non-linear system of equations. Alternatively, Moore's formula [Moor72] for the normalization constant can be used to obtain explicit formula for the preceding derivatives [TrKi80]. The system of non-linear equations can then be solved using the Newton-Raphson or other suitable method. The solution of the system is the optimum service rate vector $\boldsymbol{\mu}$ that fulfills Eq. (11.15). The maximum throughput can then be obtained with Eq. (11.11)

Example 11.2 Consider a multiprogramming computer system with a CPU and three disk devices. The product-form queueing network model is given in Fig. 11.1. The corresponding routing probabilities are:

$$p_{11} = 0.05, \quad p_{12} = 0.5, \quad p_{13} = 0.3, \quad p_{14} = 0.15, \quad p_{21} = p_{31} = p_{41} = 1.0.$$

For special values of the cost coefficients c_i and the exponents α_i (given in Table 11.3), the total budget COST = 500 and a linear memory cost function $C(K) = C_m K$ with $C_m = 50$, the optimal values of the throughput λ^*, the service rates μ_i^* and degree of multiprogramming K are given in Table 11.4.

We see that we have an optimum of throughput when degree of multiprogramming $K = 2$. The computation time for this method is high compared to the BFS method. On the other hand this method delivers exact results.

Problem 11.1 Use the BFS method for Example 11.2.

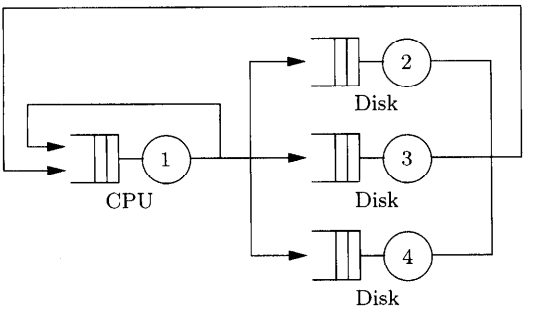


Fig. 11.1 Central-server model of the multiprogramming system in Example 11.2.

Table 11.3 Cost factors c_i and exponents α_i for the central-server model of Fig. 11.1

	c_i	α_i
CPU	131.9	0.55
Disk_1	11.5	1.00
Disk_2	54.2	0.67
Disk_2	54.2	0.67

11.3.2 Optimal Design of Storage Hierarchies

This example is intended to address the problem of optimally determining the technology and capacity of the different memory levels of a linear storage hierarchy. The storage hierarchy is used in a multiprogrammed environment. Our goal is to maximize the system throughput subject to a cost constraint. It is assumed that we know the number of storage levels as well as the capacity of the slowest level. Consider the storage hierarchy, given in Fig. 11.2 [TrSi81].

It consists of m electronically accessed levels (numbered $1, 2, \ldots, m$), as for example a cache and p mechanically accessed levels (numbered $m+1, m+$

Table 11.4 Optimal throughput λ^* with the optimal service rates μ_i^* for the central-server model of Fig. 11.1

K	λ^*	μ_1^*	μ_2^*	μ_3^*	μ_4^*
1	0.071	3.090	4.240	2.054	1.376
2	0.091	2.819	3.373	1.525	1.032
3	0.089	2.356	2.684	1.238	0.752
4	0.076	1.967	1.979	0.847	0.571
5	0.059	1.557	1.084	0.598	0.431

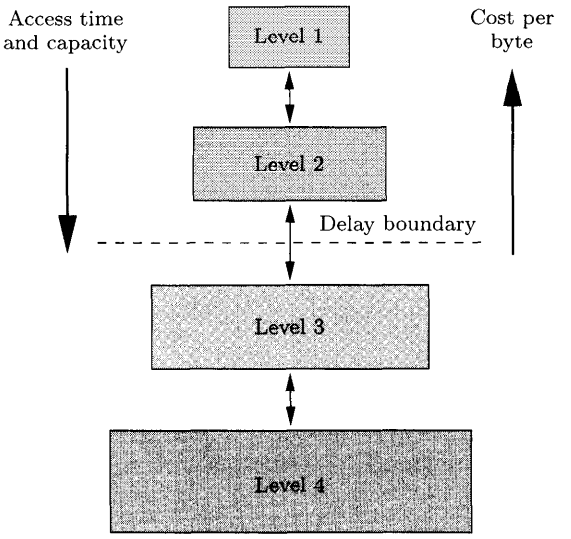


Fig. 11.2 A linear storage hierarchy.

$2, \ldots, m+p$), e.g., disk. Let $M = m + p$. These two groups are divided by a delay boundary. It is assumed that the speed increases with higher levels (lower numbers denote a faster level) and the capacity decreases with higher levels. Whenever a piece of information is not found in level 1, a request is sent to a lower level. This activity goes on until the request can be satisfied. If $i < m$ (meaning the device is above the delay boundary), it is assumed that the CPU is kept waiting until the information is retrieved, meaning that there will be no more than one request at levels $1, 2, \ldots, m$. For $m + 1 \leq i \leq m + p$, a process swap is performed to allow another process to use the CPU, meaning that several requests may be queued at each of the memory levels $m+1, m+2, \ldots, m+p$. This linear storage hierarchy is modeled using a closed queueing network where K (the multiprogramming degree) programs circulate sharing the resources of the system. The capacity of the ith level is b_i bits, which are organized into blocks of s_i bits each. The block sizes s_i, $i = 1, 2, \ldots, M$, are assumed to be known. The K active programs share the capacity of each level equally. A success function H_i denotes the probability that a storage reference is a hit at level i. A reasonable assumption is that this function depends only on b_i/K and s_i [TrSi81].

$$H_i = H_i(b_i/K, s_i) \quad i = 1, 2, \ldots, M.$$

Let the miss ratio function F_i be defined as $F_i = 1 - H_i$. The hit ratio is the probability that a storage reference generated by the CPU is found in

level i and not in levels $1, 2, \ldots, i-1$. The hit ratio h_i is given by:

$$\begin{aligned} h_i &= H_i(b_i/K, s_i) - H_{i-1}(b_{i-1}/K, s_{i-1}), \\ &= F_{i-1}(b_{i-1}/K, s_{i-1}) - F_i(b_i/K, s_i), \quad i = 1, 2, \ldots, M. \end{aligned}$$

The definition of b_0 and s_0 is such that $F_0(b_0/K, s_0) = 1$ and $F_M(b_M/K, s_M) = 0$. Let p_0 be the probability that a given request is the last memory request (i.e., the program has finished execution). Then the average total number of memory references per program is given by $1/p_0$. The probability that a memory request at level i is satisfied is then given by:

$$p_i = (1 - p_0)h_i, \quad i = 1, 2, \ldots, M.$$

The closed queueing network model of this system is shown in Fig. 11.3 [TrSi81]. All levels where requests are not queued are merged into an equiva-

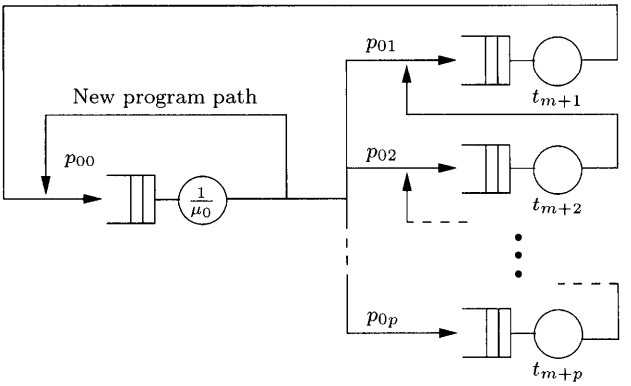


Fig. 11.3 Queueing network model of the linear storage hierarchy shown in Fig. 11.2.

lent server. The branching probabilities for the network are given as:

$$\begin{aligned} p_{00} &= \frac{p_0}{1 - \sum_{i=1}^{m} p_i}, \\ p_{0j} &= \frac{p_{j+m}}{1 - \sum_{i=1}^{m} p_i}, \quad j = 1, 2, \ldots, p. \end{aligned}$$

In [TrSi81] the service time for the equivalent server is derived:

$$\frac{1}{\mu_0} = \left(\sum_{i=1}^{m} p_i \sum_{j=1}^{i} t_j + \sum_{i=m+1}^{M} p_i \sum_{j=1}^{m} t_j \right) \cdot \frac{1}{1 - \sum_{i=1}^{m} p_i}. \tag{11.16}$$

In this context t_j denotes the average transfer time for a block of size s_{j-1} from the jth level to the $(j-1)$st level. Furthermore we assume that the service discipline at all nodes is FCFS and all service times are exponentially distributed. The goal is to maximize the system throughput, which is defined as the average rate of flow along the new program path, $\lambda = p_{00}\mu_0\rho_0$, where ρ_0 is the utilization of the equivalent server node and is given by:

$$\rho_0 = \frac{e_0}{\mu_0}\frac{G(\boldsymbol{\mu}, K-1)}{G(\boldsymbol{\mu}, K)},$$

$$\boldsymbol{\mu} = (\mu_0, \mu_1, \ldots, \mu_p), \tag{11.17}$$

$$x_0 = \frac{e_0}{\mu_0} = \sum_{i=1}^{m} p_i \sum_{j=1}^{i} t_i + \sum_{i=m+1}^{M} p_i \sum_{j=1}^{i} t_j,$$

$$= (1-p_0)\left(t_1 + \sum_{j=2}^{m} t_j F_{j-1}(b_{j-1}/K, s_{j-1})\right), \tag{11.18}$$

$$\frac{1}{\mu_i} = e_i t_{i+m} = t_{i+m} \sum_{j=1}^{p} p_{j+m},$$

$$= (1-p_0)t_{i+m}F_{i+m-1}(K_{i+m-1}/K, s_{i+m-1}), \quad i = 1, 2, \ldots, p. \tag{11.19}$$

In these expressions, $\mathbf{e} = (e_0, e_1, \ldots, e_p)$ is the vector of the visit ratios of the specified closed queueing network. Since the e_i can only be determined within a multiplicative constant, we choose:

$$e_0 = 1 - \sum_{i=1}^{m} p_i,$$

which yields:

$$e_i = \sum_{j=1}^{p} p_{j+m}.$$

The system throughput is given by:

$$\lambda(\boldsymbol{\mu}, K) = p_0 \frac{G(\boldsymbol{\mu}, K-1)}{G(\boldsymbol{\mu}, K)}.$$

The optimization problem is to maximize the throughput, or, in other words, to minimize:

$$z(\boldsymbol{\mu}, K) = \frac{1}{\lambda(\boldsymbol{\mu}, K)}.$$

Let the technology cost curve for the ith level be given by $c_i(t_i)$. The total cost of memory level i is given by $c_i(t_i)\theta_i(b_i)$, where $\theta_i(b_i)$ accounts for the

economies of scale that are present in the memory technology. The optimization problem can now be defined as follows:

$$\text{Minimize } z(\boldsymbol{\mu}, K) = \frac{1}{p_0} \frac{G(\boldsymbol{\mu}, K)}{G(\boldsymbol{\mu}, K-1)},$$

with $\boldsymbol{\mu}$ as defined in Eqs. (11.17), (11.18), and (11.19), and subject to:

$$\sum_{i=1}^{M} c_i(t_i)\theta_i(b_i) \leq \text{COST}. \tag{11.20}$$

In this case the decision variables are the capacities and the average access times for each level $(b_i, t_i, i = 1, 2, \ldots, M)$. The values m, p, K, p_0, b_M, $s_1, \ldots,$ s_{M-1}, and COST are assumed to be fixed parameters. Furthermore, we assume that F_i and θ_i are functions of the memory capacities b_i. The b_i, c_i, and θ_i are posynomial functions of the access times t_i (posynomials are generalizations of polynomials where the coefficients are positive real numbers while the exponents are allowed to be arbitrary real numbers). In the special case of $K = 1, z(\boldsymbol{\mu}, 1)$ is a polynomial function of the μ_i since:

$$z(\boldsymbol{\mu}, 1) = \frac{1}{p_0} \sum_{i=1}^{M} \mu_i.$$

The optimization problem in the case of $K = 1$ is then seen to be a standard geometric programming problem , provided we assume that b_i, c_i, and θ_i are polynomial functions of t_i. In [TrSi81] it is shown that even in the multiprogramming case, the optimization problem becomes a convex programming problem when logarithmically transforming the variables t_i and b_i (i.e., $t_i = e^{v_i}$ and $b_i = e^{\omega_i}, i = 1, 2, \ldots, M$) and requiring the functions $F_i(b_i), c_i(t_i)$ and $h_i(b_i), i = 1, 2, \ldots, M$ to be posynomial functions. Thus any local solution to the optimization problem is also its global solution.

12

Performance Analysis Tools

Performance analysis tools have acquired increased importance due to increased complexity of modern systems. It is often the case that system measurements are not available or are very difficult to get. In such cases the development and the solution of a system model is an effective method of performance assessment. Software tools that support performance modeling studies provide one or more of the following solution methods:

- Discrete-event simulation.
- Generation and (steady-state and/or transient) solution of CTMC and DTMC.
- Exact and/or approximate solution of product-form queueing networks.
- Approximate solution of non-product-form queueing networks.
- Hierarchical (multilevel) models combining one or more of the preceding methods.

If we use DES, then the system behavior can be described very accurately, but computation time and resource needs are usually extremely high. In this book queueing network solutions as well as Markov chain analysis methods have been introduced to analyze system models. Queueing networks are very easy to understand and allow a very compact system description. For a limited class of queueing networks (see Chapters 8 and 9), so-called product-form queueing networks, efficient solution algorithms (such as convolution, MVA, SCAT) are available. But many queueing networks do not fulfill the product-form requirements. In this case approximation methods can be used (see

Chapter 10). It is also possible to develop a multilevel model to approximately solve a non-product-form queueing network. If the approximation methods are not accurate enough or cannot be applied to a certain problem, as can, for example, be the case when we have non-exponentially distributed service times or when blocking is allowed in the network, then DES or CTMC is used. As we have seen in earlier chapters, the state space for even simple systems can be huge and grows exponentially with the number of nodes and the number of jobs in the network. Nevertheless, due to increasing computational power and better solution algorithms, CTMCs have acquired greater importance.

For the solution of our models we realize that:

- The application of exact or approximate solution algorithms for queueing networks is very cumbersome, error prone, and time consuming and hence not feasible to carry out by hand calculations.
- The generation and solution of the CTMC/DTMC for even small systems by hand is nearly impossible.

We therefore need tools that can automatically generate the state space of the underlying CTMC and apply exact or approximate solution algorithms and/or that have implemented exact and approximate algorithms for queueing networks. In this chapter we introduce four representative performance modeling and analysis tools: PEPSY, SPNP, MOSES, and SHARPE. As mentioned previously, tools differ not only in their solution algorithms (DES, exact and approximate solution algorithms for queueing networks, generation and transient and steady-state solution of CTMCs) but also in their input/output facilities. There are tools that have a graphical user interface (GUI) and/or a special input language (batch mode or interactive) or both. Some tools are based on special model types such as queueing networks, SPNs, CTMC models, or precedence graphs (and others) or a combination of these model types.

12.1 PEPSY

PEPSY (**p**erformance **e**valuation and **p**rediction **sy**stem) [BoKi92] has been developed at the University of Erlangen-Nürnberg. Using this tool it is possible to describe and solve PFQNs and NPFQNs. More than 30 solution algorithms are incorporated. It is possible to specify open, closed, and mixed networks where jobs cannot switch to another job class. Closed job classes are described by the number of jobs in each class, while open job classes are described by the arrival rate of jobs at the class. To compute performance measures such as throughput, utilization, or mean response time of a given network, different exact as well as approximation algorithms are provided for PFQN and NPFQN. The network is described in textual and/or graphical form. The X11-windows version of PEPSY is called XPEPSY [Kirs93]. PEPSY

can be used on almost all UNIX machines, whereas XPEPSY needs the X11 windows system. A Windows95, WindowsNT version WinPEPSY with similar features is also available but it has a restricted set of solution algorithms.

12.1.1 Structure of PEPSY

12.1.1.1 The Input File The PEPSY input file contains the specification of the system to be modeled. The specification file has a name of the form `e_name` and is divided into a standard block and possibly additional blocks that contain additional information specific to solution algorithms. The standard block contains all necessary descriptions for most of the solution methods, for example, the number of nodes and job classes, the service time distribution at each node, the routing behavior of the jobs, and a description of the classes. The input file is usually generated using the `input` program, which is provided by PEPSY. Information that is not asked for by the input program is provided using the `addition` program.

The type of each node (and its characteristics) needs to be specified: the -/M/m-FCFS, -/G/1-PS, -/G/-IS, and -/G/1-LCFS) and several other node types. For example, two types of multiple server nodes with different service time distributions -/M/m-FCFS-ASYM and -/G/m-FCFS-ASYM can be specified. The service time distribution is defined by its first and second moment, or by the service rate and the squared coefficient of variation.

For the solution of the specified models, many different algorithms are implemented in PEPSY. These algorithms can be divided into six groups:

1. Convolution algorithm for product-form networks.
2. MVA for product-form networks.
3. Approximation algorithms for product-form networks.
4. Approximation methods for non-product-form networks.
5. Automated generation and steady-state solution of the underlying CTMC.
6. DES.

In addition to these six groups, there are some methods and techniques that do not fit in any of these groups. For example, the `bounds` method performs a calculation of the upper bounds of the throughput and lower bounds of the average response time.

12.1.1.2 The Output File For each file name `e_name`, an output file with the computed performance measures is generated. It is called `xx_name` where "`name`" is the same as in the input file and "`xx`" is an abbreviation for the used method. The output file consists of a short header with the name of

the model, the corresponding input file, and the solution method used to calculate the performance measures. The performance measures of all nodes are separated by classes.

The output file contains the performance measures for each node (throughput, visit ratio, utilization, average response time, average number of jobs, average waiting times, and average queue length) and the performance measures for the entire network (throughput, average response time and average number of jobs).

12.1.1.3 Control Files As already mentioned, PEPSY offers more than 30 solution methods to compute performance measures of queueing networks. In general, each method has certain limitations and restrictions. Because it would be impossible for the user to know all these restrictions and limitations, control files are used in PEPSY. Each control file contains the limitations and restrictions for each solution method, namely the network type (open, closed, or mixed), the maximum number of nodes and classes allowed, whether the method can deal with routing probabilities or visit ratios, whether the service rates at -/M/m-FCFS nodes have to be the same for different job classes, the range of the coefficients of variation, and the maximum number of servers at a -/M/m node.

12.1.2 Different Programs in PEPSY

The following lists the most important programs of PEPSY. For a more detailed reference see [Kirs94].

input: This program is used for an interactive description of a queueing network.

addition: For some solution algorithms, additional information is needed. This information is provided by invoking the addition program.

selection: Using this program, all possible solution algorithms can be listed that are applicable to the system description (the file that was just created). The output of this program consists of two columns. The left-hand column contains all solution algorithms that can be applied directly, while the right-hand column contains the methods that can only be applied after specifying an additional block (using the addition program).

analysis: After a complete system description, the queueing network desired solution algorithm can be applied by invoking the analysis program. The results are printed on the screen as well as written to a file.

pinf: This program is invoked to obtain information about the solution method utilized.

transform: As already mentioned before, it is possible to specify the network using either routing probabilities or visit ratios. By invoking the transform program, visit ratios can be transformed into routing probabilities.

pdiff: By invoking pdiff, the results of two different solution methods can be compared.

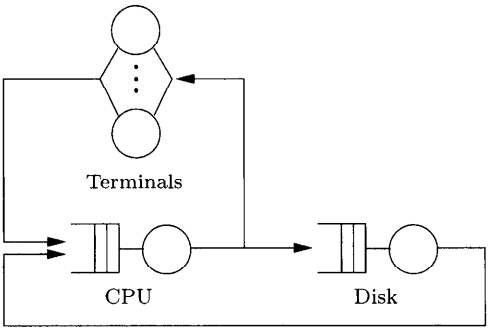


Fig. 12.1 Simple queueing network example.

12.1.3 Example of using PEPSY

The main steps in working with PEPSY are described herein. Starting with the model, first the model file is created, then the solution method is selected, and the analysis is carried out. The model to be analyzed is shown in Fig. 12.1.

At first the input program is invoked by means the command `input`. The user will then be asked to enter the type and number of job classes (a closed network with only one job class in our case), the number of nodes (three in our case), the type of each node, and its service rate. After the basic network information is specified, the routing information is entered. In PEPSY we need to consider one special point when talking about routing: open job classes contain a (combined) source/sink-node that is used in PEPSY as a reference node (outside node) for relative performance values. For closed queueing networks the outside node also needs to be specified. Finally the number of jobs in the network needs to be specified.

Now the file is saved in, say, `e_first_example`. The corresponding model description, generated by `input`, is shown in Fig. 12.2. The next step is the selection of the solution method. PEPSY supports the user with a built-in database with the limitations and restrictions of each algorithm and returns a list of applicable solution methods. In our example the result of applying the selection program to the model description is shown in Fig. 12.3. This list contains all the methods that can be used to analyze the network `e_simple_example`. In addition to the standard network description that has been entered in the beginning, some methods need further parameters.

```
#
# filename  e_simple_example
#

NUMBER NODES: 3
NUMBER CLASSES: 1

NODE SPECIFICATION

  node   |      name          |     type
---------+--------------------+--------------------
     1   |      cpu           |     -/G/1-PS
     2   |      disk          |     -/M/1-FCFS
     3   |      terminal      |     -/G/0-IS

CLASS SPECIFICATION

  class    | arrival rate   number of jobs
-----------+-----------------------------------
     1     |     -              10

CLASS SPECIFIC PARAMETERS

CLASS  1

  node       |  service_rate   squared_coeff._of_variation
-------------+-------------------------------------------
  cpu        |  10                      1
  disk       |  3                       1
  terminal   |  0.2                     1

ROUTING PROBABILITIES

from/to     | outside    cpu        disk       terminal
------------+-------------------------------------------
outside     | 0.000000  1.000000  0.000000  0.000000
cpu         | 0.000000  0.000000  0.700000  0.300000
disk        | 1.000000  0.000000  0.000000  0.000000
terminal    | 1.000000  0.000000  0.000000  0.000000
```

Fig. 12.2 Model description file e_simple_example.

```
simple_example:
    applicable solution method |  need further specification
   ----------------------------------------------------------
    ammva                      |
    bol_aky                    |
    bounds                     |
    cmva                       |
                               |  hm
    marie                      |
    mmva                       |
    monosum                    |
    multisum                   |
    num_app                    |
    num_single                 |
    pm_2                       |
    pmapp_1                    |
    priomva2c                  |
    priomva2m                  |
    recal                      |
                               |  sim2
```

Fig. 12.3 The list of applicable solution methods presented by the selection program.

```
PERFORMANCE_INDICES FOR NET:  simple_example
description of the network is in file 'e_simple_example'
the closed network was solved using the method 'marie'
jobclass 1
marie       |  lambda      e     1/mu     rho      mvz      maa      mwz     mwsl
------------+-------------------------------------------------------------------
cpu         |   3.922   1.000   0.100   0.392   0.156   0.613   0.056   0.221
disk        |   2.746   0.700   0.333   0.915   1.276   3.503   0.943   2.588
terminal    |   1.177   0.300   5.000   0.000   5.000   5.883   0.000   0.000

characteristic indices:
marie       |  lambda      mvz      maa
------------+--------------------------
            |   3.922   2.549  10.000
legend
e    : average number of visits              mu     : service rate
rho  : utilisation                           lambda: mean throughput
mvz  : average response time
maa  : average number of jobs
mwz  : average waiting time
mwsl: average queue-length
```

Fig. 12.4 Performance measures of the network e_simple_example.

For example, in order to run the `sim2` method (DES), some information about the maximum simulation time and accuracy needs to be specified. If the user is not familiar with a particular method, PEPSY assists the user with online help.

Suppose the user selects the method name `marie` and invokes the solution method by entering the command `analysis marie simple_example`. The resulting performance measures are shown on the screen and are also written into the file `a_c_simple_example`. The contents of this file are shown in Fig. 12.4.

12.1.4 Graphical User Interface XPEPSY

To demonstrate how to work with XPEPSY, the same queueing model as in Section 12.1.2 is used. After the program is started, the user can graphically draw the network as shown in Fig. 12.5. Nodes or connections can be created, moved, or deleted simply by mouse clicks. The necessary parameters are specified in the corresponding dialog boxes. Consider, for example, the node data dialog box, shown in Fig. 12.6*a* that appears when clicking on the service station in the drawing area and that is used to specify node parameters.

When the queueing network is fully specified, XPEPSY switches from the edit mode to the analysis mode and a new set of menus will appear. Now PEPSY assists you in selecting the solution method. Four possibilities are offered:

1. Select from a limited set of methods, i.e., the ones that are very well suited for the problem.

2. Select from all possible methods.

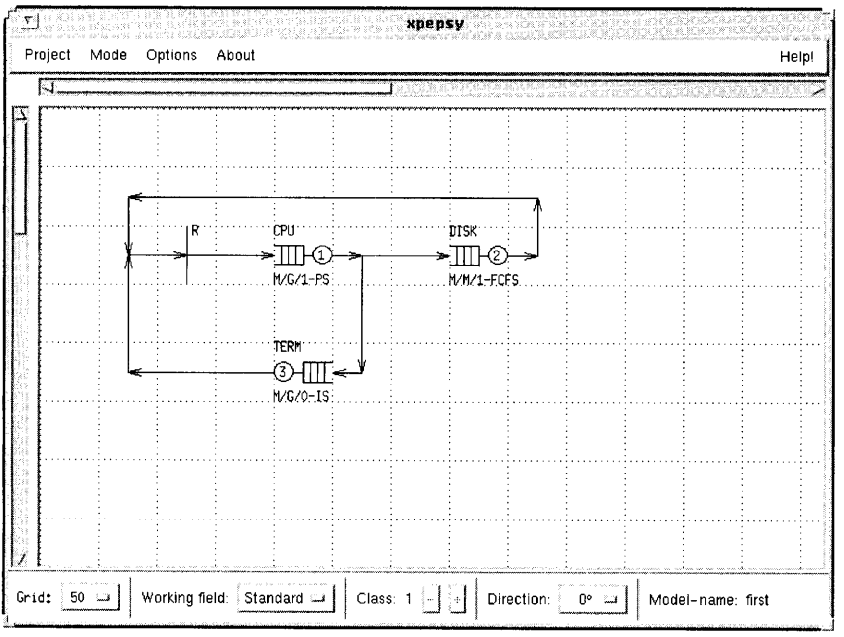


Fig. 12.5 The main XPEPSY window.

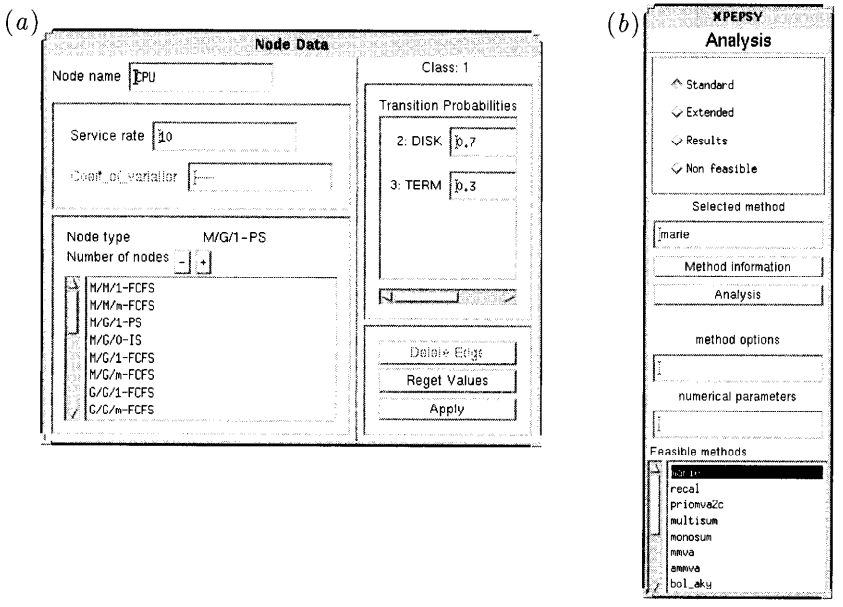


Fig. 12.6 (*a*) The node data dialog box and (*b*) the menu of solution methods corresponding to Fig. 12.5.

3. Show only methods that have already been used and for which valid results can be found without resolving the model.

4. Show all methods that cannot be used due to inherent restrictions of the solution method.

Now the solution can be carried out by simply activating the `Analysis` button from the analysis menu, shown in Fig. 12.6*b*, and the results are presented on the screen (see Fig. 12.4). The results can also be presented as a histogram or as a line drawing. In a line drawing it is also possible to show an output parameter as a function of an input parameter.

Some additional programs from PEPSY are integrated into XPEPSY as well. Using the `Method Information` button, the PEPSY internal database can be searched for information about the selected method.

12.2 SPNP

The stochastic Petri net package (SPNP) developed by Ciardo, Muppala and Trivedi, is a versatile modeling tool for performance, dependability, and performability analysis of complex systems. Input models developed based on the theory of stochastic Petri nets are solved by efficient and numerically stable algorithms. Steady-state, transient, cumulative transient, time-averaged, and up-to-absorption measures can be computed. Parametric sensitivity analysis of these measures is efficiently carried out. Some degree of logical analysis capabilities is also available in the form of assertion checking and the number and types of markings in the reachability graph. Advanced constructs available - such as marking-dependent arc multiplicities, guards, arrays of places and transitions, and subnets - reduce modeling complexity and enhance the power of expressiveness of the package. The most powerful feature is the capability to assign reward rates at the net level and subsequently compute the desired measures of the system being modeled. The model description language is CSPL, a C-like language, although no previous knowledge of the C language is necessary to use SPNP.

12.2.1 SPNP Features

The input language of SPNP is CSPL (C-based stochastic **P**etri net **l**anguage). A CSPL file is compiled using the C compiler and then linked with the precompiled files that are provided with SPNP. The full power of the C programming language can be used to increase the flexibility of the net description. An important feature of CSPL is that it is a superset of C. Thus a CSPL user can exploit C language constructs to represent a large class of SPNs within a single CSPL file. Most applications will only require a limited knowledge of the C syntax, as predefined functions are available to define SPNP objects.

Another important characteristic of SPNP is the provision of a function to input values at run time, before reading the specification of the SPN. These input values can be used in SPNP to modify the values of scalar parameters or the structure of the SPN itself. Arrays of places and transitions and subnets are two features of SPNP that are extremely useful in specifying large structured nets. A single CSPL file is sufficient to describe any legal SPN, since the user of SPNP can input at run time the number of places and transitions, the arcs among them, and any other required parameters.

The SPNP package allows the user to perform *steady-state, transient, cumulative transient*, and *sensitivity analysis* of SPNs. Steady-state analysis is often adequate to study the performance of a system, but time-dependent behavior (transient analysis) is sometimes of greater interest: instantaneous availability, interval availability, and reliability (for a fault-tolerant system); response time distribution of a program (for performance evaluation of software); computation availability (for a degradable system) are some examples. Sensitivity analysis is useful to estimate how the output measures are affected by variations in the value of input parameters, allowing the detection of system performance bottlenecks or aiding in design optimization.

Sophisticated steady-state and transient solvers are available in SPNP. In addition, the user is not limited to a predefined set of measures: Detailed expressions reflecting exactly the measures sought can be easily specified. The measures are defined in terms of reward rates associated with the markings of the SPN (see Section 2.2.3). The numerical solution methods provided in the package address the *stiffness* problems often encountered in reliability and performance models (see Section 5.1.4.2).

A number of important Petri net constructs such as marking dependency, variable cardinality arc, guards, arrays of places and transitions, subnets, and assertions facilitate the construction and debugging of models for complex systems. A detailed description of SPNP and the input language CSPL can be found in [CFMT94] and [CTM89]. SPNP has been installed at more than 100 sites and has been used to solve many practical problems at Digital Equipment Corporation, Hewlett Packard, Nortel and other corporations.

Some very powerful new features that have been added to SPNP include the capability of DES of non-Markovian nets and fluid stochastic Petri nets [CNT97].

12.2.2 The CSPL Language

Modeling with SPNP implies that an input file describing the system structure and behavior must be prepared. Such an input file can be prepared automatically via the graphical user interface that has been recently developed as discussed in Section 12.2.3. Alternatively, the user may decide to prepare such a file himself. The language designed to do so is named CSPL, a superset of the C language [KeRi78]. What distinguishes CSPL from C is a set of predefined functions specially developed for the description of SPN

```
/* begin of CSPL file */
parameters(){
  ...
}
net(){
  ...
}
assert(){
  ...
}
ac_init(){
  ...
}
ac_reach(){
  ...
}
ac_final(){
  ...
}
/* end of CSPL file */
```

Fig. 12.7 Basic structure of a CSPL input file

entities. Any legal C construct can be used anywhere in the CSPL file. All the C library functions, such as `fprintf`, `fscanf`, `log`, `exp`, etc., are available and perform as expected. The only restriction to this generic rule is that the file should not have a `main` function. In spite of being a programming language, CSPL enables the user to describe SPN models very easily. There is no need to be a programmer to fully exploit all the built-in features of SPNP. Just a basic knowledge of C is sufficient to describe SPNs effectively; although, for experienced programmers, CSPL brings the full power and generality of the C language.

Fig. 12.7 shows the basic structure of a CSPL input file with six different segments.

Parameters Segment The function `parameters` allows the user to customize the package. Several parameters establishing a specific behavior can be selected. The function `iopt` (`fopt`) enables the user to set `option` to have the integer (double-precision floating point) `value`. Any of the available options can be selected and modified. For example:

```
parameters(){
  ...
  iopt(IOP_METHOD,VAL_TSUNIF);
  iopt(IOP_PR_RGRAPH,VAL_YES);
  ...
}
```

specifies that the solution method to be used is transient solution as uniformization and that the reachability graph is to be printed. The function `input` permits the input of parameter values at run time.

Net Segment The function `net` allows the user to define the structure and parameters of an SPN model. For built-in functions that can be used inside the `net` segment, Fig. 12.8 provides some illustrations.

```
net(){
  ...
  /* places */
  place("p1");
  place("p2");
  ...
  place("pn");

  /* initial markings */
  ...
  init("p1",n);
  ...
  init("pn",2);
  ...

  /* transitions */
  ...
  trans("t1");
  trans("t2");
  ...
  trans("tm");
  ...

  /* input arcs,
     multiple input arcs */
  ...
  iarc("t1","p1");
  iarc("t3","p2");
  miarc("t5","p6",3);
  ...

  /* output arcs,
     multiple output arcs */
  ...
  oarc("t2","p1");
  oarc("t1","p2");
  moarc("t3","p6",2);
  ...

  /* inhibitor arcs,
     multiple inhibitor arcs */
  ...
  harc("t4","p5");
  mharc("t5","p3");
  ...

  /* priorities */
  ...
  priority("t3",10);
  priority("t6",5);
  ...

  /* firing probabilities
     of immediate transitions */
  ...
  probval("t7",0.4);
  probval("t8",0.6);
  ...

  /* firing rates of timed transitions */
  ...
  rateval("t1",2.0);
  rateval("t2",5.0);
  ...
}
```

Fig. 12.8 The CSPL net segment.

There are other functions such as **guards**, which allow the user to define the enabling condition of a transition as a function of tokens in various places, or functions that allow the user to define arrays of places and transitions or functions that pertain to sensitivity analysis.

Assert Segment The `assert` function allows the evaluation of a logical condition on a marking of the SPN. For example, the `assert` definition:

```
assert(){
  if(mark("p2")+mark("p3") != 4 || enabled("t11") && enabled("t7"))
    return(RES_ERROR);
  else
    return(RES_NOERR);
}
```

will stop the execution in a marking where the sum of the number of tokens in places `p2` and `p3` is not 4, or where `t11` and `t7` are both enabled.

Ac_init and Ac_reach Segment The function `ac_init` is called just before starting the reachability graph construction. It can be used to output data about

the SPN in the ". out" file. This is especially useful when the number of places or transitions is defined at run time (otherwise it is merely a summary of the CSPL file). The function `ac_reach` is called after the reachability graph construction is completed. It can be used to output data about the reachability graph in the ". out" file.

Ac_final Segment The function `ac_final` is called after the solution of the CTMC has been completed, to carry out the computation and printing of user-requested outputs. For example, the following function:

```
ac_final(){
  pr_std_average();
  pr_std_average_der();
  pr_message(goodbye);
}
```

writes in the ".out" file for each place the probability that it is not empty and its average number of tokens, and for each transition the probability that it is enabled and its average throughput, respectively. The derivatives of all the preceding standard measures with respect to a set of previously chosen parameters are computed and printed. In the end it writes the message `goodbye`. Before applying the `ac_final` function, additional functions often have to be provided. This can be done, for instance, using the construct `reward_type` as it is shown in the following example:

```
reward_type ep1() { return(mark("p1")); }
reward_type ep3() { return(mark("p3")); }
reward_type ep7() { return(mark("p7")); }
...
ac_final(){
  x = expected(ep1)*expected(ep7)+expected(ep3)*1.2;
  printf("%f",x);
}
```

Example Using the example of Fig. 12.9, we show that the application of SPNP to analyse systems that can be modeled by an SPN is a straightforward procedure. The inhibitor arcs avoid a capacity overflow in places `p1` and `p2`. The CSPL file for this SPN is shown in Fig. 12.10.

A short description of this specification follows:

Line 4-6: The reachability graph, the CTMC, and the state probabilities of the CTMC are to be printed.

Line 10-22: The places, the initial marking of the places, the transitions, their firing rates, and the input, output, and inhibitor arcs are defined.

Line 26-29: Check whether a capacity overflow in either place p1 or place p2 has occurred.

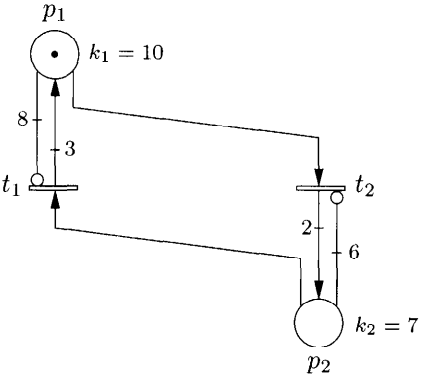


Fig. 12.9 Petri net example.

```
 1  #include "user.h"
 2
 3  parameter(){
 4    iopt(IOP_PR_RGRAPH,VAL_YES);
 5    iopt(IOP_PR_MC,VAL_YES);
 6    iopt(IOP_PR_PROB,VAL_YES);
 7  }
 8
 9  net(){
10    place("p1");
11    init("p1",1);
12    place("p2");
13    trans("t1");
14    trans("t2");
15    rateval("t1",1.0);
16    rateval("t2",1.0);
17    iarc("t1","p2");
18    iarc("t2","p1");
19    moarc("t1","p1",3);
20    moarc("t2","p2",2);
21    mharc("t1","p1",8);
22    mharc("t2","p2",6);
23  }
24
25  assert(){
26    if ((mark("p1")>10)||(mark("p2")>7))
27      return(RES_ERROR);
28    else
29      return(RES_NOERR);
30  }
31
32  ac_init(){
33    pr_net_info();
34  }
35
36  ac_reach(){
37    pr_rg_info();
38  }
39
40  reward_type s1(){ return(mark("p1")); }
41  reward_type s2(){ return(mark("p2")); }
42  reward_type p1_empty(){ if(mark("p1")==0)
43                            return(1.0);
44                          else
45                            return(0.0);}
46  reward_type p2_empty(){ if(mark("p2")==0)
47                            return(1.0);
48                          else
49                            return(0.0);}
50
51  ac_final(){
52    pr_mc_info();
53    pr_expected("Mean number of tokens in p1",s1);
54    pr_expected("Mean number of tokens in p2",s2);
55    pr_expected("Prob. that p1 is empty",p1_empty);
56    pr_expected("Prob. that p2 is empty",p2_empty);
57    pr_std_average();
58  }
```

Fig. 12.10 CSPL file for the SPN in Fig. 12.9.

Line 33-37: Data about the SPN and the reachability graph of the SPN are to be printed.

Line 40: The number of tokens in place p1 in each marking is the reward rate s1 in that marking.

Line 41: The number of tokens in place p2 in each marking is the reward rate s2 in that marking.

Line 42-49: Self-explanatory.

Line 52: Data about the CTMC and its solution are to be printed.

Line 53-54: The mean number of tokens in the places p1 and p2 are to be computed and printed.

Line 55-57: The probabilities that the places p1 and p2 are empty are to be computed and printed.

12.2.3 iSPN

Input to SPNP is specified using CSPL, but iSPN (**i**ntegrated environment for modeling using **s**tochastic **P**etri **n**ets) removes this burden from the user by providing an interface for graphical representation of the model. The use of Tcl/Tk [Welc95] in designing iSPN makes this application portable to all platforms [HWFT97].

The major components of the iSPN interface (see Fig. 12.11) are a Petri net editor, which allows graphical input of the stochastic Petri net, and an extensive collection of visualization routines to display results of SPNP and aid in debugging. Each module in iSPN is briefly described in the following:

Input Data: iSPN provides a higher level input format to CSPL, which provides great flexibility to users. iSPN is capable of executing SPNP with two different file formats: Files created directly using the CSPL and files created using iSPN's Petri net editor.

The Petri net editor (see Fig. 12.12), the software module of iSPN that allows users to graphically design the input models, introduces another way of programming SPNP: The user can draw the SPN model and establish all the necessary additional functions (i.e., rewards rates, guards, etc.) through a common environment. The Petri net editor provides several characteristics normally available only in sophisticated two-dimensional graphical editors and a lot of features designed specifically for the SPNP environment.

iSPN also provides a textural interface, which is necessary if we wish to accommodate several categories of users. Beginners may feel more comfortable using the Petri net editor, whereas experienced SPNP users

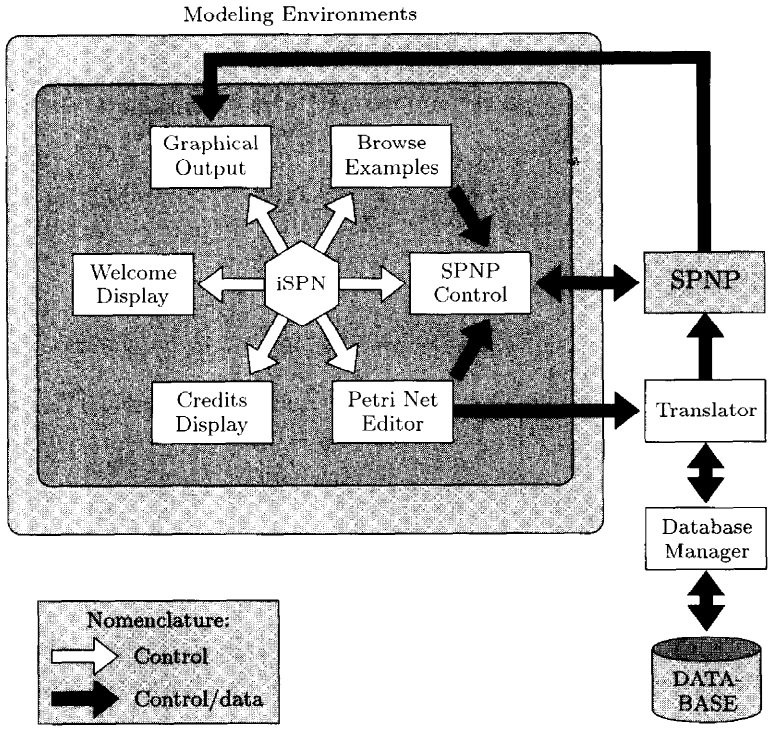


Fig. 12.11 Main software modules for iSPN.

may wish to input their models using CSPL directly. Even if the textual input is the option of choice, many facilities are offered through the integrated environment. In both cases, the "SPNP control" module provides everything a user needs to run and control the execution of SPNP without having to switch back and forth among different environments.

Output Data: The main goal of most GUIs is only to facilitate the creation of input data for its underlying package. Usually, the return communication between the software output and the GUI is neglected. One of the advantages of iSPN is the incorporation of displaying SPNP results in the GUI application. iSPN's own graphing capability allows the results of experiments to be graphically displayed in the same environment (see Fig. 12.13). Different combinations of input data may be compared against each other on one plot or viewed simultaneously. The graphical output format is created in such a way that it may be viewed by other visualization packages such as gnuplot of xvgr.

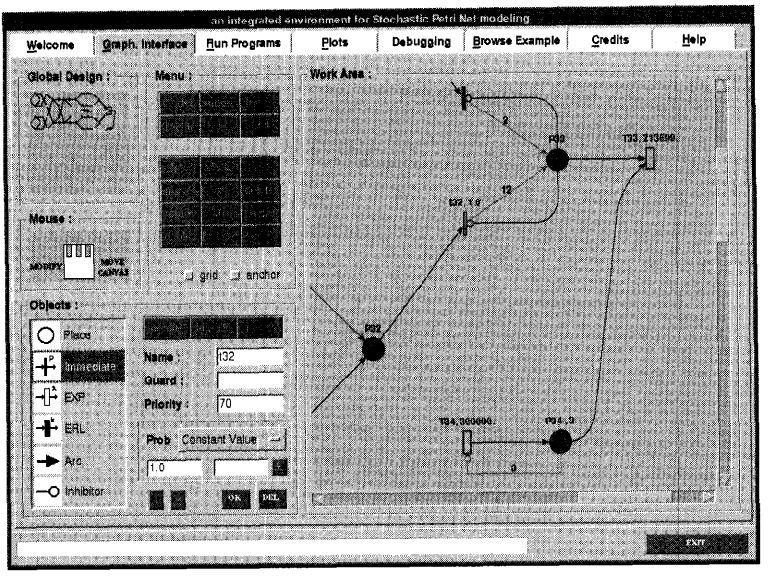


Fig. 12.12 The Petri net editor.

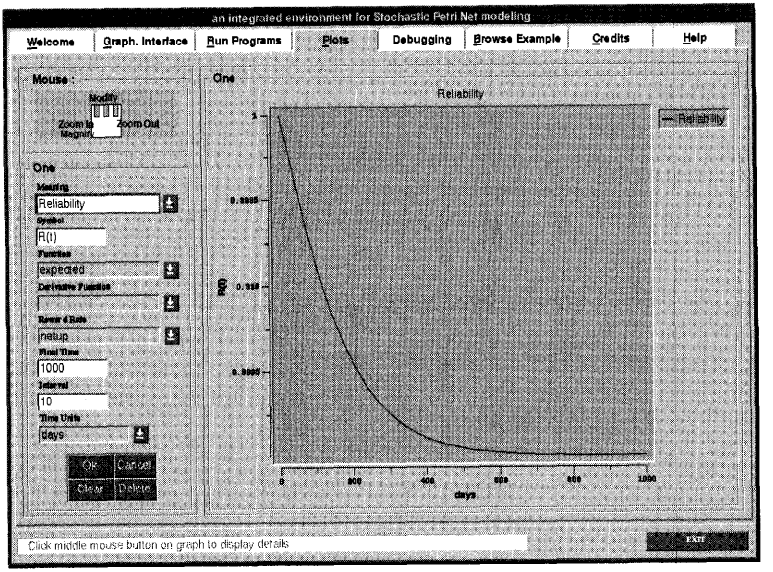


Fig. 12.13 The iSPN output page.

12.3 MOSES

In this section the CTMC-based tool MOSES (**mo**deling, **s**pecification, and **e**valuation **s**ystem), developed at the University of Erlangen, is introduced. MOSES is based on the system description language MOSEL (**mo**deling **s**pecification, and **e**valuation **l**anguage) [BoHe96]. The core of MOSEL consists of several constructs to specify the possible state and the state transitions of the CTMC to be analyzed. The basic way that MOSES computes the performance measures is shown in Fig. 12.14.

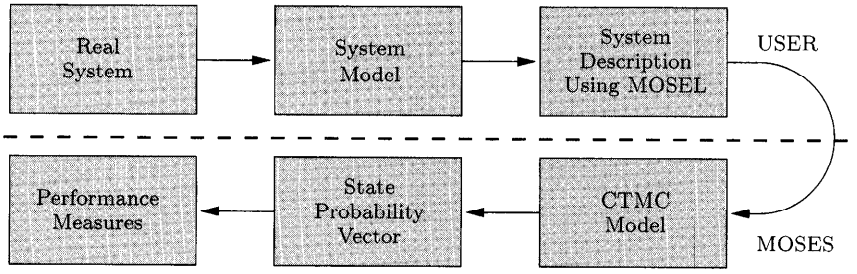


Fig. 12.14 Different steps when computing the performance measures with MOSES.

MOSES gets as input the system description (in MOSEL) and develops the underlying generator matrix **Q**. Five solution methods for the CTMC are provided (Grassmann (see Section 3.3.2), Jacobi (see Section 3.4.3), Gauss-Seidel (see Section 3.4.4), multilevel method (see [Hort96]), and uniformization (see Section 5.1.4)). From the state-probability vector, the system performance measures of interest are computed. In the following section we provide a brief overview of the model description language MOSEL.

12.3.1 The Model Description Language MOSEL

MOSEL consists of a series of segments, described in the following:

The DECLARATION Part:

`#define` This method of definition is similar to the corresponding C constructs except that it only allows numerical values (integer and floating point) and no expressions or other types of assignments.

`#enum` This construct allows the definition of a set of constants. If we do not assign any integer value to a constant name in the enumeration list, then this constant gets a default integer value. For example, the following

two enum declarations are both identical:

```
enum cpu_state {idle = 0, user = 1, kernel = 2, driver = 3};
enum cpu_state {idle,user,kernel,driver};
```

The VECTOR Description Part: This defines how a state vector looks (NODES Part) and which state vectors are prohibited (NOT Part).

`NODES Part` In this part of the specification the components of the state vector are specified. For each component of the state vector, its name and capacity (since we consider finite CTMCs) must be given. As an example, consider:

```
NODE node_1[K]      ; /* Range: 0..K */
NODE cpu[cpu_state]; /* Range: 0..3 */
```

For the definition of `cpu_state`, see the example in the `#enum construct` given previously. There is also the possibility for defining subnodes within a NODE declaration. For instance:

```
NODE N1[K; N11[1]; N12[1]] /* Node N1 could model a M/E2/1 queue */
```

Subnodes can, for example, be used for components with Erlang distributed service times.

`START Part` This optional part specifies a valid start state for the transient analysis.

`NOT Part` This construct is used to specify the prohibited system states. If we have, for example, a closed queueing network, then the sum of jobs over all stations in the network cannot be different from K (the overall number of jobs in the system).

The RULES Part: This is the most important segment in the system because the RULE constructs are used to specify the state transitions. Each rule consists of a global part and, optionally, a local part. Consider, for example, the following two rules where the first rule consists only of a global part while the second one has also a local part, consisting of two local rules:

```
FROM node_1 TO node_2 W mu_1 ;
FROM node_1 W mu_1 IF state_node_1==up
     { TO node_2  P p_12 ;
       TO node_3  P p_13 ; }
```

The first rule specifies a transition from `node_1` to `node_2` with transition rate `mu_1`. Since we do not specify a routing probability, 1.0 is chosen as default value. It should be noted here that instead of directly specifying the state transitions of the underlying CTMC (although it can be done), service

rates of stations and routing probabilities of the network are specified. This compact method of specification is much easier and less error prone. The second rule consists of two local rules. The global part states that transitions begin at `node_1` and all take place with transition rate `mu_1`. These transitions occur only if `state_node_1 == up`. In the local part we define the transition targets and probabilities. It is even possible to specify local conditions for each local rule. In this case the global as well as local conditions have to be satisfied for the transition to occur.

The RESULT Part In this part the basic performance measures of the CTMC computed from the steady-state probabilities are requested to be printed.

Loops in MOSEL Some parts of a specification happen to be very similar, differing, for example, only in one index. For situations like these, MOSEL offers the `loop` construct. Consider the following example specification:

```
FROM node_2 TO node_1 W mu_2 P p21 ;
FROM node_3 TO node_1 W mu_3 P p31 ;
FROM node_4 TO node_1 W mu_4 P p41 ;
```

It is very cumbersome and error prone to repeat every single specification since they differ from each other only in the index. These three lines of code can be shortened using the loop construct:

```
<2..4> FROM node_# TO node_1 W mu_# P p#1 ;
```

As we can see, the loop indices are specified within the angled brackets.

12.3.2 Examples

12.3.2.1 Central-Server Model In the following the central-server model is specified. By using MOSEL, the queueing network model shown in Fig. 12.15 can be specified in many different ways. Here we present two different versions: a longer one that is quite straightforward (see Fig. 12.16) and a second one that is a much shorter specification and demonstrates the power of loops (Fig. 12.17).

A short description of the lines in the specification of Fig. 12.16 follows:

Line 2: Definition of the number of jobs in the system.

Line 5-7: Definition of the state space.

Line 9: Definition of the prohibited states in the system. At any time, the sum of all jobs at all nodes must be K.

Line 13-14: Specification of all possible state transitions from node `N1` to all other nodes.

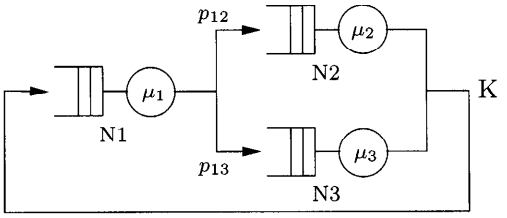


Fig. 12.15 The central-server model.

```
1   /* DECLARATION part */
2   #define K 10
3
4   /* VECTOR description part */
5   NODE N1[K];
6   NODE N2[K];
7   NODE N3[K];
8
9   NOT N1 + N2 + N3 != K;
10
11  /* RULES part */
12  FROM N1 TO N2 W mu1 P p12;
13  FROM N1 TO N3 W mu1 P p13;
14
15  FROM N2 TO N1 W mu2 P 1.0;
16  FROM N3 TO N1 W mu3 P 1.0;
17  /* RESULTS part */
18  RESULT >> rho1 = UTIL N1;
19  RESULT >> rho2 = UTIL N2;
20  RESULT >> rho3 = UTIL N3;
21
22  RESULT >> n_1 = MEAN N1;
23  RESULT >> n_2 = MEAN N2;
24  RESULT >> n_3 = MEAN N3;
25
26  RESULT >> lambda1 = rho1 * mu1;
27  RESULT >> lambda2 = rho2 * mu2;
28  RESULT >> lambda3 = rho3 * mu3;
29
30  RESULT >> t_1 = n_1 / lambda1;
31  RESULT >> t_2 = n_2 / lambda2;
32  RESULT >> t_3 = n_3 / lambda3;
```

Fig. 12.16 MOSEL implementation of the central-server model without loops.

Line 15-16: Specification of all possible transitions from nodes `N2` and `N3` to node `N1`.

Line 18-20: Specification of the utilization of a node using the `UTIL` construct (which computes the utilization of a node using the state probabilities).

Line 22-24: Specification of the mean number of jobs at a node using the `MEAN` construct.

Line 26-32: Definition of other interesting performance measures.

As we can see, the use of loops considerably reduces the length of the specification.

12.3.2.2 Fault Tolerant Multiprocessor System Here we show the usability of MOSEL for the model of a fault tolerant multiprocessor system. Fig. 12.18 shows a multiprocessor system with a finite buffer capacity. It is assumed that the failure and repair of servers are mutually independent. The following characteristics also apply:

```
1   /* DECLARATION part */
2         #define K 10
3
4   /* VECTOR description part */
5         NODE <1..3> N#[K];
6
7         NOT N1 + N2 + N3 != K;
8
9   /* RULES part */
10        FROM N1 W mu1 {<2..3> TO N# P p1# };
11        FROM <2..3> N# TO N1 W mu# P p#1;
12
13  /* RESULTS part */
14  <1..3> RESULT >> rho# = UTIL N#;
15  <1..3> RESULT >> n_# = MEAN N#;
16  <1..3> RESULT >> lambda# = rho# * mu#;
17  <1..3> RESULT >> t_# = n_# / lambda#;
```

Fig. 12.17 MOSEL implementation of the central-server system with loops.

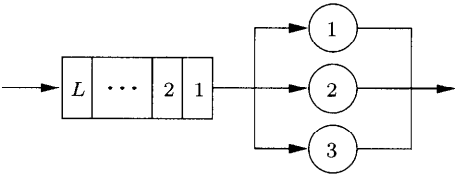


Fig. 12.18 An example of a multiprocessor system.

- The system has a finite job queue of length L.
- The arrival rate to this queue is `lambda`.
- The time to repair is exponentially distributed with the rate $1/mttr$.
- The time between failures is exponentially distributed with the rate $1/mtbf$.
- If a server that is currently working on a job fails, then the job is started again on a free server.
- If no server is currently available, then a job must wait until a server is repaired or becomes free.
- Service times are exponentially distributed with mean $1/\mu$.

The implementation in Fig. 12.19 shows one way to describe this model of the fault tolerant multiprocessor system.

Descriptions of the lines follow:

Line 2-4: Definition of the states space.

```
1  /* VECTOR description part */
2  NODE queue[L]
3  NODE active[3]
4  NODE working[3]
5
6  NOT queue+active > L+3;
7
8  /* RULES part */
9  FROME        TO  queue    W lambda;
10 FROM working TOE          W 1/mtbf;
11 FROME        TO  working  W 1/mttr;
12
13 FROM queue   TO active IF working-active > 0;
14
15 <1..3> FROM active TOE W #*mu IF active==# AND working>=#;
16 <1..2> FROM active TOE W #*mu IF active>=# AND working==#;
17
18 /* RESULTS part */
19 RESULT A = MEAN active;
20 RESULT rho = A/3;
21 RESULT IF (working > 0) p_working += PROB;
22 RESULT DIST queue;
23 RESULT q = MEAN queue;
24 RESULT >> throughput = 3*rho*mu*p_working;
```

Fig. 12.19 MOSEL implementation of the fault tolerant multiprocessor system

Line 6: Definition of the prohibited states in the system. The sum of the active jobs and the jobs in the queue shall not exceed L+3.

Line 9: Specification of the arrival of jobs in the system. (FROME: short form for FROM External).

Line 10-11: Specification of the failure and the repair of a processor. (TOE: short form for TO External).

Line 13: Specification of the transition from the queue to a processor.

Line 15-16: Specification of the possible transitions from the processor to external. If a processor fails while the job is being processed, the job is started on a free server, if available, or has to wait.

Line 19: Request the mean number of active processors.

Line 20: Request the utilization of the processors.

Line 21: Request the probability that at least one processor is working.

Line 22: Request the pmf of the number of jobs in the queue.

Line 23: Request the mean queue length.

Line 24: Request the throughput.

12.4 SHARPE

SHARPE (symbolic **h**ierarchical **a**utomated **r**eliability **p**erformance **e**valuator) tool was originally developed in 1986 by Sahner and Trivedi at Duke University [STP96]. It is implemented in C and runs on virtually all platforms. The advantage of SHARPE is that it is not restricted to one model type as PEPSY or SPNP. It offers seven different model types including product-form queueing networks, stochastic Petri nets, CTMCs, task precedence graphs and fault trees. The user can choose the model type that is most convenient for the problem. The models can also be hierarchical, which means the output of a submodel can be used as the input for another submodel.

Several graphical user interfaces for SHARPE are now available but the original version used textual, line-oriented ASCII code. An interactive or batch oriented input is possible. The syntax for both cases is the same. The possibilities that SHARPE offers depend on the chosen model type. If the user wishes to analyze a product-form queueing network, the commands for the performance measures such as throughput, utilization, mean response time at a station, and mean queue length are provided by SHARPE. MVA is used to solve product-form queueing networks. For a continuous-time Markov chain, reward rates and initial state probability vector can be specified. Steady-state, transient, and cumulative transient measures can be computed. The tool offers two steady-state solution methods. At first it starts with the SOR method and after a certain period of time, it prints the number of iteration steps and the tolerance if no solution is found. In this case the user is asked whether he or she wants to go on with the SOR method or switch to Gauss Seidel. Three different transient CTMC solution methods are available.

It is not possible in this context to describe the full power of SHARPE. Instead of describing the syntax and semantics in detail, we give examples that use the most important model types to show the reader that SHARPE is very powerful and also easy to use. SHARPE has been installed at more than 220 sites. For more detailed information the reader can consult [STP96].

12.4.1 Central-Server Queueing Network

As the first example we consider a central-server queueing network (Fig. 12.20) consisting of a CPU and two disks with the following input parameters:

$$\mu_0 = 1000/20, \qquad \mu_1 = 1000/30, \qquad \mu_2 = 1000/42.9,$$
$$p_{11} = 0.1, \qquad p_{12} = 0.667, \qquad p_{13} = 0.233,$$
$$p_{21} = p_{31} = 1.$$

At each node the scheduling discipline is FCFS and the service times are exponentially distributed. Figure 12.21 shows a SHARPE input file for this

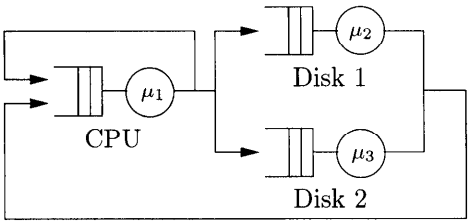


Fig. 12.20 A central-server model.

```
1   * central-server model
2
3   pfqn csm(jobs)
4   cpu disk1 p12
5   cpu disk2 p13
6   disk1 cpu 1
7   disk2 cpu 1
8   end
9   * servers
10  cpu fcfs 1000/20
11  disk1 fcfs 1000/30
12  disk2 fcfs 1000/42.918
13  end
14  * number of jobs
15  chain1 jobs
16  end
17
18  bind
19  p12 0.667
20  p13 0.223
21  end
22
23  loop i,2,10,2
24  expr tput(csm,cpu;i)
25  expr util(csm,cpu;i)
26  expr qlength(csm,cpu;i)
27  expr rtime(csm,cpu;i)
28  end
29
30  end
```

Fig. 12.21 SHARPE Input for central-server network.

model. Lines with comments begin with an asterisk. The description of the SHARPE specification is given as follows:

Line 3: Specifies the model type `pfqn` (= product-form queueing network), model named `csm`, and model parameter jobs.

Line 4-8: Specification of the routing probabilities.

Line 9-13: Specification of the nodes and the service parameters of the nodes (name, queueing discipline, service rate).

Line 15-16: Gives the number of jobs in the network.

Line 18-21: The routing probabilities are bound to specific values.

Line 23-28: Requests the computation and printing of the throughput, the utilization, the mean queue length, and the mean response time at the cpu for the number of jobs: 2, 4, 6, 8, and 10.

The output produced by SHARPE in this case is shown in Fig. 12.22. We see that the throughput grows from 29 to 45, the utilization from 0.59 to 0.90, the mean queue length from 0.82 to 4.60, and the mean response time from 0.03 to 0.10, as the number of jobs increases from 2 to 10.

```
i=2.000000                          i=8.000000
tput(csm,cpu;i):  2.9406e+01        tput(csm,cpu;i):  4.3753e+01
util(csm,cpu;i):  5.8811e-01        util(csm,cpu;i):  8.7506e-01
qlength(csm,cpu;i):  8.2331e-01     qlength(csm,cpu;i):  3.6209e+00
rtime(csm,cpu;i):  2.7998e-02       rtime(csm,cpu;i):  8.2758e-02

i=4.000000                          i=10.000000
tput(csm,cpu;i):  3.7976e+01        tput(csm,cpu;i):  4.4992e+01
util(csm,cpu;i):  7.5952e-0         util(csm,cpu;i):  8.9983e-01
qlength(csm,cpu;i):  1.7202e+00     qlength(csm,cpu;i):  4.5955e+00
rtime(csm,cpu;i):  4.5298e-02       rtime(csm,cpu;i):  1.0214e-01

i=6.000000
tput(csm,cpu;i):  4.1733e+01
util(csm,cpu;i):  8.3465e-01
qlength(csm,cpu;i):  2.6591e+00
rtime(csm,cpu;i):  6.3717e-02
```

Fig. 12.22 SHARPE output for the central-server model.

12.4.2 M/M/m/K System

As a simple example for a CTMC model, we consider an M/M/5/100 queueing system. The state diagram for this CTMC is given in Fig. 12.23. It is of birth-

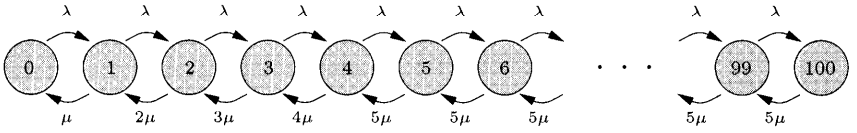


Fig. 12.23 Markov chain for the M/M/5/100 system.

death type and irreducible. Each state name gives the number of jobs in the system. The SHARPE input file is shown in Fig. 12.24.

In the first two lines the values of the arrival and service rates are bound. Then the model type (markov = CTMC) and the name (mm5) of the model are given. Subsequent lines specify the transitions between the state together with their transition rates. Then five variables are defined. `Pidle` is the steady-state probability of being in state 0. This is the probability that there are no jobs being served, i.e., that the station is idle. `Pfull` is the steady-state probability of being in state 100. This is the probability that the queue is full. `Lreject` is the rate at which jobs are rejected. `Mqueue` is the mean number of jobs in the system. This can be calculated by the built-in function sum. `Mresp` is the mean response time of accepted jobs computed using Little's theorem as `Mqueue`/(λ-`Lreject`), `expr expression` prints the value of the expression in the output file (Fig. 12.25).

Note that a recent extension to SHARPE includes a loop specification in the definition of a CTMC that can be used to make concise specification of a

```
bind LAM 5.3          100 99 5*MU          var Pidle prob(mm5,0)
bind MU 1.2           99 98 5*MU           var Pfull prob(mm5,100)
                      .                    var Lreject LAM *Pfull
markov mm5            .                    var Mqueue sum(i,0,100,i*prob(mm5,$i))
0 1 LAM               10 9 5*MU            var Mresp Mqueue/(LAM-Lreject)
1 2 LAM               9 8 5*MU             expr Pidle
2 3 LAM               8 7 5*MU             expr Pfull
3 4 LAM               7 6 5*MU             expr Lreject
4 5 LAM               6 5 5*MU             expr Mqueue
5 6 LAM               5 4 5*MU             expr Mresp
6 7 LAM               4 3 5*MU             end
7 8 LAM               3 2 5*MU
9 10 LAM              2 1 5*MU
.                     1 0 5*MU
.                     end
99 100 LAM
```

Fig. 12.24 SHARPE input file for the M/M/5/100 queue.

structured CTMC such as the one just described. For an example of the use of this feature see Section 13.3.2.

```
Pidle:   6.0678e-03
Pfull:   1.0244e-07
Lreject:  5.4293e-07
Mqueue:  9.7471e+00
Mresp:   1.8391e+00
```

Fig. 12.25 Output file for the M/M/5/100 system.

12.4.3 M/M/1/K System with Server Failure and Repair

Now we extend the M/M/1/K queueing model by allowing for the possibility that a server could fail and could be repaired. Let the job arrival rate be λ and the job service rate be μ. The processor failure rate is γ and the processor repair rate is τ. This system can be modeled using an irreducible CTMC, as shown in Fig. 12.26 for the case where $m = 1$ (one server) and $K = 10$ (the maximum number of jobs in the server and queue is 10). Each state is named with a two-digit number ij where $i \in \{0, 1, \ldots, 9, a\}$ is the number of jobs in the system (a is 10 in hexadecimal) and $j \in \{0, 1\}$ denotes the number of operational processors. SHARPE input and output files for this example are shown in Figs. 12.27 and 12.28. The probability that the system is idle is given by $\texttt{prob}(mm1k, 00) + \texttt{prob}(mm1k, 01)$, and the rate at which jobs are rejected because the system is full is given by λ $(\texttt{prob}(mm1k, a0) + \texttt{prob}(mm1k, a1))$.

The generalized stochastic Petri net (GSPN) in Fig. 12.29 is equivalent to the CTMC model and will let us vary the value of K without changing the model structure. We use this example to show how to handle GSPNs

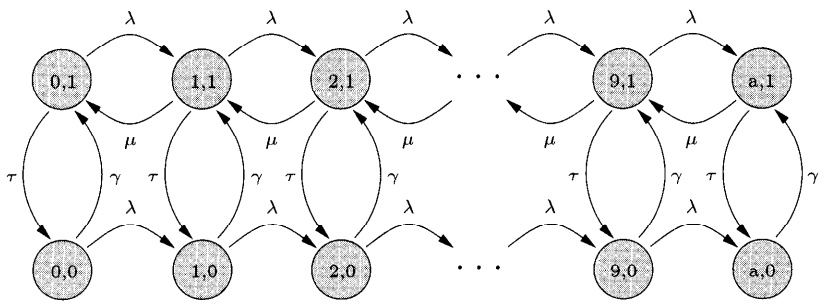


Fig. 12.26 CTMC for an M/M/1/10 system with server failure and repair.

```
markov mm1k          11 10 GAM        30 40 LAM
01 11 LAM            10 11 TAU        40 50 LAM
11 01 MU             21 10 GAM        60 70 LAM
11 21 LAM            20 21 TAU        70 80 LAM
21 11 MU             31 30 GAM        80 90 LAM
21 31 LAM            30 31 TAU        90 a0 LAM
31 21 MU             41 40 GAM        end
31 41 LAM            40 41 TAU
41 31 MU             51 50 GAM        bind
41 51 LAM            50 51 TAU        LAM 1
51 41 MU             61 60 GAM        MU 2
51 61 LAM            60 61 TAU        GAM 0.0001
61 51 MU             71 70 GAM        TAU 0.1
61 71 LAM            70 71 TAU        end
71 61 MU             81 80 GAM        var Pidle prob(mm1k,00)+prob(mm1k,01)
71 81 LAM            80 81 TAU        var Pfull prob(mm1k,a0)+prob(mm1k,a1)
81 71 MU             91 90 GAM        var Lreject LAM*PFULL
81 91 LAM            90 91 TAU        expr Pidle
91 81 MU             a1 a0 GAM        expr Lreject
91 a1 LAM            a0 a1 TAU        end
a1 91 MU             00 10 LAM
01 00 GAM            10 20 LAM
00 01 TAU            20 30 LAM
```

Fig. 12.27 SHARPE input for the M/M/1/10 system with server failure and repair.

using SHARPE. The loop in the upper part of the GSPN is a representation of an M/M/1/K queue. The lower loop models a server that can fail and be repaired. The inhibitor arc from place `server-down` to transition `service` shows that customers cannot be served while the server is not functioning. The number within each place is the initial number of tokens in the place. All of the transitions are timed, and their firing rates are shown below the transitions. The input file for this model is shown in Fig. 12.30, and the description of this file follows:

Line 1-7: The input parameters are bound to specific values.

Line 9: Model type and name of the model.

Line 11-15: Places and initial numbers of the tokens.

```
Pidle:  4.9953e-01
Lreject:  9.6239e-04
```

Fig. 12.28 SHARPE output for the M/M/1/10 system with server failure and repair.

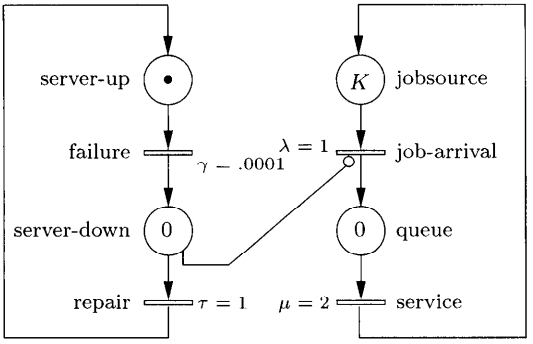


Fig. 12.29 GSPN model for queue with server failure and repair.

Line 17-21: Timed transitions.

Line 23-27: Arcs from places to transitions and arc multiplicities.

Line 29-33: Arcs from transitions to places and arc multiplicities.

Line 35-36: Inhibitor arcs and their multiplicities.

Line 38: Define a variable for the probability that the server is idle.

Line 40: Define a variable for the probability that a job is rejected at the server.

Line 42: Define a variable for the rejection rate.

Line 44: Define a variable for the average queue length at server.

Line 46: Define a variable for the throughput of server.

Line 48: Define a variable for the utilization of server.

Line 50-54: Values requested to be printed (Fig. 12.31).

This GSPN is irreducible. For GSPNs that are non-irreducible, SHARPE can compute the expected number of tokens in a place at a particular time t, the probability that a place is empty at time t, the throughput and utilization of a transition at time t, the time-averaged number of tokens in a place during the interval $(0,t)$, and the time-averaged throughput of a transition during $(0,t)$. See [STP96] for the syntax for these functions. Other model types

```
1  bind                                  28  * arcs, transitions-places
2  lambda 1.0                            29  job-arrival queue 1
3  mu 2.0                                30  service jobsource 1
4  gamma 0.0001                          31  failure serverdown 1
5  tau 0.1                               32  repair serverup 1
6  K 10                                  33  end
7  end                                   34  * inhibitor arcs
8                                        35  serverdown service 1
9  gspn mm1k-fail                        36  end
10 * places                              37
11 jobsource K                           38  var Pidle prempty(mm1k-fail,queue)
12 queue 0                               39
13 serverup 1                            40  var Preject prempty(mm1k-fail,jobsource)
14 serverdown 0                          41
15 end                                   42  var Lreject prempty(mm1k-fail,jobsource)
16 * transitions                         43
17 job-arrival ind lambda                44  var avquelength etok(mm1k-fail,queue)
18 service ind mu                        45
19 failure ind gamma                     46  var thruput tput(mm1k-fail,service)
20 repair ind tau                        47
21 end                                   48  var utilization uitl(mm1k-fail,service)
22 * arcs, places-transitions            49
23 jobsource job-arrival 1               50  expr Pidle
24 queue service 1                       51  expr Lreject, Preject
25 serverup failure 1                    52  expr avquelength
26 serverdown repair 1                   53  expr thruput, utilization
27 end                                   54  end
```

Fig. 12.30 Input for GSPN model of the system with server failure and repair.

```
Pidle:  4.9953e-01
Lreject:  9.6239e-04
Preject:  9.6239e-04
avquelength: 1.0028e+00
thruput:  9.9904e-01
utilization:  4.9952e-01
```

Fig. 12.31 Output for the GSPN model.

provided by SHARPE are multiple chain product-form queueing networks, semi-Markov chains, reliability block diagrams, fault trees, reliability graphs, and series-parallel graphs. For details see [STP96].

12.5 CHARACTERISTICS OF SOME TOOLS

In Table 12.1 other important tools are listed together with their main features. There are pure queueing network tools such as QNAP2 or PEPSY, pure Petri net tools such as SPNP or PANDA, and tools that have a more general input and are therefore more flexible such as SHARPE or MOSES. Many of them provide a GUI and/or a textual input language. The GUI is very convenient for getting accustomed to the tools, but the experienced user often prefers the use of a textual input language.

Table 12.1 Performance Evaluation Tools

Tool	Location	Reference	Modeltype				Solution Method				GUI	Language	Hierarchical
			QN	SPN	other	DES	pqfn	npqfn	m-tr	m-ss			
RESQ	IBM	[SaMa85]	*	-	-	*	*	-	-	-	*	(*)	-
QNAP2	SIMULOG	[VePo85]	*	-	-	*	*	-	-	*	*	*	-
MAOS	TU München	[FeJo90]	*	-	-	*	*	-	-	-	-	ILMAOS	*
PEPSY	Univ. Erlangen	[Kirs94]	*	-	-	*	*	*	-	*	XPEPSY	(Menue)	-
HIT	Univ. Dortmund	[BMW89]	*	-	-	*	*	-	-	*	HITGRAPHIC	HISLANG	*
SPNP	Duke Univ.	[CFMT94]	-	*	-	*	-	-	*	*	iSPN	CSPL	-
GreatSPN	Univ. Torino	[ABC+95]	-	*	-	*	-	-	*	*	*	-	-
TIMENET	TU Berlin	[GKZH94]	-	*	-	*	-	-	*	*	*	(*)	-
DSPNexpress	TU Berlin	[Lind94]	-	*	-	-	-	-	*	*	*	-	-
Ultra SAN	Univ. Illinois	[SOQW95]	-	*	-	*	-	-	*	*	*	-	-
PANDA	Univ. Erlangen	[AlDa97]	-	*	-	*	-	-	*	*	*	-	*
QPN	Univ. Dortmund	[BaKe94]	*	*	-	-	-	-	-	*	*	APNN	*
SHARPE	Duke Univ.	[STP96]	*	*	*	-	*	*	*	*	GSHARPE	*	*
MOSES	Univ. Erlangen	[BoHe96]	*	*	*	(*)	-	-	*	*	(*)	MOSEL	*
MARCA	NC State Univ.	[Stew90]	*	-	*	-	-	-	*	*	XMARCA	*	-
MACOM	Univ. Dortmund	[ScMü90]	*	-	-	-	-	-	-	*	*	(USENUM)	-
DNAmaca	Univ. Cape Town	[KnKr96]	*	*	*	-	-	-	-	*	-	C++-constructs	-

Notes: m-tr: CTMC solver transient, m-ss: CTMC solver steady-state

13 Applications

This chapter considers several large applications. The set of applications is organized into three sections. In Section 13.1, we present case studies of queueing network applications. In Section 13.2 we present case studies of Markov chains and stochastic Petri nets. In Section 13.3, case studies of hierarchical models are presented.

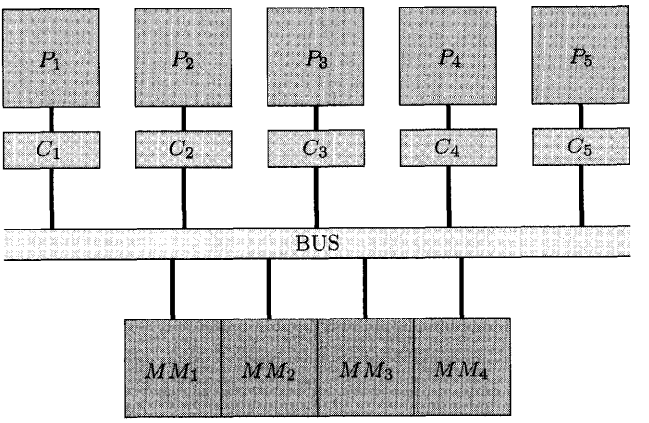
13.1 CASE STUDIES OF QUEUEING NETWORKS

Five different case studies are presented in this section. These range from multiprocessor system model, several networking applications one operating system model and a flexible production system model.

13.1.1 Multiprocessor Systems

Models of tightly coupled multiprocessor systems will be discussed first followed by models of loosely coupled systems.

13.1.1.1 Tightly Coupled Systems Consider a tightly coupled multiprocessor system with caches at each processor and a common memory, connected to the processors via a common bus (see Fig. 13.1). The system consists of m processors and a common memory with n memory modules. A processor sends a request via the common bus to one of the n memory modules when a cache miss occurs, whereupon the requested data is loaded into the cache via



P_n	Processor n	$n = 1, \ldots, 5$
C_n	Cache n	$n = 1, \ldots, 5$
MM_n	Memory Module n	$n = 1, \ldots, 4$

Fig. 13.1 A multiprocessor system with caches, common memory, and common bus.

the common bus. Figure 13.2 shows a product-form queueing network model of such a multiprocessor system.

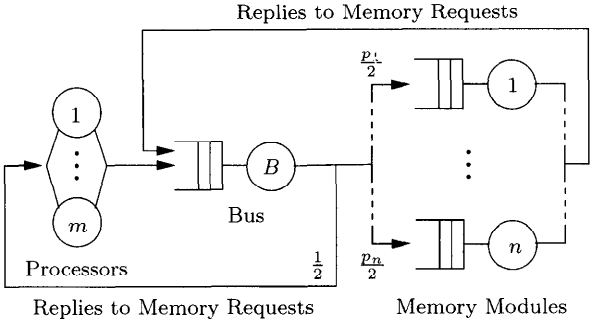


Fig. 13.2 Queueing network model of the multiprocessor system shown in Fig. 13.1.

The m processors are modeled by an IS node and the bus and the memory modules by single server nodes. The number of requests in the system is m since a processor is either working or waiting until a memory request is finished. Using this model we can calculate the utilization of the bus or the mean response time of a memory request from a processor. The mean time between two cache misses is modeled by the mean thinking time of the IS node. Other parameters that we need are the mean bus service time, the mean memory request time, and, finally, p_i, the probability of a request to memory module i. In the absence of additional information, we assume $p_i = 1/n$

$(i = 1, 2, \ldots, n)$. Note that for each cache miss, there are two service requests to the bus. It is for this reason that with probability 0.5, a completed bus request returns to the processor station. Similarly, the probability of visiting memory module i subsequent to the completion of a bus request is $p_i/2$.

We assume that the service times for the two types of bus requests have the same mean. This assumption can be easily relaxed. If we have explicit values for these parameters, then we can calculate interesting performance measures such as bus utilization and mean response time as functions of the number of processors. Another interesting measure is the increase in the mean response time assuming a fixed number of processors and memory modules, while changing the mean bus service time and the mean time between two cache misses. In Fig. 13.3 the mean response time is shown as a function of the mean bus service time and the mean time between two cache misses [Bolc91].

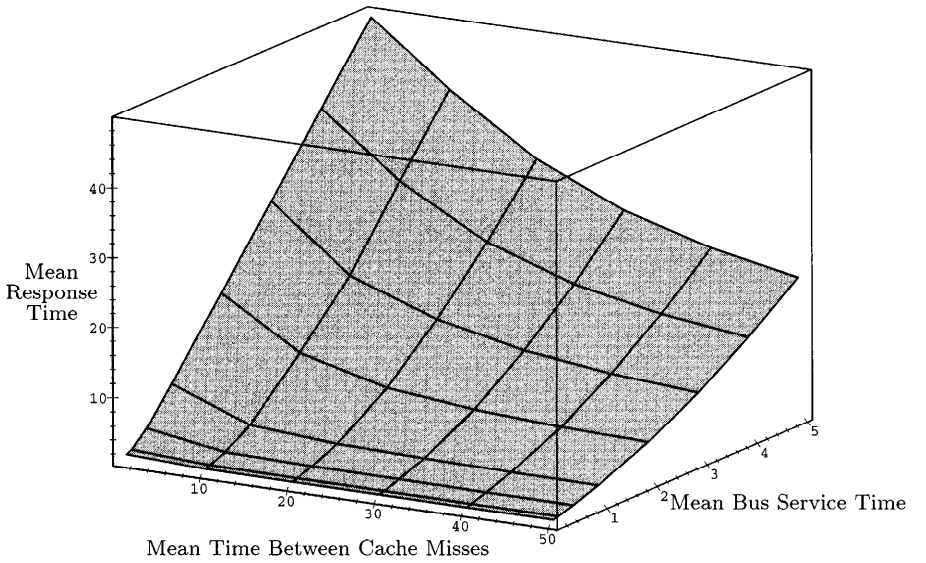


Fig. 13.3 Mean response time as a function of the bus service time, and the mean time between cache misses ($m = 5$, $n = 4$).

We can see that there is a wide area where the increase in the mean response time is tolerable (15%) compared to the case with no waiting time at the bus or a memory queue (this is the minimum value of the mean response time). On the other hand, there is also a wide area with an intolerable increase in the mean response time. Thus the analysis of such a queueing network model can help the system designer to choose parameter values in a tolerable area.

Problem 13.1 Verify the results shown in Fig. 13.3 by hand computation and using any software package available (e.g., SHARPE or PEPSY).

Problem 13.2 Improve the model described in Section 13.1.1.1 by considering different mean service times for bus requests from processor to memory and from memory to cache (Hint: Use a multiclass queueing network model).

13.1.1.2 Loosely Coupled Systems In a loosely coupled multiprocessor system, the processors have only local memory and local I/O devices, and communicate via an interconnection network by exchanging messages. A simple queueing network model of such a system is shown in Fig. 13.4. The mean

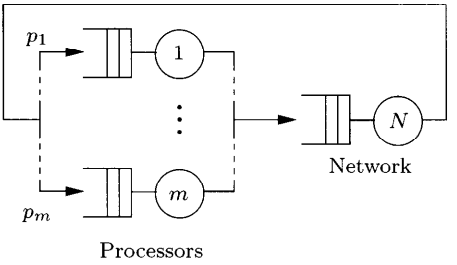


Fig. 13.4 A simple queueing network model of a loosely coupled system.

service time at the processors is the mean time between the messages, and the mean service time at the network node N is the mean message delay at the network. The routing probability p_i is the probability that a message is directed to processor i.

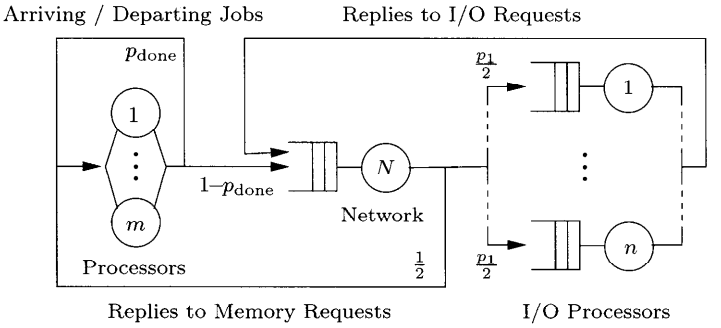


Fig. 13.5 A complex queueing network model of a loosely coupled system.

A more complex and detailed model of a loosely coupled multiprocessor system is shown in Fig. 13.5 [MAD94]. Here we assume that we have n separate I/O processors and m "computing" processors. The computing processors send I/O requests via the network N to the I/O processors and get the replies to these requests also via the network N. We assume that a job that begins at a computing processor is also completed at this processor and that the processors are not multiprogrammed and are heavily loaded (for each job that leaves the system after being processed by a processor, a new job arrives immediate-

ly at the processor). This system can be modeled by a closed product-form queueing network with m job classes (one for each computing processor) with the population of each class equal to 1 [MAD94].

As an example we consider a loosely coupled multiprocessor system with eight computing processors and $n = 2, 3, 4$ I/O processors. The mean computing processor service time is 30 msec, the mean I/O time is 50 msec, and the mean message delay at the network is 1 msec. The probability that a job leaves the system after it has been processed at a computing processor is $p_{\text{done}} = 0.05$. We assume that $p_i = 1/n$ for all i. Some interesting results from this model are listed in Table 13.1.

Table 13.1 Performance measures for the loosely coupled multiprocessor system for different numbers of I/O processors

Number of I/O Processors	2	3	4
mean response time	4.15 sec	3.18 sec	2.719 sec
throughput	$1.93\,\text{sec}^{-1}$	$2.51\,\text{sec}^{-1}$	$2.944\,\text{sec}^{-1}$
$\rho_{\text{computingprocessor}}$	0.145	0.189	0.220
ρ_{network}	0.070	0.090	0.106
$\rho_{\text{I/Oprocessor}}$	0.867	0.754	0.662

Problem 13.3 Verify the results shown in Table 13.1 by hand computation and using any software package available (e.g., PEPSY or SHARPE).

Problem 13.4 Extend the complex model described in Section 13.1.1.2 so as to allow distinct mean network service times for request from a computing processor to I/O processor and vice versa.

13.1.2 Client Server Systems

A client server system consists of client and server processes and some method of interprocess communication. Usually the client and the server processes are executing on different machines and are connected by a LAN. The client interacts with the user, generates requests for a server, transmits the request to the server, receives the results from the server, and presents the results to the user. The server responds to requests from the clients and controls access to resources such as file systems, databases, wide area networks, or printers. As an example we consider a client server system with a fixed number m of client workstations that are connected by an Ethernet network to a database server. The server consists of a single disk (node number 4) and a single CPU (node number 3). This leads to a closed product-form queueing network model shown in Fig. 13.6.

The client workstations are modeled as an IS node (node number 1) [MAD94], and the number of jobs in the closed system is equal to the number of workstations m. The Ethernet network (carrier sense multiple access with

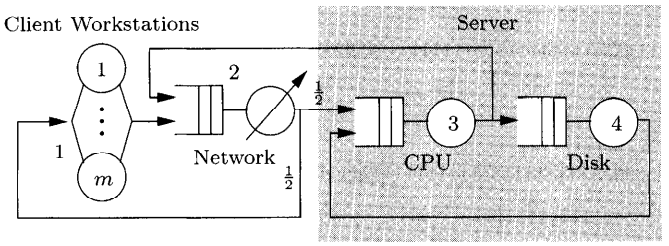


Fig. 13.6 Closed queueing network model of a client server system.

collision detection, or CSMA/CD, network) can be modeled as a server (node number 2) with the load-dependent service rate [LZGS84, MAD94, HLM96]:

$$\mu_{\text{net}}(k) = \begin{cases} \left(\frac{1}{N_p} \cdot \frac{\overline{L_p}}{B} + S \cdot C(1)\right)^{-1}, & k = 1, \\ \left(\frac{1}{N_p} \cdot \frac{\overline{L_p}}{B} + S \cdot C(k+1)\right)^{-1}, & k > 1, \end{cases} \tag{13.1}$$

where $C(k) = \big(1 - A(k)\big)/A(k)$ is the average number of collisions per request and $A(k) = (1 - 1/k)^{k-1}$ is the probability of a successful transmission and k the number of workstations that desire the use of the network.

Other parameters are described in Table 13.2 and shown in Fig. 13.6. We

Table 13.2 Parameters for the client server example

Parameter	Description
N_p	Average number of packets generated per request
B	Network bandwidth in bits per second
S	Slot duration (i.e., time for collision detection)
$\overline{L_p}$	Average packet length in bits

compute the throughput λ and other interesting performance measures using the load-dependent MVA (see Section 8.2). As an example we use the parameters from Table 13.3 and determine the throughput as a function of the number of client workstations m (see Fig. 13.7)

Problem 13.5 Verify the results shown in Fig. 13.7 by hand computation and using an available modeling package such as SHARPE or PEPSY.

13.1.3 Communication Systems

13.1.3.1 Description of the System As an example of a more complex queueing network and the performance evaluation of communication systems, we consider the LAN of a medium size enterprise [Ehre96].

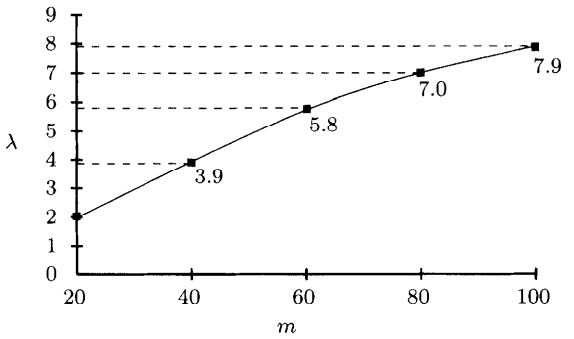


Fig. 13.7 Throughput as a function of the number of workstations.

Table 13.3 Numerical parameter values for the client-server example

$N_p = 7$		$\mu_1 = \mu_{\text{CL}} = 0.1/\text{sec}$	
$B = 10\,\text{Mb/sec}$		$\mu_2 = \mu_{\text{Net}}(k)$	
$S = 51.2\,\mu\text{sec}$		$\mu_3 = \mu_{\text{CPU}} = 16.7/\text{sec}$	
$\overline{L}_p = 1518\,\text{bits}$		$\mu_4 = \mu_{\text{Disk}} = 18.5/\text{sec}$	
$p_{12} = 1$	$p_{21} = 0.5$	$p_{32} = 0.5$	$p_{43} = 1$
	$p_{23} = 0.5$	$p_{34} = 0.5$	

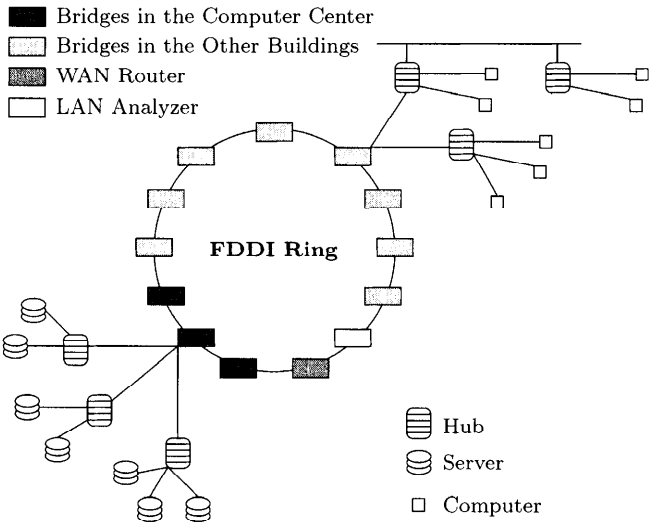


Fig. 13.8 The FDDI backbone with 13 stations.

The LAN connects several buildings that are close together and it is divided into several network sections. In each network section the devices of a building are connected to each other. Three other sections are located in one building and are used to connect the servers to the LAN. As a backbone for the sections, a fiber distributed data interface (FDDI) ring is used. Eleven bridges for the sections, a WAN router, and a LAN analyzer that measures the utilization of the LAN constitute the stations of this ring. Figure 13.8 shows the FDDI ring with the stations schematically, wherein the structure of two stations is shown in more detail.

The typical structure of a section in which all computers within a building are connected is shown in Fig. 13.9. The building has four floors. The computers on the second and third floor are connected to different segments, and the computers on the fourth and fifth floor are connected to the same segment. These three segments are connected to each other and to the FDDI ring via a multiport bridge. The computers are connected to a stack of four hubs that

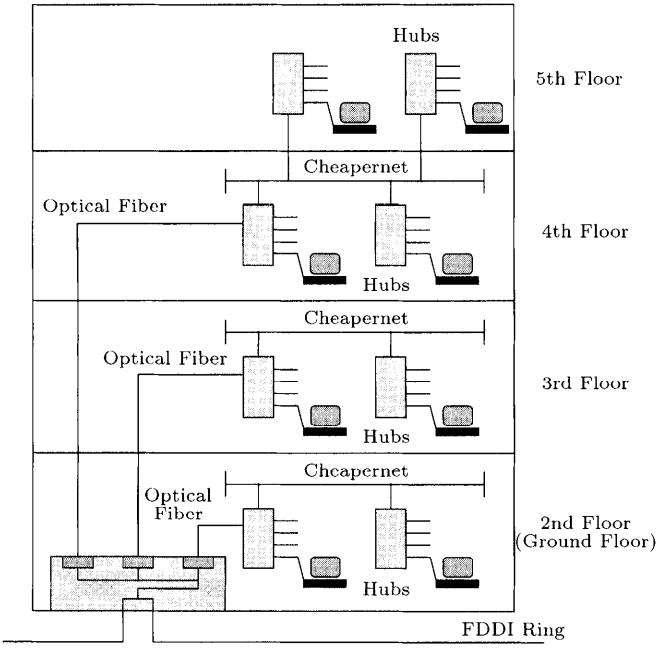


Fig. 13.9 A section of a communication system with three segments.

are connected to each other via Cheapernet, and there is a connection of the hubs to the multiport bridge via optical fibers. The CSMA/CD principle is used with a transmission rate of 10 Mb/sec which is equal for all segments. Each segment is a different collision domain.

13.1.3.2 Queueing Network Model The nodes of the queueing network model of the LAN are the FDDI ring, the Ethernets that connect the devices in the segment, the computers, the servers in the computer center, and the bridges. The FDDI ring can be considered as a server that serves the stations in the ring (12 stations without the LAN analyzer) in a fixed sequence. A station (or section) sends a request to another station via the FDDI ring, and the reply to the request is sent back to the station again via the FDDI ring. Thus we obtain a closed queueing network model for the LAN as shown in Fig. 13.10 with a simplified representation of the individual sections. The FDDI ring is modeled by a single server node. The LAN analyzer does not influence the other sections and for this reason it does not appear in the model. The WAN router can be modeled as a single server node and the WAN itself as an IS node (see Fig. 13.11). There are arrivals at the WAN router from the WAN

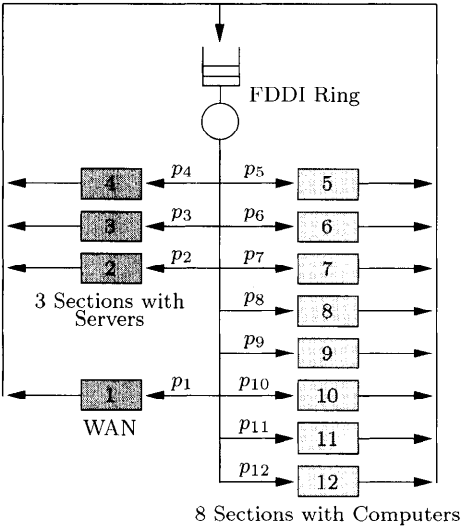


Fig. 13.10 Closed queueing network model of the LAN with a simplified representation of the individual sections.

and from the ring. The WAN router sends requests to the ring and to the WAN as well. A similar situation occurs at all the bridges/routers.

In the sections, the segments are connected to the bridge via Ethernet. The Ethernet segments are modeled by a multiple server with only one queue where the number of servers is the number of segments in the section. Because of the CSMA/CD strategy, the frames are either transferred successfully to the computers of the sections to the bridge or in the case of a collision, sent back to the Ethernet queue with collision probability q_i. Each computer sends requests to a server via the LAN and waits for the reply before it sends another request. Therefore the number of requests in the LAN equals the number of

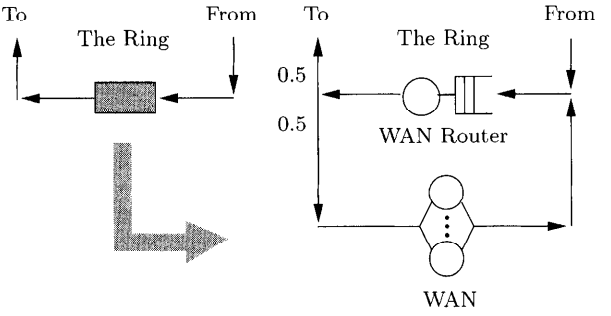


Fig. 13.11 Detailed model of the WAN and the WAN router.

active computers, and the requests do not have to wait at the computers. For this reason the computers at a section can be modeled by an IS node.

A queueing network model of a section of the LAN is shown in Fig. 13.12 and the queueing network model of a computer center section with the servers

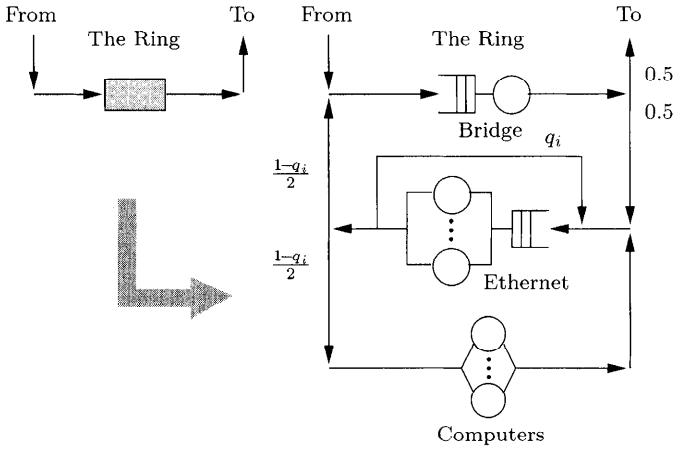


Fig. 13.12 Detailed model of a section of the LAN.

is shown in Fig. 13.13. Servers are modeled by a multiple server node with one queue where the number of servers at the node is equal to the number of real servers. The queue is necessary because there are more computers that can send requests to the servers than the number of available servers. The whole LAN model consists of 1 node for the FDDI ring, 12 nodes for the bridges, 11 nodes for the Ethernet segments, 8 nodes for the computers, 3 nodes for the servers, and 1 node for the WAN. As we will allow non-exponential service time distributions, it is a non-product-form network.

13.1.3.3 Model Parameters As parameters we need the total number of requests, the routing probabilities p_i, the mean service times $1/\mu_i$, and the

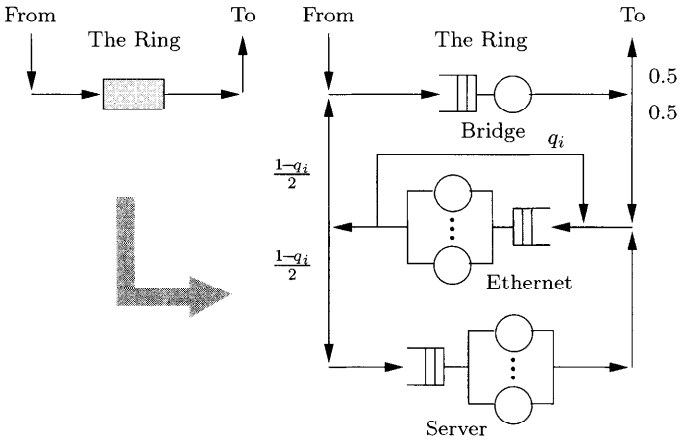


Fig. 13.13 Detailed model of a section with servers.

coefficient of variation c_i of the service times of the nodes. Furthermore, we need the number of service units m_i for the multiple server nodes. The total number of requests in the network is the mean number of active computers $K = 170$ (measured value). We estimate the routing probabilities p_i from the FDDI ring to the bridges by measuring the amount of data transferred from the ring to the bridges, which can be measured at the bridges. This estimate was done based on measurements over a two-day period (see Table 13.4).

Table 13.4 Total data transferred from FDDI ring to the bridges for a two-day period and estimated routing probabilities p_i

Section	1	2	3	4	5	6	7	8	9	10	11	12
Data/Mb	2655	1690	2800	1652	2840	1500	3000	200	1940	1180	4360	4380
p_i/%	9.5	6.2	10	5.9	10.1	5.4	10.7	0.7	6.9	4.2	14.8	15.6

Notes: 1: WAN; 2,3,4: section with servers; 5, ...,12: section with computers.

Table 13.5 Collision probabilities q_i

Section	1	2	3	4	5	6	7	8	9	10	11	12
q_i/%	1	5	1	1	1	1	1	1	1	1	3	2

The other necessary routing probabilities are shown in Figs. 13.11, 13.12, and 13.13 and in Table 13.5. The number of servers m_i of the multiple server nodes is given by the number of servers for the server nodes (Table 13.6) or by the number of segments for the Ethernet node (Table 13.7). To obtain the mean service times of the nodes, we use a LAN analyzer which can measure

Table 13.6 Number of service units m_i in server nodes

Section	2	3	4
m_i	14	23	13

Table 13.7 Number of service units m_i (= number of segments) in Ethernet nodes

Section	2	3	4	5	6	7	8	9	10	11	12
m_i	4	7	2	4	5	3	1	4	2	5	5

the interarrival times and the length of the frames transferred by the FDDI ring. The results of the measurement are shown in Tables 13.8 and 13.9. From

Table 13.8 Empirical pmf of the interarrival times of the frames.

Interarrival Time (μs)	%
≤ 5	3.0
5–20	0.9
20–82	15.3
82–328	42.7
329–1300	27.1
1300–5200	1.0

Table 13.9 Empirical pmf of the frame length L

Length (Bytes)	%
≤ 32	11.1
32–63	10.0
64–95	33.0
96–127	7.9
128–191	6.8
192–511	4.1
512–1023	5.8
1024–1526	21.3

these tables we obtain the mean values and the square coefficients of variation (Table 13.10) of the interarrival times and frame lengths. Given the measured throughput $\lambda = 1/346\,\mu\text{sec} = 2890$ per sec, and the routing probabilities (see Table 13.4), the traffic equations are solved to produce the arrival rates of the individual sections (see Table 13.11). Normally in a closed network, relative throughputs are computed from routing probabilities or visit ratios. However, in this case we have measured throughput at one node and hence the relative throughputs at each node are also the actual throughputs.

The mean service time of the FDDI ring is given by the sum of the mean transfer time of a frame and the mean waiting time until the token arrives at a station [MSW88]. We assume *one-limited service*, i.e., a station transmits at most one frame when the token arrives. An upper limit of the service time of the ring is the token rotation time. The utilization $\rho_i = \lambda_i/\mu$ is the probability that a station wishes to transmit a frame, and $1-\rho_i$ is the probability that the token is transferred to the next station without transmitting a frame. In this case the transfer of the token can be considered as the transfer of a frame with

Table 13.10 Characteristic values of the interarrival times and the frame length

	Interarrival Times of the Frames	Length of the Frames
Mean value	346 μsec	382.5 Byte
Squared coefficient of variation	1.48	1.67

Table 13.11 Arrival rates of frames at the stations

Station	1	2	3	4	5	6	7	8	9	10	11	12
λ_i	275	179	289	171	292	156	309	20	199	121	428	451

length 0. Accordingly, the mean token rotation time $\overline{T}_r$ can be calculated as follows:

$$\overline{T}_r = U + R^{-1} \cdot \overline{L} \cdot \sum \rho_i. \tag{13.2}$$

Here U denotes the free token rotation time (22μsec), R denotes the transfer rate (100 Mb/sec), and $\overline{L}$ denotes the mean frame length (see Table 13.10).

With:

$$\sum \rho_i = \frac{\sum \lambda_i}{\mu} = \frac{\lambda}{\mu}, \tag{13.3}$$

the approximation:

$$\overline{T}_r \approx \frac{1}{\mu}$$

and Eq. (13.2), it follows that the service rate of the FDDI ring is given by:

$$\mu = \frac{R - \lambda \overline{L}}{U \cdot R}. \tag{13.4}$$

With $\lambda = 2890$/sec, $\overline{L} = 382.5$ Bytes (see Table 13.10), we obtain the mean service rate:

$$\mu = 41435/\,\text{sec},$$

and the mean service time at the FDDI ring:

$$\overline{T}_r \approx \frac{1}{\mu} = 24\ \mu\text{sec}.$$

The variance of the token rotation time T_r is given by [MSW88]:

$$\sigma^2_{T_r} = R^{-2} \left(\rho \cdot \text{var}(L) + \rho \cdot \left(1 - \rho \sum p_i^2 \right) \overline{L}^2 \right), \tag{13.5}$$

where $\sigma_{T_r}^2$ and $\overline{L}^2$ can be calculated using the values from Table 13.9, the values of p_i are given in Table 13.4, and $\rho = \lambda/\mu = 0.070$ is given by the values of λ and μ. Then we obtain the squared coefficient of variation:

$$c_{T_r}^2 = 0.3.$$

The service time at each bridge is deterministic with the forwarding rate of 10,000 frames/sec. And, of course, the coefficient of variation is 0 in this case.

The service time of the Ethernet is given by the sum of the transfer time T_t and the delay time T_d of a frame. To obtain the transfer time T_t we need the mean frame length, which we get from Table 13.9 given that the minimum frame size is 72 bytes in the CSMA/CD case. Thus, 54.1% of the frames have a size between 72 and 95 bytes and it follows:

$$\overline{L}_{eth} = 395 \text{ bytes} = 3160 \text{ bit}, \quad c_{L_{eth}}^2 = 1.51.$$

Given a transfer rate of 10 Mb/sec (see Table 13.12), we finally have:

$$\overline{T}_t = \frac{3160 \text{ bit}}{10 \text{ Mb/sec}} = 316 \,\mu\text{sec} \quad c_{T_t}^2 = 1.51.$$

The mean delay time $\overline{T}_d$ of the Ethernet can be calculated using Fig. 13.9 and Table 13.12:

$$\begin{aligned}\overline{T}_d &= 0.011 \text{ km} \cdot 5 \,\mu\text{sec/km} + 0.005 \text{ km} \cdot 4.3 \,\mu\text{sec/km} + 0.05 \text{ km} \cdot 4.8 \,\mu\text{sec/km} \\ &= 0.3 \,\mu\text{sec}.\end{aligned}$$

Table 13.12 Characteristics of the optical fiber, the Cheapernet, and the twisted pair

	Transfer Rate	Mean Length	Signal Time
Optical fiber	10 Mb/sec	11 m	5 μsec/km
Cheapernet	10 Mb/sec	5 m	4.3 μsec/km
Twisted pair	10 Mb/s	50 m	4.8 μsec/km

In this case, $\overline{T}_d$ can be neglected compared to $\overline{T}_t$ and we have:

$$\frac{1}{\mu_{eth}} = 316 \,\mu\,\text{sec}\,.$$

To obtain the service rates of the IS node (WAN or computers in a section) we use the formula:

$$\overline{K}_i = \frac{\lambda_i}{\mu_i},$$

and have:

$$\mu_i = \frac{\lambda_i}{\overline{K}_i}. \tag{13.6}$$

The values of the λ_i are listed in Table 13.11. In order to get the $\overline{K}_i$ we need the mean number of active computers. This value cannot be measured directly but we have the mean total number of active computers which is 170. To get an approximation, for the $\overline{K}_i$, we divide 170 by the number of sections, which is 12 (see Table 13.4 or Fig. 13.17). This yields:

$$\overline{K}_i \approx \frac{170}{12} = 14.17.$$

For sections with servers (multiple server node) we use:

$$\rho_i = \frac{\lambda_i}{m_i \mu_i},$$

and obtain:

$$\mu_i = \frac{\lambda_i}{m_i \rho_i}, \tag{13.7}$$

and can calculate the service rates using the utilization ρ_i of the servers, which was measured approximately as 90%, and the number of servers m_i from Table 13.6. The values of the service rates are listed in Table 13.13.

Table 13.13 Service rates of the computers, servers, and the WAN

Station	1	2	3	4	5	6	7	8	9	10	11	12
μ_i	19.4	14.2	14.0	14.6	20.6	11.0	21.8	14.1	14.8	8.5	30.2	31.8

Now the closed non-product-form queueing network model of the considered communication system is completely defined. We solve it using Marie's method (see Section 10.1.4.2).

13.1.3.4 Results The closed queueing network model from Section 13.1.3.2 together with the model parameters (as derived in Section 13.1.3.3) is solved using the queueing network package PEPSY (see Chapter 12). The PEPSY input file for this example is given in Fig. 13.14. In PEPSY, either routing probabilities p_{ij} or visit ratios e_i can be used in the input file. Here we use the visit ratios, which are calculated from the originally given routing probabilities using Eq. (7.5). We have a closed queueing network, therefore we have the total number of jobs $K = 170$ as input. From the input file, PEPSY produces an output file, shown in Fig 13.15, with all performance measures. Since we have a non-product-form network, Marie's method was used for the analysis.

In the output file we see that the computed utilization of the servers is $\rho_{\text{serv}} = 0.9$, which matches with the estimated value from actual measurement, and that the utilization of the ring is $\rho_r = 0.07$, which also matches with the measured value. The mean queue length of the ring is negligible and queue

```
#
# filename  e_lan170
#

NUMBER NODES: 36
NUMBER CLASSES: 1

NODE SPECIFICATION

  node  |  name          |     type       |  node  |  name          |     type
--------+----------------+----------------+--------+----------------+----------------
    1   |  ring          |  -/G/1-FCFS    |   19   |  pc-b7         |  -/G/O-IS
    2   |  bridge-cc2    |  -/G/1-FCFS    |   20   |  bridge-b8     |  -/G/1-FCFS
    3   |  eth-cc2       |  -/G/4-FCFS    |   21   |  eth-b8        |  -/G/1-FCFS
    4   |  serv-cc2      |  -/G/14-FCFS   |   22   |  pc-b8         |  -/G/O-IS
    5   |  bridge-cc3    |  -/G/1-FCFS    |   23   |  bridge-b9     |  -/G/1-FCFS
    6   |  eth-cc3       |  -/G/7-FCFS    |   24   |  eth-b9        |  -/G/4-FCFS
    7   |  serv-cc3      |  -/G/23-FCFS   |   25   |  pc-b9         |  -/G/O-IS
    8   |  bridge-cc4    |  -/G/1-FCFS    |   26   |  bridge-b10    |  -/G/1-FCFS
    9   |  eth-cc4       |  -/G/2-FCFS    |   27   |  eth-b10       |  -/G/2-FCFS
   10   |  serv-cc4      |  -/G/13-FCFS   |   28   |  pc-b10        |  -/G/O-IS
   11   |  bridge-b5     |  -/G/1-FCFS    |   29   |  bridge-b11    |  -/G/1-FCFS
   12   |  eth-b5        |  -/G/4-FCFS    |   30   |  eth-b11       |  -/G/5-FCFS
   13   |  pc-b5         |  -/G/O-IS      |   31   |  pc-b11        |  -/G/O-IS
   14   |  bridge-b6     |  -/G/1-FCFS    |   32   |  bridge-b12    |  -/G/1-FCFS
   15   |  eth-b6        |  -/G/5-FCFS    |   33   |  eth-b12       |  -/G/5-FCFS
   16   |  pc-b6         |  -/G/O-IS      |   34   |  pc-b12        |  -/G/O-IS
   17   |  bridge-b7     |  -/G/1-FCFS    |   35   |  wanrouter     |  -/G/1-FCFS
   18   |  eth-b7        |  -/G/3-FCFS    |   36   |  wan           |  -/G/O-IS

CLASS SPECIFICATION

  class   | arrival rate   number of jobs
----------+-------------------------------
    1     |    -              170

CLASS SPECIFIC PARAMETERS

CLASS  1                ( sc_o_v = squared coefficent of variation )

 node        | service_rate | sc_o_v | visit_rat | node        | service_rate | sc_o_v | visit_rat
-------------+--------------+--------+-----------+-------------+--------------+--------+----------
 ring        |   41345      |   0.3  |  9.901    |  pc-b7      |   21.81      |   1    |  1.059
 bridge-cc2  |   9999       |   0.1  |  1.228    |  bridge-b8  |   9999       |   0.1  |  0.139
 eth-cc2     |   3164       |   1.51 |  1.292    |  eth-b8     |   3164       |   1.5  |  0.14
 serv-cc2    |   14.2       |   1    |  0.614    |  pc-b8      |   14.11      |   1    |  0.069
 bridge-cc3  |   9999       |   0.1  |  1.98     |  bridge-b9  |   9999       |   0.1  |  1.366
 eth-cc3     |   3164       |   1.51 |  2        |  eth-b9     |   3164       |   1.51 |  1.38
 serv-cc3    |   14.0       |   1    |  0.99     |  pc-b9      |   14.11      |   1    |  0.683
 bridge-cc4  |   9999       |   0.1  |  1.168    |  bridge-b10 |   9999       |   0.1  |  0.832
 eth-cc4     |   3164       |   1.51 |  1.18     |  eth-b10    |   3164       |   1.51 |  0.84
 serv-cc4    |   14.6       |   1    |  0.584    |  pc-b10     |   8.54       |   1    |  0.416
 bridge-b5   |   9999       |   0.1  |  2        |  bridge-b11 |   9999       |   0.1  |  2.93
 eth-b5      |   3164       |   1.51 |  2.02     |  eth-b11    |   3164       |   1.51 |  3.021
 pc-b5       |   20.61      |   1    |  1        |  pc-b11     |   30.21      |   1    |  1.465
 bridge-b6   |   9999       |   0.1  |  1.069    |  bridge-b12 |   9999       |   0.1  |  3.089
 eth-b6      |   3164       |   1.51 |  1.08     |  eth-b12    |   3164       |   1.51 |  3.152
 pc-b6       |   11.01      |   1    |  0.535    |  pc-b12     |   31.83      |   1    |  1.545
 bridge-b7   |   9999       |   0.1  |  2.119    |  wanrouter  |   9999       |   0.1  |  1.881
 eth-b7      |   3164       |   1.51 |  2.1      |  wan        |   19.41      |   1    |  0.941
```

Fig. 13.14 PEPSY input file for the LAN example.

length at the servers is $\overline{Q}_{cc2} = 3.1$, $\overline{Q}_{cc3} = 1.9$, and $\overline{Q}_{cc4} = 3.2$. The mean response time $\overline{T}$ for the LAN is 0.59 sec.

Now we can use this model for some experiments. For example, we can change the number of active computers K. As we can then see in Table 13.14, the server nodes are the bottleneck and the number of active computers should not exceed 250 because the queue lengths become very large. The mean response time $\overline{T}$ is not influenced much by the number of active computers K in our example. Table 13.15 demonstrates the influence of changing the number of servers at server nodes. If the number of servers is reduced to $m_2 = 10$, $m_3 = 16$, and $m_4 = 9$, then we have a bottleneck at the server nodes. The situation deteriorates if we further reduce the number of servers. The utilization of the ring decreases when the number of servers decreases, whereas the utilization of the server nodes increases.

```
PERFORMANCE_MEASURE FOR NETWORK:  lan170
description of the network is in file 'e_lan170'
the closed network was solved using the method 'marie'
jobclass 1
```

marie	lambda	e	1/mu	rho	mvz	maa	mwz	mwsl
ring	2879.034	9.901	0.000	0.070	0.000	0.377	0.000	0.089
bridge-cc2	357.080	1.228	0.000	0.036	0.000	0.037	0.000	0.001
eth-cc2	375.690	1.292	0.000	0.030	0.000	0.119	0.000	0.000
serv-cc2	178.540	0.614	0.070	0.898	0.088	15.669	0.017	3.096
bridge-cc3	575.749	1.980	0.000	0.058	0.000	0.060	0.000	0.002
eth-cc3	581.564	2.000	0.000	0.026	0.000	0.184	0.000	0.000
serv-cc3	287.874	0.990	0.071	0.894	0.078	22.479	0.007	1.916
bridge-cc4	339.634	1.168	0.000	0.034	0.000	0.035	0.000	0.001
eth-cc4	343.123	1.180	0.000	0.054	0.000	0.113	0.000	0.004
serv-cc4	169.817	0.584	0.069	0.895	0.087	14.858	0.019	3.227
bridge-b5	581.564	2.000	0.000	0.058	0.000	0.060	0.000	0.002
eth-b5	587.380	2.020	0.000	0.046	0.000	0.186	0.000	0.000
pc-b5	290.782	1.000	0.049	0.000	0.049	14.109	0.000	0.000
bridge-b6	310.846	1.069	0.000	0.031	0.000	0.032	0.000	0.001
eth-b6	314.045	1.080	0.000	0.020	0.000	0.099	0.000	0.000
pc-b6	155.568	0.535	0.091	0.000	0.091	14.130	0.000	0.000
bridge-b7	616.167	2.119	0.000	0.062	0.000	0.064	0.000	0.003
eth-b7	622.274	2.140	0.000	0.066	0.000	0.199	0.000	0.003
pc-b7	307.938	1.059	0.046	0.000	0.046	14.119	0.000	0.000
bridge-b8	40.419	0.139	0.000	0.004	0.000	0.004	0.000	0.000
eth-b8	40.709	0.140	0.000	0.013	0.000	0.013	0.000	0.000
pc-b8	20.064	0.069	0.071	0.000	0.071	1.422	0.000	0.000
bridge-b9	397.208	1.366	0.000	0.040	0.000	0.041	0.000	0.001
eth-b9	401.279	1.380	0.000	0.032	0.000	0.127	0.000	0.000
pc-b9	198.604	0.683	0.071	0.000	0.071	14.075	0.000	0.000
bridge-b10	241.931	0.832	0.000	0.024	0.000	0.025	0.000	0.000
eth-b10	244.257	0.840	0.000	0.039	0.000	0.078	0.000	0.001
pc-b10	120.965	0.416	0.117	0.000	0.117	14.165	0.000	0.000
bridge-b11	852.282	2.931	0.000	0.085	0.000	0.090	0.000	0.005
eth-b11	878.453	3.021	0.000	0.056	0.000	0.278	0.000	0.000
pc-b11	425.996	1.465	0.033	0.000	0.033	14.101	0.000	0.000
bridge-b12	898.226	3.089	0.000	0.090	0.000	0.096	0.000	0.006
eth-b12	916.545	3.152	0.000	0.058	0.000	0.290	0.000	0.000
pc-b12	449.258	1.545	0.031	0.000	0.031	14.114	0.000	0.000
wanrouter	546.961	1.881	0.000	0.055	0.000	0.057	0.000	0.002
wan	273.626	0.941	0.051	0.000	0.051	14.097	0.000	0.000

```
characteristic indices:
```

marie	lambda	mvz	maa
	290.782	0.585	170.000

```
legend
e   : average number of visits          mu    : service rate
rho : utilisation                       lambda: mean throughput
mvz : average response time
maa : average number of jobs
mwz : average waiting time
mwsl: average queue-length
```

Fig. 13.15 PEPSY output file for the queueing network model of the communication system (cc_i = computing center$_i$, b_i = building$_i$).

These two experiments show that it is easy to study the influence of the variation in the parameters or the structure of the LAN once we have constructed and parameterized the queueing network model. We can use PEPSY, or any other queueing network package such as QNAP2 [VePo85] or RESQ [SaMa85], for analysis. To obtain the mean response time for a computer, its service time has to be subtracted from mean system response time $\overline{T} = 0.585$ and we obtain for the computers in building 5, for example, the mean response time 0.54 sec.

Problem 13.6 Verify the results of the communication system model just described using another queueing network package such as RESQ or QNAP2.

Table 13.14 Performance measures for the communication system for different numbers of active computers K

K	100	130	150	170	180	200	300
ρ_2	0.55	0.71	0.81	0.90	0.93	0.97	0.99
ρ_3	0.55	0.71	0.81	0.89	0.92	0.97	0.99
ρ_4	0.55	0.71	0.81	0.90	0.92	0.97	0.99
$\overline{Q}_2$	0.05	0.6	1.4	3.1	4.8	7.9	50
$\overline{Q}_3$	0.02	0.3	0.8	1.9	2.9	8.3	30
$\overline{Q}_4$	0.11	0.6	1.4	3.2	4.8	9.3	43
$\overline{T}$	0.56	0.56	0.57	0.59	0.60	0.64	0.94
ρ_{ring}	0.043	0.055	0.063	0.070	0.072	0.075	0.077

Notes: $m_2 = 14$, $m_3 = 23$, $m_4 = 13$.

Table 13.15 Performance measures for the communication system for different number of servers m_2, m_3, and m_4 of section 1, section 2, and section 3, respectively

m_2	7	10	14	16	18	28
m_3	12	16	23	18	26	46
m_4	7	9	13	16	17	26
ρ_2	0.999	0.96	0.90	0.69	0.73	0.47
ρ_3	0.95	0.98	0.89	0.998	0.88	0.47
ρ_4	0.92	0.98	0.90	0.64	0.71	0.47
$\overline{Q}_2$	63	10	3.1	0.4	0.5	0
$\overline{Q}_3$	9.6	13	1.9	28	0.7	0
$\overline{Q}_4$	7.0	23	3.2	0.2	0.5	0
$\overline{T}$	1.05	0.77	0.59	0.67	0.56	0.56
ρ_{ring}	0.039	0.053	0.070	0.061	0.072	0.073

Note: $K = 170$.

13.1.4 UNIX Kernel

We now develop a performance model of the UNIX operating system [GrBo96]. A job in the considered UNIX system is in the *user* context when it executes user code, in the *kern* context when it executes code of the operating system kernel, or in the *driv* context when it executes driver code to set up or perform an I/O operation. In Fig. 13.16 the life-cycle of a UNIX job is shown.

The job always starts in the *kern* context from where it switches to the *user* context with probability p_{user} or to the *driv* context with probability p_{io}. After a mean service time s_{user}, the job returns from the *user* context to the *kern* context. From the *driv* context, where the job remains s_{drive} time units, the job returns to the *kern* context with probability p_{drivdone} or starts a I/O operation with probability $1 - p_{\text{drivdone}}$. After finishing the I/O, the job returns to the context *driv* and again remains there (with mean time s_{driv})

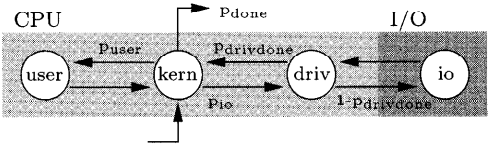


Fig. 13.16 Model of a UNIX job.

Table 13.16 Routing probabilities of the monoprocessor model

	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)
(1,1)	0	p_{drivdone}	0	$1 - p_{\text{drivdone}}$	0	0
(1,2)	p_{io}	p_{done}	p_{user}	0	0	0
(1,3)	0	1	0	0	0	0
(2,1)	1	0	0	0	0	0
(2,2)	0	0	0	0	0	0
(2,3)	0	0	0	0	0	0

and returns to the *kern* context with probability p_{drivdone}. A job can only leave the system when it is in the *kern* context (with probability p_{done}). The mean service time in the *kern* context is s_{kern}. The I/O operation is carried out on an I/O device, whereas the remaining three activities are serviced by the CPU.

13.1.4.1 Model of the UNIX Kernel The UNIX kernel just described can be modeled by a closed non-product-form queueing network with two nodes (CPU, I/O) and three job classes, priorities, and class-switching (see Fig. 13.17).

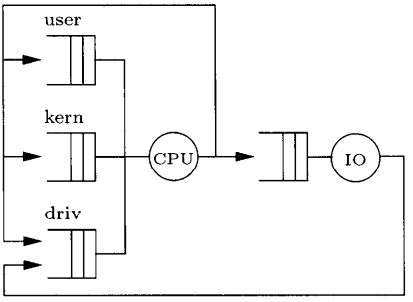


Fig. 13.17 The monoprocessor model.

Since we have only one processor, we call this the *monoprocessor model.* The three job classes correspond to the *user*, *kern*, and *driv* context. To obtain a closed network, we assume that when a job leaves the system a new job immediately arrives at the *kern* queue.

A state in the routing DTMC of this model is described as a pair (node number, class number) with:

node number 1 : CPU,
node number 2 : I/O,

class number 1 : *driv*,
class number 2 : *kern*,
class number 3 : *user*.

The routing probabilities are shown in Table 13.16. In the following we summarize the assumptions we make:

- The network is closed and there are $K = 10$ jobs in the system.
- The service times are exponentially distributed.
- The system has three job classes *user, kern, driv* with the priority order driv > kern > user.
- Class switching is allowed.
- Jobs in the context *user* can always be preempted by *driv* and *kern* jobs. Other preemptions are not possible.
- The time unit used in this section is msec.
- The following parameter values are used:

$$\begin{array}{ll} p_{\text{io}} = 0.05, & s_{\text{user}} = \text{varied from } 0.25 \text{ to } 20.0, \\ p_{\text{done}} = 0.01/0.005, & s_{\text{kern}} = 1.0, \\ p_{\text{drivdone}} = 0.4, & s_{\text{driv}} = 0.5. \end{array}$$

- The I/O system consists of several devices that are assumed to form a composite node (see Section 8.4) with load-dependent service times (which were measured from the real system):

$$\begin{array}{ll} s_{\text{io}}(1) = 28.00, & s_{\text{io}}(6) = 12.444, \\ s_{\text{io}}(2) = 18.667, & s_{\text{io}}(7) = 12.000, \\ s_{\text{io}}(3) = 15.555, & s_{\text{io}}(8) = 11.667, \\ s_{\text{io}}(4) = 14.000, & s_{\text{io}}(9) = 11.407, \\ s_{\text{io}}(5) = 13.067, & s_{\text{io}}(10) = 11.200. \end{array}$$

The parameters s_{user}, s_{kern}, and s_{driv} are the mean service times of class i ($i = 1, 2, 3$) at the CPU (node 1), and $s_{\text{io}}(k)$ is the mean load-dependent service time of class 1 jobs at the I/O devices (node 2). At the CPU (node 1) we have a mixture of preemptive and non-preemptive service strategy, while at the I/O devices (Node 2) we have only one job class with the FCFS service discipline. The following concepts are used to solve this model:

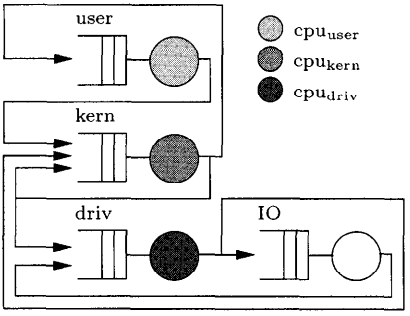


Fig. 13.18 The transformed monoprocessor model.

- Chain concept (see Section 7.3.6)
- Extended shadow technique with mixed priority strategy and class switching (see Section 10.3.2.2).

Since we have three job classes, we split the CPU into three CPUs, one for each job class (see Fig. 13.18). Approximate iterative solution of the resulting model can be carried out by PEPSY or SHARPE.

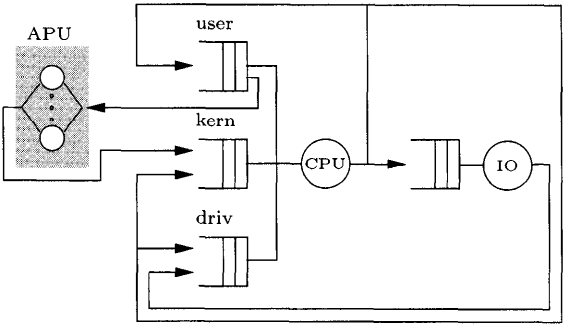


Fig. 13.19 The master slave model.

13.1.4.1.1 The Master Slave Model For this model, shown in Fig. 13.19, we have the same basic assumptions as for the monoprocessor model but now we also have two APUs. An APU is an additional processor (associate processing unit) that works with the main CPU. It is assumed that only jobs in the *user* context can be processed at the APU. Since this is the only job class that can be processed at the APU, no preemption can take place there. If the APU as well as the CPU are free and there are jobs in the user queue, then the processor (CPU, APU) on which the job is served is decided randomly. If we apply the shadow transformation to the original model, we get the transformed master slave model, shown in Fig. 13.20.

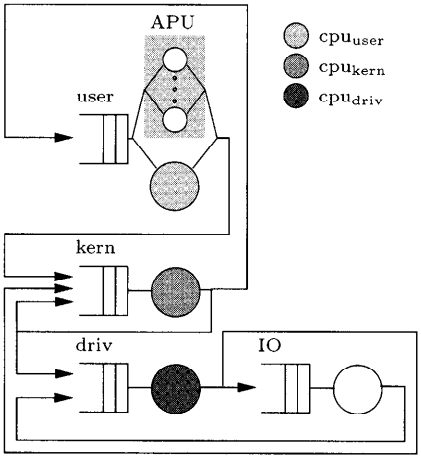


Fig. 13.20 The transformed master slave model.

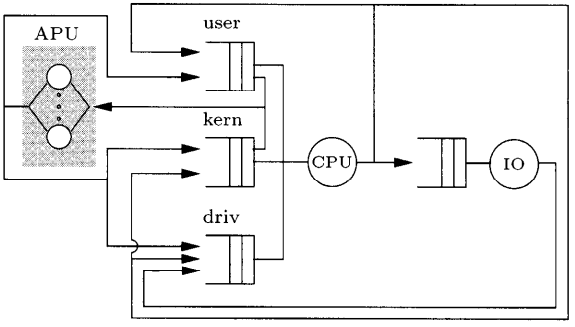


Fig. 13.21 The associated processor model.

13.1.4.1.2 The Associated Processor Model The assumptions for the associated processor model, shown in Fig. 13.21, are the same as for the master slave model but now jobs in the *kern* context can also be processed on the APU. Since jobs in *kern* context have higher priority than jobs in *user* context, preemption can also take place at the APU. The model needs to be transformed so that it can be solved approximately using the shadow technique in combination with the MVA. The transformed associated processor model is shown in Fig. 13.22.

13.1.4.2 Analysis We now solve the three non-product-form network models using approximate techniques and compare the results to the ones obtained from the (exact) numerical solution of the underlying CTMC. We show the computation of the performance measures for the monoprocessor model step

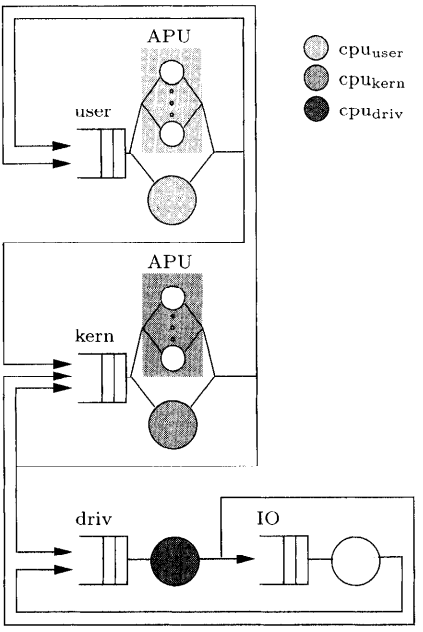


Fig. 13.22 The transformed associated processor model

by step. The procedure is analogous for the other two models and therefore only the results are given.

13.1.4.2.1 The Monoprocessor Model The model, given in Fig. 13.17, contains three job classes and two nodes:

class 1 : *driv*,
class 2 : *kern*,
class 3 : *user*,

node 1 : CPU,
node 2 : I/O.

The priority order of the job classes is $driv > kern > user$. Jobs in the context *user* can always be preempted by *driv* and *kern* jobs. Other preemptions are not possible. The chosen numerical parameter values are:

$$p_{11,12} = p_{\text{drivdone}} = 0.4, \qquad p_{11,21} = 1 - p_{\text{drivdone}} = 0.6,$$
$$p_{12,11} = p_{\text{io}} = 0.05, \qquad p_{12,12} = p_{\text{done}} = 0.01,$$
$$p_{12,13} = p_{\text{user}} = 0.94, \qquad p_{13,12} = 1.0,$$
$$p_{21,11} = 1.0,$$
$$s_{11} = s_{\text{driv}} = 0.5, \qquad \epsilon = 0.001,$$
$$s_{12} = s_{\text{kern}} = 1.0, \qquad N = 5 \text{ jobs},$$
$$s_{13} = s_{\text{user}} = 1.5,$$
$$s_{21} = s_{\text{io}} = \text{load dep.}$$

Now the analysis of the monoprocessor model can be done in the following nine steps:

STEP 1 Compute the visit ratios e_{ir} using Eq. (7.15):

$$e_{11} = e_{\text{driv}} = 12.5, \qquad e_{12} = e_{\text{kern}} = 100,$$
$$e_{13} = e_{\text{user}} = 94, \qquad e_{21} = e_{\text{io}} = 7.5.$$

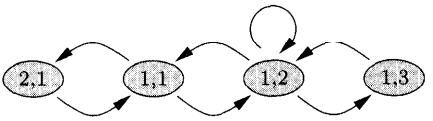


Fig. 13.23 The routing DTMC for the monoprocessor model.

STEP 2 Determine the chains in the network. The DTMC state diagram for the routing matrix in the monoprocessor model is given in Fig. 13.23. As we can see, every state can be reached from every other state, which means that we have only one chain in the system.

STEP 3 Transform the network into the shadow network (see Fig. 13.18). The corresponding parameter values for the transformed model are:

$$s_{11} = 0.5, \qquad s_{22} = 1.0, \qquad s_{33} = 1.0, \qquad s_{41} = \text{load dep.}$$
$$e_{11} = 12.5, \qquad e_{22} = 100, \qquad e_{33} = 94, \qquad e_{41} = 7.5.$$

STEP 4 Compute the visit ratios e^*_{iq} per chain and recall that we have only one chain:

$$e^*_{iq} = \frac{\sum\limits_{r \in \pi_q} e_{ir}}{\sum\limits_{r \in \pi_q} e_{1r}}.$$

Then we get:

$$e^*_{11} = \frac{e_{11}}{e_{11}} = 1, \qquad e^*_{21} = \frac{e_{22}}{e_{11}} = 8,$$
$$e^*_{31} = \frac{e_{33}}{e_{11}} = 7.52, \qquad e^*_{41} = \frac{e_{41}}{e_{11}} = 0.6.$$

STEP 5 Compute the scale factors α_{ir}, using the equation:

$$\alpha_{ir} = \frac{e_{ir}}{\sum\limits_{s \in \pi_q} e_{is}},$$

and get:

$$\alpha_{11} = \frac{e_{11}}{e_{11}+e_{12}+e_{13}} = 1.0, \qquad \alpha_{31} = \frac{e_{31}}{e_{31}+e_{32}+e_{33}} = 0,$$
$$\alpha_{12} = \frac{e_{12}}{e_{11}+e_{12}+e_{13}} = 0, \qquad \alpha_{32} = \frac{e_{32}}{e_{31}+e_{32}+e_{33}} = 0,$$
$$\alpha_{13} = \frac{e_{13}}{e_{11}+e_{12}+e_{13}} = 0, \qquad \alpha_{33} = \frac{e_{33}}{e_{31}+e_{32}+e_{33}} = 1.0,$$
$$\alpha_{21} = \frac{e_{21}}{e_{21}+e_{22}+e_{23}} = 0, \qquad \alpha_{41} = \frac{e_{41}}{e_{41}+e_{42}+e_{43}} = 1.0,$$
$$\alpha_{22} = \frac{e_{22}}{e_{21}+e_{22}+e_{23}} = 1.0, \qquad \alpha_{42} = \frac{e_{42}}{e_{41}+e_{42}+e_{43}} = 0,$$
$$\alpha_{23} = \frac{e_{23}}{e_{21}+e_{22}+e_{23}} = 0, \qquad \alpha_{43} = \frac{e_{43}}{e_{41}+e_{42}+e_{43}} = 0.$$

STEP 6 Compute the mean service times for the chain:

$$s_{iq}^* = \frac{1}{\mu_{iq}} = \sum_{r \in \pi_q} s_{ir} \cdot \alpha_{ir},$$

$$s_{11}^* = 0.5, \qquad s_{21}^* = 1.0, \qquad s_{31}^* = 1.5, \qquad s_{31}^* = \text{load dep.}$$

STEP 7 Start the shadow iterations:

Iteration 1:

$$\rho_{11}^* = 0.037, \quad \rho_{21}^* = 0.594, \quad \tilde{s}_{11}^{(1)*} = 0.5,$$
$$\rho_{31}^* = 0.838, \quad \rho_{41}^* = 0.659, \quad \tilde{s}_{21}^{(1)*} = 1.03842,$$
$$\lambda_{11}^* = 0.075, \quad \lambda_{21}^* = 0.594, \quad \tilde{s}_{31}^{(1)*} = 4.06504,$$
$$\lambda_{31}^* = 0.558, \quad \lambda_{41}^* = 0.045, \quad \tilde{s}_{41}^{(1)*} = \text{load dep.}$$

Iteration 2:

$$\rho_{11}^* = 0.016, \quad \rho_{21}^* = 0.27, \quad \tilde{s}_{11}^{(2)*} = 0.5,$$
$$\rho_{31}^* = 0.994, \quad \rho_{41}^* = 0.289, \quad \tilde{s}_{21}^{(2)*} = 1.01626,$$
$$\lambda_{11}^* = 0.033, \quad \lambda_{21}^* = 0.26, \quad \tilde{s}_{31}^{(2)*} = 2.10084,$$
$$\lambda_{31}^* = 0.245, \quad \lambda_{41}^* = 0.02, \quad \tilde{s}_{41}^{(2)*} = \text{load dep.}$$

Iteration 3:

$$\rho_{11}^* = 0.030, \quad \rho_{21}^* = 0.481, \quad \tilde{s}_{11}^{(3)*} = 0.5,$$
$$\rho_{31}^* = 0.935, \quad \rho_{41}^* = 0.525, \quad \tilde{s}_{21}^{(3)*} = 1.03092,$$
$$\lambda_{11}^* = 0.033, \quad \lambda_{21}^* = 0.473, \quad \tilde{s}_{31}^{(3)*} = 3.16373,$$
$$\lambda_{31}^* = 0.245, \quad \lambda_{41}^* = 0.02, \quad \tilde{s}_{41}^{(3)*} = \text{load dep.}$$

Iteration 15: Finally, after 15 iterations, the solution converges and we have the following results.

$$\begin{aligned}
\rho_{11}^{*} &= 0.025, & \rho_{21}^{*} &= 0.402, & \tilde{s}_{11}^{(15)*} &= 0.5, \\
\rho_{31}^{*} &= 0.969, & \rho_{41}^{*} &= 0.435, & \tilde{s}_{21}^{(15)*} &= 1.0256, \\
\lambda_{11}^{*} &= 0.049, & \lambda_{21}^{*} &= 0.392, & \tilde{s}_{31}^{(15)*} &= 2.61780, \\
\lambda_{31}^{*} &= 0.368, & \lambda_{41}^{*} &= 0.029, & \tilde{s}_{41}^{(15)*} &= \text{load dep.}
\end{aligned}$$

STEP 8 Retransform the chain values into class values:

$$\begin{aligned}
\lambda_{11} &= \alpha_{11}\lambda_{11}^{*} = 0.049, \\
\lambda_{22} &= \alpha_{22}\lambda_{21}^{*} = 0.392, \\
\lambda_{33} &= \alpha_{33}\lambda_{31}^{*} = 0.368, \\
\lambda_{41} &= \alpha_{41}\lambda_{41}^{*} = 0.029.
\end{aligned}$$

If we rewrite the results of the shadow network in terms of the original network, we get the following final results:

$$\begin{aligned}
\lambda_{11}^{(\text{approx})} &= 0.049, & \lambda_{12}^{(\text{approx})} &= 0.392, \\
\lambda_{13}^{(\text{approx})} &= 0.368, & \lambda_{21}^{(\text{approx})} &= 0.029.
\end{aligned}$$

The overall throughput of the network is then given by:

$$\lambda^{(\text{approx})} = 1000 \cdot p_{\text{done}} \cdot \lambda_{12}^{(\text{approx})} = 3.92.$$

STEP 9 Verify the results: To verify the results, we construct and solve the underlying CTMC using MOSES. The exact overall throughput is given by:

$$\lambda^{(\text{exact})} = 3.98.$$

The difference between the exact and the approximate throughput is small.

The other two models were also analyzed using the extended shadow technique with class switching in a similar manner.

The overall system throughput of the UNIX models is plotted as a function of the user service time in Fig. 13.24. There is nearly no difference between the approximate values and the exact values obtained from the numerical solution of the CTMC for the chosen parameter set. As expected, the associated processor model has the highest system throughput. In a more detailed study it was found that for the master slave model the throughput cannot be increased any more by adding additional APUs because the CPU is the bottleneck. For systems that spend a lot of time doing system jobs (UNIX is said to spend approximately 50% of its time doing system work) it is worth having a parallel system kernel. For more I/O-intensive jobs, the number of

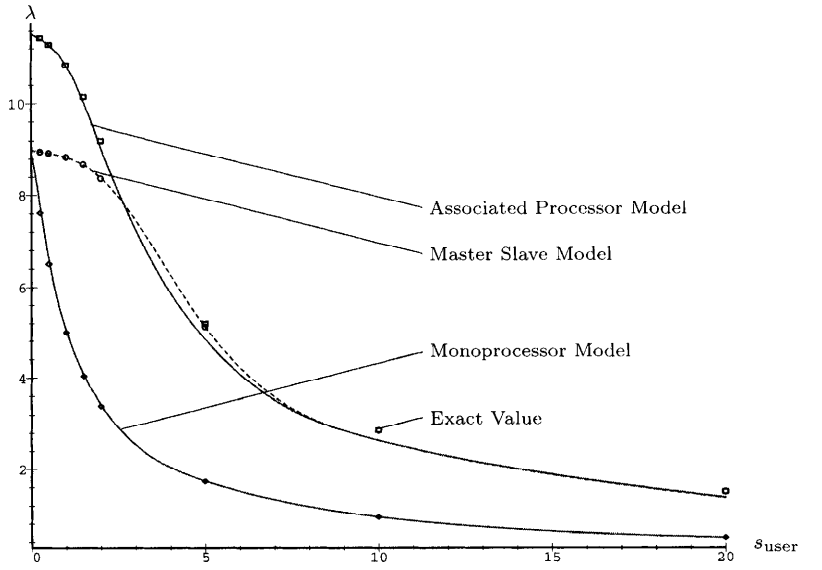


Fig. 13.24 Throughput for the different models covered in Section 13.1.4.

peripheral devices must also be increased, otherwise the processors have to wait too long for I/O jobs to be completed. In this case it is not worth having a multiprocessor system. Another very important result from this study is that for $s_{\text{user}} \geq 3$ msec, the master slave and associated processor model behave nearly the same and only for $s_{\text{user}} < 3$ *msec* does the associated processor model gives better results than the master slave model.[1] However, it does not seem to matter as to which of the two (an associated processor configuration or a master slave one) configurations is used. On the other hand, if the system is used in a real-time environment where response times are required to be extremely short, the associated processor model delivers better performance than the master slave configuration and it is worthwhile to have a completely parallel kernel.

The computation time to solve the model was approximately 45 minutes if the CTMC method is used. Using the shadow technique, the results were generated within seconds for the entire parameter set. By contrast, when this non-product-form network was solved using DES, it took nearly 120 hours of computing time. The impact of priorities is shown for the master slave model in Table 13.17. Here p_{io} is varied while p_{done} is constant. On the right-hand side of the table the throughput for the network without priorities (nonpre), the original values with priorities (orig = obtained from numerical solution of the CTMC), and approximate values (approx = obtained using the shadow

[1]This result means that if the system will be used for very time consuming jobs, it is worthwhile to have a parallel kernel.

approximation) are given. As can be seen in the table, for some parameter values the impact of priorities cannot be neglected.

Table 13.17 Impact of the priorities ($p_{\text{done}} = 0.01$)

s_{user}	p_{io}	Throughput		
		Nonpre	Approx	Orig
0.50	0.9	0.6593	0.6594	0.6593
0.50	0.01	9.0049	9.8765	9.8765
0.50	0.001	9.0897	9.9875	9.9875
1.00	0.9	0.6593	0.6594	0.6593
1.00	0.01	8.2745	9.8736	9.8716
1.00	0.001	8.3398	9.9859	9.9832
1.50	0.9	0.6593	0.6594	0.6592
1.50	0.01	7.6517	9.8194	9.7666
1.50	0.001	7.7027	9.9395	9.8983
2.00	0.9	0.6593	0.6594	0.6592
2.00	0.01	7.1078	9.5178	9.4700
2.00	0.001	7.1489	9.6326	9.4000
2.50	0.9	0.6592	0.6593	0.6592
2.50	0.01	6.6170	8.8123	8.6144
2.50	0.001	6.6485	8.8802	8.5100

13.1.5 Flexible Production Systems

Queueing network models combined with numerical solution methods are also a very powerful paradigm for the performance evaluation of production systems. In this section we demonstrate how to model and analyze such systems using open and closed queueing networks.

13.1.5.1 An Open Network Model Consider a simple production system that can be modeled as an open queueing network. The system contains the following stations (Fig. 13.25):

- A load station where the workpieces are mounted on to the pallet (LO).
- Two identical lathes (LA).
- Three identical milling machines (M).
- A transfer system (T) that does the transfer between the stations and consists of two automatically controlled vehicles.
- A station to unload the pallet (U), which removes the workpieces from the production system.

The identical milling machines and lathes are modeled by multiple server nodes. With these assumptions we obtain the queueing network model shown

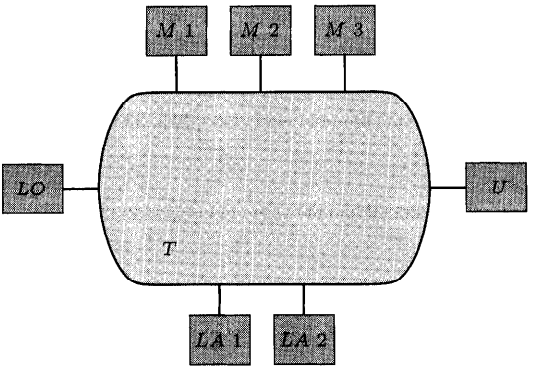


Fig. 13.25 Layout of a simple production system.

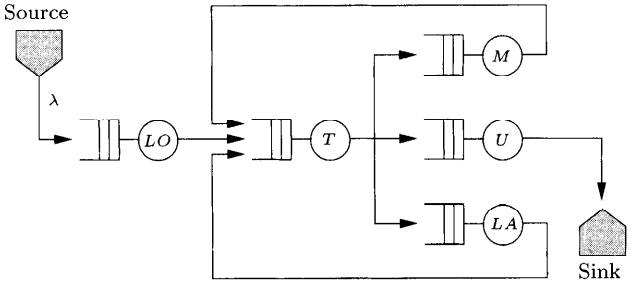


Fig. 13.26 Open network model of the production system shown in Fig. 13.25.

in Fig. 13.26. The production system produces different products and therefore the machines are used in several different ways. For example, 60% of the workpieces that leave the milling machine are passed to the station that unloads the pallet, the rest are transferred to the lathes. Table 13.18 contains the probabilities q_{ij}, $i = LO, M, LA$, $j = M, LA, U$ that the transfer system T moves workpieces from a station i to station j.

To obtain from the transfer probabilities q_{ij} the probabilities of routing from the transfer system T to the lathes $p_{T,LA}$, milling machines $p_{T,M}$, or the unload station $p_{T,U}$, we have to weight the individual probabilities q_{ij} with the individual arrival rates λ_i, $i = LO, M, LA$. The arrival rate λ_T to the transfer system can be easily obtained from Fig. 13.26:

$$\lambda_T = \lambda_{LO} + \lambda_M + \lambda_{LA}, \tag{13.8}$$

Table 13.18 Routing table for the transfer system of the production system

i \ j	M	LA	U
LO	0.5	0.5	0
M	0	0.4	0.6
LA	0.7	0	0.3

and it follows that:

$$\begin{aligned} p_{T,M} &= \frac{\lambda_{LO}}{\lambda_T} \cdot q_{LO,M} + \frac{\lambda_{LA}}{\lambda_T} \cdot q_{LA,M}, \\ p_{T,LA} &= \frac{\lambda_{LO}}{\lambda_T} \cdot q_{LO,LA} + \frac{\lambda_M}{\lambda_T} \cdot q_{M,LA}, \\ p_{T,U} &= \frac{\lambda_{LA}}{\lambda_T} \cdot q_{LA,U} + \frac{\lambda_M}{\lambda_T} \cdot q_{M,U}. \end{aligned} \tag{13.9}$$

Then, using the values from Table 13.18:

$$\begin{aligned} p_{T,M} &= \frac{1}{\lambda_T} \left(\lambda_{LO} \cdot 0.5 + \lambda_{LA} \cdot 0.7\right), \\ p_{T,LA} &= \frac{1}{\lambda_T} \left(\lambda_{LO} \cdot 0.5 + \lambda_M \cdot 0.4\right), \\ p_{T,U} &= \frac{1}{\lambda_T} \left(\lambda_{LA} \cdot 0.3 + \lambda_M \cdot 0.6\right), \end{aligned}$$

we finally obtain the matrix of routing probabilities p_{ij} for the queueing network model (see Fig. 13.26 and Table 13.19). In order to apply Jackson's

Table 13.19 Routing matrix for the queueing model of the production system

i \ j	outside	LO	LA	M	U	T
outside	0	1	0	0	0	0
LO	0	0	0	0	0	1
LA	0	0	0	0	0	1
M	0	0	0	0	0	1
M	0	0	0	0	0	1
U	1	0	0	0	0	0
T	0	0	p_{LA}	p_M	p_U	0

theorem for open product-form networks (see Section 7.3.4), we make the following assumptions:

- The service times at the stations and the interarrival times at the load station LO are exponentially distributed.
- The service discipline at each station is FCFS.
- The system is stable.

Table 13.20 Arrival rates λ_{0i}, service rates μ_i (in 1/h), and number of servers m_i for the model of Fig. 13.26

i	λ_{0i}	μ_i	m_i
LO	15	20	1
LA	0	10	2
M	0	7	3
U	0	20	1
T	0	24	2

The service rates μ_i, the arrival rates λ_{0i}, and number of servers m_i are listed in Table 13.20. Now we use Eq. (7.1) and the values of the routing probabilities from Table 13.19 to determine the arrival rates λ_i:

$$\lambda_i = \lambda_{0i} + \sum_j \lambda_j \cdot p_{ji}, \quad i,j = LO, LA, M, U, T, \tag{13.10}$$

and obtain:

$$\begin{aligned}
\lambda_{LO} &= \lambda_{0LO} = 15, \\
\lambda_{LA} &= \lambda_T \cdot p_{T,LA} = \lambda_{LO} \cdot 0.5 + \lambda_M \cdot 0.4 = 14.58, \\
\lambda_M &= \lambda_T \cdot p_{T,M} = \lambda_{LO} \cdot 0.5 + \lambda_{LA} \cdot 0.7 = 17.71, \\
\lambda_U &= \lambda_T \cdot p_{T,U} = \lambda_{LO} = 15, \\
\lambda_T &= \lambda_{LO} + \lambda_{LA} + \lambda_M = 47.29.
\end{aligned}$$

Performance measures, which were calculated using Jackson's theorem, are listed in Table 13.21.

We see that the transfer system T is heavily loaded and has a very long queue, and that its utilization is nearly 100%. If we use an additional vehicle, the utilization and the queue length are reduced substantially to

$$\rho_T = 0.66, \quad \overline{Q}_T = 0.82, \quad \overline{W}_T = 0.02.$$

The work in progress (WIP, the mean number of workpieces in the system) and the mean response time $\overline{T}$ for both cases are listed in Table 13.22. To improve the performance we could increase the number of milling machines because they have a very high utilization, $\rho_M = 0.84$, or the number of lathes. The results for these changes are shown in Table 13.23. A further increase in

Table 13.21 Performance measures for the production system

i	$\overline{Q}_i$	$\overline{W}_i$	ρ_i
LO	2.25	0.15	0.75
LA	1.66	0.11	0.73
M	3.86	0.22	0.84
U	2.25	0.15	0.75
T	65.02	1.38	0.985

Table 13.22 Work in progress (WIP) and mean response time $\overline{T}$ for 2 and 3 vehicles in the transfer system

m_T	WIP	$\overline{T}$
2	82.5	5.5
3	18.3	1.2

the number of the vehicles in the transfer system has a negligible influence on the performance. Maximum improvement can be achieved by increasing the number of milling machines, because that station is the bottleneck ($\rho_i = 0.84$, see Table 13.21). Although we have used a very simple model to approximate the behavior of the production system, it does provide insight into the system behavior and enables us to identify bottlenecks.

13.1.5.2 A Closed Network Model Now we consider the example of a production system that can be modeled as a closed multiclass queueing network [SuHi84]. The production system consists of (see Fig. 13.27):

- Two load/unload stations (LU).
- Two lathes with identical tools (LA).

Table 13.23 WIP and $\overline{T}$ for different numbers of vehicles, milling machines, and lathes

m_T	m_{LA}	m_M	WIP	$\overline{T}$
3	2	3	18.3	1.22
4	2	3	17.6	1.18
3	3	3	16.9	1.12
3	2	4	15.0	1.00
3	3	4	13.6	0.90

- Three machine centers with different tools (M).
- A central transfer system with eight transporters (T).

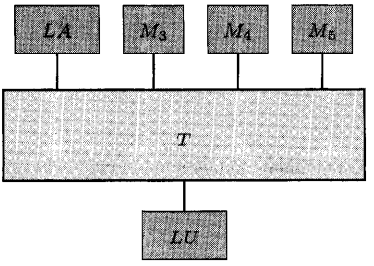


Fig. 13.27 Layout of the production system for a closed multiclass network model.

In the production system, workpieces for three different product classes are manufactured. A fixed number K_i of pallets for each class circulates through the system. In the first class the workpieces are mounted on the pallets in the LU station then they are moved by the transfer system either to the lathes LA or to the third M_3. Finally, they are shipped back to the LU station where they are unmounted. For workpieces of the other two classes, the procedure is similar, but they use machine M_4 and M_5, respectively. The load/unload station LU and the lathes LA can be modeled as a multiple server node and the machines M_3, M_4, and M_5 as single server nodes. Since there are more transporters in the transfer system T than circulating pallets, it is modeled by an IS node. The parameters for the model are summarized in Table 13.24.

Table 13.24 Parameter values for the stations

Station$_i$	m_i	$1/\mu_i$	e_{1i}	e_{2i}	e_{3i}
LU	2	7.5	2	2	2
LA	2	28	0.5	0	0
M_3	1	12	0.5	0	0
M_4	1	30	0	1	0
M_5	1	15	0	0	1
T	8	2.5	2	2	2

The numbers of pallets K_i, which are the numbers of workpieces in class i, are given by:

$$K_1 = 4, \quad K_2 = 1, \quad K_3 = 2.$$

With these parameters we obtain the queueing network models for these three classes, given in Fig. 13.28. Furthermore we assume that each network fulfills the product-form assumptions. Now we have all the necessary input param-

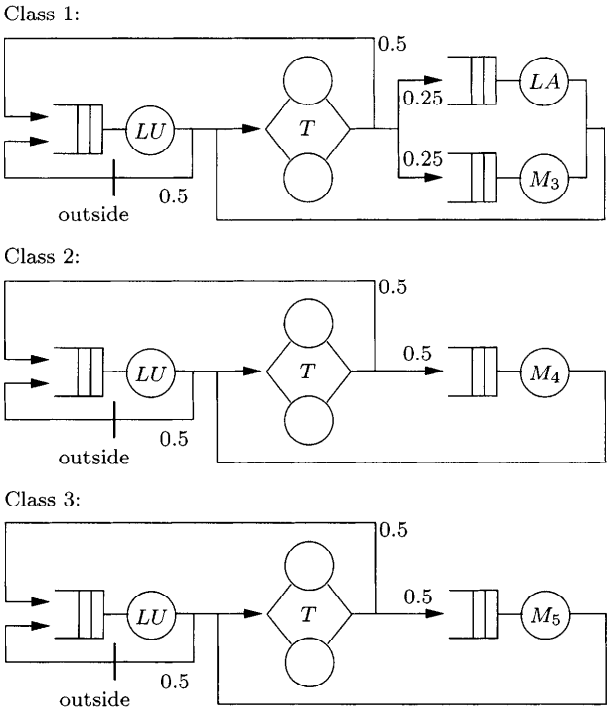


Fig. 13.28 Closed queueing model of the production system shown in Fig. 13.27.

eters and can use any solution method for closed product-form queueing networks such as MVA, convolution, or SCAT to obtain performance measures. In Table 13.25, the throughput and the mean system response time for the different classes are shown. The total throughput is 0.122 workpieces/min or 7.32

Table 13.25 Performance measures for workpieces

Class	Throughput	Mean Response Time
1	0.069	58.35
2	0.016	63.88
3	0.037	53.77

workpieces/h. For other interesting performance measures of the machines, see Table 13.26.

Using the queueing network model, the system can be optimized. In Table 13.26 we can see that the load/unload stations are the bottleneck and, therefore, we increase the number of these stations to three and four, respectively. The results for the modified system models are given in Table 13.27,

Table 13.26 Performance measures for the stations

Station$_i$	m_i	ρ_i	Mean number in Service	Mean Queue Length $\overline{Q}_i$
LU	2	0.91	1.82	1.68
LA	2	0.48	0.96	0.12
M_3	1	0.41	0.41	0.19
M_4	1	0.47	0.47	0.00
M_5	1	0.56	0.56	0.16
T	8	0.08	0.61	0.00

Table 13.27 Utilizations of the stations for different number of servers at the LU stations and routing probabilities of Table 13.28

Station$_i$ \ m_{LU}	1	2	3	4	4a	4b	4c	4d
LU	0.997	0.91	0.73	0.58	0.55	0.56	0.56	0.77
LA	0.27	0.48	0.58	0.61	0.60	0.60	0.60	0.82
M_3	0.23	0.41	0.49	0.52	0.56	0.54	0.56	0.77
M_4	0.26	0.47	0.57	0.59	0.68	0.64	0.59	0.82
M_5	0.30	0.56	0.68	0.72	0.52	0.62	0.62	0.85
T	0.04	0.08	0.092	0.096	0.09	0.09	0.08	0.129
$\rho_{max} - \rho_{min}$	0.77	0.50	0.240	0.200	0.16	0.10	0.06	0.08

columns 2, 3 and 4. In column 1, the number of load/unload stations is reduced to one. In this case the system works, but with a severe bottleneck at LU, the mean response time is greater than 100 units, and the mean queue length at LU is 3.23. For $m_{LU} = 3$, the utilization of the LU station is still

Table 13.28 Routing probabilities

	p_{23}	p_{24}	p_{34}	p_{35}
a	0.2	0.8	0.2	0.8
b	0.1	0.9	0.1	0.9
c	0.2	0.8	0.1	0.9
d	$K_1 = 8, K_2 = 2, K_3 = 4$			

$\rho_{LU} = 0.73$, and the system is still unbalanced, which means that the utilization of the different stations differ considerably, as seen in the last column of Table 13.27. For $m_{LU} = 4$, the system is more balanced, but now $\rho_{M_5} = 0.72$ is too high. To get a more balanced system, a fraction of the workpieces of class 2 are serviced at station M_3 (see Fig. 13.29). Now the system is better balanced especially in case 4*c*, but the utilizations are relatively low ($\rho_i \approx 0.6$).

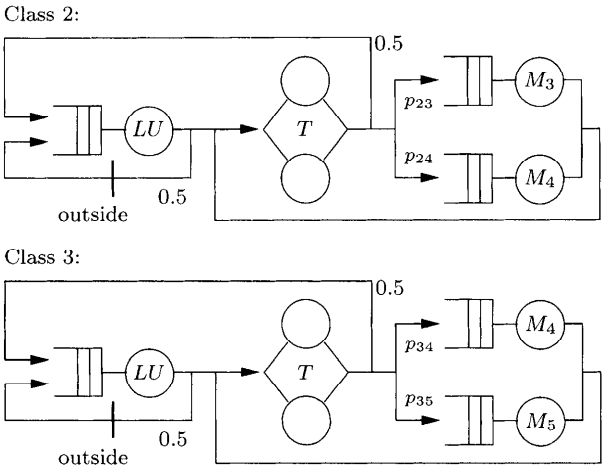


Fig. 13.29 Queueing network model of the production system shown in Fig. 13.28, with different routing for class 2 and 3.

Therefore, we double the number of pallets in the system ($K_1 = 8$, $K_2 = 2$, and $K_3 = 4$) and obtain the values of the utilization in column 4d. The system is now balanced at a higher level of utilization ($\rho_i \approx 0.8$). Increasing the number of LU servers leads to a higher throughput and a lower mean response time (see Table 13.29). If we increase the number of pallets to $K_1 = 8$, $K_2 = 2$ and $K_3 = 4$, the mean response time increases along with the throughput (see Table 13.30).

Table 13.29 Performance measures with four LU servers

Class	Throughput	Mean Response Time
1	0.086	46.29
2	0.020	50.61
3	0.048	41.97

Problem 13.7 Verify the results of the closed queueing network model of the production system by hand computation and by using the SHARPE software package.

13.2 CASE STUDIES OF MARKOV CHAINS

In this section we show the use of CTMC models either constructed directly or via a higher-level specification.

Table 13.30 Performance measures with four LU servers and a higher number of pallets

Class	Throughput	Mean Response Time
1	0.117	68.21
2	0.026	76.56
3	0.063	63.54

Notes: $K_1 = 8$, $K_2 = 2$, $K_3 = 4$.

13.2.1 Wafer Production System

In this section we formulate and solve a CTMC model of the photolithographic process of a wafer production system. The system is specified using a queueing network formalism, although for this product-form network we use the package MOSEL to automatically generate and solve the underlying CTMC. In [Rand86] the photolithographic process is divided into the following five steps:

1. Clean the wafer.
2. Put on photoresist.
3. Bake wafer and remove solvent by evaporation.
4. Align mask, and illuminate and develop the wafer.
5. Etch the wafer and remove photoresist.

The photolithographic processing is carried out by three machine types:

- The spinning machine does the first three steps (machine type 1).
- The masking machine does step four (machine type 2).
- The etching machine does step five (machine type 3).

From the first machine type, three machines are available. From the second machine type, we have machines from different producers. These machines are called masking 1 and masking 2. The third machine type does the steps of etching and removing the photoresist. The job of removing the photoresist is done by machine five in case the wafer has not been produced correctly. Since three layers of photoresist have to be put on, the wafer needs to be processed again after a successful masking and spinning. In Fig. 13.30, the open queueing network model of the wafer production system is shown and in Table 13.31 its parameter values are given.

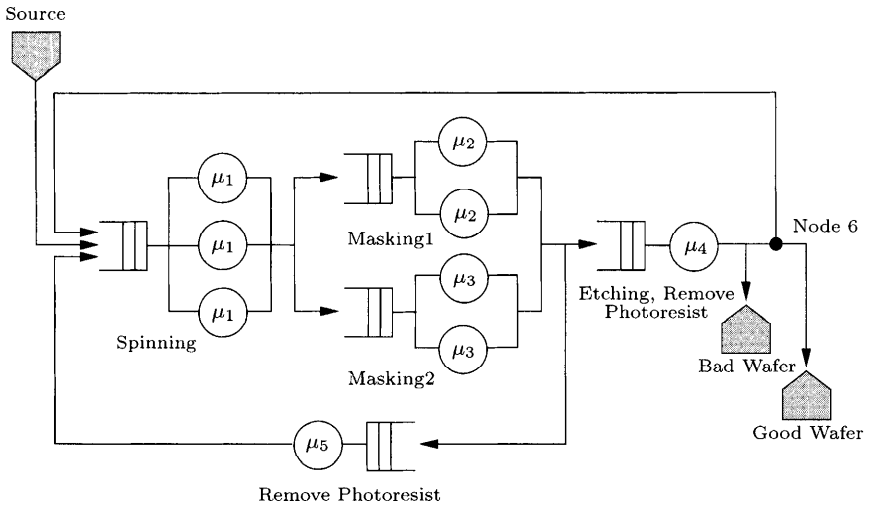


Fig. 13.30 Open network model of the wafer production system discussed in Section 13.2.1.

In the representation of the queueing network model we use node 6 as a dummy node with zero service time. We use node 6 in order to easily determine the routing probabilities, but it is, of course, also possible to determine the routing probabilities without the introduction of node 6. Since it is an open network, we limit the number of wafers in the system to K so that the underlying CTMC is not infinite.

The MOSEL specification of the network is shown in Fig. 13.31; note the use of the loop construct to reduce the size of the specification. WIP is the work in progress, i.e., the mean number of wafers in the system. This value needs to be as small as possible for good system performance.

Results of the analysis are shown in Fig. 13.32. Here we plot the mean response time as a function of the number of machines at station 1. There are several things that we can learn from the plot:

- Having more than five machines at station 1 does not reduce the mean response time very much. Therefore it is not cost effective to have more than five machines at station 1.

- Increasing the number of machines at stations 2 and 3 from two to three reduces the mean response time considerably, while adding more machines is not any more cost effective.

- We get a large decrease in the mean response time if we increase the number of machines at station 1 from three to four and if we increase the number of machines at stations 2 and 3 from two to three.

Table 13.31 Parameter values for the wafer production system discussed in Section 13.2.1

m_i	p_{ij}	μ_i	λ
$m_1 = 3 \ldots 10$	$p_{12} = 0.5$	$\mu_1 = 1$	$\lambda = 1$
$m_2 = 2 \ldots 4$	$p_{13} = 0.5$	$\mu_2 = 1$	
$m_3 = 2 \ldots 4$	$p_{24} = 0.9$	$\mu_3 = 1$	
$m_4 = 2$	$p_{25} = 0.1$	$\mu_4 = 2$	
$m_5 = 1$	$p_{34} = 0.9$	$\mu_5 = 1$	
	$p_{35} = 0.1$		
	$p_{40} = 0.1$		
	$p_{46} = 0.9$		
	$p_{60} = 2/3$		
	$p_{61} = 1/3$		

It is possible to extend this model to allow for machine failures and repairs.

Problem 13.8 Formulate the wafer production problem as a stochastic reward net and solve it using SPNP.

13.2.2 Polling Systems

Polling systems are very important in modeling computer and communication systems. The generic *polling model* is a system of multiple queues visited by a single server in cyclic order (see Fig. 13.33). For an exposition and performance models of polling systems, we refer the reader to [Taka90]. In this subsection we wish to illustrate the use of SPNs in automatically generating and solving the underlying CTMC for polling system performance models. Our treatment is based on the paper by Ibe and Trivedi [IbTr90].

In Fig. 13.34, the GSPN for a three-station finite population single service polling system is shown. The potential customers that can arrive at station j are indicated by tokens in place $P_{jI}, j = 1, 2, 3$. This place contains M_j tokens where M_j denotes the population size of customers that potentially arrive at the station. The firing rate of the transitions t_{lj} is marking-dependent, meaning that when the number of tokens in place P_{jI} is k, then the firing rate of t_{lj} is given by $\lambda_j k, 0 \leq k \leq M_j$. A marking-dependent firing rate is represented by the # symbol and placed next to transition t_{lj}. A token in place P_{jP} represents the condition that the server is polling station j; a token in place P_{jB} represents the condition that a customer has arrived at station j, and P_{jS} represents the condition that the server has arrived at station j.

The server commences polling station $j+1$ after finishing service at station j or found no customer there when it arrived. Whenever the timed transition t_{lj} fires, a customer arrives at station j. The server has finished polling station j when the timed transition t_{rj} fires. Such a GSPN can be easily input to any of the SPN tools such as SHARPE or SPNP, whereby the underlying CTMC will be automatically generated and solved for steady-state or transient

```
/* Nodes */
<1..5> NODE Buffer_#[K];
<1..5> NODE Active_#[m_#];
       NODE Station_6 [K];
       NODE num  [K];

/* Rules */
       FROME TO Buffer_1, num W lambda;
<1..5>   FROM Buffer_# TO Active_#;

<1..m_1> FROM Active_1 TO Buffer_2  W #*mu_1 P p12 IF Active_1 == #;
<1..m_1> FROM Active_1 TO Buffer_3  W #*mu_1 P p13 IF Active_1 == #;

<1..m_2> FROM Active_2 TO Buffer_4  W #*mu_2 P p24 IF Active_2 == #;
<1..m_2> FROM Active_2 TO Buffer_5  W #*mu_2 P p25 IF Active_2 == #;

<1..m_3> FROM Active_3 TO Buffer_4  W #*mu_3 P p34 IF Active_3 == #;
<1..m_3> FROM Active_3 TO Buffer_5  W #*mu_3 P p35 IF Active_3 == #;

<1..m_4> FROM Active_4, num TOE     W #*mu_4 P p40 IF Active_4 == #;
<1..m_4> FROM Active_4 TO node_6    W #*mu_4 P p46 IF Active_3 == #;

         FROM Active_5 TO Buffer_1  W mu_5;
         FROM Station_6, num TOE    P p60;
         FROM Station_6 TO Buffer_1 P p61;

/* Results */
/* Mean number of active machines */
<1..5>   RESULT >> A_# = MEAN Active_#;

/* Utilization of the machines */
<1..5>   RESULT >> rho_# = A_#/m_#;

/* Mean buffer length */
<1..5>   RESULT >> Q_# = MEAN Buffer_#;

/* Mean number of wafers (WIP) */
         RESULT >> WIP = MEAN num;

/* Mean system time */
         RESULTS >> T = WIP / lambda;
```

Fig. 13.31 MOSEL specification of the wafer production system.

behavior, and resulting performance measures calculated. We refer the reader to [STP96] and [IbTr90].

Other kinds of polling schemes have also been considered by [IbTr90]. In the gated service polling system, only those customers are served by the server that arrive prior to the server's arrival at station j. The SRN model of station 1 in a finite population gated service polling system is shown in Fig. 13.35. The difference between the GSPN model of the single service polling system is that now a variable multiplicity arc (indicated by the zigzag sign) is needed. Therefore, the new model is a stochastic reward net model that cannot be solved by pure GSPN solvers such as SHARPE or GreatSPN [STP96, ABC+95]. However, SPNP is capable of handling such models. For further details we refer the reader to [IbTr90] and [ChTr92].

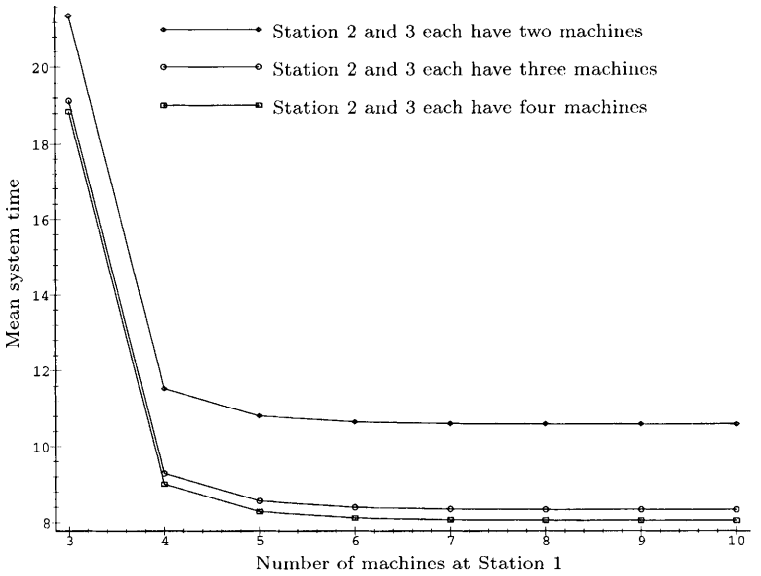


Fig. 13.32 Results for different number of machines.

The SRN (or GSPN) models can be solved using SPNP. The SPNP input file contains the specification of timed transitions and corresponding firing rates, immediate transitions, the input and output arc for each transition, and the initial number of tokens in each place. From this information SPNP generates the reachability graph, eliminates all vanishing markings, and constructs the underlying CTMC. In order to obtain the steady-state probability of each marking, a combination of SOR and the Gauss-Seidel method is used. It is possible to obtain the mean number of tokens in each place as well as more general reward-based measures. Furthermore, transient and sensitivity analysis can also be carried out.

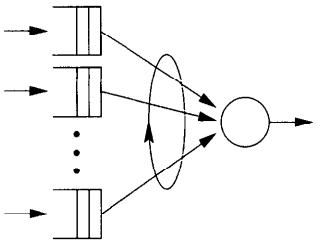


Fig. 13.33 A polling queue.

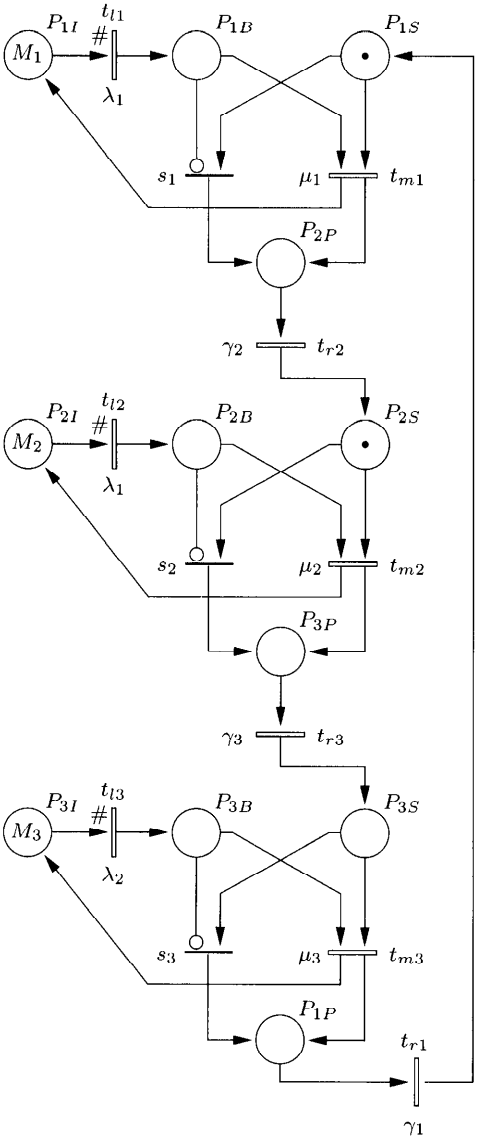


Fig. 13.34 GSPN Model for the single service polling system.

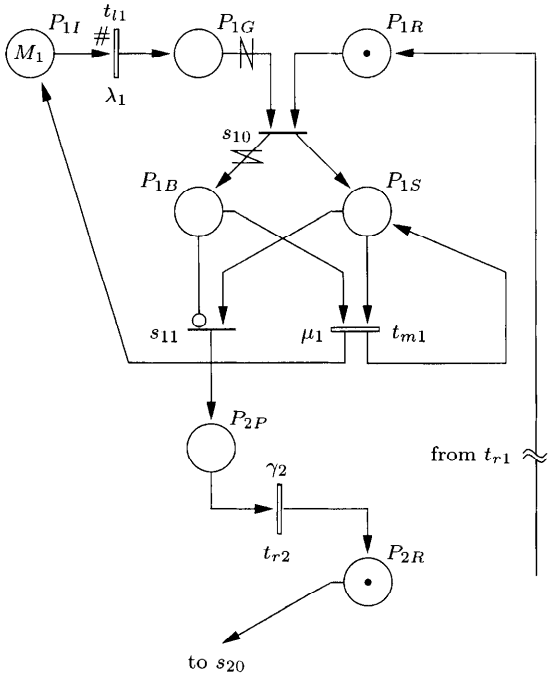


Fig. 13.35 SRN Model of gated service polling system.

Problem 13.9 Specify and solve the GSPN model of the single server polling system using SHARPE. Determine the size of the underlying CTMC as a function of the number of customers $M_1 = M_2 = M_3 = M$. Determine the steady-state and the transient performance measures.

Problem 13.10 Specify and solve the SRN model of the gated service polling system using SPNP. Determine the steady-state and transient performance measures.

Problem 13.11 Extend the polling model of Fig. 13.34 to an Erlang-3 distributed polling time. Run the resulting model on both SHARPE and SPNP.

13.2.3 Client Server Systems

Consider a distributed computing system with one file server and N workstations, interconnected by a LAN. Because the communication medium is shared by several workstations, the server and each workstation should have access to the network before it starts transmitting request/reply messages. It is assumed that a workstation does not start generating a new request until it has received the reply of its previous request. This assumption is based on the fact that in many situations future operations at a client workstation

depend on the outcome of the current operation (current service request). As long as a workstation's request is outstanding, it can continue its local processing. Our model deals only with the aspect of a user's processing that requires access to the server. Our exposition here is based on [ICT93].

In our system model, we assume that each time the server captures the access right to the network it transmits at most one reply message. The order of service at the server is FCFS.

It is important to distinguish between the client server models here and the single buffer polling system presented in Section 13.2.2. The difference is that in a single buffer polling system the station does not wait for reply to its message. Thus in a polling system, as soon as a station transmits its message, it is ready to generate a new one after the reply to the last transmitted message has been received. The interdependencies of the access control of the network and workload between the client workstations and the server also make the analysis of client server systems different from the traditional analysis of plain LANs. We consider client server systems based on a token ring network. For another model with CSMA/CD, see [ICT93].

The first step of the analysis is to find an appropriate SRN model for the system. Since there might be a confusion between the word token in a Petri net or in a token ring network, we refer to token in a ring network as *network token* and to the token associated with the SRN as the PN token. We assume that the time until a client generates a request is exponentially distributed with mean $1/\lambda$. The time for the network to move from one station to the next station (polling time) is also assumed to be exponentially distributed with mean $1/\gamma$. The processing time of a request at a server is exponentially distributed with mean $1/\mu$, and the time required for the server to process a request is assumed to be exponentially distributed with mean $1/\beta$. The system consists of N clients and one server. We note that the assumption of exponential distribution can be realized by resorting to phase-type expansions.

As can be seen in Fig. 13.36, we consider a tagged client and lump the remaining clients into one *superclient*. This superclient subsystem enables us to closely approximate $(N-1)$ multiples of the single client subsystem so that we reduce the number of states of the underlying CTMC.

The part of the SRN model that deals with the tagged client is described in Fig. 13.37. The place P_{TI} contains one PN token and represents the condition that the client is idle (in the sense the client does not generate a network-based request). The firing rate of the timed transition t_{ta} is λ. Thus, the firing of t_{ta} represents the event that the client has generated a request. The place P_{TA} represents the condition that the network token has arrived. Suppose that there is a PN token in P_{TS}. Then if there is a PN token in P_{TA}, the timed transition t_{ts} whose mean time to fire is the mean request transmission time is enabled; otherwise the immediate transition s_1 fires. The place P_{SI} represents the condition that the client's request has arrived at the server, and the place P_{TW} represents the condition that the client is waiting to receive a reply from the server. The place P_{SP} represents the condition that the client

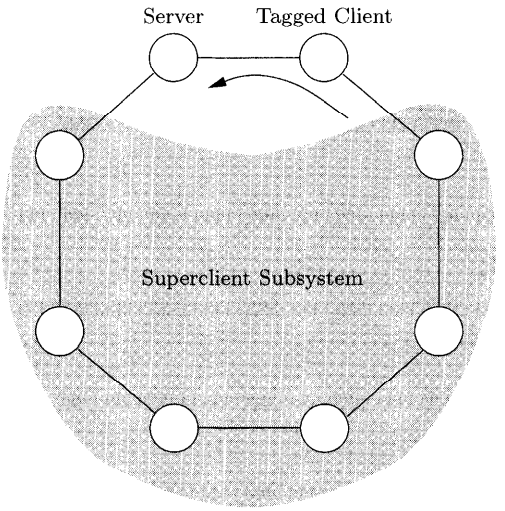


Fig. 13.36 A possible configuration of network relative to the tagged client.

has finished transmitting its request, if it is reached through t_{ts}, or the client has no request to transmit, if it is reached via s_1. The timed transition t_{sp} represents the event that the network token is being passed to its neighbor, the server in this configuration. The place P_{SS} represents the condition that the server has received the network token and can commence transmitting a reply, if one is ready.

It should be noted that in this model, a network token is released from a station as soon as it finishes transmitting its packet. This operation of the token ring network is called the *single-token* operation and is used in IEEE 802.5 token ring networks [Ibe87]. Single-token operation requires a station that has completed its packet transmission to release a network token only after its packet header has returned to the station. In the case that the round-trip delay of the ring is lower than the transmission time, the two schemes become identical.

Let us consider now the superclient subsystem. The corresponding SRN for this subsystem is shown in Fig. 13.38 and can be interpreted as follows: Place P_{OI} initially contains $N-1$ PN tokens and represents the condition that no member of the superclient subsystem has generated a request. The number of tokens can generally be interpreted as the number of idle members of the subsystem. The firing rate of the timed transition t_{oa} is marking-dependent so that if there are l PN tokens in place P_{OI}, the firing rate of t_{oa} is given by $l \cdot \lambda (0 \leq l \leq N-1)$. The places P_{OW}, P_{OS}, P_{TP}, and P_{OA} play the same role respectively as P_{TW}, P_{TS}, P_{SP} and P_{TA} of Fig. 13.37. Similarly, the transitions s_2, t_{tp}, and t_{os} correspond respectively to s_1, t_{sp}, and t_{ts} of Fig. 13.37. Furthermore, the two additional places P_{C2} and P_{C1} are introduced and used to keep a count of the number of times the PN

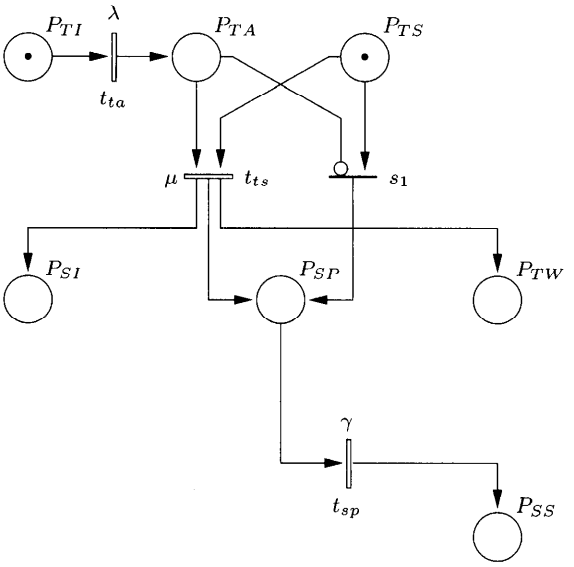


Fig. 13.37 SRN for tagged client subsystem.

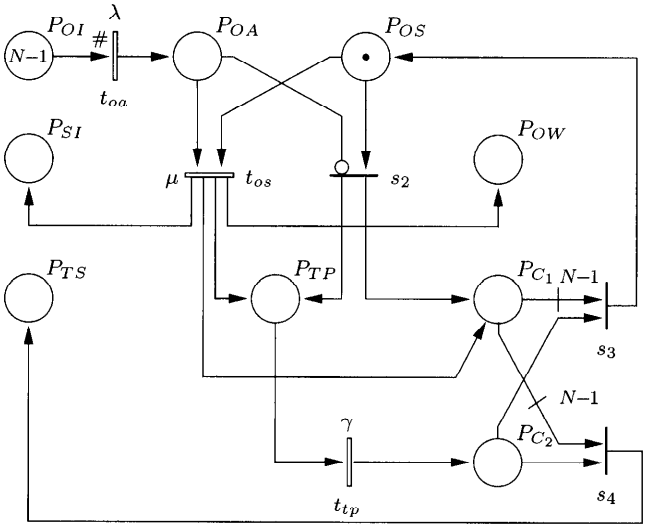


Fig. 13.38 SRN for the superclient subsystem in token ring network.

token has gone through P_{OS}. As soon as each member of the superclient subsystem has been polled, a PN token is deposited in both places P_{C1} and P_{C2}. Note that P_{C2} receives a PN token only after a walk time has been completed while P_{C1} receives the PN token immediately after a request, if one is transmitted, or prior to the commencement of the polling of the next member, if no transmission takes place. A regular arc of multiplicity $N-1$ from place P_{C1} is input to transition s_4 and an inhibitor arc of multiplicity $N-1$ is input from P_{C1} to the immediate transition s_3. Transition s_3 is enabled if a PN token is in place P_{C2} and fewer than $N-1$ PN tokens are in place P_{C1}. This represents the condition that the network token leaves the superclient subsystem if fewer than $N-1$ clients have been polled. If the number of PN tokens in place P_{C1} reaches $N-1$, implying all members of the subsystem have been polled, then with a PN token in P_{C2}, s_4 fires immediately and a PN token is deposited in P_{TS}. In this situation the tagged client now has access to the network after all members of the superclient subsystem had their chance to transmit their requests.

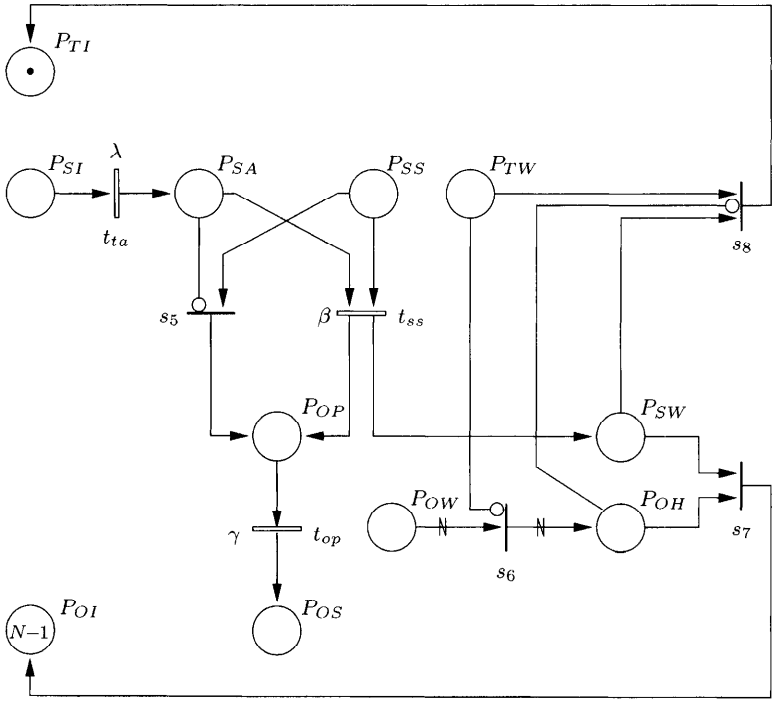


Fig. 13.39 SRN for the server subsystem.

The SRN for the subsystem is shown in Fig. 13.39. The places P_{OP}, P_{SS}, and P_{SA} serve the same purpose as P_{SP}, P_{TS}, and P_{TA} in Fig. 13.37. A token in place P_{SI} represents the condition that the server has received a request, while the place P_{TW} (P_{OW}) represents the condition that a client

from the superclient subsystem (the tagged client) is waiting for a reply to its request. The firing rates of the timed transitions t_{op}, t_{ss}, and t_{sa} are given by γ, β, and λ respectively. The condition that the server has completed serving one request is represented by the place P_{SW} and a determination is to be made regarding whose request was just serviced. In order to separate the requests from the superclient subsystem the server receives before that of the tagged client from those the server receives after that of the tagged client, the immediate transition s_6 is used. To denote that the input arc from P_{OW} to s_6 has a variable multiplicity, the zigzag sign on that arc is used.

Let us assume P_{OW} contains k PN tokens, $0 < k \leq N-1$, and P_{TW} has no token. Then the immediate transition s_6 is enabled and fires immediately by removing all k PN tokens from P_{OW} and deposits them in P_{OH}. The number of clients from the superclient subsystem whose requests were received by the server before that of the tagged client is therefore given by the marking of P_{OH}. The tagged client's request cannot be replied to until requests from the superclient subsystem that arrived before it have been replied to, or stated differently. Due to the FIFO queue at the server, the immediate transition s_8 cannot fire as long as there is a PN token in P_{OH}. As soon as the immediate transition s_7 fires, a PN token is deposited in P_{OI}, which means that a waiting member of the superclient subsystem becomes idle again. The complete SRN model for the system is shown in Fig. 13.40.

Problem 13.12 Specify and run the client server SRN using the SPNP package. Compute the average response time as a function of the number of stations N. In each case, keep track of the number of states and the number of non-zero transitions in the underlying CTMC.

Problem 13.13 Extend the previously described model to allow for an Erlang-2 distributed polling time. Solve the new SRN model using SPNP.

13.2.4 ISDN Channel

13.2.4.1 The Baseline Model In this section we present a CTMC modeling a simplified version of an ISDN channel. Typically, heterogeneous data are transported across an ISDN channel. In particular, discrete and continuous media are served at the same time, although different qualities of service requirements are imposed by different types of media on the transport system. It is well known that voice data is very sensitive to delay and delay variation (jitter) effects. Therefore, guarantees must be given for a maximum end-to-end delay bound as far as voice data is concerned. On the other hand, continuous media streams such as voice can tolerate some fraction of data loss without suffering from perceivable quality degradation. Discrete data, in contrast, are very loss sensitive but require only weak constraints as far as delay is concerned. Trade-off analysis is therefore necessary to make effective use of limited available channel bandwidth.

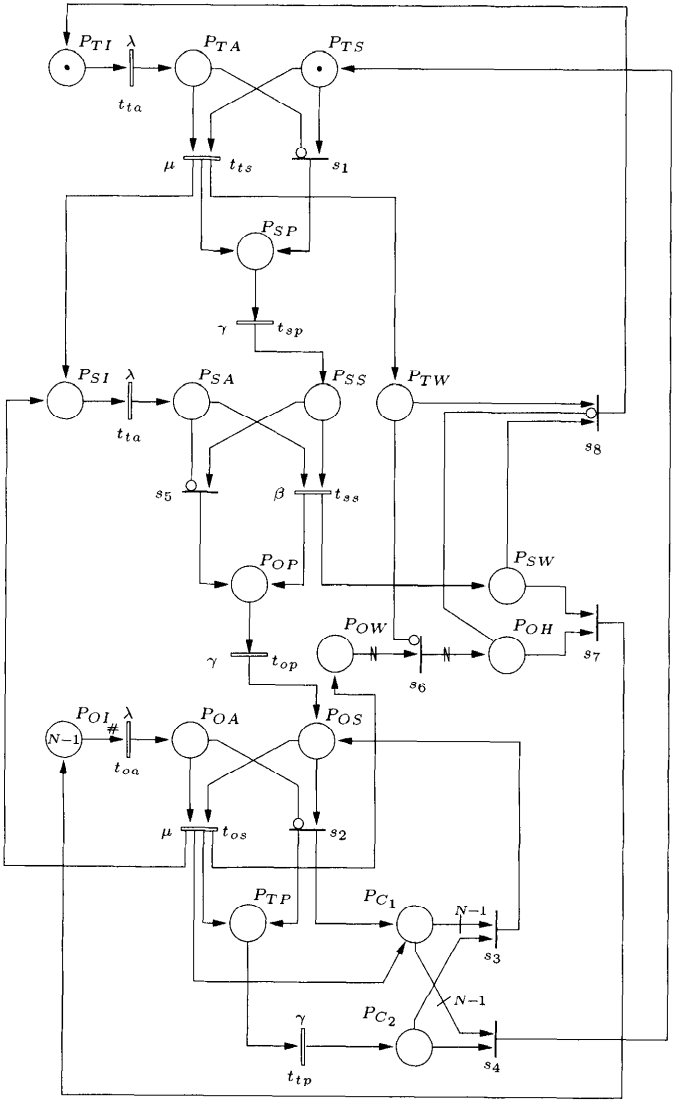


Fig. 13.40 SRN for the client server system ($N > 1$).

In our simplified scenario, a single buffer scheme is assumed that is shared between a voice stream and discrete data units. For the sake of simplicity, the delay budget available for voice packets to be spent at the channel is assumed to be just enough for a voice data unit transmission. Any queueing delay would lead to a violation of the given delay guarantee. Therefore, voice packets arriving at a non-empty system are simply rejected and discarded. Rejecting voice data, on the other hand, is expected to have a positive impact on the loss rate of data packets due to sharing of limited buffer resources. Of course, the question arises to what extent the voice data loss must be tolerated under the given scenario.

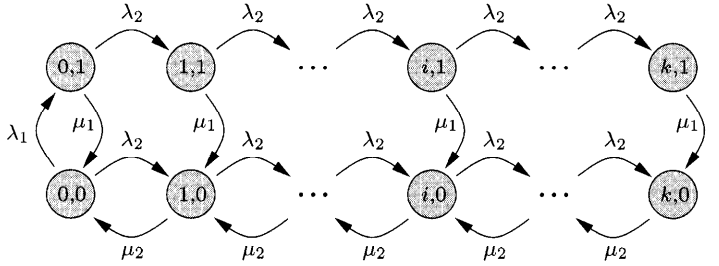


Fig. 13.41 CTMC modeling an ISDN channel with voice and discrete data.

Table 13.32 Parameter values for the ISDN channel in Fig. 13.41

Parameter	Value
λ_1	1.0
λ_2	2.0
μ_1	5.0
μ_2	10.0
k	10

For the time being, we assume Poisson arrivals for discrete data packets with parameter λ_2 and for voice data units with parameter λ_1. The transmission times of data packets are exponentially distributed with mean $1/\mu_2$ and for voice data with mean $1/\mu_1$. Buffer capacity is limited to $k = 10$. The resulting CTMC is depicted in Fig. 13.41 and the parameters summarized in Table 13.32. States are represented by pairs (i, j) where $i, 0 \leq i \leq k$ denotes the number of discrete data packets in the channel and $j, 0 \leq j \leq 1$ indicates the presence of a voice data unit in the channel.

Under these assumptions and the model shown in Fig. 13.41, some numerical computations are carried out using the SHARPE software package. First, transient and steady-state results for the rejection probability of data packets

are presented in Fig. 13.42 and for the rejection of voice data in Fig. 13.43. Comparing the results, it can be seen that the rejection probability of voice data very quickly approaches the steady-state value of a little more than 33%. Rejection probability of discrete data is much smaller, on the order of 10^{-6}, and steady-state is also reached in coarser time scale. It was assumed that the system was initially empty.

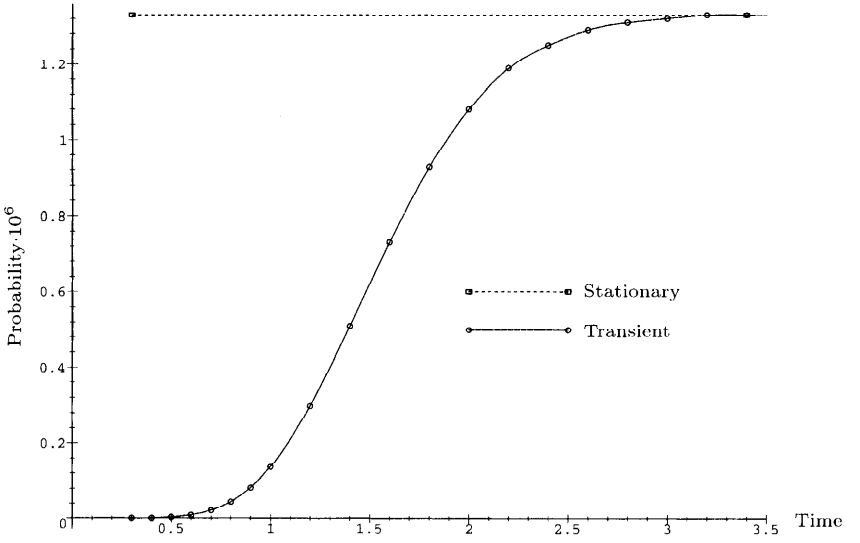


Fig. 13.42 Rejection probability for data packets.

The channel-idle probability of 66% is approached in a similar time scale as the rejection probability reaches its steady-state for voice data, as can be seen in Fig. 13.44. Often, a GSPN model specification and automated generation of the underlying CTMC is much more convenient and less error prone due to possible state space explosion. The GSPN in Fig. 13.45 is equivalent to the CTMC in Fig. 13.41 [STP96]. The GSPN was also solved using the SHARPE package.

The results of Fig. 13.46 illustrate the qualitative and quantitative difference in rejection probabilities of voice vs. discrete data. Note that rejection probabilities increase for voice data as channel capacity is increased. This effect is due to the reduced loss rate of discrete data data packets as a function of increased buffer size.

Figure 13.47 shows the impact that the arrivals of discrete data packets have on voice rejection probabilities as compared to rejection of discrete data packets themselves. Both steady-state and transient probabilities are presented. Note that voice data is much more sensitive to an increase in λ_2.

13.2.4.2 Markov Modulated Poisson Processes Markov modulated Poisson processes (MMPP) are introduced to represent correlated arrival streams and

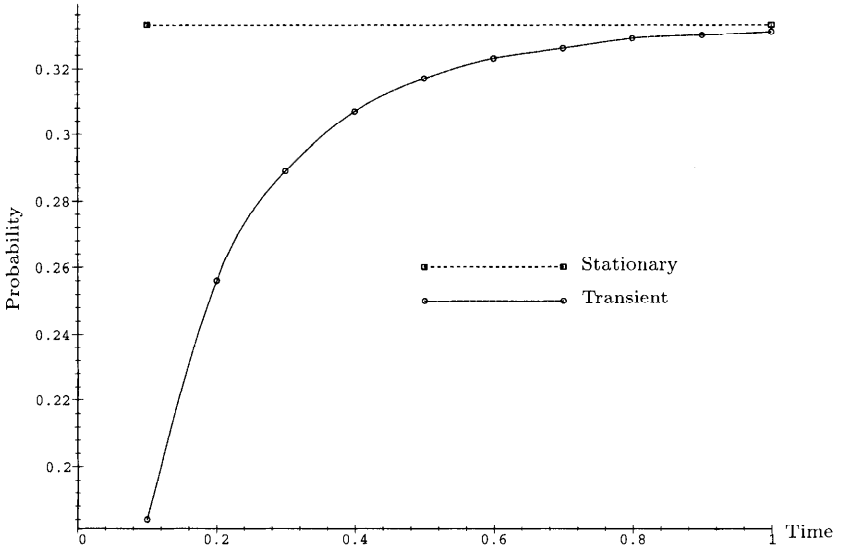


Fig. 13.43 Rejection probability for voice data units.

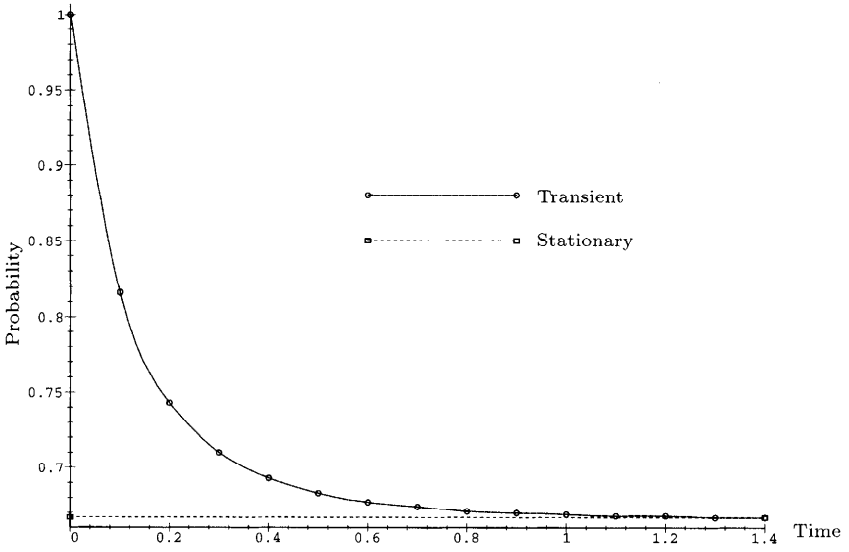


Fig. 13.44 Channel idle probability as a function of time.

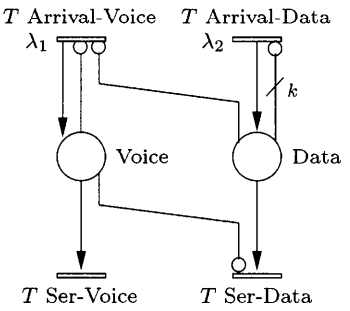


Fig. 13.45 SPN representation of the CTMC in Fig. 13.41.

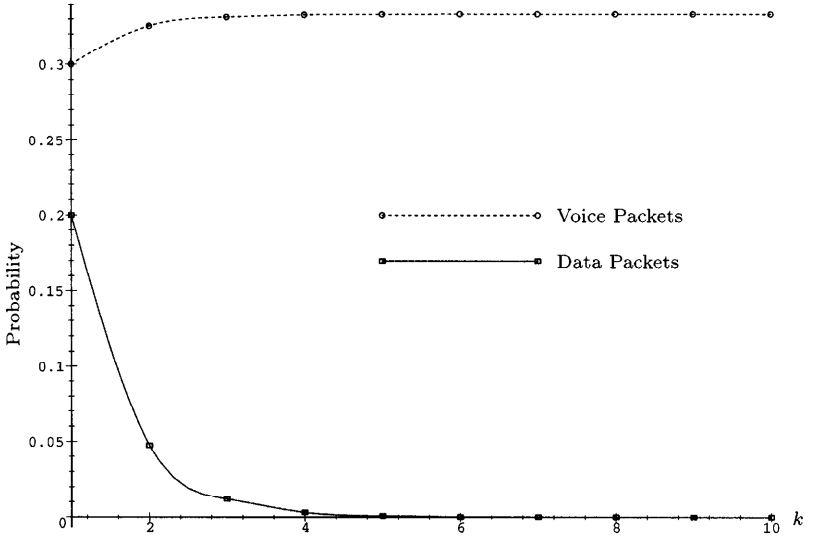


Fig. 13.46 Rejection probabilities as a function of channel capacity k: voice vs. data.

bursty traffic more precisely. An MMPP is a stochastic arrival process with an arrival rate governed by an m-state CTMC [FiMe93]. The arrival rate of an MMPP is assumed to be modulated according to an independent CTMC. The GSPN model of Fig. 13.45 is now modified so as to have two-state MMPP data arrivals as shown in Fig. 13.48 [STP96]. A token moves between the two places MMPP_1 and MMPP_2. When transition $T_{\text{arrival-data1}}$ is enabled and there are fewer than k packets transmitted across the channel, then discrete data is arriving with rate λ_{21}. Alternatively, the arrival rate is given by λ_{22}. If λ_{21} differs significantly from λ_{22}, bursty arrivals can be modeled this way. Note that the MMPP process is governed by the firing rates of T_{21} and T_{12}, which are given by rates a and b, respectively.

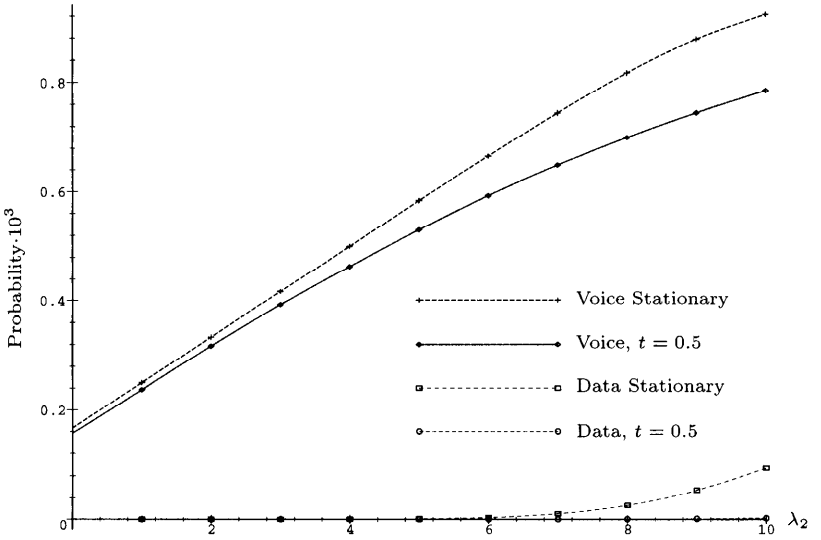


Fig. 13.47 Rejection probabilities as a function of λ_2 and time: voice vs. discrete data.

The results in Fig. 13.49 indicate that for small buffer sizes a Poisson traffic model assumption leads to more pessimistic rejection probability predictions than those based on MMPP assumptions. This effect is quickly reversed for discrete data if channel capacity is increased. For larger capacities there is hardly any difference in rejection probability for voice data, regardless of Poisson- or MMPP-type traffic model. In general, though, correlated or bursty traffic can make a significant difference in performance measures as opposed to merely Poisson-type traffic. But our modeling methods based on GSPN (and SRN), which allow us to compactly specify and automatically generate

Table 13.33 Parameters to the MMPP-based ISDN model in Fig. 13.48

Parameter	Value
λ_1	1.0
λ_{21}	9.615
λ_{22}	0.09615
μ_1	5.0
μ_2	10.0
a	8.0
b	2.0
k	10

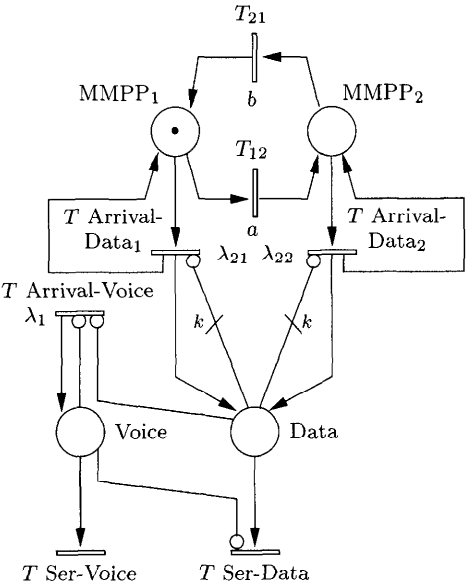


Fig. 13.48 A GSPN including an MMPP modulating data packet arrival rates.

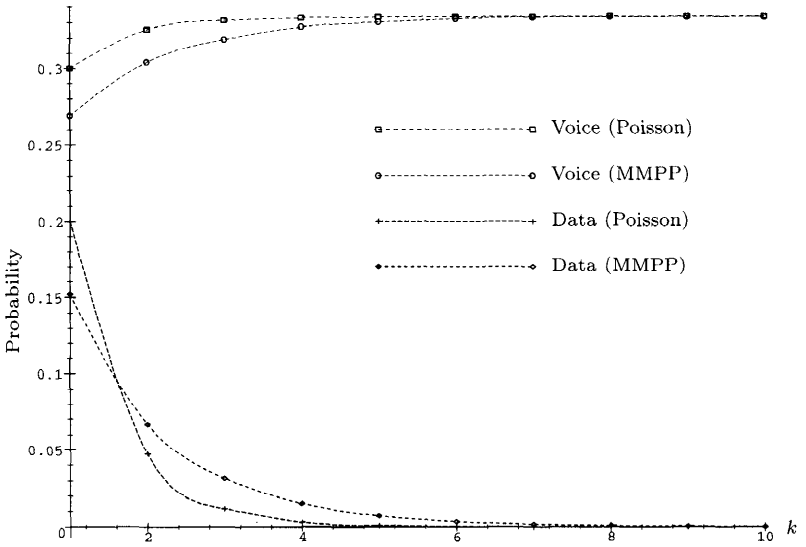


Fig. 13.49 Rejection probabilities as a function of channel capacity k: Poisson vs. MMPP.

and solve large CTMCs and MRMs for steady-state and transient behavior, are capable of handling such non-Poisson traffic scenarios.

13.2.5 ATM Network under Overload

Consider a variant of the ATM network model studied in [WLTV96], consisting of three nodes, N_1, N_2, and N_3. Two links, N_1N_3 and N_2N_3, have been established as shown in Fig. 13.50. Now suppose that the network connection

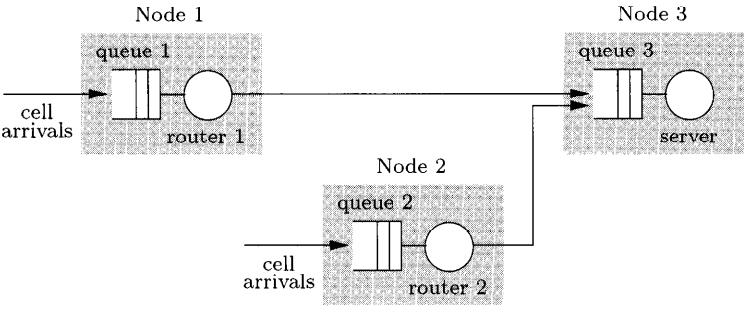


Fig. 13.50 The [WLTV96] queueing model before rerouting.

between N_1 and N_3 is down. Before the global routing table is updated, the cells destined to node N_3 arriving at node N_1 will be rerouted to node N_2 (see Fig. 13.51). Redirected cells from node N_1 may be rejected and thus lost if the input queue of node N_2 is full. Right after the rerouting starts, the cell loss probability of N_1 will overshoot because node N_2 is overloaded.

In the original paper, the burstiness of the ATM traffic is modeled by a two-state MMPP arrival process. The cell transmission time is modeled by an Erlang-5 distribution. Here we approximate the cell arrival process by a Poisson process and the cell service time by an exponential distribution. We can easily allow MMPP arrivals and Erlangian service time, but the underlying CTMC will become larger. First, we analyze the simple case before the link

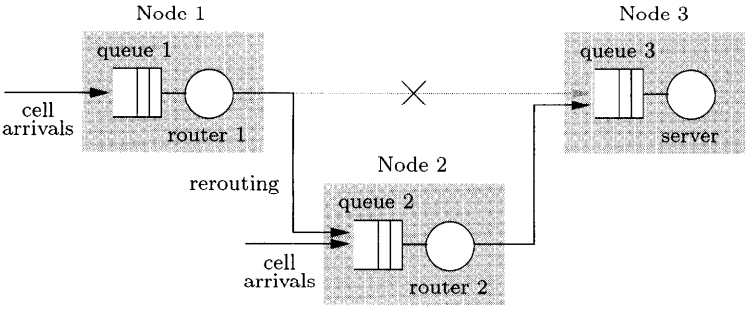


Fig. 13.51 The [WLTV96] queueing model after rerouting.

N_1N_3 breaks down. Here we assume that node N_1 and node N_2 have the same structure. That is, their buffer sizes and the service rates are the same. Furthermore, we assume that the job arrival rates are the same for the two nodes and that the buffer size at node N_3 is large enough to handle all the jobs coming from node N_1 and node N_2. Thus node N_1 and node N_2 can be modeled by a finite-buffer M/M/1/N queue, where N is the buffer size.

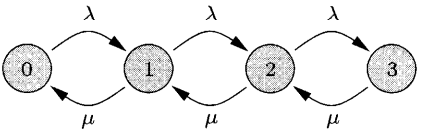


Fig. 13.52 The CTMC for the $M/M/1/3$ queue.

The CTMC and the GSPN models for the node are shown in Fig. 13.52 and Fig. 13.53 respectively. We use SHARPE to find the steady-state probabilities for each state as well as the steady-state cell loss probability before the link N_1N_3 breaks down. Figure 13.54 shows the SHARPE input file of the CTMC and the GSPN models of the M/M/1/3 queue together with their outputs. Both models give the same results for the cell loss probability.

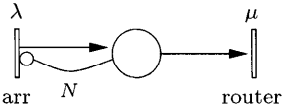


Fig. 13.53 The GSPN model for the M/M/1/N queue.

When the link N_1N_3 breaks down, the state transition diagram of the subnet can be given as shown in Fig. 13.55. Each state is identified by a pair (i, j) with $i, j \in \{0, \ldots, 3\}$, representing the number of cells in the buffers of node N_1 and node N_2. The cell arrival rates are λ_1 and λ_2 and the cell transmission rates are μ_1 and μ_2. States 30, 31, 32, and 33 represent the buffer full states for node N_1, while states 03, 13, 23, and 33 represent the buffer full states for node N_2.

The loss probability (L_1) of cells destined to node N_3 going through node N_1 is determined by two elements:

1. The probability that node N_1 is full, denoted by p_1.

2. The probability that the rerouted cells are dropped because node N_2 is full, denoted by d.

In short: $L_1 = p_1 + (1 - p_1)d$.

The loss probability (L_2) of cells destined to node N_3 going directly through node N_2 is simply the probability that node N_2 is full, which is denoted by p_2.

```
bind
lam 0.1
mu      1/2.73
N   3
end

format 6

markov noReroutingM
0  1  lam
1  2  lam
2  3  lam
3  2  mu
2  1  mu
1  0  mu
end

* steady-state probability state 0
var  p0 prob(noReroutingM,0)
expr p0

* steady-state probability state 1
var  p1 prob(noReroutingM,1)
expr p1

* steady-state probability state 2
var  p2 prob(noReroutingM,2)
expr p2

* steady-state probability state 3
var  p3 prob(noReroutingM,3)
expr p3

* steady-state cell loss probability
var  Lm prob(noReroutingM,3)
expr Lm

gspn  noReroutingG
      * places
que   0
end

* timed trans
arr       ind lam
router    ind mu
end

* immediate trans
end

* input arcs
que router 1
end

* output arcs
arr que 1
end

* inhibitor arcs
que arr N
end

* steady-state cell loss probability
var  Lg 1- util(noReroutingG,arr)
expr Lg
end

* Result:
* p0: 7.310607e-01
* p1: 1.995796e-01
* p2: 5.448523e-02
* p3: 1.487447e-02
* Lm: 1.487447e-02
* Lg: 1.487447e-02
```

Fig. 13.54 SHARPE input file for the CTMC and GSPN models of the M/M/1/3 queue together with the results.

In Figs. 13.56, 13.57, and 13.58 the SHARPE input file for the irreducible Markov reward model from Fig. 13.55 is given. From line 1 to line 6, we assign the values to the input parameters, and from lines 10 through 62, the Markov reward model is defined. Lines 10 through 26 describe the CTMC. By using loops in the specification of Markov chain transitions, we can generate the Markov chain automatically. Without the loop functionality, it would be cumbersome to specify the Markov chain manually, especially when the state space gets large. From lines 29 through 45, the reward rates $r_{i,j}$ are assigned to the states (i, j). By default, any state not assigned a reward rate is assumed to have the reward rate 0. In order to analyze the transient behavior of the system, the initial state probabilities are assigned from the values for the steady-state probabilities before the link N_1N_3 breaks down. In other words:

$$\pi_{i,j} = \pi_{s,i} \times \pi_{s,j},$$

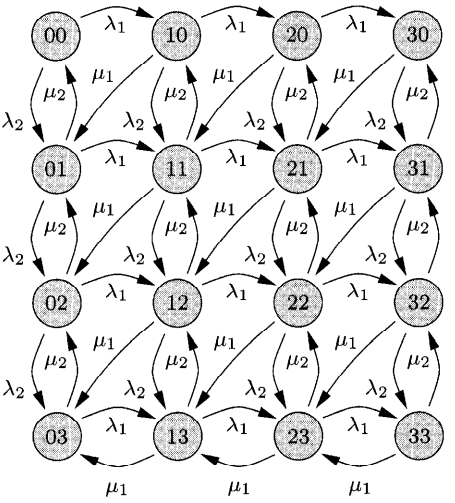


Fig. 13.55 The CTMC for Fig. 13.52 after rerouting.

where $\pi_{i,j}$ is the initial state probability for state (i, j) in the transient model. The probability $\pi_{s,i}$ is the steady-state probability of node N_1 in state i before the link breaks down. Similarly, the probability $\pi_{s,j}$ is the steady-state probability of node N_2 in state j before the link breaks down. The fact we wish to carry out transient analysis of an irreducible CTMC and hence need to assign initial state probabilities is indicated to SHARPE with the keyword `readprobs` in line 10. The assignment of the initial probabilities is done from line 46 to line 62.

Reward rates for computing the average queue length of node N_1 are written in a separate file called `reward.que1length`. In this file, we assign the reward rates to indicate the number of jobs in the first node. The contents of the file are as follows:

```
bind
r00 0       r01 0       r02 0       r03 0
r10 1       r11 1       r12 1       r13 1
r20 2       r21 2       r22 2       r23 2
r30 3       r31 3       r32 3       r33 3
end
```

In line 65, the keyword `include` is used to read the file. From lines 67 through 69, we ask for the steady-state expected queue length of N_1. This result is achieved by asking for the expected steady-state reward rate using the keyword `exrss`. In line 72, we assign the new reward rates so that the steady-state probability that node N_1 is full can be calculated. Thereafter we bind the value of $p1$ to p. Similarly, from lines 83 through 121, we assign different reward rates and ask for the steady-state average queue length of

```
bind
lam1    0.1
lam2    0.1
mu1     1/2.73
mu2     1/2.73
end

format 5

markov rerouting readprobs
loop i, 0, 2
  loop j, 0, 2
    $(j)_$(i) $(j+1)_$(i)   lam1
    $(j+1)_$(i) $(j)_$(i+1) mu1
  end
end
loop i, 0, 2
  loop j, 0, 3
    $(j)_$(i) $(j)_$(i+1)   lam2
    $(j)_$(i+1) $(j)_$(i)   mu2
  end
end
loop i, 0, 2
  $(i)_3 $(i+1)_3 lam1
  $(i+1)_3 $(i)_3 mu1
end

reward
0_0     r00
1_0     r10
2_0     r20
3_0     r30
0_1     r01
1_1     r11
2_1     r21
3_1     r31
0_2     r02
1_2     r12
2_2     r22
3_2     r32
0_3     r03
1_3     r13
2_3     r23
3_3     r33
end
0_0     0.53444974708449
0_1     0.14590480208172
0_2     0.03983201038346
0_3     0.01087414045033
1_0     0.14590480208172
1_1     0.03983201673616
1_2     0.01087414040931
1_3     0.00296864077281
2_0     0.03983201038346
2_1     0.01087414040931
2_2     0.00296864028815
2_3     0.00081043891908
3_0     0.01087414045033
3_1     0.00296864077281
3_2     0.00081043891908
3_3     0.00022124985778
end
```

Fig. 13.56 First part of input file for the Markov reward model.

```

* reward rates length queue1
include reward.que1length

var que1length exrss (rerouting)
echo steady-state queue1 length
expr que1length

* reward rates queue1 to be full
include reward.que1full

var p1 exrss (rerouting)
echo steady-state probability
     that queue1 is full
expr p1

bind
p p1
end

* reward rates for length queue2
include reward.que2length

var que2length exrss (rerouting)
echo steady-state queue2 length
expr que2length

* reward rates queue2 to be full
include reward.que2full

var p2 exrss (rerouting)
echo steady-state probability
     that queue 2 is full
expr p2

* reward rates throughput queue1
include reward.tput1

var tputa11 exrss (rerouting)

* throughput cell arrivals queue 1
bind
tputa1  tputa11
end

* reward rates throughput queue2
include reward.tput2

var tputa22 exrss (rerouting)

* throughput cell arrival queue 2
bind
tputa2  tputa22
end


* reward rates throughput router2
include reward.tputr2
```

Fig. 13.57 Second part of input file for the Markov reward model.

```
119
120  * find throughput for router 2
121  var tputr2 exrss (rerouting)
122
123  * find steady-state drop probability for rerouted cells
124  var d  (tputa2+tputa1-tputr2)/tputa1
125
126  var L1 p+(1-p)*d
127  echo steady-state loss probability of cells destined to
128  echo node N3 going through node N1
129  expr L1
130
131  bind k 3
132
133  * start the transient analysis
134  loop t , 3, 90 , 3
135
136  bind
137  * pt is transient probability that queue1 is full.
138    pt      sum(i,0,k, tvalue(t;rerouting,3_$(i)))
139
140  * p2t is transient probability that queue2 is full.
141    p2t     sum(i,0,k, tvalue(t;rerouting,$(i)_3))
142
143  * tputa2t is transient throughput queue2
144    tputa2t lam2 * ( 1 - sum(i,0,k, tvalue(t;rerouting,$(i)_3)))
145
146  * tputa1t is transient throughput for queue1
147    tputa1t lam1 * ( 1 - sum(i,0,k, tvalue(t;rerouting,3_$(i))))
148
149  * tputr2t is transient throughput for router2
150    tputr2t mu2 * (1 - sum(i,0,k, tvalue(t;rerouting,$(i)_0)))
151
152  * dt is transient drop probability for rerouted cells
153    dt      (tputa2t + tputa1t - tputr2t)/tputa1t
154
155  * L1t is loss probability of cells destined to node N3
156  * going through node N1.
157    L1t     pt +(1-pt)* dt
158
159  end
160
161  end
162
163  end
```

Fig. 13.58 Third part of input file for the Markov reward model.

node N_2, the steady-state probability for node N_2 to be full, the throughput of node N_1, node N_2, and router 2. Finally we get the cell drop probability for the rerouted cells and the loss probability of cells destined to node N_3 going through node N_1. With these steps, we can see the power of the Markov reward model: By just changing the reward rates, we can get different performance measures, such as throughput, average queue length, and rejection probability for the arriving cells.

From line 133 to line 163, we perform transient analysis of the model. This result is achieved by using the keyword `tvalue`. With `tvalue`, we can get the probability that the system is in state (i, j) at time t. We use the nested `sum` functions to add up the transient probabilities. From lines 138 through 150, we use the `evaluated word` notation. An evaluated word is made up of one

or more occurrences of $(n). Whenever the evaluated word is used, the string $(n) is expanded into an ASCII string whose digits are the number n. For example, in line 138, the state name 3_$(i) includes the states: 3_0, 3_1, 3_2, and 3_3. In Figs. 13.59 and 13.60 the results from the analysis are shown.

```
* steady-state queue1 length
  que1length:   3.53173e-01

* the steady-state probability that queue1 is full
  p1:   1.48745e-02

* steady-state queue2 length
  que2length:   8.06567e-01

* the steady-state probability that queue2 is full
  p2:   7.95456e-02

* The steady-state loss probability of cells destined to
* node N3 going through node N1
  L1:   9.00124e-02
```

Fig. 13.59 Steady state results.

```
t=3.000000
 pt       <- 0.014874
 p2t      <- 0.033728
 tputa2t  <- 0.096627
 tputa1t  <- 0.098513
 tputr2t  <- 0.144398
 dt       <- 0.515083
 L1t      <- 0.522296

t=6.000000
 pt       <- 0.014874
 p2t      <- 0.051051
 tputa2t  <- 0.094895
 tputa1t  <- 0.098513
 tputr2t  <- 0.162341
 dt       <- 0.315357
 L1t      <- 0.325541

t=9.000000
 pt       <- 0.014874
 p2t      <- 0.062675
 tputa2t  <- 0.093732
  .
  .
  .
 L1t      <- 0.092832

t=33.000000
 pt       <- 0.014874
 p2t      <- 0.079335
 tputa2t  <- 0.092067
 tputa1t  <- 0.098513
 tputr2t  <- 0.182903
 dt       <- 0.077924
 L1t      <- 0.091639

t=36.000000
 pt       <- 0.014874
 p2t      <- 0.079424
 tputa2t  <- 0.092058
 tputa1t  <- 0.098513
 tputr2t  <- 0.182962
 dt       <- 0.077225
 L1t      <- 0.090951

t=39.000000
 pt       <- 0.014874
 p2t      <- 0.079475
 tputa2t  <- 0.092052
  .
  .
  .
 L1t      <- 0.090024

t=63.000000
 pt       <- 0.014874
 p2t      <- 0.079545
 tputa2t  <- 0.092046
 tputa1t  <- 0.098513
 tputr2t  <- 0.183044
 dt       <- 0.076279
 L1t      <- 0.090019

t=66.000000
 pt       <- 0.014874
 p2t      <- 0.079545
 tputa2t  <- 0.092045
 tputa1t  <- 0.098513
 tputr2t  <- 0.183044
 dt       <- 0.076276
 L1t      <- 0.090016

t=69.000000
 pt       <- 0.014874
 p2t      <- 0.079545
 tputa2t  <- 0.092045
  .
  .
  .
 L1t      <- 0.090012
```

Fig. 13.60 Results of transient analysis.

From the results, we can see that although both router 1 and router 2 have the same cell arrival rate, the steady-state probability for node N_2 to be full is more than four times bigger than that of node N_1. This condition occurs because router 2 has two cell sources, one coming from the original cell source, the other from the rerouted cells coming from router 1. However,

the loss probability of cells destined to node N_3 going directly through N_2 is smaller than that going through N_1 first. From the transient results, we can answer questions such as:

- How long does it take to reach steady state after the line breakdown?
- What is the loss probability at a particular time before the system is in steady state?

These are important questions that need to be considered when we make performance and reliability analysis. Figure 13.61 is derived from the transient analysis.

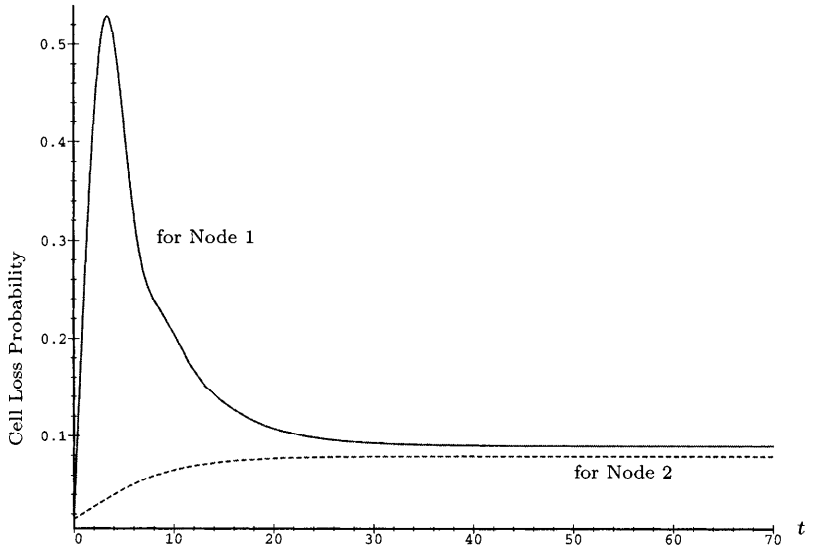


Fig. 13.61 Cell loss probability for the two nodes in the network discussed in Section 13.2.5.

Problem 13.14 Modify the model of overload described in Section 13.2.5 to allow for two-state MMPP sources and three-stage Erlangian service. You may find it convenient to use GSPN rather than a direct use of CTMC.

13.3 CASE STUDIES OF HIERARCHICAL MODELS

We discuss two examples of hierarchical models in this section. Several other hierarchical models are discussed in Chapter 10. For further exposition on hierarchical models, see [MaTr93].

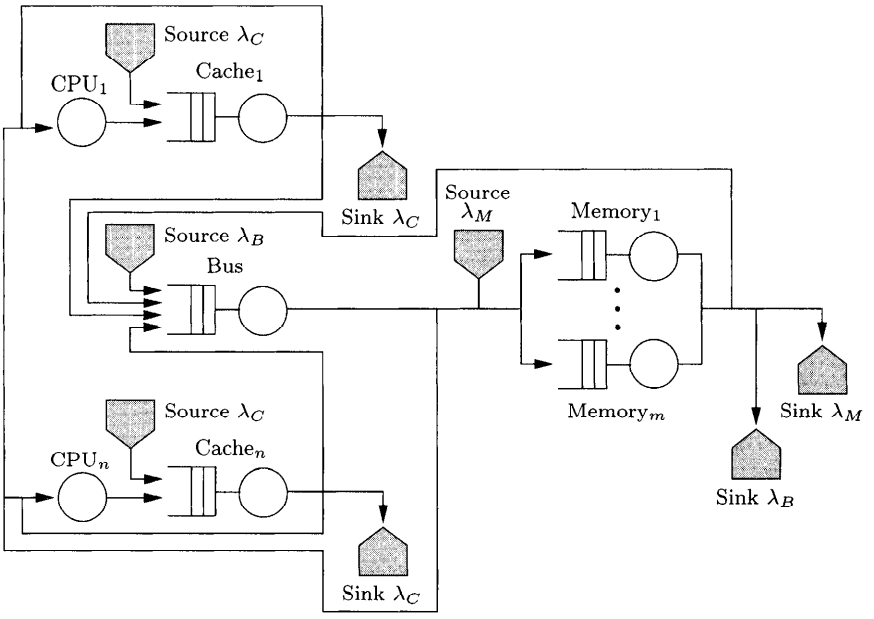


Fig. 13.62 The outer model for a multiprocessor system with cache protocols.

13.3.1 A Multiprocessor with Different Cache Strategies

Section 13.1.1 introduces different models for loosely and tightly coupled multiprocessor systems. Now we model multiprocessor systems with different cache coherence protocols. Our treatment is based on [YBL89]. The outer model is a product-form queueing network, some of whose parameters are obtained from several lower level CTMC models. The outer model shown in Fig. 13.62 (where $n = 2$ CPUs are assumed).

The model consists of open and closed job classes where closed job classes capture the flow of normal requests through the system and open job classes model the additional load imposed by the cache. For the queueing network (outer model), the following notation is used:

m	Number of memory modules
n	Number of CPUs
μ_{cpu}	Service rate at the CPU
μ_{cache}	Service rate at the cache
μ_{bus}	Service rate at the bus
μ_{mem}	Service rate at the main memory modules

and for the CTMC cache models (inner models), we use:

C	Overall number of blocks in each cache
C_{sc}	Number of shared cache blocks (blocks shared by all caches)
C_{pc}	Number of private cache blocks
C_{ic}	Number of instruction cache blocks
S	Degree of sharing
f_w	Probability that data/instructions are written
$1 - f_w$	Probability that data/instructions are read
h	Probability of a given request to a private cache being a hit
u	Probability that a previously accessed block has not been modified
d	Probability that a private block, selected for replacement, is modified and needs to be written back
p_{ic}	Probability that instructions are accessed during a memory request
$1 - p_{\text{ic}}$	Probability that a data block is accessed during a memory request; these blocks can be either shared or private

The cache behavior is modeled by the jobs of the open job classes. These additional customers induce additional traffic:

- The traffic at the cache due to requests from other caches (loading of missed blocks).
- The requests from other processors due to state updates.
- Additional bus traffic due to invalidation signals.

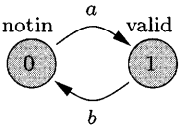


Fig. 13.63 CTMC for an instruction cache block (inner model 1).

If we access blocks in a cache, we have to differentiate between accessing private blocks, shared blocks, and instructions. Private blocks can be owned only by one cache at a time. No other cache is allowed to have a copy of these blocks at the same time. Since it is possible to read and modify private blocks, they have to be written back to the main memory after being modified. In order to keep shared block consistent, a cache coherence protocol needs to be used because shared blocks can be modified by any other processor that also owns this block. Items in the instruction cache can only be read but never be

modified. Therefore no cache coherence protocol is necessary for instruction blocks. Now we develop and solve three inner models: For an instruction cache block, for a private cache block, and for a shared cache block. A block in the *instruction cache* can either be not in the cache or valid. If a block is selected for replacement, it is overwritten (no write back is necessary) since these blocks can not be modified. This assumption leads to the CTMC, shown in Fig. 13.63.

The transition rates are given by:

$$a = \frac{(1-f_w)p_{\text{ic}}}{C_{\text{ic}}}\mu_{\text{cpu}}, \quad b = \frac{1-h}{C}\mu_{\text{cpu}},$$

with $C_{\text{ic}} = p_{\text{ic}}C$. Steady-state probabilities of this simple CTMC are easily obtained:

$$\pi_0^{\text{ic}} = \frac{b}{a+b}, \qquad \pi_1^{\text{ic}} = \frac{a}{a+b}. \tag{13.11}$$

Since *private cache blocks* can also be modified, they have to be written back

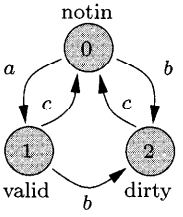


Fig. 13.64 CTMC for a private cache block (inner model 2).

after being modified. Modified blocks are called *dirty*. The corresponding state diagram is shown in Fig. 13.64. The transition rates are given by:

$$a = (1-p_{\text{ic}})(1-S)\frac{1-f_w}{C_{\text{pc}}}\mu_{\text{cpu}},$$

$$b = (1-p_{\text{ic}})(1-S)\frac{f_w}{C_{\text{pc}}}\mu_{\text{cpu}},$$

$$c = \frac{1-h}{C}\mu_{\text{cpu}},$$

with $C_{\text{pc}} = (1-p_{\text{ic}}(1-S)C$. Steady-state probabilities of this CTMC are:

$$\pi_0^{\text{pc}} = \frac{c}{a+b+c}, \qquad \pi_1^{\text{pc}} = \frac{ac}{(a+b+c)(b+c)}, \qquad \pi_2^{\text{pc}} = \frac{b}{b+c}. \tag{13.12}$$

For private and instruction cache blocks no coherence protocol is necessary, while for *shared blocks* a *cache coherence protocol* must be used since these blocks can become inconsistent in different caches. One of the first cache

coherence protocols found in the literature is Goodman's write-once scheme [Good83]. When using this protocol, the cache blocks can be in one of five states:

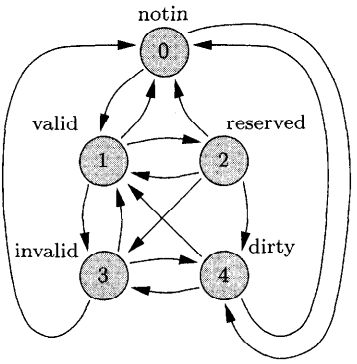


Fig. 13.65 CTMC for shared cache blocks (inner model 3).

1. Notin: the block is not in the local cache.
2. Invalid: the block is in the local cache but not consistent with other caches.
3. Valid: the block is in the local cache and consistent with the main memory.
4. Reserved: the block is in the local cache, consistent with the main memory, and not in any other cache.
5. Dirty: the block is in the local cache, inconsistent with the main memory, and not in any other cache.

If a read miss to a shared cache block occurs and a block in another cache exists that is in state dirty, this cache supplies the block as well as writing it back to the main memory. Otherwise (no cache has a dirty copy), each cache with a copy of that block sets the block state to valid. In case of a write miss the block is loaded from memory or, if the block is dirty, the block is loaded from the cache that has a dirty copy (this cache then invalidates its copy). When a write hit to an already dirty or reserved cache block occurs, the write proceeds locally, and if the state is valid, the block is written. In case of a write hit to an invalid cache block, the local state of the cache block is changed. Finally, if a block is loaded into the cache, it can happen that there is not enough space in the cache and a block in the cache is to be selected for replacement. Blocks are replaced in the order invalid, dirty, reserved, and valid.

The state diagram of the CTMC for shared cache blocks is shown in Fig. 13.65. Transition rates for this CTMC are:

- The processor under consideration generates a read request to a shared block:

$$\text{notin} \rightarrow \text{valid} : \frac{S(1-f_w)(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}},$$

$$\text{invalid} \rightarrow \text{valid} : \frac{S(1-f_w)(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}}.$$

  If the cache block is in state dirty, reserved, or valid, the read request can proceed without any state change and bus transaction.

- Another processor generates a read request to a shared block:

$$\text{reserved} \rightarrow \text{valid} : \frac{(n-1)S(1-f_w)(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}},$$

$$\text{dirty} \rightarrow \text{valid} : \frac{(n-1)S(1-f_w)(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}}.$$

  If the cache block is in the states valid, invalid, or notin, no state change occurs.

- The processor under consideration generates a write request to a shared block:

$$\text{valid} \rightarrow \text{reserved} : \frac{(n-1)Sf_w(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}},$$

$$\text{reserved} \rightarrow \text{dirty} : \frac{(n-1)Sf_w(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}},$$

$$\text{invalid} \rightarrow \text{dirty} : \frac{(n-1)Sf_w(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}},$$

$$\text{notin} \rightarrow \text{dirty} : \frac{Sf_w(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}}.$$

  If a block is alrcady dirty, the write request proceeds locally without any state change and bus transaction.

- Another processor generates a read request to a shared block:

$$\text{dirty} \rightarrow \text{invalid} : \frac{(n-1)Sf_w(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}},$$

$$\text{reserved} \rightarrow \text{invalid} : \frac{(n-1)Sf_w(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}},$$

$$\text{valid} \rightarrow \text{invalid} : \frac{(n-1)Sf_w(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}}.$$

- A write or read miss in the local cache can result in a page replacement, which leads to the following state transitions:

$$\text{reserved} \rightarrow \text{notin} : \frac{1-h}{C}\mu_{\text{cpu}},$$

$$\text{dirty} \rightarrow \text{notin} : \frac{1-h}{C}\mu_{\text{cpu}},$$

$$\text{valid} \rightarrow \text{notin} : \frac{1-h}{C}\mu_{\text{cpu}},$$

$$\text{invalid} \rightarrow \text{notin} : \frac{1-h}{C}\mu_{\text{cpu}}.$$

Let:

$$a = \frac{1-h}{C}\mu_{\text{cpu}}, \quad b = \frac{S(1-f_w)(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}}, \quad c = \frac{Sf_w(1-p_{\text{ic}})}{C_{\text{sc}}}\mu_{\text{cpu}},$$

with $C_{\text{sc}} = (1-p_{\text{ic}})S \cdot C$. Then the steady-state probabilities of the CTMC are given by:

$$\begin{aligned}
\pi_0^{\text{sc}} &= \frac{c}{a+b+c}, \\
\pi_1^{\text{sc}} &= \frac{a(\pi_0^{\text{sc}} + \pi_3^{\text{sc}}) + (n-1)a(1-\pi_0^{\text{sc}} - \pi_3^{\text{sc}})}{(n-1)a+nb+c}, \\
\pi_2^{\text{sc}} &= \frac{b\pi_1^{\text{sc}}}{(n-1)a+nb+c}, \\
\pi_3^{\text{sc}} &= \frac{b(n-1)(1-\pi_0^{\text{sc}})}{a+nb+c}, \\
\pi_4^{\text{sc}} &= 1 - \pi_0^{\text{sc}} - \pi_1^{\text{sc}} - \pi_2^{\text{sc}} - \pi_3^{\text{sc}}.
\end{aligned} \tag{13.13}$$

Now we are ready to parameterize the outer model. To analyze the outer model given in Fig. 13.62, we need the visit ratios for each station and the arrival rates of the open job classes. We see from the following that these parameters are functions of the state probabilities of the three inner models:

Visit Ratio for a Remote Cache i: A cache miss is either supplied by a remote cache that has a dirty copy or by one of the main memory modules if no dirty copy exists. Because the probability of a dirty and valid copy are $1-(1-\pi_4^{\text{sc}})^{n-1}$ and $1-(1-\pi_1^{\text{ic}})^{n-1}$, respectively, the visit ratio of the remote cache i is given by:

$$\begin{aligned}
e_{C_i} = &\frac{(1-p_{\text{ic}})S(\pi_0^{\text{sc}} + \pi_3^{\text{sc}})(1-(1-\pi_4^{\text{sc}})^{n-1})}{n-1} \\
&+ \frac{p_{\text{ic}}\pi_0^{\text{ic}}(1-(1-\pi_1^{\text{ic}})^{n-1})}{n-1}.
\end{aligned}$$

Visit Ratio to Main Memory Module i: The main memory is involved when a cache miss to a private block occurs, a cache miss to a shared block occurs and no other cache has a dirty copy, and a cache miss to instructions occurs and no other cache contains these instructions. Hence the visit ratio is:

$$e_{M_i} = \frac{(1-p_{\text{ic}})(1-S)(1-h) + (1-p_{\text{ic}})S(\pi_0^{\text{sc}} + \pi_3^{\text{sc}})(1-\pi_4^{\text{sc}})^{n-1}}{m} + \frac{p_{\text{ic}}\pi_0^{\text{ic}}(1-\pi_1^{\text{ic}})^{n-1}}{m}.$$

Visit Ratio to the Bus: The bus is used to broadcast invalidations and to transmit missed blocks. Hence the visit ratio is:

$$e_B = 2(1-p_{\text{ic}})S(\pi_0^{\text{sc}} + \pi_3^{\text{sc}}) + 2(1-p_{\text{sc}})(1-S)(1-h) + (1-p_{\text{sc}})Sf_w\pi_1^{\text{sc}} + (1-p_{\text{sc}})(1-S)f_w u + p_{\text{ic}}\pi_0^{\text{ic}}.$$

Once we know the visit ratios and the structure of the network, we can compute the routing probabilities, using Eq. (7.5). As a result, we get for the routing probabilities:

$$p_{\text{cache}_i \to \text{proc}_i} = \frac{(e_{C_i} - e_B) + \sum_{j=1}^{m} e_{M_j}}{e_{C_i}}, \quad \forall i = 1, \ldots, n,$$

$$p_{\text{cache}_i \to \text{bus}} = \frac{e_B - \sum_{j=1}^{m} e_{M_j}}{e_{C_i}}, \quad \forall i = 1, \ldots, n,$$

$$p_{\text{bus} \to \text{proc}_i} = \frac{e_B - \sum_{j=1}^{m} e_{M_j}}{e_B}, \quad \forall i = 1, \ldots, n,$$

$$p_{\text{bus} \to \text{mem}_j} = \frac{e_{M_j}}{e_B} \quad \forall j = 1, \ldots, m.$$

Next we obtain the arrival rates for each open customer class:

Derivation of λ_C: The possible customers of a local cache are the requests from its owner processor $(1-p_{\text{ic}})(1-S)(1-h)$, the loading of missed blocks $(1-p_{\text{ic}})S(\pi_0^{\text{sc}} + \pi_3^{\text{sc}}) + p_{\text{ic}}\pi_0^{\text{ic}}$, the requests from other processors for state updating due to a write request to a shared block $(n-1)(1-p_{\text{ic}})Sf_w(\pi_1^{\text{sc}} + \pi_2^{\text{sc}})$, and requests from other processors for state updating due to a read request for a reserved block $(n-1)(1-p_{\text{ic}})S(1-f_w)\pi_2^{\text{sc}}$. Therefore the open customer arrival rate, denoted by λ_C, is given by:

$$\lambda_C = \rho_i \mu_{cpu_i} \big((1-p_{\text{ic}})(1-S)(1-h) + (1-p_{\text{ic}})S(\pi_0^{\text{sc}} + \pi_3^{\text{sc}}) + p_{\text{ic}}\pi_0^{\text{ic}} + (n-1)(1-p_{\text{ic}})Sf_w(\pi_1^{\text{sc}} + \pi_2^{\text{sc}}) + (n-1)(1-p_{\text{ic}})S(1-f_w)\pi_2^{\text{sc}}\big). \tag{13.14}$$

Derivation of λ_B: A selected block needs to be written back to main memory only if it is private and modified or shared and dirty. The probability for a shared block to be selected is given by $C_{\text{sc}}(1-\pi_0^{\text{sc}})/C$ and the probability for a private block to be selected is given by $C_{\text{pc}}(1-\pi_0^{\text{pc}})/C$. The selected block is written back if it is private and modified or shared and dirty. The arrival rate at the bus is, therefore, given by:

$$\begin{aligned}\lambda_B =& \rho_i \mu_{cpu_i} n\big((1-p_{\text{ic}})(1-S)(1-h) + (1-p_{\text{ic}})S\pi_0^{\text{sc}}\big) \\ & \cdot \left(\frac{C_{\text{pc}}(1-\pi_0^{\text{pc}})}{C} d + \frac{C_{\text{sc}}(1-\pi_0^{\text{sc}})}{C}\pi_4^{\text{sc}}\right).\end{aligned} \tag{13.15}$$

Derivation of λ_M: There are two possibilities: A shared read results in a miss and one of the remote caches has a dirty copy $(1-p_{\text{ic}})S(1-f_w)(\pi_0^{\text{sc}}+\pi_3^{\text{sc}})(1-(1-\pi_4^{\text{sc}})^{n-1})$, or a write hit to a valid shared block occurs and a write through operation is performed $(1-p_{\text{ic}})Sf_w + (1-p_{\text{ic}})(1-S)f_w u$. The arrival rate at the memory modules is, therefore, given by:

$$\begin{aligned}\lambda_M =& \rho_i \mu_{cpu_i} n\big((1-p_{\text{ic}})S(1-f_w)(\pi_0^{\text{sc}}+\pi_3^{\text{sc}})(1-(1-\pi_4^{\text{sc}})^{n-1}) \\ & + (1-p_{\text{ic}})Sf_w + (1-p_{\text{ic}})(1-S)f_w u\big).\end{aligned} \tag{13.16}$$

Because the arrival rates λ_C, λ_B, and λ_M are functions of server utilizations that are computed from the solution of the outer model, the outer model needs to be solved using fixed-point iteration. The solution algorithm is sketched in the following:

STEP 1 Solve the inner model for shared/private/instruction cache blocks to obtain the steady-state probabilities (see Eqs. (13.11), (13.12), and (13.13)).

STEP 2 Determine the arrival rates λ_C (Eq. (13.14)), λ_B (Eq. (13.15)), and λ_M (Eq. (13.16)) of the open customer classes, assuming the initial value $\rho_i = 1$ and using the steady-state probabilities of the inner model.

STEP 3 Solve the outer model using the computed arrival rates to obtain new values for the utilizations ρ_i.

STEP 4 The computed utilizations are used to determine the new arrival rates.

STEP 5 The last two steps are repeated until the results of two successive iteration steps differ less than ϵ.

For our example we assume the following parameter values (all time units are msec):

$$\begin{array}{lll} C = 2048 & \mu_{\text{cpu}} = 0.25 & h = 0.98 \\ S = 0.1 & \mu_{\text{cache}} = 1.0 & f_w = 0.3 \\ u = 0.1 & \mu_{\text{bus}} = 1.0 & p_{\text{ic}} = 0.1 \\ d = 0.4 & \mu_{\text{mem}} = 0.25 & \end{array}$$

Table 13.34 Performance measures for the cache model with $\mu_{\text{bus}} = 1.00$

No. of Processors	ρ_{cpu}	ρ_{bus}	System Power
2	0.475	0.356	0.950
3	0.454	0.426	1.278
4	0.438	0.495	1.754
5	0.425	0.562	2.127
6	0.413	0.619	2.478
7	0.402	0.679	2.817
8	0.391	0.734	3.131
9	0.377	0.785	3.395
10	0.363	0.821	3.634
15	0.279	0.902	4.180
20	0.219	0.902	4.374

In Tables 13.34 and 13.35 the performance measures for different mean bus service times are given as a function of the number of processors. Here, the measure *system power* is the sum of all processor utilizations. In Table 13.34 we see that by increasing the number of processors from 2 to 20, the system power increases by a factor of 4.6. Because the system bottleneck is the bus, it makes more sense to use a faster bus or to use a dual bus instead of increasing the number of processors. In Table 13.35 the results are shown for a faster bus with $\mu_{\text{bus}} = 1.5$. As a result, the system power is increased by a factor of 6.5.

Table 13.35 Performance measures for the cache model with $\mu_{\text{bus}} = 1.50$

No. of Processors	ρ_{cpu}	ρ_{bus}	System Power
2	0.511	0.255	1.023
3	0.486	0.303	1.458
4	0.470	0.354	1.882
5	0.459	0.404	2.297
6	0.451	0.451	2.707
7	0.445	0.501	3.115
8	0.439	0.549	3.515
9	0.432	0.601	3.892
10	0.427	0.643	4.269
15	0.379	0.817	5.688
20	0.331	0.909	6.614

13.3.2 Performability of a Multiprocessor System

We consider a model that was developed by Trivedi, Sathaye, et al.[TSI+96] to help determine the optimal number of processors needed in a multiprocessor system. A Markov reward model is used for this purpose. In this model, it is assumed that all failure events are mutually independent and that a single repair facility is shared by all the processors. Assume that the failure rate

of each processor is α. A processor fault is covered with probability c and is not covered with probability $1-c$. Subsequent to a covered fault, the system comes up in a degraded mode after a brief reconfiguration delay, while after an uncovered fault, a longer reboot action is required. The reconfiguration times are assumed to be exponentially distributed with mean $1/\gamma$, the reboot times are exponentially distributed with mean $1/\delta$, and the repair times are exponentially distributed with mean $1/\beta$. It is also assumed that no other event can take place during a reconfiguration or a reboot phase. The CTMC modeling the failure/repair behavior of this system is shown in Fig. 13.66. In state i, $1 \leq i \leq n$, the system is up with i processors functioning, and $n-i$ processors are waiting for repair. In states (c_{n-i}), $i = 0, \cdots, n-2$, the system is undergoing a reconfiguration. In states (b_{n-i}), $i = 0, \cdots, n-2$, the system is being rebooted. In state 0, the system is down waiting for all processors to be repaired.

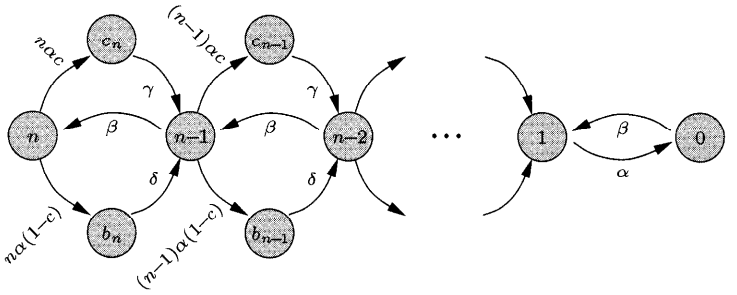


Fig. 13.66 CTMC for computing the performability of a multiprocessor system.

The reward rate in a state with i processors functioning properly correspond to some measure of performance in that configuration. Here the loss probability is used as a performance measure. An M/M/i/b queueing model is used to describe the performance of the multiprocessor system, which is represented by the PFQN shown in Fig. 13.67. This model contains two stations. station *mp* is the processor station with i processors; it has type *ms* for multiple servers, each server having service rate μ. The other station is *source*, which represents the job source with rate λ. Because there is a limited number b of buffers available for queueing the jobs, the closed product-form network with a fixed number b of jobs is chosen.

The loss probability can be obtained via the throughput *tput* of station *mp*. Note that we have used a two-level hierarchical model. The lower level (inner model) is the PFQN (Fig. 13.67), while the outer model is the CTMC (Fig. 13.66). For state i of the CTMC, the PFQN with i processors is evaluated, and the resulting loss probability is used as a reward rate in state i. Then the expected steady-state reward rate is the overall performability measure of

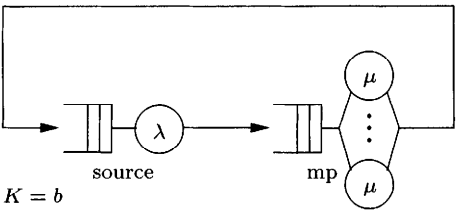


Fig. 13.67 PFQN model of the M/M/i/b queue.

the system, given by

$$\text{EXRSS} = \sum_{i=1}^{n} r_i \pi_i,$$

where π_i is the steady-state probability of state i, and r_i is the loss probability of station mp. We consider just the operational state i because in the down states c_i, b_i and 0, the loss probability (reward rate) is 1.

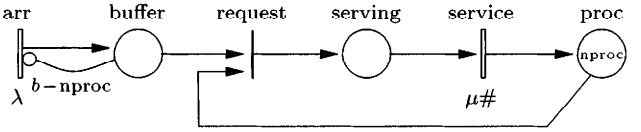


Fig. 13.68 GSPN model of the M/M/i/b queue.

It is also possible to model the lower level model using the GSPN shown in Fig. 13.68. The initial number *nproc* of tokens in place *proc* means that there are *nproc* processors available. When a new job arrives in place *buffer*, a token is taken from place *proc*. Jobs arrive at the system when transition *arr* fires. There is a limitation for new jobs entering the system caused by the inhibitor arc from place *buffer* to transition *arr*. Thus *arr* can only fire when the system is not already full. There can be only b jobs in the system altogether, *nproc* being served (in place *serving*) and $b-nproc$ in place *buffer*. The firing rates are λ for transition *arr* and $k\mu$ for transition *service*. Here k is the number of tokens in place *serving*; the notation for this marking-dependent firing rate and in Fig.13.68 is $\mu\#$. The expected reward rate in the steady state is called the *total loss probability* (TLP). It is defined as the fraction of jobs rejected either because the buffer is full or the system is down.

A possible SHARPE input file for this hierarchical performability model is shown in Fig. 13.69. Descriptions of the lines of the SHARPE input file are as follows:

Line 3-9: Assign the values to the input parameters.

```
1  format 8
2  bind
3  gamma 1/6000
4  beta 12
5  tau 1.0
6  delta 360
7  mu 100
8  C 0.98
9  b 10
10 end
11
12 gspn queue (nproc,lambda)
13 * places
14 buffer 0
15 serving 0
16 proc nproc
17 end
18 * timed transitions
19 arr ind lambda
20 service dep serving mu
21 end
22 * immediate transitions
23 request ind 1
24 end
25 * input arcs
26 buffer request 1
27 serving service 1
28 proc request 1
29 end
30 * output arcs
31 arr buffer 1
32 request serving 1
33 service  proc 1
34 end
35 * inhibitor arcs
36 buffer arr b-nproc
37 end
38
39 func R(nproc,L) \
40   1-util(queue,arr;nproc,L)
41
42 markov one(L)
43 1 0 gamma
44 0 1 tau
45 reward
46 1 R(1,L)
47 0 1
48 end
49
50 markov two(L)
51 2 x2 2* gamma*C
52 2 y2 2* gamma*(1-C)
53 x2 1 delta
54 y2 1 beta
55 1 2 tau
56 1 0 gamma
57 0 1 tau
58 reward
59 2 R(2,L)
60 x2 1
61 1 R(1,L)
62 0 1
63 end
64
65 include markov.3
66 include markov.4
67 * ...
68
69 loop L,50,200,50
70    expr exrss(one;L)
71    expr exrss(two;L)
72    expr exrss(three;L)
73    expr exrss(four;L)
74 * ...
75 end
76 end
```

Fig. 13.69 Input file for multiprocessor performability model.

Line 12-37: Define the GSPN model; the model has two parameters, the number of processors *nproc* and the job arrival rate *lambda*.

Line 39-40: Define a function to use for reward rates.

Line 42-48: Specification of a Markov model for $nproc = 1$ processor.

Line 46-47: Assign reward rates to all states.

Line 50-63: Specification of a Markov model for $nproc = 2$ processors.

Line 59-62: Assign reward rates to all states.

Line 65: Input file with specification of a Markov model for 3 processors.

Line 66: Input file with specification of a Markov model for 4 processors.

Line 67: Define models for more processors.

Line 69: Let the arrival rate L vary from 50 to 200 by increments of 50.

Line 70-75: Get the TLP for each number of processors.

In Fig. 13.70 the total loss probability and system unavailability (TLP with $\lambda = 0$) are plotted as functions of the number of processors. The difference between the unavailability and the total loss probability is caused only by limited buffer size b.

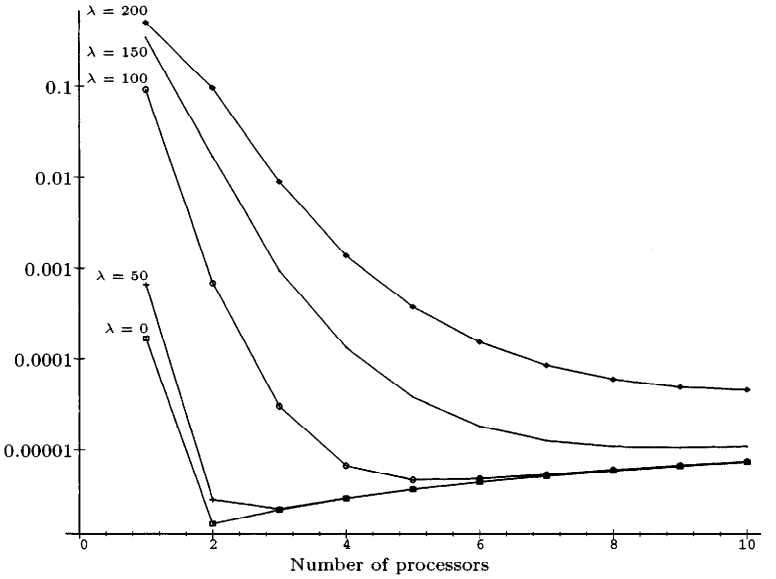


Fig. 13.70 TLP and unavailability.

Glossary

$\#(P_i, m)$	number of tokens at a particular place P_i in marking m
$1/\mu_i$	mean service time of the jobs at the ith node
$\alpha(r)$	factor for computing the shadow service time
argmin	function that returns the index of the minimum value
ABA	asymptotic bounds analysis
AMVA	asymmetric MVA
APU	associate processing unit
ASCAT	asymmetric SCAT
ASPA	average sub-system population algorithm
ASUM	asymmetric SUM
ATM	asynchronous transfer mode
$b_{i,j}$	probability that a job at node i leaves the node after the jth phase
B	normalized blocking factor
B_N	blocking factor
BCMP network	queueing network with several job classes, different queueing strategies, and generally distributed service times, named after Baskett, Chandy, Muntz, and Palacios
BFS	Bolch, Fleischmann and Schreppel method
BJB	balanced job bounds analysis

BOTT	bottapprox method
c_{Ai}	coefficient of variation of the interarrival time
$\hat{c}_A$	Coefficient of variation of the batch interarrival time
c_{Bi}	coefficient of variation of the service time
c_{B_∞}	coefficient of variation of the service time when applying the closing method
$c_i(j)$	capacity function of node i; the number of jobs completed by station i if there are j jobs in the queue
$c_X = \frac{\sigma_X}{\bar{X}}$	normalized standard deviation or coefficient of variation
ceiling	function that rounds up to the next integer
cl	index for closed job classes
corr	correction factor
C_k	Cox distribution with k phases
$C(\boldsymbol{\mu})$	linear cost function
CCNC	coalesce computation of normalizing constants
CDF	cumulative distribution function
CPU	central processing unit
CSMA/CD	carrier sense multiple access with collision detection
CTMC	continuous time Markov chain
$d(\mathbf{S})$	function of the jobs in the network
$driv$ **context**	a job executes driver code to set up or perform an I/O operation
D	deterministic distribution
$D_{irs}(\mathbf{K})$	change of contribution in the SCAT algorithm if a class-s job is removed from the network
D^+	output arcs
D^-	input arcs
DA	diffusion approximation
DAC	distribution analysis by chain
DES	discrete-event simulation
DTMC	discrete time Markov chain
ϵ	stopping condition
$\mathbf{e}$	vector of the visit ratios $\mathbf{e} = (e_0, e_1, \ldots, e_N)$
e_i	visit ratio; the mean number of visits of a job to the ith node, also known as the relative arrival rate
e_∞	visit ratio of the closing node when applying the closing method

e_{iq}^{*}	number of visits at node i in chain q, also called visit ratio
$ec_i(k)$	effective capacity function
$\mathcal{ERG}$	extended reachability graph
E_k	Erlang distribution with k phases
E_r	set of pairs (j, s) that can be reached from pair (i, r) in a finite number of steps
$f(c_A, c_B, \rho)$	Kulbatzki approximation
f_{ol}	fraction of all I/Os that overlap with a primary task
$f_{\mathbf{X}}(\mathbf{s};\mathbf{t})$	probability density function
$f_X(x) = \frac{dF_X(x)}{dx}$	probability density function
$\mathbf{fix}_{ir}$	functions needed for the fixed-point iteration
floor	function that rounds to the next lower integer
F_i	miss ratio function $F_i = 1 - H_i$
$F_i(k_i)$	functions corresponding to the state probabilities $\pi_i(k_i)$ of the ith node
$F_{ir}(\mathbf{K})$	contribution of class-r jobs at the ith node for a given population $\mathbf{K}$
$F_X(x) = P(X \leq x)$	cumulative distribution function
FB	full batch policy
FCFS	first-come-first-served
FDDI	fiber distributed data interface
FES	flow equivalent server method
FTCS	symposium on fault tolerant computing
$g(\lambda)$	core function of the summation method
$\Gamma(\alpha)$	gamma distribution
G	general distribution, also used for the normalization constant
$G(K)$	normalization constant of a single class queueing network
$G(\mathbf{K})$	normalization constant of a multiclass queueing network
$G_{\mathbf{KLB}}$	correction factor Krämer/Langenbach-Belz
$G_n(k)$	auxilary function for computing $G(K)$
$G_n(\mathbf{k})$	auxiliary function for computing $G(\mathbf{K})$
$G_N^{(i)}(k)$	normalization constant of the network with node i removed from the network
$G_N^{(i)}(\mathbf{k})$	normalization constant of the multiclass network with node i removed from the network

GE	Gaussian elimination algorithm
GI	general distribution with independent interarrival times
GreatSPN	analysis package for analyzing stochastic Petri nets
GREEDY	special case of the minimum batch policy
GTH	Grassmann, Taksar, and Heyman algorithm
GUI	graphical user interface
h	function used in bottleneck analysis
h_i	hit ratio of the cache
$h(\rho, m)$	factor for computing $\overline{W}_{GI/D/m}$ in the case of the Cosmetatos approximation
H_k	hyperexponential distribution with k phases
$H(\pi)$	entropy function
HOL	non-preemptive head-of-line queueing strategy
I/O	input/output
I	$= [i^-, i^+]$ intervall
I_{μ_n}	$=[\mu_n^-, \mu_n^+]$ intervall of the service rates
IMMD IV	institute for mathematical machines and data processing IV
IS	infinite server
ISDN	integrated services digital network
iSPN	integrated environment for modeling using stochastic Petri nets
k_i	number of jobs at the ith node
$(k_1, k_2, \ldots, k_N)$	state of the network
k_{ir}	number of jobs in the rth class at the ith node
kern **context**	job executes kern code
K	constant number of jobs in a closed network, also used for the number of jobs in the system
K_r	number of jobs of the rth class in the network
K^*	overall number of jobs in the fictious network
$\mathbf{K} = (K_1, \ldots, K_R)$	number of jobs in the various classes, known as the population vector
$\mathbf{K}_s$	mean number of class-s jobs at node i that an arriving class-r job sees given the network population $\mathbf{k}$
K	mean number of jobs in the system
$\overline{K}_{FJ}$	mean number of parallel-sequential jobs with fork-join synchronization
$\overline{K}_i$	mean number of jobs at node i

$\overline{K}_{ir}$	mean number of class-r jobs at node i
$\overline{K}^*_{iR^*}$	overall number of class-R^* jobs at node i in the fictious network
$\overline{K}^*_i$	mean number of jobs at the ith node in the pseudo open network
λ	throughput
$\lambda^{(0)}$	initial value for the throughput (used in bottleneck analysis)
λ_{0i}	arrival rate of jobs from outside to the ith node
$\lambda_1(\lambda_{02})$	throughput of the primary task at node 0
λ_2^{max}	maximum throughput of the open class in case of parallel processing with asynchronous tasks
λ_i	overall arrival rate of jobs at the ith node
$\hat{\lambda}$	arrival rate of batches
$\lambda_{ij,r}$	arrival rate of class-r jobs from node i to node j
λ_{ir}	throughput, the rate at which jobs of the rth class are serviced and leave the ith node
$\lambda^*_{iR^*}$	throughput of the fictious network at node i, class -R*
λ_{opt}	optimistic bound, largest possible throughput
λ_{pes}	pessimistic bound, lowest possible throughput
λ_r	overall throughput
$L(\theta)$	likelihood estimation
$L(\lambda, \mu_1, \ldots, \mu_N, y_1, y_2)$	Lagrange function with objective function λ and two Lagrange multipliers y_1 and y_2
L_i	load factor of open classes regarding to station i
$L_X(s)$	Laplace transform of the pdf: Laplace-Stieltjes transform of the CDF
LAN	local area network
LB	local balance property
LBANC	local balance algorithm for normalizing constants
LBPS	last batch processor sharing
LCFS	last-come-last-served
Little's theorem	Little's theorem, also known as Little's law
LST	Laplace-Stieltjes transform
LT	Laplace transform
μ_i	service rate of the jobs at the ith node
μ_∞	service rate of the closing node when applying the closing method
μ_{ir}	service rate of the ith node for jobs of the rth class

$\boldsymbol{\mu}^m = (\mu_1, \mu_2, \cdots, \mu_m)$	vector of service rates
m_0	$m_0 \in \mathcal{M}$ denotes the initial marking of a PN
m_i	number of parallel servers at the ith node
maxval	function that delivers the maximum value of a set of numbers
M	exponential distribution
$\overline{M}_{ir}$	mean number of customers of class-i who arrive during the waiting time of the tagged customer and receive service before him
$M \Rightarrow M$	Markov implies Markov property
$\mathbf{M}(t)$	time-averaged behavior of the CTMC
MB	minimum batch policy
MEM	maximum entropy method
MMPP	Markov modulated poisson process
MOSEL	modeling, specification and evaluation language
MOSES	modeling, specification and evaluation system
MRM	Markov reward model
MTTA	mean time to absorption
MTTF	mean time to failure
MTTR	mean time to repair
MVA	mean value analysis
MVALDMX	mean value analysis load dependent mixed
$\boldsymbol{\nu}(n)$	vector of the state probabilities at time n
$\nu_i(k)$	conditional throughput of node i with k jobs
$\boldsymbol{\nu}$	vector of the state probabilities of a discrete time Markov chain (DTMC)
$\tilde{\boldsymbol{\nu}}$	limiting state probabilities
nc_{Dm}	factor of the Cosmetatos approximation
$\mathcal{N}$	product-form network
$N_{i,r}$	number of class-r jobs at node i in the shadow node
$\overline{N}_{ir}$	mean number of customers of class-i found in the queue by the tagged (priority r) customer who receive service before him
NPFQN	non-product-form queueing network
op	index for open job classes
$\hat{\theta}$	estimate of θ
ODE	ordinary differential equation

One-limited service	station transmits at most one frame when the token arrives
$\pi^{(L)}$	probability of loss
$\pi_m^{(L)}$	probability that all servers are occupied
π_g	individual state for all $g \subseteq G$
$\pi_i(k)$	marginal probability that the ith node contains exactly $k_i = k$ jobs
$\pi_i^{(W)}$	state probability of node i of a non-lossy system
$\pi_j(v)$	unconditional state probabilities
π_k	probability of the number of jobs in the system
$\pi(\mathbf{S}_1, \ldots, \mathbf{S}_N)$	state probability of a network with multiple job classes
$\boldsymbol{\pi}(u)$	state probabilitiy vector at any instant of time u
$\dot{\boldsymbol{\pi}}$	derivative of the transition probabilities
$p_{0,js}$	probability in an open network that a job from outside enters the jth node as a job of the sth class
p_{0j}	probability that a job entering the network from outside first enters the jth node
p_{i0}	probability that a job leaves the network just after completing service at node i
p_{ij}	probability that a job is transferred to the jth node after service completion at the ith node (routing probability)
$p_{ij}(t)$	simplified transition probabilities for time-homogeneous CTMCs
$p_{ij}(u, v)$	transition probability of the CTMC to travel from state i to state j during $[u, v)$
$p_{ij}^{(n)}(k, l)$	probability that the Markov chain transits from state i at time k to state j at time l in exactly $n = l - k$ steps
$p_{ir,0} = \sum_{j=1}^{N} \sum_{s=1}^{R} p_{ir,js}$	probability in an open network that a job of the rth class leaves the network after having been serviced at the ith node
$p_{ir,js}$	probability that a job of the rth class at node i is transferred to the sth class at node j (routing probability)
$\phi(y, t)$	time-average accumulated reward
$\psi(y, t)$	distribution of the accumulated reward over a finite time $[0, t)$, also called performability
$\psi_{i,r}$	correction factor for computing the shadow service time in the case of a network with class switching and mixed priorities
pdf	probability density function

pmf	probability mass function
pri_r	priority of a class-r job with respect to a quota node
$\Phi(x)$	CDF of the standard normal distribution
P	places of a Petri net
P_m	waiting probability
$\dot{\mathbf{P}}$	derivative of the matrix of the transition probabilities
$\mathbf{P}$	stochastic transition matrix
$\mathbf{P}(u,v)$	matrix of the transition probabilities
$\mathbf{P}^{(n)}$	matrix of the n-step transition probabilities
PANDA	Petri net analysis and design assistant
PASTA	Poisson arrival see time averages theorem
PEPSY	performance evaluation and prediction system
PFQN	product-form queueing network
PN	Petri net
$PR(n)$	scaling function of the extended SCAT algorithm
PRIOMVA	extended MVA for the analysis of priority queueing networks
PRIOSUM	extension of the summation method to priority networks
PRS	preemptive resume queueing strategy
PS	processor sharing
$q_r(t)$	priority of class-r at time t
Q	queue length
$\mathbf{Q}$	infinitesimal generator matrix
$\hat{Q}$	mean queue length for batches
$\overline{Q}$	mean queue length
$\overline{Q}_i$	mean number of tasks waiting for the join synchronization in queue Q_i
$\overline{Q}_{ir}$	mean queue length of class-r jobs at the ith node
$\overline{Q}_{\mathbf{batch}}$	mean queue length for individual customers in a batch system
QNAP	queueing network analysis package
ρ	utilization
ρ_{bott}	utilization of the bottleneck node
ρ_i	utilization of node i
$\rho_i r$	utilization of node i by class-r jobs
$\rho_k^{(W)}$	specific utilization needed for calculating the utilization of an asymmetric non-lossy system

ρ_k^L	utilization of the kth server of a lossy system
r_i	reward rate assigned to state i
r_r	starting priority of class-r
$\mathcal{RG}$	reduced reachability graph
$\mathcal{RS}$	reachability set
R	number of priority classes
$\overline{R}$	mean remaining service time of a busy server, also called mean residual life
RECAL	recursion by chain algorithm
RESQ	research queueing network package
RPS	rotational position sensing system
RR	round robin
$\sigma_X^2 = \mathbf{var}(X)$	variance of X or second central moment
$s_{(i,r)}^{virt}$	virtual service time of class-r jobs at quota node i
s_{ci}	mean contention time
$s_{i,r}$	service time of class-r jobs at node i
$\tilde{s}_{i,r}$	service time of class-r jobs at shadow node i
s_{iq}^*	service rate in chain q at node i
$s_{ir}(j)$	load-dependent mean service time or normal service time
s_{li}	mean latency time
s_{si}	mean seek time
s_{ti}	mean transfer time
$\mathbf{S}_i = (k_{i1}, \ldots, k_{iR})$	state of the ith node
$\mathbf{S} = (\mathbf{S}_1, \ldots, \mathbf{S}_N)$	overall state of the network with multiple classes
S_N	synchronization overhead
SB	station balance property
SCAT	self correcting approximation technique
SHARPE	symbolic hierarchical automated reliability performance evaluator
SIRO	service in random order
SMP	semi-Markov process
SOR	successive over relaxation
SPN	stochastic Petri net
SPNP	stochastic Petri net package
SQL	structured query language
SRN	stochastic reward net

SUM	summation method
$\mathcal{T}$	tangible markings
T	response time
$\overline{T}$	mean response time
$\overline{T}_{FJ}$	mean response time of parallel-sequential jobs with fork-join synchronization
$\overline{T}_i$	mean response time at node i
$\overline{T}_{ir}$	mean response time of class-r jobs at node i
$\overline{T}_J$	mean join time
$\overline{T}_{J,i}$	mean time spent by task i waiting for the join synchronization in queue Q_i
$\overline{T}_{\text{opt}}$	optimistic bound, lowest possible mean response time
$\overline{T}_{\text{pes}}$	pessimistic bound, greatest possible mean response time
$\overline{T}_r$	mean target rotation time
$\overline{T}_S$	mean response time of a purely sequential job
$\overline{T}_{S,i}$	mean response time of task i belonging to a purely sequential job
T_d	delay time of a frame
T_t	transfer time of a frame
Tcl/Tk	tool command language/tool kit
TLP	total loss probability
TLP(t)	transient expected total loss probability
TPM	transition probability matrix
Type 1 blocking	blocking after service, transfer blocking, production blocking, or non-immediate blocking
Type 2 blocking	blocking before service, service blocking, communication blocking, or immediate blocking
Type 3 blocking	repetitive service, or rejection blocking
user **context**	job executes user code
U	number of chains
Upper time limit	time limit within which jobs need to be served in a real time system
$\mathcal{V}$	vanishing markings
ω	relaxation parameter
ω_0	optimal relaxation parameter
$\omega_0^{\approx}$	approximation of the optimal relaxation parameter
$\omega_{r,k}$	factor for computing the shadow service time
W	waiting time

$W(i,j)$	weighting function of the extended SCAT algorithm
$\overline{W}$	mean waiting time
$\overline{W}_0$	mean remaining service time of a job in service
$\overline{W}_i$	mean waiting time at node i
$\overline{W}_{ir}$	mean waiting time of class-r jobs at node i
$\overline{W}_r$	mean waiting time of an arriving customer of priority class-r
WAN	wide area network
WEIRDP	queueing strategy where the first job in the queue is assigned to the pth part of the processor and the rest of the jobs are assigned to the $1-p$th part of the processor
WinPEPSY	Windows version of the queueing network package PEPSY
WIP	mean number of workpieces in the system (work in progress)
ξ_r	set of jobs that cannot be preempted by class-r jobs
x_{ave}	average relative utilization of the individual nodes in the network
x_i	relative utilization
$x_{\max}$	maximum relative utilization
x_{sum}	sum of the relative utilizations
$X_t : t \in T$	family of random variables
$\overline{X}$	$=E[X]$, mean value or expected value
$\overline{X^n}$	$= E[X^n]$, nth moment
XPEPSY	X-version of the queueing network package PEPSY
$Y(t)$	accumulated reward in the finite time horizon $[0,t)$
$z_i(k)$	number of possibilities to distribute k tasks to $n_i + 1$ nodes (including the join queue Q_i)
$Z(t)$	instantaneous reward rate at time t
$Z_{FJ}(k)$	number of states in the fork-join sub-network

Bibliography

AbBo97. M. Abu El-Qomsan and G. Bolch. Analyse von Warteschlangennetzen mit asymmetrischen Knoten und mehreren Auftragsklassen. Technical report TR-I4-97-07, Universität Erlangen-Nürnberg, IMMD IV, 1997.

ABC84. M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2:93–122, May 1984.

ABC+95. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* Wiley, New York, 1995.

Abde82. A. Abdel-Moneim. Weak Lumpability in Finite Markov Chains. *Journal of Applied Probability*, 19:685–691, 1982.

AbMa93. H. Abdalah and R. Marie. The uniformized power method for transient solutions of Markov processes. *Computers and Operations Research*, 20(5):515–526, June 1993.

ABP85. I. Akyildiz, G. Bolch, and M. Paterok. Die erweiterte Parametrische Analyse für geschlossene Warteschlangennetze. Number 110 in Informatik-Fachberichte, pages 170–185, Berlin, 1985. Springer.

AgBu83. S. Agrawal and J. Buzen. The Aggregate Server Method for Analyzing Serialization Delays in Computer Systems. *ACM Transactions on Computer Systems*, 1(2):116–143, May 1983.

AgTr82. J. Agre and S. Tripathi. Modeling Reentrant and Nonreentrant Software. *ACM Sigmetrics Performance Evaluation Review*, 11(4):163–178, 1982.

AkBo83. I. Akyildiz and G. Bolch. Erweiterung der Mittelwertanalyse zur Berechnung der Zustandswahrscheinlichkeiten für geschlossene und gemischte Netze. Number 61 in Informatik-Fachberichte, pages 267–276, Berlin, 1983. Springer.

AkBo88a. I. Akyildiz and G. Bolch. Mean Value Analysis Approximation for Multiple Server Queueing Networks. *Performance Evaluation*, 8(2):77–91, April 1988.

AkBo88b. I. Akyildiz and G. Bolch. Throughput and Response Time Optimization in Queueing Network Models of Computer Systems. In T. Hasegawa, H. Takagi, and Y. Takahashi, editors, *Proc. Int. Seminar on Performance of Distributed and Parallel Systems*, pages 251–269, Kyoto, Japan, Amsterdam, December 1988. Elsevier.

AKH97. S. Allmaier, M. Kowarschik, and G. Horton. State Space Construction and Steady-State Solution of GSPNs on a Shared Memory Multiprocessor. In *Proc. 7th Int. Workshop on Petri Nets and Performance Models (PNPM '97)*, pages 112–121, St. Malo, France, Los Alamitos, CA, June 1997. IEEE Computer Society Press.

Akvo89. I. Akyildiz and H. von Brand. Exact Solutions for Open, Closed and Mixed Queueing Networks with Rejection Blocking. *Theoretical Computer Science*, 64(2):203–219, 1989.

Akyi87. I. Akyildiz. Exact Product Form Solution for Queueing Networks with Blocking. *IEEE Transactions on Computers*, 36(1):122–125, January 1987.

Akyi88a. I. Akyildiz. Mean Value Analysis for Blocking Queueing Networks. *IEEE Transactions on Software Engineering*, 14(4):418–428, April 1988.

Akyi88b. I. Akyildiz. On the Exact and Approximate Throughput Analysis of Closed Queueing Networks with Blocking. *IEEE Transactions on Software Engineering*, 14(1):62–70, January 1988.

AlDa97. S. Allmaier and S. Dalibor. PANDA – Petri Net Analysis and Design Assistant. In *Proc. 7th Int. Workshop on Petri Nets and*

Performance Models (PNPM '97), pages 58–60, St. Malo, France, Rennes, France, June 1997. INRIA.

Alle90. O. Allen. *Probability, Statistics and Queueing Theory with Computer Science Applications.* Academic Press, New York, 2nd edition, 1990.

AlPe86. T. Altiok and H. Perros. Approximate Analysis of Open Networks of Queues with Blocking: Tandem Configurations. *AIIE Transactions*, 12(3):450–461, March 1986.

Alti82. T. Altiok. Approximate Analysis of Exponential Tandem Queues with Blocking. *European Journal of Operational Research*, 11:390–398, October 1982.

AMM65. B. Avi-Itzhak, W. Maxwell, and L. Miller. Queueing with alternating priorities. *Operations Research*, 13(2):306–318, March–April 1965.

Baer85. M. Baer. Verlustsysteme mit unterschiedlichen mittleren Bedienungszeiten der Kanäle. *Wissenschaftliche Zeitschrift der Hochschule für Verkehrswesen–Friedrich List*, 1985. Sonderheft 15.

BaIa83. S. Balsamo and G. Iazeolla. Some Equivalence Properties for Queueing Networks with and without Blocking. In A. Agrawala and S. Tripathi, editors, *Proc. Performance '83*, pages 351–360, Amsterdam, 1983. North-Holland.

BaKe94. F. Bause and P. Kemper. *QPN-Tool for the qualitative and quantitative analysis of Queueing Petri Nets.* Number 794 in LNCS. Springer, Berlin, 1994.

Bard79. Y. Bard. Some Extensions to Multiclass Queueing Network Analysis. In *Proc. 4th Int. Symp. on Modelling and Performance Evaluation of Computer Systems*, volume 1, pages 51–62, Vienna, New York, February 1979. North-Holland.

Bard80. Y. Bard. A Model of Shared DASD and Multipathing. *Communications of the ACM*, 23(10):564–572, October 1980.

Bard82. Y. Bard. Modeling I/O Systems with Dynamic Path Selection and General Transmission Networks. *ACM Sigmetrics Performance Evaluation Review*, 11(4):118–129, 1982.

BaTh94. J. Barner and K.-C. Thielking. Entwicklung, Implementierung und Validierung von analytischen Verfahren zur Analyse von Prioritätsnetzen. Studienarbeit, Universität Erlangen-Nürnberg, IMMD IV, 1994.

BBA84. S. Bruell, G. Balbo, and P. Afshari. Mean Value Analysis of Mixed, Multiple Class BCMP Networks with Load Dependent Service Stations. *Performance Evaluation*, 4:241–260, 1984.

BBC77. R. Brown, J. Browne, and K. Chandy. Memory Management and Response Time. *Communications of the ACM*, 20(3):153–165, March 1977.

BBS77. G. Balbo, S. Bruell, and H. Schwetmann. Customer Classes and Closed Network Models–A Solution Technique. In B. Gilchrist, editor, *Proc. IFIP 7th World Computer Congress*, pages 559–564, Amsterdam, August 1977. North-Holland.

BCH79. O. Boxma, J. Cohen, and N. Huffels. Approximations of the Mean Waiting Time in a M/G/s-Queueing System. *Operations Research*, 27:1115–1127, 1979.

BCMP75. F. Baskett, K. Chandy, R. Muntz, and F. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22(2):248–260, April 1975.

Beau78. M. Beaudry. Performance-related reliability measures for computing systems. *IEEE Transactions on Computers*, 27(6):540–547, 1978.

Bega93. K. Begain. Using subclasses of PH distributions for the modeling of empirical distributions. In *Proc. 18th ICSCS*, pages 141–152, Cairo, Egypt, 1993.

BeGo81. P. Bernstein and N. Goodman. Concurrency Control in Distributed Database Systems. *Computing Surveys*, 13(2):185–221, June 1981.

BeMe78. F. Beutler and B. Melamed. Decomposition and Customer Streams of Feedback Networks of Queues in Equilibrium. *Operations Research*, 26(6):1059–1072, 1978.

BFH88. M. Bär, K. Fischer, and G. Hertel. *Leistungsfähigkeit–Qualität–Zuverlässigkeit*, chapter 4. transpress Verlagsgesellschaft mbH, Berlin, 1988.

BFS87. G. Bolch, G. Fleischmann, and R. Schreppel. Ein funktionales Konzept zur Analyse von Warteschlangennetzen und Optimierung von Leistungsgrößen. *Informatik-Fachberichte*, (154):327–342, 1987.

BGJ92. G. Bolch, M. Gaebell, and H. Jung. Analyse offener Warteschlangennetze mit Methoden für geschlossene Warteschlangennetze. In *Operations Research Proc. 1992*,

pages 324–332, Aachen, Germany, Berlin, September 1992. DGOR–Deutsche Gesellschaft für Operations Research, Springer.

BKLC84. R. Bryant, A. Krzesinski, M. Lakshmi, and K. Chandy. The MVA Priority Approximation. *ACM Transactions on Computer Systems*, 2(4):335–359, November 1984.

BMW89. H. Beilner, J. Mäter, and N. Weißenberg. Towards a Performance Modelling Environment: News on HIT. In R. Puigjaner, editor, *Proc. 3rd Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Palma de Mallorca*, volume 1, Amsterdam, September 1989. North-Holland.

BoBr84. G. Bolch and W. Bruchner. Analytische Modelle symmetrischer Mehrprozessoranlagen mit dynamischen Prioritäten. *Elektronische Rechenanlagen*, 26(1):12–19, 1984.

BoFi93. G. Bolch and M. Fischer. Bottapprox: Eine Engpaßanalyse für geschlossene Warteschlangennetze auf der Basis der Summationsmethode. In H. Dyckhoff, U. Derigs, M. Salomon, and H. Tijms, editors, *Operations Research Proc. 1993*, pages 511–517, Amsterdam, Berlin, August 1993. DGOR–Deutsche Gesellschaft für Operations Research, Springer.

BoHe96. G. Bolch and H. Herold. MOSEL, A new Language for the Markov Analyzer MOSES. Technical report TR-I4-96-02, Universität Erlangen-Nürnberg, IMMD IV, 1996.

BoKi92. G. Bolch and M. Kirschnick. PEPSY-QNS - Performance Evaluation and Prediction SYstem for Queueing NetworkS. Technical report TR-I4-21-92, Universität Erlangen-Nürnberg, IMMD IV, October 1992.

BoKo81. O. Boxma and A. Konheim. Approximate Analysis of Exponential Queueing Systems with Blocking. *Acta Informatica*, 15:19–66, 1981.

Bolc83. G. Bolch. Approximation von Leistungsgrößen symmetrischer Mehrprozessorsysteme. *Computing*, 31:305–315, 1983.

Bolc89. G. Bolch. *Leistungsbewertung von Rechensystemen*. Leitfäden und Monographien der Informatik. B.G.Teubner Verlagsgesellschaft, Stuttgart, 1989.

Bolc91. G. Bolch. Analytische Leistungsbewertung–Methoden und Erfolge. *PIK–Praxis der Informationsverarbeitung und Kommunikation*, 4:231–241, 1991.

BoRo94. G. Bolch and K. Roggenkamp. Analysis of Queueing Networks with Asymmetric Nodes. In M. Woodward, S. Datta, and

S. Szumko, editors, *Proc. Computer and Telecommunication Systems Performance Engineering Conf.*, pages 115–128, London, 1994. Pentech Press.

BoSc91. G. Bolch and A. Scheuerer. Analytische Untersuchungen Asymmetrischer Prioritätsgesteuerter Wartesysteme. In W. Gaul, A. Bachem, W. Habenicht, W. Runge, and W. Stahl, editors, *Operations Research Proc. 1991*, pages 514–521, Stuttgart, Berlin, September 1991. DGOR–Deutsche Gesellschaft für Operations Research, Springer.

BoTr86. A. Bobbio and K. Trivedi. An Aggregation Technique for the Transient Analysis of Stiff Markov Chains. *IEEE Transactions on Computers*, 35(9):803–814, September 1986.

Bran82. A. Brandwajn. Fast Approximate Solution of Multiprogramming Models. *ACM Sigmetrics Performance Evaluation Review*, 11(4):141–149, 1982.

BrBa80. S. Bruell and G. Balbo. *Computational Algorithms for Closed Queueing Networks.* Operating and programming systems series. North-Holland, Amsterdam, 1980.

BrSe91. I. Bronstein and K. Semendjajew. *Taschenbuch der Mathematik.* B.G.Teubner Verlagsgesellschaft, Stuttgart, 1991.

BRT88. J. Blake, A. Reibman, and K. Trivedi. Sensitivity analysis of reliability and performability measures for multiprocessor systems. In *Proc. 1988 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Computer Systems*, pages 177–186, Santa Fee, N.M., May 1988.

Brum71. S. Brumelle. Some Inequalities for Parallel-Server Queues. *Operations Research*, (19):402–413, 1971.

Bux81. W. Bux. Local-Area Subnetworks: A Performance Comparison. *IEEE Transactions on Communication*, 29(10):1465–1473, October 1981.

Buze71. J. Buzen. *Queueing Network Models of Multiprogramming.* PhD thesis, Div. of Engineering and Applied Physics, Harvard University, 1971.

Buze73. J. Buzen. Computational Algorithms for Closed Queueing Networks with Exponential Servers. *Communications of the ACM*, 16(9):527–531, September 1973.

BVDT88. M. Boyd, M. Veeraraghavan, J. Dugan, and K. Trivedi. An Approach to Solving Large Reliability Models. In *Proc. IEEE/AIAA DASC Symp.*, San Diego, 1988.

CBC+92. G. Ciardo, A. Blakemore, P. Chimento, J. Muppala, and K. Trivedi. *Linear algebra, Markov chains and queueing models*, volume 48, chapter Automated Generation and Analysis of Markov Reward Models using Stochastic Reward Nets, pages 145–191. Springer, New York, 1992.

CFMT94. G. Ciardo, R. Fricks, J. Muppala, and K. Trivedi. *SPNP Users Manual Version 4.0.* Duke University, Department of Electrical Engineering, Durham, N.C., March 1994.

CGM83. A. Chesnais, E. Gelenbe, and I. Mitrani. On the Modeling of Parallel Access to Shared Data. *Communications of the ACM*, 26(3):196–202, March 1983.

Chan72. K. Chandy. The Analysis and Solutions for General Queuing Networks. In *Proc. 6th Annual Princeton Conf. on Information Sciences and Systems*, pages 224–228, Princeton, N.J., March 1972. Princeton University Press.

ChLa74. A. Chang and S. Lavenberg. Work Rates in Closed Queueing Networks with General Independent Servers. *Operations Research*, 22:838–847, 1974.

ChLa83. K. Chandy and M. Lakshmi. An Approximation Technique for Queueing Networks with Preemptive Priority Queues. Technical report, Department of Computer Sciences, University of Texas, February 1983.

ChMa83. K. Chandy and A. Martin. A Characterization of Product–Form Queuing Networks. *Journal of the ACM*, 30(2):286–299, April 1983.

ChNe82. K. Chandy and D. Neuse. Linearizer: A Heuristic Algorithm for Queueing Network Models of Computing Systems. *Communications of the ACM*, 25(2):126–134, February 1982.

ChSa80. K. Chandy and C. Sauer. Computational Algorithms for Product Form Queueing Networks. *Communications of the ACM*, 23(10):573–583, October 1980.

CHT77. K. Chandy, J. Howard, and D. Towsley. Product Form and Local Balance in Queueing Networks. *Journal of the ACM*, 24(2):250–263, April 1977.

ChTr92. H. Choi and K. Trivedi. Approximate Performance Models of Polling Systems Using Stochastic Petri Nets. In *Proc IEEE INFOCOM '92*, volume 3, pages 1–9, Florence, Italy, Los Alamitos, CA, May 4–8 1992. IEEE Computer Society Press.

CHW75a. K. Chandy, U. Herzog, and L. Woo. Approximate Analysis of General Queueing Networks. *IBM Journal of Research and Development*, 19(1):43–49, January 1975.

CHW75b. K. Chandy, U. Herzog, and L. Woo. Parametric Analysis of Qucucing Networks. *IBM Journal of Research and Development*, 19(1):36–42, January 1975.

Chyl86. P. Chylla. *Zur Modellierung und approximativen Leistungsanalyse von Vielteilnehmer-Rechensystemen.* Dissertation, Faculty for Mathematics and Computer Science, Technical University Munich, 1986.

ChYu83. W. Chow and P. Yu. An Approximation Technique for Central Server Queueing Models with a Priority Dispatching Rule. *Performance Evaluation*, 3:55–62, 1983.

CiGr87. B. Ciciani and V. Grassi. Performability evaluation of fault-tolerant satellite systems. *IEEE Transactions on Computers*, 35(4):403–409, 1987.

Cinl75. E. Cinlar. *Introduction to Stochastic Processes.* Prentice-Hall, Englewood Cliffs, N.J., 1975.

CiTr93. G. Ciardo and K. Trivedi. A Decomposition Approach for Stochastic Reward Net Models. *Performance Evaluation*, 18(1):37–59, July 1993.

CMST90. G. Ciardo, R. Marie, B. Sericola, and K. Trivedi. Performability analysis using semi-Markov reward processes. *IEEE Transactions on Computers*, 39(10):1251–1264, 1990.

CMT90. M. Calzarossa, R. Marie, and K. Trivedi. System Performance with User Behavior Graphs. *Performance Evaluation*, 11(3):155–165, July 1990.

CMT91. G. Ciardo, J. Muppala, and K. Trivedi. On the Solution of GSPN Reward Models. *Performance Evaluation*, 12(4):237–254, July 1991.

CNT97. C. Ciardo, D. Nicol, and K. Trivedi. Discrete-event Simulation of Fluid Stochastic Petri Nets. In *Proc. 7th Int. Workshop on Petri Nets and Performance Models (PNPM '97)*, pages 217–225, St. Malo, France, Los Alamitos, CA, June 1997. IEEE Computer Society Press.

CoGe86. A. Conway and N. Georganas. RECAL–A New Efficient Algorithm for the Exact Analysis of Multiple-Chain Closed Queuing Networks. *Journal of the ACM*, 33(4):768–791, October 1986.

CoHo86. E. J. Coffman and M. Hofri. Queueing models of secondary storage devices. *Queueing Systems*, 1(2):129–168, September 1986.

CoSe84. P. Courtois and P. Semal. Bounds for positive eigenvectors of nonnegative matrices and for their approximations by decomposition. *Journal of the ACM*, 31(4):804–825, 1984.

Cosm76. G. Cosmetatos. Some Approximate Equilibrium Results for the Multiserver Queue (M/G/r). *Operations Research Quarterly, USA*, pages 615–620, 1976.

Cour75. P. Courtois. Decomposability, Instabilities and Saturation in Multiprogramming Systems. *Communications of the ACM*, 18(7):371 377, July 1975.

Cour77. P. Courtois. *Decomposability: Queueing and Computer System Applications.* Academic Press, New York, 1977.

Cox55. D. Cox. A Use of Complex Probabilities in the Theory of Stochastic Processes. In *Proc. Cambridge Philosophical Society*, volume 51, pages 313–319, 1955.

Crom34. C. Crommelin. Delay probability formulae. *Post Off. Electronic Engineer*, 26:266–274, 1934.

CSL89. A. Conway, E. de Souza e Silva, and S. Lavenberg. Mean Value Analysis by Chain of Product Form Queueing Networks. *IEEE Transactions on Computers*, 38(3):432–442, March 1989.

CTM89. G. Ciardo, K. Trivedi, and J. Muppala. SPNP: Stochastic Petri Net package. In *Proc. 3rd Int. Workshop on Petri Nets and Performance Models (PNPM '89)*, pages 142–151, Kyoto, Japan, Los Alamitos, CA, December 1989. IEEE Computer Society Press.

DaBh85. C. Das and L. Bhuyan. Bandwith availability of multiple-bus multiprocessors. *IEEE Transactions on Computers*, 34(10):918–926, 1985.

Dadu96. H. Daduna. Discrete Time Queueing Networks: Recent Developments. In *Proc. Performance '96*, Lausanne, October 8 1996. Tutorial Lecture Notes, Revised Version.

DeBu78. P. Denning and J. Buzen. The Operational Analysis of Queueing Network Models. *Computing Surveys*, 10(3):225–261, September 1978.

DoIy87. L. Donatiello and B. Iyer. Analysis of a composite performance reliability measure for fault-tolerant systems. *Journal of the ACM*, 34(4):179–199, 1987.

Dosh90. B. Doshi. Single Server Queues With Vacations–A Survey. *Queueing Systems*, 1, 1(1):29–66, 1990.

DrLa77. S. Dreyfus and A. Law. *The art and theory of dynamic programming.* Academic Press, New York, 1977.

DuCz87. A. Duda and T. Czachórski. Performance Evaluation of Fork and Join Synchronization Primitives. *Acta Informatica*, 24:525–553, 1987.

Duda87. A. Duda. Approximate Performance Analysis of Parallel Systems. In *Proc. 2nd Int. Workshop on Applied Mathematics and Performance/Reliability Models and Computer/Communication Systems*, pages 189–202, 1987.

EaSe83. D. Eager and K. Sevcik. Performance Bound Hierarchies for Queueing Networks. *ACM Transactions on Computer Systems*, 1(2):99–115, May 1983.

EaSe86. D. Eager and K. Sevcik. Bound Hierarchies for Multiple-Class Queueing Networks. *Journal of the ACM*, 33(1):179–206, January 1986.

Ehre96. D. Ehrenreich. Erweiterung eines Kommunikationsnetzes auf der Basis von Leistungsuntersuchungen. Diplomarbeit, Universität Erlangen-Nürnberg, IMMD IV, July 1996.

FeJo90. R. Feix and M. R. Jobmann. MAOS - Model Analysis and Optimization System. Technical report, University of Munich, FB Informatik, 1990.

Fell68. W. Feller. *An Introduction to Probability Theory and its Applications.* Volume 1 of [Fell71], 3rd edition, 1968.

Fell71. W. Feller. *An Introduction to Probability Theory and its Applications*, volume 2. Wiley, New York, 2nd edition, 1971.

FiMe93. W. Fischer and K. Meier-Hellstern. The Markov-Modulated Poisson Process (MMPP) cookbook. *Performance Evaluation*, 18:149–171, 1993.

Flei89. G. Fleischmann. *Modellierung und Bewertung paralleler Programme.* Dissertation, Universität Erlangen-Nürnberg, IMMD IV, 1989.

FoGl88. B. Fox and P. Glynn. Computing Poisson probabilities. *Communications of the ACM*, 31(4):440–445, April 1988.

Förs89. C. Förster. Implementierung und Validierung von MWA–Approximationen. Studienarbeit, Universität Erlangen-Nürnberg, IMMD IV, 1989.

FrBe83. D. Freund and J. Bexfield. A New Aggregation Approximation Procedure for Solving Closed Queueing Networks with Simultaneous Resource Possession. *ACM Sigmetrics Performance Evaluation Review*, pages 214–223, August 1983. Special Issue.

GaKe79. F. Gay and M. a. Ketelsen. Performance evaluation for gracefully degrading systems. In *Proc. IEEE Int. Symp. on Fault-Tolerant Computing, FTCS*, pages 51–58, Los Alamos, CA, Los Alamitos, CA, 1979. IEEE Computer Society Press.

GBB98. S. Greiner, G. Bolch, and K. Begain. A generalized analysis technique for queueing networks with mixed priority strategy and class switching. *Computer Communications*, 1998.

GCS+86. A. Goyal, W. Carter, E. de Souza e Silva, S. Lavenberg, and K. Trivedi. The System AVailability Estimator (SAVE). In *Proc. 16th Int. Symp. on Fault-Tolerant Computing, FTCS*, pages 84–89, Vienna, Austria, Los Alamitos, CA, July 1986. IEEE Computer Society Press.

Gele75. E. Gelenbe. On Approximate Computer System Models. *Journal of the ACM*, 22(2):261–269, April 1975.

GeMi80. E. Gelenbe and I. Mitrani. *Analysis and Synthesis of Computer Systems*. Academic Press, London, 1980.

GePu87. E. Gelenbe and G. Pujolle. *Introduction to Queueing Networks*. Wiley, Chichester, 1987.

GeTr83. R. Geist and K. Trivedi. The Integration of User Perception in the Heterogeneous M/M/2 Queue. In A. Agrawala and S. Tripathi, editors, *Proc. Performance '83*, pages 203–216, Amsterdam, 1983. North-Holland.

GKZH94. R. German, C. Kelling, A. Zimmermann, and G. Hommel. TimeNET–A Toolkit for Evaluating Non-Markovian Stochastic Petri Nets. Technical report, Technical University Berlin, 1994.

GLT87. A. Goyal, S. Lavenberg, and K. Trivedi. Probabilistic Modeling of Computer System Availability. *Annals of Operations Research*, 8:285–306, March 1987.

GoNe67a. W. Gordon and G. Newell. Closed Queuing Systems with Exponential Servers. *Operations Research*, 15(2):254–265, April 1967.

GoNe67b. W. Gordon and G. Newell. Cyclic Queuing Systems with Restricted Queues. *Operations Research*, 15(2):266–277, April 1967.

Good83. J. Goodman. Using cache memory to reduce processor-memory traffic. In *Proc. 10th Int. Symp. on Computer Architecture*, pages 124–131, New York, 1983. IEEE.

GoTa87. A. Goyal and A. Tantawi. Evaluation of performability in acyclic Markov chains. *IEEE Transactions on Computers*, 36(6):738–744, 1987.

GrBo96. S. Greiner and G. Bolch. Approximative analytische Leistungsbewertung am Beispiel eines UNIX-basierten Multiprozessor Betriebssystems. volume 11 of *Informatik Forschung und Entwicklung*, pages 112–124, Berlin, 1996. Springer.

GrHa85. H. Gross and C. Harris. *Fundamentals of Queueing Theory*. Wiley, New York, 2nd edition, 1985.

GTH85. W. Grassmann, M. Taksar, and D. Heyman. Regenerative Analysis and Steady State Distribution for Markov Chains. *Operations Research*, 33:1107–1116, September 1985.

Hahn88. T. Hahn. Implementierung und Validierung der Mittelwertanalyse für höhere Momente und Verteilungen. Studienarbeit, Universität Erlangen-Nürnberg, IMMD IV, 1988.

Hahn90. T. Hahn. Erweiterung und Validierung der Summationsmethode. Diplomarbeit, Universität Erlangen-Nürnberg, IMMD IV, 1990.

HaSp95. T. Hanschke and T. Speck. AMS–An APL Based Modeling System for Production Planning. In *Operations Research Proceedings 1994*, pages 318–323, Berlin, 1995. Springer.

HaTr93. R. Haverkort and K. Trivedi. Specification and Generation of Markov Reward Models. *Discrete-Event Dynamic Systems: Theory and Applications*, 3:219–247, 1993.

HaYo81. L. Hagemann and D. Young. *Applied Iterative Methods*. Academic Press, New York, 1981.

HBAK86. K. Hoyme, S. Bruell, P. Afshari, and R. Kain. A Tree-Structured Mean Value Analysis Algorithm. *ACM Transactions on Computer Systems*, 4(2):178–185, November 1986.

Hend72. W. Henderson. Alternative Approaches to the Analysis of the M/G/1 and G/M/1 Queues. *Operations Research*, 15:92–101, 1972.

HeSo84. D. Heyman and M. Sobel. *Stochastic Models in Operations Research*, volume 1 and 2. McGraw-Hill, New York, 1984.

IIeTr82. P. Heidelberger and K. Trivedi. Queueing Network Models for Parallel Processing with Asynchronous Tasks. *IEEE Transactions on Computers*, 31(11):1099–1109, November 1982.

HeTr83. P. Heidelberger and K. Trivedi. Analytic Queueing Models for Programs with Internal Concurrency. *IEEE Transactions on Computers*, 32(1):73–82, January 1983.

HLM96. G. Haring, J. Lüthi, and S. Majumdar. Mean Value Analysis for Computer Systems with Variabilities in Workload. In *Proc. IEEE Int. Computer Performance and Dependability Symp. (IPDS '96)*, pages 32–41, Urbana-Champaign, September 1996.

HMT91. D. Heimann, N. Mittal, and K. Trivedi. Dependability Modeling for Computer Systems. In *Proc. Annual Reliability and Maintainability Symposium*, pages 120–128, Orlando, FL, January 1991.

Hort96. G. Horton. On the Multi-Level Algorithm for Markov Chains. In *Proc. Cooper Mountain Conf. on Iterative Methods*, Cooper Mountain, C.O., April 1996.

Hova81. A. Hordijk and N. van Dijk. Networks of Queues with Blocking. In F. Kylstra, editor, *Proc. Performance '81*, pages 51–65, Amsterdam, 1981. North-Holland.

Howa71. R. Howard. *Dynamic Probabilistic Systems*, volume 2: Semi-Markov and Decision Processes. Wiley, New York, 1971.

HsLa87. C. Hsieh and S. Lam. Two Classes of Performance Bounds for Closed Queueing Networks. *Performance Evaluation*, 7(1):3–30, February 1987.

HsLa89. C. Hsieh and S. Lam. PAM–A Noniterative Approximate Solution Method for Closed Multichain Queueing Networks. *Performance Evaluation*, 9(2):119–133, April 1989.

Husl81. R. Huslende. A combined evaluation of performance and reliability for degradable systems. In *Proc. 1981 ACM SIGMETRICS Conf. on Measurements and Modeling of Computer Systems*, Performance Evaluation Review, pages 157–164, Las Vegas, Nevada, September 1981.

HWFT97. C. Hirel, S. Wells, R. Fricks, and K. Trivedi. iSPN: an Integrated Environment for modeling using Stochastic Petri Nets. In *Tool Descriptions 7th Int. Workshop on Petri Nets and Performance Models (PNPM '97)*, pages 17–19, St. Malo, France, Los Alamitos, CA, June 1997. IEEE Computer Society Press.

Ibe87. O. Ibe. Performance comparison of explicit and implicit token-passing networks. *Computer Communications*, 10(2):59–69, 1987.

IbTr90. O. Ibe and K. Trivedi. Stochastic Petri net models of polling systems. *IEEE Journal on Selected Areas in Communications*, 8(10):146–152, December 1990.

ICT93. O. Ibe, H. Choi, and K. Trivedi. Performance Evaluation of Client-Server Systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(11):1217–1229, November 1993.

IPM82. J. Ignazio, D. Palmer, and C. Murohy. A Multicriteria Approach to Supersystem Architecture Definition. *IEEE Transactions on Computers*, (C-31):410–418, 1982.

Jack57. J. Jackson. Networks of Waiting Lines. *Operations Research*, 5(4):518–521, 1957.

Jack63. J. Jackson. Jobshop-Like Queuing Systems. *Management Science*, 10(1):131–142, October 1963.

Jaek91. H. Jaekel. Analytische Untersuchung von Multiple Server Systemen mit Prioritäten. Diplomarbeit, Universität Erlangen-Nürnberg, 1991.

Jain91. R. Jain. *The Art of Computer Systems Performance Analysis.* Wiley, New York, 1991.

JaLa82. P. Jacobson and E. Lazowska. Analyzing Queueing Networks with Simultaneous Resource Possession. *Communications of the ACM*, 25(2):142–151, February 1982.

JaLa83. P. Jacobson and E. Lazowska. A Reduction Technique for Evaluating Queueing Networks with Serialization Delays. In A. Agrawala and S. Tripathi, editors, *Proc. Performance '83*, pages 45–59, Amsterdam, 1983. North-Holland.

Jens53. A. Jensen. Markoff chains as an aid in the study of Markoff processes. *Skandinavian Aktuarietidskr.*, 36:87–91, 1953.

Kauf84. J. Kaufmann. Approximation Methods for Networks of Queues with Priorities. *Performance Evaluation*, 4:183–198, 1984.

KeRi78. B. Kernighan and D. Ritchie. *The C Programming Language.* Prentice-Hall, Englewood Cliffs, N.J., 1978.

Kero86. T. Kerola. The Composite Bound Method for Computing Throughput Bounds in Multiple Class Environments. *Performance Evaluation*, 6(1):1–9, March 1986.

KeSn78. J. Kemeny and J. Snell. *Finite Markov Chains.* Springer, New York, 2nd edition, 1978.

KGB87. S. Kumar, W. Grassmann, and R. Billington. A Stable Algorithm to Calculate Steady-State Probability and Frequency of a Markov System. *IEEE Transactions on Reliability*, R-36(1), April 1987.

KGTA88. D. Kouvatsos, H. Georgatsos, and N. Tabet-Aouel. A Universal Maximum Entropy Algorithm for General Multiple Class Open Networks with Mixed Service Disciplines. Technical report DDK/PHG-1. N, Computing Systems Modelling Research Group, Bradford University, England, 1988.

Kimu85. T. Kimura. Heuristic Approximations for the Mean Waiting Time in the GI/G/s Queue. Technical report B55, Tokyo Institute of Technology, 1985.

King70. J. Kingman. Inequalities in the Theory of Queues. *Journal of the Royal Statistical Society*, (Series B 32):102–110, 1970.

King90. J. King. *Computer and Communication Systems Performance Modeling.* Prentice-Hall, Englewood Cliffs, N.J., 1990.

Kirs91. M. Kirschnick. Approximative Analyse von WS-Netzwerken mit Batchverarbeitung. Diplomarbeit, Universität Erlangen-Nürnberg, IMMD IV, 1991.

Kirs93. M. Kirschnick. XPEPSY Manual. Technical report TR-I4-18-93, Universität Erlangen-Nürnberg, IMMD IV, September 1993.

Kirs94. M. Kirschnick. The Performance Evaluation and Prediction SYstem for Queueing NetworkS (PEPSY-QNS). Technical report TR-I4-18-94, Universität Erlangen-Nürnberg, IMMD IV, June 1994.

Klei65. L. Kleinrock. A conservation law for a wide class of queueing disciplines. *Naval Research Logistic Quart.*, (12):181–192, 1965.

Klei75. L. Kleinrock. *Queueing Systems*, volume 1: Theory. Wiley, New York, 1975.

Klei76. L. Kleinrock. *Queueing Systems*, volume 2: Computer Applications. Wiley, New York, 1976.

KnKr96. W. Knottenbelt and P. Kritzinger. A Performance Analyzer for the Numerical Solution of General Markov Chains. Technical report, Data Network Architectures Laboratory, Computer Science Department, University of Cape Town, 1996.

KoAl88. D. Kouvatsos and J. Almond. Maximum Entropy Two-Station Cyclic Queues with Multiple General Servers. *Acta Informatica*, 26:241–267, 1988.

Koba74. H. Kobayashi. Application of the Diffusion Approximation to Queueing Networks, Part 1: Equilibrium Queue Distributions. *Journal of the ACM*, 21(2):316–328, April 1974.

Koba78. H. Kobayashi. *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology.* Addison-Wesley, Reading, MA, 1978.

Koba79. H. Kobayashi. A Computational Algorithm for Queue Distributions via Polya Theory of Enumeration. In M. Arato and M. Butrimenko, editors, *Proc. 4th Int. Symp. on Modelling and Performance Evaluation of Computer Systems*, volume 1, pages 79–88, Vienna, Austria, February 1979. North-Holland.

Köll74. J. Köllerström. Heavy Traffic Theory for Queues with Several Servers. *Journal of Applied Probability*, 11:544–552, 1974.

KoRe76. A. Konheim and M. Reiser. A Queueing Model with Finite Waiting Room and Blocking. *Journal of the ACM*, 23(2):328–341, April 1976.

KoRe78. A. Konheim and M. Reiser. Finite Capacity Queuing Systems with Applications in Computer Modeling. *SIAM Journal on Computing*, 7(2):210–299, May 1978.

Kouv85. D. Kouvatsos. Maximum Entropy Methods for General Queueing Networks. In *Proc. Int. Conf. on Modelling Techniques and Tools for Performance Analysis*, pages 589–608, 1985.

KrLa76. W. Krämer and M. Langenbach-Belz. Approximate Formulae for General Single systems with Single and Bulk Arrivals. In *Proc. 8th Int. Teletraffic Congress (ITC)*, pages 235–243, Melbourne, 1976.

Krze87. A. Krzesinski. Multiclass Queueing Networks with State-Dependent Routing. *Performance Evaluation*, 7(2):125–143, June 1987.

KTK81. A. Krzesinski, P. Teunissen, and P. Kritzinger. Mean Value Analysis for Queue Dependent Servers in Mixed Multiclass Queueing Networks. Technical report, University of Stellenbosch, South Africa, July 1981.

Kuba86. P. Kubat. Reliability analysis for integrated networks with application to burst switching. *IEEE Transactions on Communication*, 34(6):564–568, 1986.

Kühn79. P. Kühn. Approximate Analysis of General Queuing Networks by Decomposition. *IEEE Transactions on Communication*, 27(1):113–126, January 1979.

Kulb89. J. Kulbatzki. Das Programmsystem PRIORI: Erweiterung und Validierung mit Hilfe von Simulationen. Diplomarbeit, 1989.

Kulk96. V. Kulkarni. *Modeling and Analysis of Stochastic Systems.* Chapman & Hall, London, 1996.

LaLi83. S. Lam and Y. Lien. A Tree Convolution Algorithm for the Solution of Queueing Networks. *Communications of the ACM*, 26(3):203–215, March 1983.

Lam81. S. Lam. A Simple Derivation of the MVA and LBANC Algorithms from the Convolution Algorithm. Technical report 184, University of Texas at Austin, November 1981.

LaRe80. S. Lavenberg and M. Reiser. Stationary State Probabilities at Arrival Instants for Closed Queueing Networks with Multiple Types of Customers. *Journal of Applied Probability*, 17:1048–1061, 1980.

Lave83. S. Lavenberg. *Computer Performance Modeling Handbook.* Academic Press, New York, 1983.

LaZa82. E. Lazowska and J. Zahorjan. Multiple Class Memory Constrained Queueing Networks. *ACM Sigmetrics Performance Evaluation Review*, 11(4):130–140, 1982.

LeWi89. Y. Levy and P. Wirth. A unifying approach to performance and reliability objectives. Teletraffic Science for New Cost-Effective Systems, Networks and Services, ITC-12, pages 1173–1179. Elsevier, Amsterdam, 1989.

Lind94. C. Lindemann. *Stochastic Modeling using DSPNexpress.* Oldenburg, Munich, 1994.

Litt61. J. Little. A Proof of the Queuing Formula $L = \lambda W$. *Operations Research*, 9(3):383–387, May 1961.

Luca91. D. Lucantoni. New Results on the single server queue with a Batch Markovian Arrival Process. *Stochastic Models*, 7(1):1–46, 1991.

Luca93. D. Lucantoni. The BMAP/G/1 queue: A tutorial. In L. Donatiello and R. Nelson, editors, *Models and Techniques for Performance Evaluation of Computer and Communications Systems.* Springer, New York, 1993.

LZGS84. E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik. *Quantitative System Performance–Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Englewood Cliffs, N.J., 1984.

MAD94. D. Menascé, V. Almeida, and L. Dowdy. *Capacity Planning and Performance Modeling–From Mainframes to Client-Server Systems.* Prentice-Hall, Englewood Cliffs, N.J., 1994.

MaRa95. S. Majumdar and R. Ramadoss. Interval-Based Performance Analysis for Computer Systems. In P. Dowd and E. Gelenbe, editors, *Proc. 3rd Int. Workshop on Modeling Analysis and Simulation of Computer and Telecommunication Systems*, pages 345–351, Durham, N.C., Los Alamitos, CA, 1995. IEEE Computer Society Press.

Marc74. W. Marchal. *Simple Bounds and Approximations in Queueing Systems.* D.sc. dissertation, George Washington University, Department of Operations Research, Washington, D.C., 1974.

Marc78. W. Marchal. Some simpler bounds on the mean queueing time. *Operations Research*, 26(6), 1978.

Mari78. R. Marie. Méthodes itératives de résolution de modèles mathématiques de systèmes informatiques. *R.A.I.R.O. Informatique/Computer Science*, 12(2):107–122, 1978.

Mari79. R. Marie. An Approximate Analytical Method for General Queueing Networks. *IEEE Transactions on Software Engineering*, 5(5):530–538, September 1979.

Mari80. R. Marie. Calculating Equilibrium Probabilities for $\lambda(n)/C_k/1/N$ Queues. *ACM Sigmetrics Performance Evaluation Review*, 9(2):117–125, 1980.

Mart72. J. Martin. *System analysis for data transmission.* Prentice-Hall, Englewood Cliffs, N.J., 1972.

MaSt77. R. Marie and W. Stewart. A Hybrid Iterative-Numerical Method for the Solution of a General Queueing Network. In *Proc. 3rd Symp. on Measuring, Modelling and Evaluating Computer Systems*, pages 173–188, Amsterdam, 1977. North-Holland.

MaTr93. M. Malhotra and K. Trivedi. A Methodology for Formal Specification of Hierarchy in Model Solution. In *Proc. 5th Int. Workshop on Petri Nets and Performance Models (PNPM '93)*, pages 258–267, Toulouse, France, October 1993.

McKe88. J. McKenna. A New Proof and a Tree Algorithm for RECAL. In P. Courtois and G. Latouche, editors, *Proc. Performance '87*, pages 3–16, Amsterdam, 1988. North-Holland.

McMi84. J. McKenna and D. Mitra. Asymptotic Expansions and Integral Representations of Moments of Queue Lengths in Closed Markovian Networks. *Journal of the ACM*, 31(2):346–360, April 1984.

MCT94. J. Muppala, G. Ciardo, and K. Trivedi. Stochastic Reward Nets for Reliability Prediction. *Communications in Reliability, Maintainability and Serviceability (An nternational journal published by SAE international)*, 1(2):9–20, July 1994.

MeDü97. H. de Meer and O. Düsterhöft. Controlled Stochastic Petri Nets. In *Proc. 16th IEEE Symposium on Reliable Distributed Systems (SRDS'97)*, pages 18 25, Durham N.C., Los Alamitos, CA, October 1997. IEEE Computer Society Press.

Meer92. H. de Meer. Transiente Leistungsbewertung und Optimierung rekonfigurierbarer fehlertoleranter Rechensysteme. Dissertation 10, Universität Erlangen-Nürnberg, IMMD IV, October 1992.

MeFi97. H. de Meer and S. Fischer. Controlled Stochastic Petri Nets for QoS Management. In K. Irmscher, C. Mittasch, and K. Richter, editors, *Proc. 9th ITG/GI Conf. on Measurement, Modeling and Performance Evaluation of Computer and Communication Systems (MMB '97)*, pages 161–171, Freiberg/Sachsen, Germany, Berlin–Offenbach, September 1997. VDE Verlag.

MeNa82. D. Menasce and T. Nakanishi. Optimistic Versus Pessimistic Concurrency Control Mechanisms in Database Management Systems. *Information Systems*, 7(1):13–27, 1982.

MeSe97. H. de Meer and H. Ševčíková. PENELOPE: dependability evaluation and the optimization of performability. In R. Marie, R. Plateau, and G. Rubino, editors, *Proc. 9th Int. Conf. on Modeling Techniques ant Tools for Computer Performance Evaluation*, Lecture Notes in Computer Science 1245, pages 19–31, St. Malo, France, Berlin, 1997. Springer.

Meye80. J. Meyer. On Evaluating the Performability of Degradable Computing Systems. *IEEE Transactions on Computers*, 29(8):720–731, August 1980.

Meye82. J. Meyer. Closed-Form Solutions of Performability. *IEEE Transactions on Computers*, 31(7):648–657, July 1982.

Mitz97. U. Mitzlaff. *Diffusionsapproximation von Warteschlangensystemen.* Dissertation, Technical University Clausthal, Germany, July 1997.

MMKT94. J. Muppala, V. Mainkar, V. Kulkarni, and K. Trivedi. Numerical Computation of Response Time Distributions Using Stochastic Reward Nets. *Annals of Operations Research*, 48:155–184, 1994.

MMT94. M. Malhotra, J. Muppala, and K. Trivedi. Stiffness-Tolerant Methods for Transient Analysis of Stiff Markov Chains. *International Journal on Microelectronics and Reliability*, 34(11):1825–1841, 1994.

MMT96. J. Muppala, M. Malhotra, and K. Trivedi. *Reliability and maintenance of complex systems*, chapter Markov Dependability Models of Complex Systems: Analysis Techniques, pages 442–486. Springer, Berlin, 1996.

MMW57. C. Mack, T. Murphy, and N. Webb. The efficiency of N Machines unidirectionally patrolled by one operative when walking time and repair time are constant. *Journal of the Royal Statistical Society*, Series B 19(1):166–172, 1957.

Moor72. F. Moore. Computational Model of a Closed Queuing Network with Exponential Servers. *IBM Journal of Research and Development*, 16(6):567–572, November 1972.

MSA96. M. Meo, E. de Souza e Silva, and M. Ajmone Marsan. Efficient Solution for a Class of Markov Chain Models of Telecommunication Systems. *Performance Evaluation*, 27/28(4):603–625, October 1996.

MSW88. P. Martini, O. Spaniol, and T. Welzel. File Transfer in High Speed Token Ring Networks: Performance Evaluation by approximate Analysis and Simulation. *IEEE Journal on Selected Areas in Communications*, 6(6):987–996, July 1988.

MTBH93. H. de Meer, K. Trivedi, G. Bolch, and F. Hofmann. Optimal Transient Service Strategies for Adaptive Heterogeneous Queueing Systems. In O. Spaniol, editor, *Proc. 7th GI/ITG Conf. on Measurement, Modelling and Performance Evaluation of Computer and Communication Systems (MMB '93)*, Informatik aktuell, pages 166–170, Aachen, Germany, Berlin, September 1993. Springer.

MTD94. H. de Meer, K. Trivedi, and M. Dal Cin. Guarded Repair of Dependable Systems. *Theoretical Computer Science*, 128:179–210, July 1994. Special Issue on Dependable Parallel Computing.

Munt72. R. Muntz. Network of Queues. Technical report, University of Los Angeles, Department of Computer Science, 1972. Notes for Engineering 226 C.

Munt73. R. Muntz. Poisson Departure Process and Queueing Networks. In *Proc. 7th Annual Princeton Conf. on Information Sciences and Systems*, pages 435–440, Princeton, N.J., March 1973. Princeton University Press.

MüRo87. B. Müller-Clostermann and G. Rosentreter. Synchronized Queueing Networks: Concepts, Examples and Evaluation Techniques. *Informatik-Fachberichte*, (154):176–191, 1987.

MuTr91. J. Muppala and K. Trivedi. Composite performance and availability analysis using a hierarchy of stochastic reward nets. In G. Balbo, editor, *Proc. 5th Int. Conf. on Modeling Techniques and Tools for Computer Performance Evaluation*, pages 322–336, Torino, Amsterdam, 1991. Elsevier Science (North Holland).

MuTr92. J. Muppala and K. Trivedi. Numerical Transient Solution of Finite Markovian Queueing Systems. In U. Bhat and I. Basawa, editors, *Queueing and Related Models*, pages 262–284. Oxford University Press, Oxford, 1992.

MuWo74a. R. Muntz and J. Wong. Asymptotic Properties of Closed Queueing Network Models. In *Proc. 8th Annual Princeton Conf. on Information Sciences and Systems*, pages 348–352, Princeton, N.J., March 1974. Princeton University Press.

MuWo74b. R. Muntz and J. Wong. Efficient Computational Procedures for Closed Queueing Network Models. In *Proc. 7th Hawaii Int. Conf. on System Science*, pages 33–36, Hawaii, January 1974.

NaFi67. T. Naylor and J. Finger. Verification of Computer Simulation Models. *Management Science*, 14:92–101, October 1967.

NaGa88. W. Najjar and J. Gaudiot. Reliability and performance modeling of hypercube-based multiprocessors. In G. Iazeolla, P. Courtois, and O. Boxma, editors, *Computer Performance and Reliability*, pages 305–320. North-Holland, Amsterdam, 1988.

NeCh81. D. Neuse and K. Chandy. SCAT: A Heuristic Algorithm for Queueing Network Models of Computing Systems. *ACM Sigmetrics Performance Evaluation Review*, 10(3):59–79, 1981.

NeOs69. G. Newell and E. Osuna. Properties of vehicle-actuated signals: II. One-way streets. *Transportation Science*, 3:99–125, 1969.

NeTa88. R. Nelson and A. Tantawi. Approximate Analysis of Fork/Join Synchronization in Parallel Queues. *IEEE Transactions on Computers*, 37(6):739–743, June 1988.

Neut81. M. Neuts. *Matrix-Geometric Solutions in stochastic Models: An Algorithmic Approach.* The Johns Hopkins University Press, Baltimore, M.D., 1981.

Newe69. G. Newell. Properties of vehicle-actuated signals: I. One-way streets. *Transportation Science*, 3:30–52, 1969.

Nico90. V. Nicola. Lumpability of Markov Reward Models. Technical report, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1990.

NNHG90. V. Nicola, M. Nakajama, P. Heidelberger, and A. Goyal. Fast Simulation of Dependability Models with General Failure, Repair and Maintenance Processes. In *Proc. 20th Annual Int. Symp. on Fault-Tolerant Computing Systems*, pages 491–498, Newcastle upon Tyne, UK, Los Alamitos, CA, June 1990. IEEE Computer Society Press.

Noet79. A. Noetzel. A Generalized Queueing Discipline for Product Form Network Solutions. *Journal of the ACM*, 26(4):779–793, October 1979.

OnPe86. R. Onvural and H. Perros. On Equivalences of Blocking Mechanisms in Queueing Networks with Blocking. *Operations Research Letters*, 5(6):293–298, December 1986.

OnPe89. R. Onvural and H. Perros. Some Equivalences between Closed Queueing Networks with Blocking. *Performance Evaluation*, 9(2):111–118, April 1989.

PeAl86. H. Perros and T. Altiok. Approximate Analysis of Open Networks of Queues with Blocking: Tandem Configurations. *IEEE Transactions on Software Engineering*, 12(3):450–461, March 1986.

Perr81. H. Perros. A Symmetrical Exponential Open Queue Network with Blocking and Feedback. *IEEE Transactions on Software Engineering*, 7(4):395–402, July 1981.

Perr94. H. Perros. *Queueing Networks with Blocking.* Oxford University Press, Oxford, 1994.

PeSn89. H. Perros and P. Snyder. A Computationally Efficient Approximation Algorithm for Feed-Forward Open Queueing Networks with Blocking. *Performance Evaluation*, 9(9):217–224, June 1989.

Petr62. C. A. Petri. *Kommunikation mit Automaten.* Dissertation, University of Bonn, Bonn, Germany, 1962.

PGTP96. A. Pfening, S. Garg, S. Telek, and A. Puliafito. Optimal rejuvenation for tolerating soft failures. *Performance Evaluation*, 27/28:491–506, October 1996.

Pitt79. B. Pittel. Closed Exponential Networks of Queues with Saturation: The Jackson Type Stationary Distribution and Its Asymptotic Analysis. *Mathematics of Operations Research*, 4:367–378, 1979.

PuAi86. G. Pujolle and W. Ai. A Solution for multiserver and multiclass open queueing networks. *INFOR*, 24(3):221–230, 1986.

RaBe74. E. Rainville and P. Bedient. *Elementary Differential Equations.* MacMillan, New York, 1974.

Rand86. S. Randhawa. Simulation of a Wafer Fabrication Facility Using Network Modeling. *Journal of the Paper Society of Manufacturing Engineers*, 1986.

RaTr95. A. Ramesh and K. Trivedi. Semi-numerical transient analysis of Markov models. In R. Geist, editor, *Proc. 33rd ACM Southeast Conf.*, pages 13–23, 1995.

Reis81. M. Reiser. Mean-Value Analysis and Convolution Method for Queue-Dependent Servers in Closed Queueing Networks. *Performance Evaluation*, 1, 1981.

ReKo74. M. Reiser and H. Kobayashi. Accuracy of the Diffusion Approximation for Some Queueing Systems. *IBM Journal of Research and Development*, 18(2):110–124, March 1974.

ReKo75. M. Reiser and H. Kobayashi. Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms. *IBM Journal of Research and Development*, 19(3):283–294, May 1975.

ReLa80. M. Reiser and S. Lavenberg. Mean-Value Analysis of Closed Multichain Queuing Networks. *Journal of the ACM*, 27(2):313–322, April 1980.

ReTr88. A. Reibman and K. Trivedi. Numerical Transient Analysis of Markov Models. *Computers and Operations Research*, 15(1):19–36, 1988.

ReTr89. A. Reibman and K. Trivedi. Transient analysis of cumulative measures of Markov model behavior. *Stochastic Models*, 5(4):683–710, 1989.

RST89. A. Reibman, R. Smith, and K. Trivedi. Markov and Markov Reward Model Transient Analysis: An Overview of Numerical Approaches. *European Journal of Operational Research*, 40:257–267, 1989.

SaCh81. C. Sauer and K. Chandy. *Computer Systems Performance Modelling.* Prentice-Hall, Englewood Cliffs, N.J., 1981.

SaMa85. C. Sauer and E. MacNair. The Evolution of the Research Queueing Package. In D. Potier, editor, *Proc. Int. Conf. on Modelling Techniques and Tools for Performance Analysis*, pages 5–24, Paris, Amsterdam, 1985. North-Holland.

SaTr87. R. Sahner and K. Trivedi. Performance and Reliability Analysis using Directed Acyclic Graphs. *IEEE Transactions on Software Engineering*, 13(10):1105–1114, October 1987.

Saue81. C. Sauer. Approximate Solution of Queueing Networks with Simultaneous Resource Possession. *IBM Journal of Research and Development*, 25(6):894–903, November 1981.

Saue83. C. Sauer. Computational Algorithms for State-Dependent Queueing Networks. *ACM Transactions on Computer Systems*, 1(1):67–92, February 1983.

Schr70. L. Schrage. An alternative proof of a conservation law for the queue G/G/1. *Operations Research*, (18):185–187, 1970.

Schw79. P. Schweitzer. Approximate Analysis of Multiclass Closed Networks of Queues. In *Proc. Int. Conf. on Stochastic Control and Optimization*, pages 25–29, Amsterdam, June 1979.

Schw84. P. Schweitzer. Aggregation Methods for Large Markov Chains. In G. Iazeolla and other, editors, *Mathematical Computer Performance and Reliability*. North-Holland, Amsterdam, 1984.

ScMü90. M. Sczittnick and B. Müller-Clostermann. MACOM – A Tool for the Markovian Analysis of Communication System. In *Proc. 4th Int. Conf. on Data Communications Systems and their Performance*, Barcelona, June 1990.

SeMi81. K. Sevcik and I. Mitrani. The Distribution of Queuing Network States at Input and Output Instants. *Journal of the ACM*, 28(2):358–371, April 1981.

Sevc77. K. Sevcik. Priority Scheduling Disciplines in Queueing Network Models of Computer Systems. In *Proc. IFIP 7th World Computer Congress*, pages 565–570, Toronto, Amsterdam, 1977. North-Holland.

ShBu77. A. Shum and J. Buzen. The EPF Technique: A Method for Obtaining Approximate Solutions to Closed Queueing Networks with General Service Times. In *Proc. 3rd Symp. on Measuring, Modelling and Evaluating Computer Systems*, pages 201–220, 1977.

ShYa88. J. Shanthikumar and D. Yao. Throughput Bounds for Closed Queueing Networks with Queue-Dependent Service Rates. *Performance Evaluation*, 9(1):69–78, November 1988.

SLM86. E. de Souza de Silva, S. Lavenberg, and R. Muntz. A Clustering Approximation Technique for Queueing Networks with a Large

Number of Chains. *IEEE Transactions on Computers*, 35(5):419–430, May 1986.

SmBr80. C. Smith and J. Browne. Aspects of Software Design Analysis: Concurrency and Blocking. *ACM Sigmetrics Performance Evaluation Review*, 9(2):245–253, 1980.

SMK82. C. Sauer, E. MacNair, and J. Kurose. The Research Queueing Package: Past, Present and Future. *Proc. National Computer Conf.*, pages 273–280, 1982.

SmTr90. R. Smith and K. Trivedi. The Analysis of Computer Systems Using Markov Reward Processes. In H. Takagi, editor, *Stochastic Analysis of Computer and Communication Systems*, pages 589–629. Elsevier Science (North Holland), Amsterdam, 1990.

SoGa89. E. de Souza e Silva and H. Gail. Calculating availability and performability measures of repairable computer systems using randomization. *Journal of the ACM*, 36(1):171–193, 1989.

SoLa89. E. de Souza e Silva and S. Lavenberg. Calculating Joint Queue-Length Distributions in Product-Form Queuing Networks. *Journal of the ACM*, 36(1):194–207, January 1989.

SOQW95. W. Sanders, W. Obal, A. Qureshi, and F. Widjanarko. The UltraSAN modeling environment. *Performance Evaluation*, 24(1):89–115, October 1995.

Spir79. J. Spirn. Queuing Networks with Random Selection for Service. *IEEE Transactions on Software Engineering*, 5(3):287–289, May 1979.

SRT88. V. Sarma, A. Reibman, and K. Trivedi. Optimization Methods in Computer System Design. In R. Levary, editor, *Engineering Design: Better Results Through Operations Research Methods.* Elsevier Science (North Holland), Amsterdam, 1988.

Stew90. W. Stewart. *MARCA: MARCOF CHAIN ANALYZER, A Software Package for Makrov Modelling, Version 2.0.* North Carolina State University, Department of Computer Science, Raleigh, N.C., 1990.

Stew94. W. Stewart. *Introduction to Numerical Solution of Markov Chains.* Princeton University Press, Princeton, N.J., 1994.

StGo85. W. Stewart and A. Goyal. Matrix methods in large dependability models. Research report RC 11485, IBM T.J. Watson Research Center, Yorktown Heights, N.Y., November 1985.

STP96. R. Sahner, K. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems - An Example-Based Approach Using the SHARPE Software Package.* Kluwer Academic Publishers, Boston, M.A., 1996.

STR88. R. Smith, K. Trivedi, and A. Ramesh. Performability Analysis: Measures, an Algorithm and a Case Study. *IEEE Transactions on Computers*, 37(4):406–417, April 1988.

Stre86. J. Strelen. A Generalization of Mean Value Analysis to Higher Moments: Moment Analysis. *ACM Sigmetrics Performance Evaluation Review*, 14(1):129–140, May 1986.

SuDi84. R. Suri and G. Diehl. A New Building Block for Performance Evaluation of Queueing Networks with Finite Buffers. *ACM Sigmetrics Performance Evaluation Review*, 12(3):134–142, August 1984.

SuDi86. R. Suri and G. Diehl. A Variable Buffer-Size Model and its Use in Analyzing Closed Queueing Networks with Blocking. *Management Science*, 32(2):206–225, February 1986.

SuHi84. R. Suri and R. Hildebrant. Modeling flexible manufacturing systems using mean value analysis. *Journal of Manufacturing Systems*, 3(1):27–38, 1984.

Taka75. Y. Takahashi. A Lumping Method for Numerical Calculations of Stationary Distributions of Markov Chains. Research report B 18, Tokyo Institute of Technology, Department of Information Sciences, Tokyo, June 1975.

Taka90. H. Takagi. *Stochastic Analysis of Computer And Communication Systems.* North-Holland, Amsterdam, 1990.

Taka93. H. Takagi. *Queueing Analysis: A Foundation of Performance Evaluation*, volume 1-3. North-Holland, Amsterdam, 1991–1993.

Tane95. A. Tanenbaum. *Distributed Operating Systems.* Prentice-Hall, Englewood Cliffs, N.J., 1995.

Tay87. Y. Tay. *Locking Performance in Centralized Databases.* Academic Press, New York, 1987.

ThBa86. A. Thomasian and P. Bay. Analytic Queueing Network Models for Parallel Processing of Task Systems. *IEEE Transactions on Computers*, 35(12):1045–1054, December 1986.

Thom83. A. Thomasian. Queueing Network Models to Estimate Serialization Delays in Computer Systems. In A. Agrawala and S. Tripathi,

editors, *Proc. Performance '83*, pages 61–81, Amsterdam, 1983. North-Holland.

ThRy85. A. Thomasian and I. Ryu. Analysis of Some Optimistic Concurrency Control Schemes Based on Certification. *ACM Sigmetrics Performance Evaluation Review*, 13(2):192–203, August 1985.

Tijm86. H. Tijms. *Stochastic Modelling and Analysis: A Computational Approach.* Wiley, New York, 1986.

TMWH92. K. Trivedi, J. Muppala, S. Woolet, and B. Haverkort. Composite performance and dependability analysis. *Performance Evaluation*, 14:197–215, 1992.

Tows80. D. Towsley. Queuing Network Models with State-Dependent Routing. *Journal of the ACM*, 27(2):323–337, April 1980.

Triv82. K. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* Prentice-Hall, Englewood Cliffs, N.J., 1982.

TrKi80. K. Trivedi and R. Kiniki. A Model for Computer Configuration Design. *IEEE Computer*, 13(4):47–54, April 1980.

TrSi81. K. Trivedi and T. Sigmon. Optimal Design of Linear Storage Hierarchies. *Journal of the ACM*, 28(2):270–288, 1981.

TSI+96. K. S. Trivedi, A. S. Sathaye, O. C. Ibe, R. C. Howe, and A. Aggarwal. Availability and Performance-Based Sizing of Multiprocessor Systems. *Communications in Reliability, Maintainability and Serviceability*, 1996.

TSIH90. K. Trivedi, A. Sathaye, O. Ibe, and R. Howe. Should I add a processor? In *Proc. 23rd Annual Hawaii Int. Conf. on System Sciences*, pages 214–221, Los Alamitos, CA, January 1990. IEEE Computer Society Press.

TuSa85. S. Tucci and C. Sauer. The Tree MVA Algorithm. *Performance Evaluation*, 5(3):187–196, August 1985.

VePo85. M. Veran and D. Potier. QNAP 2: A Portable Environment for Queueing Systems Modelling. In D. Potier, editor, *Proc. Int. Conf. on Modelling Techniques and Tools for Performance Analysis*, pages 25–63, Paris, Amsterdam, 1985. North-Holland.

Wals85. R. Walstra. Nonexponential Networks of Queues: A Maximum Entropy Analysis. *ACM Sigmetrics Performance Evaluation Review*, 13(2):27–37, August 1985.

Welc95. B. Welch. *Practical Programming in Tcl and Tk.* Prentice-Hall, Upper Saddle River, N.J., 1995.

Whit83a. W. Whitt. Performance of the Queueing Network Analyzer. *Bell System Technical Journal*, 62(9):2817–2843, November 1983.

Whit83b. W. Whitt. The Queueing Network Analyzer. *Bell System Technical Journal*, 62(9):2779–2815, November 1983.

Wilh77. N. Wilhelm. A General Model for the Performance of Disk Systems. *Journal of the ACM*, 24(1):14–31, January 1977.

WLTV96. C. Wang, D. Logothetis, K. Trivedi, and I. Viniotis. Transient Behavior of ATM Networks under Overloads. In *Proc. IEEE INFOCOM '96*, pages 978–985, San Francisco, CA, Los Alamitos, CA, March 1996. IEEE Computer Society Press.

Wolf82. R. Wolff. Poisson arrivals see time averages. *Operations Research*, 30:223–231, 1982.

Wong75. J. Wong. *Queueing Network Models for Computer Systems.* PhD thesis, Berkley University, School of Engineering and Applied Science, October 1975.

Wu82. L. Wu. Operational models for the evaluation of degradable computing systems. In *Proc. 1982 ACM SIGMETRICS Conf. on Measurements and Modeling of Computer Systems*, Performance Evaluation Review, pages 179–185, College Park, Maryland, New York, April 1982. Springer.

YaBu86. D. Yao and J. Buzacott. The Exponentialization Approach to Flexible Manufacturing System Models with General Processing Times. *European Journal of Operational Research*, (24):410–416, 1986.

YBL89. Q. Yang, L. Bhuyan, and B. Liu. Analysis and Comparison of Cache Coherence Protocols for a Packet-Switched Multiprocessor. *IEEE Transactions on Computers*, 38(8):1143–1153, August 1989.

ZaWo81. J. Zahorjan and E. Wong. The Solution of Separable Queueing Network Models Using Mean Value Analysis. *ACM Sigmetrics Performance Evaluation Review*, 10(3):80–85, 1981.

ZES88. J. Zahorjan, D. Eager, and H. Sweillam. Accuracy, Speed, and Convergence of Approximate Mean Value Analysis. *Performance Evaluation*, 8(4):255–270, August 1988.

ZSEG82. J. Zahorjan, K. Sevcik, D. Eager, and B. Galler. Balanced Job Bound Analysis of Queueing Networks. *Communications of the ACM*, 25(2):134–141, February 1982.

Index

N

T

U

VW

XYZ