CHAPTER 1

INTRODUCTION

Queueing network models are the most widely used analytical method for estimating the standard performance measures of computer systems. With these models, calculations of throughput are typically within 5% of actual values, and mean response times are within 30% [DENN78]. Their success has been enhanced by simple and efficient solution algorithms and good commercial packages such as BEST/1 [BGS80, BUZE78a], CADS [IRA77] and RESQ [SAUE82a, SAUE82b].

The simplicity of analytic queueing network models limits their utility. They cannot directly represent certain behaviors common in computer systems. These behaviors include simultaneous resource possession, queueing for passive resources such as memory and semaphores, preemptive priorities at some servers, and the blocking of one server by the actions of another. Ignoring these behaviors can lead to substantial errors in the estimates of performance measures. Because of the computational efficiency of queueing network models, analysts have sought ways to represent these behaviors by extending queueing network models. A bewildering array of approximate models has been the result.

In reviewing the literature on approximate queueing models, one is led to a long list of questions. For example:

1. When should one replace a subnetwork with a flow equivalent server?

2. When should one add servers to the model to represent queueing delays for nonphysical resources?
3. When should one split a server into two or more separate servers?
4. Why does iteration show up in some approximations but not in others?
5. If iteration is used, does it converge to a unique solution?
6. Is there any systematic way of analyzing the errors in the solutions of models?
7. Is there any common structure in the processes by which each model was constructed?

The goal of the research reported in this monograph has been to critically study some well known approximate methods and find answers to these questions. The result is a *metamodel* of the process of developing system models.

This metamodel is a general model of the modeling process. We will show, by considering a large number of examples, that the structure of the modeling process envisioned in metamodeling is capable of representing a wide variety of approximations. The central principle of this structure is that an approximation involves mapping a "complex" model into one or more "simpler" models, solving these simpler models, and mapping the solution back into an estimate of the solution of the original model. We employ two general techniques, behavior sequence transformations and state space transformations, to develop, describe and analyze model transformations. Results of our study include a catalog of common transformations, several new methods for analyzing preemptive priority scheduling, the method of aggregate servers for analyzing serialization delays, general theorems about convergence of iterative approximations, and the discovery that some queueing systems may have multiple stable solutions for key performance quantities. These convergence theorems enabled us to prove the convergence

of

1) The Bard-Schweitzer approximation for large networks [SCHW79],
2) The Jacobson-Lazowska method for simultaneous resource possession [JACO82] in a simpler way, and
3) The shadow CPU approximation for preemptive priority scheduling [SEVC77].

In the remainder of this chapter, we review analytic queueing network models; they are the focus of the most of the discussion in later chapters. We then review the principal approximations reported in the literature. We next set forth the plan of the subsequent chapters, in which the components of modeling process are taken up. We conclude the chapter with a review of related prior work.

1.1 Queueing Network Models

In this section, we provide a brief overview of queueing network models. We begin with single resource models and then discuss queueing networks, multiple workloads, product form solutions, computational algorithms, and robustness of queueing network models. We conclude the section by reviewing the history of the queueing networks.

1.1.1 Single Resource Models

A single resource model is the most elementary form of a queueing model. It is a description of a service system comprising an arrival process, a completion or service process, a queueing mechanism, a set of one or more units of a resource that renders the service, and a buffer space for holding the waiting customers at the resource.[1] An analytical model typically gives an

1. Kendall's shorthand notation *A/B/C/D/E* is widely used to classify single resource queueing models. In this notation, *A* describes the arrival process, *B* specifies comple-

algorithm for computing $p(n)$, the proportion of time n customers are in the queue, given the parameters

λ (or $\lambda(n)$) - mean arrival rate (possibly conditioned on n), and

S (or $S(n)$) - mean service time (also possibly conditioned on n).

If N is the maximum number of customers observed at the server, the solution is [BUZE76a]:

$$p(0) = \frac{1}{1 + \sum_{n=1}^{N} \prod_{k=0}^{n-1} \lambda(k)\, S(k+1)}$$

$$p(n) = p(0) \prod_{k=0}^{n-1} \lambda(k)\, S(k+1), \quad \text{for } n = 1, \ldots, N. \tag{1.1}$$

Under the assumptions of Homogeneous Arrivals (HA: $\lambda(n) = \lambda$) and Homogeneous Service (HS: $S(n) = S$) [BUZE76a, DENN78, BRUM82], the solution reduces to

$$p(n) = (1 - \lambda S)(\lambda S)^n. \tag{1.2}$$

1.1.2 Queueing Network Models

A queueing network models is a collection of single resource models arranged in the same configuration as a real system. The parameters of this model are:

V_i - Number of times each customer visits (or requests service at) device i.

---

tion or service process, C denotes the number of units of resources that render the service, D specifies the maximum number of customers that can be queued at the server, and E is the size of the population. Some examples of this notation are

M/*M*/1 - Poisson (*M*arkov) input, exponential (*M*arkov) service times, 1 server;

M/*G*/1 - Poisson (*M*arkov) input, *G*eneral (arbitrary) service time distribution function, 1 server;

GI/*M*/1 - *G*eneral, *I*ndependently distributed interarrival times, exponential (*M*arkov) service times, 1 server.

S_i (or $S_i(n)$) - mean service time per visit to device i.

An analytical model typically gives an algorithm for computing the solution $p(\underline{n}) = p(n_1, \ldots, n_K)$, the proportion of time n_1 customers are present at device 1, . . . , and n_K customers are at device K. If the arrival rate is specified, the model is *open*. If the total number of customers $N = n_1 + \cdots + n_K$ is fixed, the model is *closed*.

1.1.3 Multiple Workloads

Customers in the network can be differentiated by tagging them with their type or *class* r; a network with multiple classes of customers is called a *multiclass network*. Separate parameters V_{ir} and S_{ir} are specified for each class; separate performance measures are computed for each class. Some classes may be open (i.e., their λ_r's are given and the number of customers of these classes in the system is not fixed), and others may be closed (i.e., their N_r are specified). A network with both open and closed classes is called a *mixed* network. The state of the system is typically defined to be

$$\underline{n} = ((n_{11}, \ldots, n_{1R}), \ldots, (n_{K1}, \ldots, n_{KR})),$$

where n_{ir} is the number of class r customers present at device i.

1.1.4 Product Form Solution

The direct solution $p(\underline{n})$ of a general queueing network would involve a very expensive solution of the global balance equations [HERZ75, STEW78]. Under the following assumptions, however, $p(\underline{n})$ has the "product form" and is efficiently computable:

a. *One Step Behavior*: A state transition can occur only due to a departure of a single customer from one resource to another or outside the system, or due to arrival of a customer from the outside,

b. *Flow Balance*: The number of arrivals (in each class) at a device must equal the number of departures (in each class) from the device,

c. *Device Homogeneity*: A device's service rate for a particular class does not depend on the state of the system in any way except for the total device queue length and the designated class's queue length. This assumption essentially implies that:

   - *Single Resource Possession*: A customer may not be present (waiting for service or receiving service) at two or more devices at the same time,
   - *No Blocking*: A device renders service whenever customers are present; its ability to render service is not controlled by any other device.
   - *Independent Customer Behavior*: Interaction among customers is limited to queueing for physical devices, e.g., there should not be any synchronization requirements.
   - *Local Information*: A device's service rate depends only on local queue length and not on the state of the rest of the system.
   - *Fair Service*: If service rates differ by class, the service rate for a class depends only on the queue length of that class at the device and not on the queue lengths of other classes. This means that the server does not discriminate against customers in a class depending on the queue lengths in other classes.

d. *Routing Homogeneity*: The customer routing should be state independent.

Because of their intuitive meaning, we will call the assumptions (c) and (d) the *autonomous behavior* assumptions. With the assumptions (a)-(d), the solution has following product form:

$$p(\underline{n}) = \frac{F_1(\underline{n}_1) \cdots F_K(\underline{n}_K)}{G}. \tag{1.3}$$

In this expression G is a normalization constant such that all probabilities sum to 1, $\underline{n}_i = (n_{i1}, \ldots, n_{iR})$, where n_{ir} is the number of class r customers at device i, and F_i is the device factor for device i:

$$F_i(\underline{n}_i) = f_i(n_{i1} + \cdots + n_{iR}) \left(\prod_{r=1}^{R} \prod_{j=1}^{n_{ir}} V_{ir} S_{ir}(j) \right) \tag{1.4}$$

where f_i is a factor affecting the overall device service rate depending on the total queue length.

Another set of assumptions for "product form" solution is provided by Baskett, Chandy, Muntz and Palacios [BASK75]; these assumptions are:

a. *Allowable Scheduling Disciplines*: The following disciplines are allowed: First-Come-First-Served (FCFS), Processor Sharing (PS), Last-Come-First-Served-Preemptive-Resume (LCFS-PR), and Infinite Server (IS) or delay service.

b. *Service Time Distribution*: The service times at a FCFS server should be exponentially distributed; moreover, the S_{ir} should be same for all classes. The service times at PS, LCFS-PR, and IS can have any distribution that has a rational Laplace transform. The mean service times for different classes may also be different.

c. *State Dependent Service Rates*: The service rate (time) at a FCFS server can depend only on the total queue length of the server. The service rate for a class at PS, LCFS-PR and IS servers can also depend on the queue length for that class, but not on the queue length of other classes. Moreover, the overall service rate of a subnetwork can depend on the total number of customers in the subnetwork.

d. *Interarrival Time Distribution*: In open networks, the time between successive customer arrivals for a class should be exponentially distributed. No bulk arrivals are permitted.

Note that the fair service and autonomous operation assumptions are central to this set of assumptions too. These assumptions lead to the condition of "local balance" [CHAN77], i.e., the flow rate into a state due to arrival of a class r job equals the flow rate out of that state due to departure of a class r job. When local balance conditions are met, the system has a product form solution.

1.1.5 Fast Computational Algorithms

The existence of fast computational algorithms is the strong point of product form solution. Two major algorithms are the Convolution algorithm [BRUE80, BUZE71, BUZE73] and the Mean-Value-Analysis algorithm [REIS80, SCHW80, ZAHO81]. These algorithms are alternative ways to exploit the recursive structure of the product form solution and are specified in Box 1.1. Variations of these two basic algorithms are presented in [CHAN80, HOYM82, LAM83].

1.1.6 Robustness and Accuracy

Performance measures predicted by queueing networks are usually sufficiently accurate: utilization estimates are typically within 5% of observed values and response time and queue length estimates are generally within 30% [DENN78].

Closed queueing networks are robust. The errors in the parameter estimates and homogeneity assumptions are usually not magnified in performance measures [GORD80, SURI83b, WILL76]. Open network

Product Form Algorithms for Load Independent Single Class Networks

Notation:

$X(n)$ - network throughput with n customers
$\bar{n}_k(n)$ - mean queue length for server k
$R_k(n)$ - mean response time at server k

The Convolution Algorithm [BRUE80, BUZE71, BUZE73]:

$G_1(0) := 1$
for $n = 1, \ldots, N$ **loop**
$G_1(n) := V_1 S_1 G_1(n-1)$
end loop

for $k = 2, \ldots, K$ **loop**
$G_k(0) := 1$
for $n = 1, \ldots, N$
$G_k(n) := G_{k-1}(n) + V_k S_k G_k(n-1)$ **end loop**
end loop

$$X(N) := \frac{G_K(N-1)}{G_K(N)}$$

The Mean-Value-Analysis (MVA) Algorithm [REIS80, SCHW80, ZAHO81]:

for $k = 1, \ldots, K$
$\bar{n}_k(0) := 0$
end loop

for $n = 1, \ldots, N$
for $k = 1, \ldots, K$
$R_k(n) := S_k(1 + \bar{n}_k(n-1))$ **end loop**

$$X(n) := \frac{n}{\sum_{k=1}^{K} V_k R_k(n)}$$

for $k = 1, \ldots, K$
$\bar{n}_k(n) := V_k R_k(n) X(n)$ **end loop**
end loop

Box 1.1: The Convolution and the MVA algorithms.

performance measures are, however, more sensitive to the errors. Both the parameter estimation errors and the homogeneity assumption errors may be magnified by a factor of the order of $1/(1-U)$, where U is the server utilization [BRUM82].

1.1.7 History

Theory

Open models, usually single server models, have been studied for long time. Kleinrock [KLEI75] provides an extensive treatment. Jackson [JACK57, JACK63], and Gordon and Newell [GORD67] showed that the solution of a closed network of exponential servers has the product form. Baskett, Chandy, Muntz, and Palacios extended the range of the product form networks to include multiclass, mixed networks with a variety of state dependent behavior [BASK75]. Limited forms of state dependent routing are allowed [TOWS80]. The set of product form scheduling disciplines has been extended to include Random Selection [SPIR79], and Load Balancing [AFSH82].

After noticing that many statements about performance are valid without requiring any distributional assumptions, Buzen and Denning developed an *operational approach* to queueing network modeling [BUZE76a, BUZE76b, DENN78, BUZE80a, BUZE80b]. This approach does not require any distributional assumptions; it relies only on the measurable quantities of the system; its homogeneity assumptions are testable. Within this approach, analysis of errors is possible [BRUM82, DENN82, KOWA81, SURI83b].

Practice

Single resource models have long been used to analyze specific aspects of system behavior, e.g., scheduling disciplines, and buffer allocation [COFF73, KLEI75, KLEI76].

The first successful application of queueing network models to computer systems was Scherr's machine repairman model of the Compatible-Time-Sharing-System [SCHE67]. Other applications rapidly followed [BUZE71, KELL76, MOOR71, SAUE75a]. The growth of the applications has been aided by commercially available packages such as BEST/1 [BGS80, BUZE78a], CADS [IRA77] and RESQ [SAUE82a, SAUE82b].

Algorithms

Though the concept of product form solution was known since 1957 [JACK57], queueing network model applications did not really take off before 1971, when Buzen reported an algorithm for computing the normalization constant G for exponential networks and computing performance measures in terms of G [BUZE71]. The algorithm has now been extended to compute performance measures for all product form networks [BRUE80, SAUE83]. The other major algorithm, Mean Value Analysis, computes the performance measures directly [REIS80, SAUE83, SCHW80, ZAHO81]. Some variations of these algorithms are also available. CCNC (Coalesce Computation of Normalizing Constants) is useful when storage is at a premium, and LBANC (Local balance Algorithm for Normalizing Constants) is useful when the number of queues is small but the number of customers is very large [CHAN80]. Tree-Convolution [LAM83] and Tree-MVA [HOYM82] algorithms are useful for analyzing large, but sparse networks. Distributed systems are examples of such sparse networks.

In addition to the exact solution algorithms discussed above, some algorithms have been developed to solve large networks approximately. They will be discussed in Section 1.2.8.

1.2 Approximate Models of Complex Computer Systems

Many practical computer systems violate the homogeneity assumptions required for product form solution. Approximate methods for such cases modify the network model of the system, by adding servers or changing parameters, until a product form model is found that represents the actual behaviors accurately [CHAN78].

In this section we briefly review some of the principal approximate methods. We first present some methods that overcome following non-homogeneous behaviors:

General service time distribution at FCFS servers,

Memory queueing,

Simultaneous resource possession in I/O subsystems,

Preemptive priority scheduling,

Serialization (critical sections),

Internal program concurrency (FORK/JOIN), and

Blocking.

We conclude the section by considering approximate methods for solving large product form networks. Throughout, unless otherwise stated, we assume a closed network with population N, servers $1, \ldots, K$, visit ratios $\{V_k\}$, service functions $\{S_k(n)\}$, and system throughput X_0.

1.2.1 General Service Time Distribution at an FCFS Server

Occasionally, the service time of a customer at a FCFS server has a high coefficient of variation (CV >> 1). Networks that include such servers in general do not have a product form solution because the exponential service time assumption is violated. (The CV of an exponentially distributed random variable is 1.) Consequently, ignoring high CV can lead to significant errors. We now discuss some methods for modeling systems containing FCFS servers with non-exponential (CV $\neq$ 1) service times.

Shum and Buzen [SHUM76, SHUM77] observed that within a multiplicative factor $F_i(n)$ (Eqns. (1.1) and (1.2)) in the product form solution is, in fact, the queue length distribution for an $M/M/1/N$ queue. They proposed that $F_i(n)$ for a general service time server can be taken as the queue length distribution of an $M/G/1/N$ queue. Assuming a value for network throughput, X_0, they compute arrival rates to each queue and use the $M/G/1/N$ queueing function to compute the device factors. The device factors are then used to evaluate device utilizations from the product form solution algorithms. From the device utilizations, they evaluate the output rate of each queue. If the output rates do not match the arrival rates, the procedure is repeated with another guess for network throughput X_0. The algorithm normally provides fairly accurate solutions but occasionally fails to find a flow-balanced solution.

Chandy, Herzog and Woo (CHW) [CHAN75a] construct K submodels, $M_1, \ldots, M_K$, one for each queue. The submodel M_k consists of the general server k and a flow-equivalent server that represents its complement in the network. The complementary network consists of exponential servers whose initial service function, $S_k^*(n)$, is computed from input parameters by ignoring

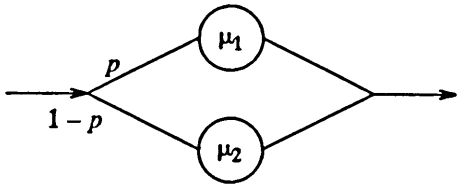
Table 1.1: Service function correction in the CHW method [CHAN75a].

Throughput	Queue Length	Service Function
$X_k/V_k < X_0(1-\epsilon)$	$\sum_l \bar{n}_l > N(1+\epsilon)$	$\hat{S}_k^*(n) = S_k^*(n)X_0/X_k$
$X_k/V_k > X_0(1+\epsilon)$	$\sum_l \bar{n}_l < N(1-\epsilon)$	$\hat{S}_k^*(n) = S_k^*(n)X_0/X_k$
$\lvert(X_k/V_k - X_0)/X_0\rvert < \epsilon$	$\lvert(\sum_l \bar{n}_l - N)/N\rvert \geq \epsilon$	$\hat{S}_k^*(n) = S_k^*(n)\sum_j \bar{n}_j/N$
$\lvert(X_k/V_k - X_0)/X_0\rvert \geq \epsilon$	$\lvert(\sum_l \bar{n}_l - N)/N\rvert < \epsilon$	$\hat{S}_k^*(n) = S_k^*(n)X_0/X_k$

Note: These service functions are used only in characterizing the complementary network in M_i, $i \neq k$. The two queue model M_k represents the queue k as general server. A rationale for these corrections is provided in [TOLO79].

the CV. M_k is solved to obtain the mean queue length $\bar{n}_k$ and the throughput X_k for server k. Because the M_k's are solved independently, the sum of the queue lengths ($\sum_k \bar{n}_k$) may not equal N, the network population; moreover, the local throughputs (X_k's) may not satisfy the forced-flow law ($X_k = V_k X_0$, where $X_0 = (\sum_k X_k/V_k)/K$). When this happens, the $S_k^*(n)$'s are corrected as shown in Table 1.1, and the M_k's are solved with the new estimates of the service functions.

Marie's method [MARI78] is similar to the CHW method. Marie treats each submodel M_k as an $M(\lambda(n))/G/1$ queue, where $\lambda(n)$ is the arrival rate to the queue and equals the throughput of the complementary network with $N-n$ customers. Assuming that queue k's service time has a rational Laplace transform, he computes $S_k^*(n)$ directly rather than by solving for $\bar{n}_k$ and X_k and using Table 1.1. (Marie's method always has $\sum_k \bar{n}_k = N$ and $X_0 = V_k X_k$.) The initial value of $S_k^*(n)$ is obtained by ignoring the CV of queue k, and then, is iteratively corrected.



$$E(S) = \frac{1}{\mu} = \frac{p}{\mu_1} + \frac{1-p}{\mu_2}$$

$$E(S^2) = \frac{2p}{\mu_1^2} + \frac{2(1-p)}{\mu_2^2}$$

$$CV = \frac{\sqrt{E(S^2) - E(S)^2}}{E(S)}$$

Figure 1.1: A hyperexponential server.

Balbo has extensively studied and compared these three methods [BALB79]. His recommendation is to ignore the problem if the CV is small ($0.5 < CV < 2$). Otherwise, Shum and Buzen's method should be used if the network contains only one FCFS server with general service times. If there are two or more non-exponential servers in the network, Marie's method is preferable. The CHW method is not robust and is not as good as other two methods. It is also much more cumbersome to implement.

Zahorjan, Lazowska and Garner's method is based on the theory of near-decomposability [ZAHO83]. We illustrate their method with an example. Assume that the service time at the CPU can be represented as a two stage hyperexponential server as shown in Figure 1.1. With the assumption that $\mu_1 >> \mu_2$ and $p >> (1-p)$, the states of the network can be decomposed into two nearly-decomposable sets. The first set represents the states in which stage 2 (rate μ_2) of the hyperexponential is not in service; these states are the states of the network M_1 in which the CPU is replaced by stage 1 (rate μ_1) of

the hyperexponential. The second aggregate represents the states in which the stage 2 of the hyperexponential server is always busy; these states are isomorphic with the states of the network M_2 which has $N - 1$ customers and in which the CPU is replaced by the stage 2 of the hyperexponential. The conditional probability distribution $p(s \mid i)$, the probability that the system is in state s given that the system is in the aggregate i, can be obtained by solving product form model M_i. The unconditional steady state probability is

$$p(s) = p(s \mid i)\, w_i,$$

where w_i is the probability that the system is in aggregate i; the w_i are given by:

$$w_1 = \frac{\mu_2 p}{\mu_2 + U_1\mu_1(1-p)},$$
$$w_2 = 1 - w_1,$$

where U_1 is the utilization of stage 1 in M_1. Other performance measures are the weighted averages of the measures of M_1 and M_2.

1.2.2 Memory Queueing

Memory queueing occurs in two stages. First a job has to wait for memory partition or a scheduling token to become available before it can compete for CPU and I/O devices. Then, once scheduled, it experiences paging delays that depend on the current level of multiprogramming. Because a job holds two limited resources at the same time (memory and CPU or an I/O device), the autonomous behavior assumption is violated and the system does not have a product form solution.

Most approximations for this problem are applications of the decomposition principle [CHAN75b, DENN78]. Consider a network with a subsystem containing non-homogeneous behavior. Construct a flow-equivalent server for the subsystem by analyzing the subsystem in isolation under fixed load. Replace the subsystem by its flow-equivalent server in the original network. If the state transitions within the subsystem occur at a much greater rate than interactions between the subsystem and the rest of the system, not much error is introduced by this replacement [COUR75, COUR77].

In the memory queueing problem, we replace the central subsystem by a flow-equivalent server and equat the memory queue with the queue at the flow-equivalent server. This method is directly applicable to single class networks [BRAN74, CHAN75b, COUR75, COUR77] and is examined in detail in Section 2.5.

A direct application of the decomposition technique to multiclass networks produces a flow-equivalent server with service function $S_{Er}(n_1, \ldots, n_R)$. A network containing servers with such service functions does not have a product form solution. Sauer [SAUE81a] used global balance analysis to solve it. Brandwajn [BRAN82], and Lazowska and Zahorjan [LAZO82] used the following approximation:

$$S_{Er}(n) \approx S_{Er}(\bar{n}_1, \ldots, \bar{n}_{r-1}, n, \bar{n}_{r+1}, \ldots, \bar{n}_R),$$

where $S_{Er}(\bar{n}_1, \ldots, \bar{n}_{r-1}, n, \bar{n}_{r+1}, \ldots, \bar{n}_R)$ is the service time for class r when there are n class r and $\bar{n}_j$ class j customers ($j \neq r$) in the subsystem, and $\bar{n}_j$ is the average number of class j customers in the subsystem. With this approximation, the network is effectively decomposed into R single class product form networks. These networks are iteratively solved to determine $\bar{n}_1, \ldots, \bar{n}_R$. This approximation is examined further in Chapter 4.

1.2.3 Simultaneous Resource Possession in I/O Subsystems

Memory queueing was an instance of simultaneous resource possession. Another instance of simultaneous resource possession occurs when a disk is blocked and cannot transfer data because another disk is using the channel. If this blocking is neglected, the resulting model usually underestimates the response time. Several researchers have developed approximate solutions for this problem. We discuss some of them below.

Wilhelm [WILH77] models each disk as an (open) $M/G/1$ queue. The basic service time of disk includes the time for seek, preparation (or search), and data transfer phases. It is elongated to include the effect of channel contention. A model for the probability that the channel (the path to the CPU) is free when requested is used to compute elongation of the service time. His model assumes that only a single path exists for each disk and disks are not shared between different CPUs. This assumption simplifies the calculation of the free path probability. Bard removes this assumption and computes the free path probability by using the maximum entropy principle [BARD80].

Jacobson and Lazowska [JACO82] analyze a closed system containing simultaneous resource possession. Their method generates two models that provide parameters for each other. The first model represents the channel queueing as a pure delay whose value comes from the other model; it computes the delay caused by seek, search and data transfer operations. The second model represents the seek-search-transfer queueing as pure delay whose value comes from the first model; it computes the channel queueing. This technique is applicable to a variety of simultaneous resource possession problems and is discussed in detail in Chapter 3.

1.2.4 Priority Scheduling

Computer systems containing devices at which some job classes have priority over others violate the fair service assumption necessary for product form solution.

Sevcik [SEVC77] analyzed such systems by replacing preemptive priority server with a shadow server for each priority class. The service rate of a shadow server is degraded to reflect the contention from higher-priority customers. This method is discussed in detail in Chapter 2. Improvements on this method are presented in Chapters 3 and 4.

Bryant and Krzesinski [BRYA83], and Chandy and Laksmi [CHAN83] have proposed extensions to Mean Value Analysis (MVA) for analyzing priority systems. They introduce no new servers; instead, they modify the formula for response times at priority servers (see Box 1.2). These two methods are further discussed in Chapter 4.

Agrawal, Buzen and Shum [AGRA84] analyze systems with preemptive priority scheduling at CPU by replacing the CPU by a set of equivalent servers, one server for each priority level. The service times at the equivalent servers are such that the response times at the priority CPU assuming Poisson arrivals are equal to that at the equivalent servers. This method is discussed in Appendix C.

1.2.5 Serialization

Critical sections and database record access are two examples of serialized or single-threaded processing. An exact queueing network model of systems with serialization does not have a product form solution because jobs simultaneously hold a "serialization token" and physical devices.

Response time in MVA at a FCFS, PS, LCFS-PR server

$$R_{ir}(\underline{N}) = S_{ir}\left(1 + \sum_{j=1}^{R} \bar{n}_{ij}(\underline{N} - \underline{1}_r)\right)$$

$(\underline{1}_r = (n_1, \ldots, n_R)$ **with** $n_r = 1$ **and** $n_j = 0, j \neq r.)$

Bryant-Krzesinski's response time formula for a preemptive priority server

$$R_{ir}(\underline{N}) = \frac{S_{ir} + \sum_{j=1}^{r} \bar{n}_{ij}(\underline{N} - \underline{1}_r)S_{ij}}{1 - \sum_{j=1}^{r-1} S_{ij}V_{ij}X_j(\underline{N})}$$

Chandy-Laksmi's response time formula for a preemptive priority server

$$R_{ir}(\underline{N}) = \frac{S_{ir} + \sum_{j=1}^{r} \bar{n}_{ij}(\underline{N} - \underline{1}_r)S_{ij}}{1 - \sum_{j=1}^{r-1} S_{ij}V_{ij}X_j(\underline{N} - \bar{n}_{ij}(\underline{N})\underline{1}_j)}$$

(Class j **has priority over class** r **if** $j < r.$**)**

Box 1.2: Response time expressions for MVA-like priority algorithms.

Agrawal and Buzen [AGRA83] use a shadow server for each critical section in a given computer system. The shadow server's service time is the mean time for that phase of serialized processing. Service times of other devices is degraded to represent concealment of the load imposed by jobs in serialized phases. The details of this method are presented in Chapter 3.

Agre and Tripathi [AGRE82] model reentrant software by representing each software module as a separate server. They allow a module to receive processing only from one device. The service rate of each module server is a function of the number of customers at other modules that execute on the same device. This model is solved using global balance analysis.

Smith and Browne [SMIT80b] divide a job's execution into three phases. Phase 1 is the time from job initiation until the critical section request. Phase 2 is the critical section processing phase. Phase 3 is the processing after the critical section execution. They include a FCFS server WQ (for Wait Queue) to simulate the delay for critical section entry. The service time of WQ, S_{WQ}, equals R_2, the residency time of Phase 2. On switching form Phase 1 to Phase 2, the customer is routed to WQ with the probability

$$P_{WQ} = 1 - \left(\frac{R_1 + R_3}{R_1 + R_2 + R_3}\right)^{N-1},$$

where R_i is the residency time of Phase i. This routing is independent of the number of customers presently in the critical section; hence it does not represent the concurrency constraint satisfactorily.

Jacobson and Lazowska [JACO83] use a two level model for the system. The low level model computes critical section residency times given a fixed load equal to the average number of active non-serialized customers and the average number of customers in each critical section. The high level model uses these residency times to determine the average number of customers

inside each critical section and the average number of customers outside all critical sections. The two models are iteratively solved.

Thomasian [THOM83] has developed two different techniques for modeling serialization delays. First is an iterative technique that is similar to Smith and Browne's technique [SMIT80b] considered earlier; it uses an improved estimate for P_{WQ}. The second is based on state aggregation: the states are aggregated such that the number of customers in each processing phase is the same for all states in an aggregate state. The aggregate state model is solved using global balance solution techniques. This technique is further discussed in Chapter 4.

1.2.6 Internal Program Concurrency

Internal program concurrency results from FORK-JOIN operations and overlapped CPU-I/O processing within a job. FORK and JOIN operations imply nonautonomous behavior of jobs; the child processes are created together and the parent process cannot continue until all child processes have finished. Similar remarks hold for overlapped CPU-I/O processing. (Compare with nonautonomous behavior of disks and channels). No product form solution exists for such systems.

Towsley, Chandy and Browne [TOWS78] model CPU-I/O overlap by replacing the I/O subsystem by its composite flow-equivalent server. The state equations of the resulting network are solved numerically.

For processes containing FORK and JOIN operations, Smith and Browne [SMIT80a] divide jobs into primary and secondary chains. A primary chain job is the one with the largest expected concurrent execution time (i.e., the one which is expected to perform JOIN last); other jobs form a secondary chain. The time from the process (primary job) initiation until the FORK plus

the time from the secondary job JOIN until the primary job completion is represented as pure delay for secondary jobs.

Heidelberger and Trivedi [HEID82, HEID83] have developed approximate models for analyzing asynchronous concurrency (i.e., FORK but no JOIN) and synchronous concurrency (i.e., both FORK and JOIN). In their asynchronous concurrency model [HEID82], forked jobs are represented as open classes. The throughput of the forked jobs should equal the throughput of the parent jobs because a parent process forks one child process of each kind. The model is iteratively solved until balanced throughputs are obtained. They present two techniques for modeling FORK and JOIN operations [HEID83]. The first one involves state aggregation: all states in which the number of active jobs of each type (parent or child process r) is the same are aggregated and the resulting model is solved using global balance solution methods. The second method employs a multiclass model in which each parent and child job is in a separate class. This model also employs additional delay servers to represent the mean synchronization delays between the processes; these delays are iteratively computed. (The second method is similar to that used by Smith and Browne [SMIT80a].) All three techniques are discussed further in Chapter 4.

1.2.7 Blocking

Blocking is a general phenomenon that occurs when the operation of a server is suspended because of the unavailability of resources elsewhere in the network. Examples include:

- finite buffers -- in store and forward communication networks a node cannot transmit a message until the destination node has a buffer available,

- Ethernet -- a node cannot transmit because another node is transmitting on the bus,
- token rings -- a node is blocked until it gets the token, and
- I/O subsystems -- a disk is blocked because the channel is busy transferring data for another disk.

Because of the nonautonomous behavior of these resources, a queueing network with blocking does not have a product form solution. A variety of techniques have been developed to deal with this problem. A brief discussion of some of these techniques follows.

Labetoulle and Pujolle [LABE80] study packet switched networks with finite buffer size at each node by analyzing each queue separately. They model each queue as a $GI/G/1/M_i$ queue with loss (M_i is the number of buffers at node i.). The arrival process (GI) at a queue i is the conjunction of the completion processes (G) of the queues whose output comes to i. If an arrival from queue j to queue i is rejected, it returns to j and a new service of the returned job immediately begins at j. Each queue is solved using diffusion approximation [GELE75, BADE76], and these solutions are reconciled iteratively.

Almes and Lazowska propose a simple markov model for the symmetrical Ethernet control policies [ALME79]. In this model, the message arrival rate is independent of the number of messages already queued, and the message delivery rate of the network when n nodes desire to transmit equals the network capacity times the instantaneous throughput efficiency of the network, E;

$$E = \frac{1}{1 + \frac{1 - A}{A}},$$

where A is the message transmission (the Ether acquisition) probability;

$$A = (1 - 1/n)^{n-1}.$$

Gelenbe and Mitrani [GELE82] model the Ethernet control policies by using a two step iterative procedure. In the first step, they analyze each station in isolation assuming that global parameters, e.g., probability of blocking and probability of determining that a transfer is in progress, are known. In the second step, the results of these single station analyses are used to obtain characteristics of global load and to compute the unknown parameters for station analyses.

Kuehn [KUEH79] analyzes token rings and cyclic service by treating each node i as an $M/G/1$ server. The service time parameters of server i are determined by analyzing token scans in two parts: scans in which no message is transmitted at node i (c'), and the scans in which a message is transmitted from node i (c''). The means and variance of c' and c'' are used to compute the waiting time of the customer.

1.2.8 Approximate Analysis of Large Product Form Networks

Although product form solution algorithms are efficient, their cost rises exponentially with the number of classes in the network. The cost can become prohibitive for sufficiently large networks. As an illustration, consider the MVA equations for a load independent network:

$$R_{ir}(\underline{n}) = S_{ir}\left(1 + \sum_{j=1}^{R} \bar{n}_{ij}(\underline{n} - \underline{1}_r)\right), \quad i = 1, \ldots, K,$$

$$X_r(\underline{n}) = \frac{n_r}{\sum_{i=1}^{K} V_{ir} R_{ir}},$$

$$\bar{n}_{ir}(\underline{n}) = V_{ir} R_{ir}(\underline{n}) X_r(\underline{n}), \quad i = 1, \ldots, K.$$

These equations are solved for $\underline{n} = (\underline{0})$ to $\underline{n} = \underline{N} = (N_1, \ldots, N_R)$. The cost of

solving this network is $RK\prod_{r=1}^{R}(N_r + 1)$ multiplications and and equal number of additions [ZAHO80]. (The same amount of computation is required for the Convolution algorithm.) This number grows rapidly with number of classes R, number of customers in each class, $N_1, \ldots, N_R$; it can become very large. For example a network with 15 servers, 5 classes, and 10 customers in each class requires about 48 million operations. The storage requirement for this network is 322,102 words for the Convolution algorithm and 1,390,895 words for the MVA algorithm. The computational costs for networks with load-dependent servers are much higher; if all the 15 servers in the above example are load-dependent, the number of required operations is about 1500 times as great, and the storage requirement for MVA is about 9 times (2 times for Convolution) as great.

There are two main approaches to reducing this cost. The first approach is to avoid recursion in the computation by solving for the mean values at a single, given load $\underline{N}$. The second approach is to reduce the complexity by reducing the number of classes (R), number of customers in each class (N_r), or the number of servers (K) in the network. The second approach is discussed and analyzed in detail by Zahorjan [ZAHO80]; an overview of the the techniques appears later in Chapter 4. Following examples illustrate the first approach.

Bard-Schweitzer Approximation [SCHW79]:

The principle is to estimate the queue length $\bar{n}_{ir}(\underline{N} - \underline{1}_r)$ from $\bar{n}_{ir}(\underline{N})$ in the response time equations of MVA, thereby obviating the recursion to obtain $\bar{n}_{ir}(\underline{N} - \underline{1}_r)$. The resulting iterative algorithm is:

Initially assume $\hat{\bar{n}}_{ir}(\underline{N}) = \dfrac{N_r}{K}$, $i = 1, \ldots, K$, $r = 1, \ldots, R$.

Repeat

$$\bar{n}_{ir}(\underline{N}) = \hat{\bar{n}}_{ir}(\underline{N})$$

$$\bar{n}_{ir}(\underline{N} - \underline{1}_j) = \begin{cases} \bar{n}_{ir}(\underline{N}) & j \neq r \\ \dfrac{N_r - 1}{N_r}\, \bar{n}_{ir}(\underline{N}) & j = r \end{cases}$$

$$R_{ir}(\underline{N}) = S_{ir}\left(1 + \sum_{j=1}^{R} \bar{n}_{ij}(\underline{N} - \underline{1}_r)\right)$$

$$X_r(\underline{N}) = \frac{N_r}{\sum_{i=1}^{K} V_{ir} R_{ir}(\underline{N})}$$

$$\hat{\bar{n}}_{ir}(\underline{N}) = X_r V_{ir} R_{ir}(\underline{N})$$

until $|\hat{\bar{n}}_{ir}(\underline{N}) - \bar{n}_{ir}(\underline{N})| < \epsilon$.

Linearizer Approximation [CHAN82]:

Chandy and Neuse's Linearizer algorithm is a generalization of the Bard-Schweitzer approximation [CHAN82]. It iteratively estimates queue lengths at loads $\underline{N}$, $\underline{N} - \underline{1}_j$ and $\underline{N} - \underline{1}_j - \underline{1}_l$ to eliminate the need for recursion. The estimator is

$$\bar{n}_{ir}(\underline{N} - \underline{1}_j) = \begin{cases} \bar{n}_{ir}(\underline{N}) + N_r D_{irj}(\underline{N}) & j \neq r \\ \dfrac{N_r - 1}{N_r}\, \bar{n}_{ir}(\underline{N}) + (N_r - 1) D_{irr}(\underline{N}) & j = r, \end{cases}$$

where $D_{irj}(\underline{N})$ is the change in the fraction of class r jobs at queue i resulting from removal of one class j job, i.e.,

$$D_{irj}(\underline{N}) = \begin{cases} \dfrac{\bar{n}_{ir}(\underline{N} - \underline{1}_j)}{N_r} - \dfrac{\bar{n}_{ir}(\underline{N})}{N_r} & j \neq r \\ \dfrac{\bar{n}_{ir}(\underline{N} - \underline{1}_r)}{N_r - \underline{1}_r} - \dfrac{\bar{n}_{ir}(\underline{N})}{N_r} & j = r. \end{cases}$$

The Linearizer algorithm assumes that D_{irj} is a constant. Note that assuming $D_{irj}(\underline{N}) = 0$ gives the same expressions as the Bard-Schweitzer algorithm.

Bound Analysis [DENN78, ZAHO82, EAGE83a]:

A third class of approximations for the solutions of large networks is based on bounds on performance measures. Bounds are, however, available only for single class networks. Bottleneck analysis [DENN78] gives asymptotic upper bounds on network throughput

$$X_0(N) \leq \min\left(\frac{N}{\sum V_i S_i}, \frac{1}{V_b S_b}\right),$$

where subscript b indicates the bottleneck device, i.e., the device with largest $V_i S_i$.

Balanced Job Bound analysis [ZAHO82] provides tighter bounds by comparing the given system with two balanced systems: a faster one in which all devices have the demand equal to the average ($\sum V_i S_i / K$), and a slower one in which all devices have demand equal to the bottleneck ($V_b S_b$). The bounds are:

$$\frac{N}{\left(\dfrac{\sum V_i S_i}{V_b S_b} + N - 1\right) V_b S_b} \leq X(N) \leq \frac{N}{(K + N - 1)\dfrac{\sum V_i S_i}{K}}.$$

Note that $\sum V_i S_i / V_b S_b$ is the number of devices in a network in which the demand at each server is $V_b S_b$, and $\sum V_i S_i / K$ is the average demand per device in a balanced K server network.

A hierarchy of upper and lower bounds on throughput can be obtained by analyzing system with initial queue length bounds for $\bar{n}_i(n_0)$, and then using MVA equations to compute bounds for $n_0, \ldots, N$ customers [EAGE83a]. A tradeoff exists between the cost and the tightness of bounds: smaller n_0 means higher cost and potentially tighter bounds.

1.3 Plan of the Monograph

Most presentations of the approximations reviewed above begin with the specific nonhomogeneous behavior to be modeled, propose a solution, and illustrate its validity with a few examples. Few try to fit their models into any larger scheme of modeling. Fewer still include systematic studies of errors or validations of their proposals. Consequently, it is difficult for the observer to appreciate the generality of the result or safely apply the method in new contexts.

These difficulties hamper communication among researchers, practitioners, and students. They limit the complexity of the problems that can be solved. With increasing complexity of models, the situation only worsens.

The goal of this work is to set forth a single description of the modeling process underlying all the approximations reviewed above. This description is based upon a general queueing network model of the system. This model directly corresponds to a set of states and state transitions; the transitions are represented as a transition rate matrix. There are three approaches to obtaining a solution of a model:

1. *Direct:* Solve the global balance equations implied by the transition rate matrix on the state space.

2. *Sampling:* Observe a sample path (behavior sequence) of the system, or of a faithful simulation, and estimate the desired performance metrics.
3. *Approximation*: Transform the model to a simpler one, solve it, and use the results as an estimate of the original solution.

This monograph focuses on approximation.

Chapter 2 shows that the modeling process can be viewed as a series of simple steps, each of which is is the construction of a transformation from a given model (M_0) to a simpler but less accurate model (M). The modeler must specify a forward map that gives the parameters of M in terms of the parameters and the performance measures of M_0; and a reverse map that gives the performance metrics of M_0 in terms of those of M. The exact solution of M_0 consists of solving the state space equations for M_0. The approximate solution consists of three steps:

1. Map forward to M.
2. Solve M.
3. Map reverse to M_0.

The goal is that these three steps be much faster than a direct solution of M_0 without significant loss of accuracy.

In Chapter 2 we show that the above basic pattern can be used hierarchically -- i.e., the model M can be further transformed and simplified. We also show that iteration arises naturally in the solution of models -- i.e., the three basic steps may need repetition to solve for unknown metrics in M_0. We illustrate the generality of this description by applying it to some of the principal approximations reviewed earlier.

Chapter 3 studies the class of approximations based on adding extra servers, called shadow servers, to represent the additional queueing delay caused by nonhomogeneous behavior in the original system. An important aid

in model development is the behavior sequence transformation, a picture of the way a typical job uses resources as it moves through the system. The method is used to show how the Sevcik shadow CPU approximation and the Jacobson-Lazowska surrogate server approximation can be derived. It is then used to derive a new approximation, the aggregate servers method for modeling serialization.

Chapter 4 generalizes further by characterizing the state spaces underlying models and the ways in which the modeling process manipulates these spaces. The resulting description of model transformations is compact. We use state space transformations to develop a better model, based on a load dependent shadow CPU, for the preemptive priority systems. We also catalog a set of common transformations and illustrate them; they are load concealment, state aggregation, server aggregation, load separation, class aggregation, load scaling, response time modeling, and delay server introduction.

Chapter 5 takes up consistency requirements and convergence proofs for iterative algorithms. We explain why convergence proofs are difficult to obtain in performance modeling. Then, based on the monotonicity properties of queueing networks, we propose two practical techniques for proving convergence. We use these techniques for proving convergence of Bard-Schweitzer algorithm, the Jacobson-Lazowska method for simultaneous resource possession, and the shadow CPU algorithm for preemptive priority at CPU. The basic theorem about convergence predicts the possibility of two or more stable solutions for some networks. This prediction is corroborated with the shadow CPU algorithm, which correctly predicts two possible values for high-priority CPU utilization of a simulated system.

Chapter 6 concludes the monograph by outlining the metamodeling methodology and providing a perspective.

Appendix A presents numerical studies comparing Sevcik's original shadow CPU algorithm, our modifications of the algorithm, the Bryant-Krzesinski priority algorithm, and the Chandy-Laksmi priority algorithm.

Appendix B extends the aggregate server method for open and multi-class mixed networks.

Appendix C presents a general technique for developing approximate analysis methods for a large class of problems. This technique involves replacing a non-product form subsystem by a set of equivalent servers such that the response time at the equivalent servers under an assumed arrival process is same as that at the subsystem. The technique was developed after the metamodeling framework had been formulated. Therefore, it illustrates the usefulness of the research reported in the main body of this monograph.

Appendix D presents some considerations for design and simulation of preemptive priority systems.

As a running example throughout this work, we use a system with preemptive priority scheduling at CPU. In Chapter 2, Sevcik's algorithm applied to this problem illustrates how iteration arises in the basic modeling pattern. In Chapter 3, behavior sequence transformations applied to this problem lead to a better approximation extending Sevcik's. In Chapter 4, state space transformations applied to this problem lead to an even better approximation with a load-dependent server. In Chapter 5, it illustrates the role of consistency requirements, exemplifies one of the techniques for proving convergence and exhibits multiple solutions in some cases. In Appendix C, response time preservation (RTP) transformation applied to this problem yield another efficient solution technique. Each time, we learn

something new about the preemptive priority systems and their modeling through the medium of this example.

1.4 Related Work

Our review of approximate solution techniques in Section 1.2 showed a variety of studies employing heuristics that seem to work but which have not been studied systematically for their general properties. Relatively little effort has been made to understand the process of approximate model development. This section reviews the few works that have.

An early approach to solving complex models of computer systems is based on the concept of near-decomposability; it was proposed by Simon and Ando [SIMO61] and applied to queueing systems by Courtois [COUR77]. The basic idea is simple. Start with a state space that accurately describes the system. Aggregate subsets of states such that (1) the transitions within a subset can be analyzed by assuming that the transitions between subsets do not exist, and (2) the transitions between subsets can be studied without regard to the transitions within subsets.[2] Each aggregate can be solved for its state occupancies internally as if it were a closed system. A macro model is solved for the probability of being in each aggregate. The probability of being in a state is estimated as the product of the two foregoing probabilities.

The process of aggregating the states can be recursively applied to obtain a hierarchy of state-models. Courtois has studied the requirements for near-decomposability and the errors introduced by hierarchical aggregation [COUR77]. In many computer system applications, the errors are small. He

2. If such an aggregation is possible, then system is *nearly-decomposable*. An intuitive condition for near-decomposability is that the transitions within a subset of states be much more frequent than the transitions from the subset to other subsets (i.e., the rest of the states).

also showed that if the service rate of resource R_{k-1} is an order of magnitude higher than the service rate of resource R_k, resources $R_0, \ldots, R_L$ can be aggregated one by one to obtain a linear hierarchy of aggregate resources $A_0, \ldots, A_L$, where A_k is an aggregate of R_k and A_{k-1}.

Chandy, Herzog and Woo's "Norton's Theorem", discussed in Section 1.2.2 on memory queueing, is another method to aggregate resources [CHAN75b].[3] In contrast to Courtois's method, the Norton's theorem has its roots in complete decomposability of product form networks: any set of resources in a product form network is exactly replaceable by its composite flow-equivalent server. Vantilborgh *et al.* have shown that this approach does not introduce significant errors in non-product form networks if the routing matrix is nearly-decomposable [VANT80].

Another general approach to model development was used by Browne *et al.* [BROW75]. They used a *macro* model of the system in which major subsystems are represented as single servers. They obtained the parameters of these servers by analyzing *micro* models for corresponding subsystems. This approach is hierarchical and permits successive refinements of the model.

Similar to Browne *et al.*'s hierarchical approach is the *isolation method* proposed by Labetoulle and Pujolle [LABE80]. They first partition the system into N solvable subsystems and define the input and output interfaces

---

3. Recall, the service function $S(n)$ of the equivalent server is determined by computing the throughput $X(n)$ of the subsystem in isolation under fixed load n and setting $S(n) = 1/X(n)$. This process is the equivalent of setting the service time of other resources to zero, or "short-circuiting" them. In an analogy with electrical circuits, Chandy, Herzog and Woo call this technique as an application of Norton's Theorem. Denning and Buzen call it "On-line behavior = Off-line behavior" because we have equated its on-line (in-system behavior) with its off-line (in isolation) behavior [DENN78].

(processes) for each subsystem. They then iteratively solve the subnetworks to obtain a consistent solution for the original network.

These approaches are similar in spirit but different in details. The major similarity is their use of the principle of hierarchically reducing a complex model to simpler models. The differences arise from their historical roots, their domains of apparent applicability, and their solution algorithms. The differences can easily mask the strong underlying similarities. One of the purposes of this work is to identify the general modeling process being used and show how existing approximate models illustrate this process.