Chapter 1

Probability Theory and Performance Evaluation

In this chapter, we outline the elements of probability theory. Most readers will have taken courses on probability and statistics, and this chapter is meant mainly as a refresher on techniques that are used in the papers collected in this book. Probability is a vast subject, and the Further Reading section contains a brief list of some introductory volumes.

1.1 Introduction

It is possible to make up a rigorous definition of probability, based on sigma algebras and measure theory, but our interests are much more practical. From our point of view, there are some things, called *events*, which have a given probability of occurring. For example, if you have a "fair" coin, the probability of getting a tail upon tossing it is 0.5. This is a way of saying that if you toss a fair coin N times, then

$$\lim_{N\to\infty} \frac{\text{Number of Tails out of } N \text{ Tosses}}{N} = \frac{1}{2} \tag{1.1}$$

Two events that cannot occur at the same time are called *mutually exclusive*. For example, if you have a memory module with one I/O port, you cannot have two simultaneous reads in

progress. Suppose that events $e_1, e_2, \cdots, e_n$ are mutually exclusive, and $P(e_i)$ is the probability of e_i occurring. Define E as the event that one of $e_1, e_2, \cdots, e_n$ occurs. Then,

$$\text{Prob}\{E \text{ occurs}\} = \sum_{i=1}^{n} P(e_i) \tag{1.2}$$

The probability of any event must lie between 0 and 1. An event that occurs with probability 1 is said to happen *almost surely*. This term points up a common misconception about probabilities: namely, that an event with probability 1 is certain to occur, and an event with probability 0 will never occur. This is not true. For example, suppose we run an experiment which consists of choosing a random point in an interval $[0, 1]$. There is an uncountable infinity of such points, and so the probability that we choose a given point (say, 0.55), is zero. In fact, *every* outcome in this experiment will have probability zero!

Given events $A_1, A_2, \cdots, A_n$ the event that they all occur is expressed by $A_1 \cap A_2 \cap \cdots \cap A_n$, while the event that at least one of them occurs is denoted by $A_1 \cup A_2 \cup \cdots \cup A_n$. The notation arises from the fact that we can denote the event probabilities by area in a Venn diagram. In such a diagram, $\text{Prob}(A_i)$ would be proportional to the area occupied by A_i. The probability of the event that either A_1 or A_2 (or both) occurred is proportional to the area of $A_1 \cup A_2$. Similarly, the probability of both A_1 and A_2 occurring is represented by the area covered by $A_1 \cap A_2$.

We have

$$\text{Prob}(A_1 \cup A_2) = \text{Prob}(A_1) + \text{Prob}(A_2) - \text{Prob}(A_1 \cap A_2) \tag{1.3}$$

The $\text{Prob}(A_1 \cap A_2)$ term corrects for the double-counting that occurs when both events occur. We can extend Equation 1.3 recursively. For example,

$$\begin{aligned}
\text{Prob}(A_1 \cup A_2 \cup A_3) &= \text{Prob}(A_1 \cup A_2) + \text{Prob}(A_3) - \text{Prob}([A_1 \cup A_2] \cap A_3) && (1.4)\\
\text{Prob}([A_1 \cup A_2] \cap A_3) &= \text{Prob}([A_1 \cap A_3] \cup [A_2 \cap A_3]) \\
&= \text{Prob}(A_1 \cap A_3) + \text{Prob}(A_2 \cap A_3) - \text{Prob}([A_1 \cap A_3] \cap [A_2 \cap A_3]) \\
&= \text{Prob}(A_1 \cap A_3) + \text{Prob}(A_2 \cap A_3) - \text{Prob}(A_1 \cap A_2 \cap A_3) && (1.5)
\end{aligned}$$

Thus,

$$\begin{aligned}
\text{Prob}(A_1 \cup A_2 \cup A_3) &= \text{Prob}(A_1) + \text{Prob}(A_2) + \text{Prob}(A_3) \\
&\quad -\text{Prob}(A_1 \cap A_2) - \text{Prob}(A_1 \cap A_3) - \text{Prob}(A_2 \cap A_3) \\
&\quad +\text{Prob}(A_1 \cap A_2 \cap A_3) && (1.6)
\end{aligned}$$

It is probably easier to see this graphically, using a Venn diagram. See Figure 1.1, and relate the area enclosed by the circles to the probabilities in Equation 1.6.

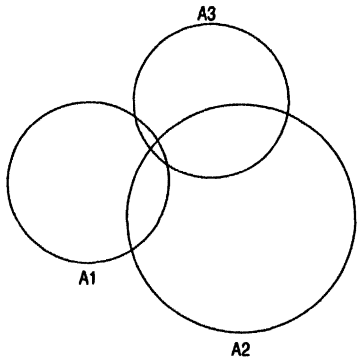

Figure 1.1. Venn Diagram for Prob $(A_1 \cup A_2 \cup A_3)$

If $A_1, \cdots, A_n$ is the set of all possible events (n may or may not be finite), then

$$\text{Prob}\,(A_1 \cup A_2 \cup \cdots \cup A_n) = 1 \tag{1.7}$$

Two events A and B are said to be *independent* if

$$\text{Prob}\,(A \cap B) = \text{Prob}\,(A) \times \text{Prob}\,(B) \tag{1.8}$$

For example, if successive tosses of a fair coin are independent, the probability that we have a head followed by a tail is $0.5 \times 0.5 = 0.25$. We can extend Equation 1.8 recursively. If A, B, C are independent events, then

$$\text{Prob}\,(A \cap B \cap C) = \text{Prob}\,(A) \times \text{Prob}\,(B) \times \text{Prob}\,(C) \tag{1.9}$$

One of the most useful concepts in probability theory is *conditional probability.* We denote by Prob $(B|A)$ the probability that event B occurs, *given that* event A has occurred. The fundamental equation relating to conditional probability is Bayes' law:

$$\text{Prob}\,(B|A) = \frac{\text{Prob}\,(A \cap B)}{\text{Prob}\,(A)} \tag{1.10}$$

We can rewrite Bayes' law as follows:

$$\text{Prob}\,(A \cap B) = \text{Prob}\,(B|A) \times \text{Prob}\,(A) \tag{1.11}$$

This leads to the following useful construct

$$\text{Prob}\,(B|A) \;=\; \frac{\text{Prob}\,(A \cap B)}{\text{Prob}\,(A)}$$

$$= \frac{\text{Prob}\,(A|B) \times \text{Prob}\,(B)}{\text{Prob}\,(A)} \tag{1.12}$$

1.2 Random Variables

A random variable can be formally defined as a mapping from the set of events to the real line. That is, a random variable is a function, which associates a real number with each event. The real number usually denotes some parameter of physical interest. For example, if the event is accessing memory, we can define a random variable t_{access} which is the memory access time. Suppose we are given that an access is to cache with probability p_{cache}, to main memory with probability p_{main}, to disks with probability p_{disks}, and to tape with probability p_{tape}. Denote by $t(cache)$, $t(main)$, $t(disks)$, and $t(tape)$ the access times associated with these various media. We can now write

$$t_{access} = \begin{cases} t(cache) & \text{with probability } p_{cache} \\ t(main) & \text{with probability } p_{main} \\ t(disks) & \text{with probability } p_{disks} \\ t(tape) & \text{with probability } p_{tape} \end{cases} \tag{1.13}$$

Associated with each random variable, X, is a *probability distribution function* (PDF),

$$F_X(x) = \text{Prob}\,\{X \leq x\} \tag{1.14}$$

Assuming that $t(cache) < t(main) < t(disks) < t(tape)$, we can write the PDF of t_{access} as

$$F_{t_{access}}(t) = \begin{cases} 0 & \text{if } t < t(cache) \\ p_{cache} & \text{if } t(cache) \leq t < t(main) \\ p_{main} + p_{cache} & \text{if } t(main) \leq t < t(disks) \\ p_{disks} + p_{main} + p_{cache} & \text{if } t(disks) \leq t < t(tape) \\ 1 & \text{otherwise} \end{cases} \tag{1.15}$$

If the PDF is a differentiable function (that is, it can be differentiated), its derivative is called the *probability density function*, (pdf). If the PDF takes discrete jumps (as was the case with t_{access} in our example), a pdf does not exist and we can define instead a *probability mass function* (pmf),

$$m_X(x) = \text{Prob}\,\{X = x\} \tag{1.16}$$

For example, the pmf of t_{access} is given by

$$m_{t_{access}}(t) = \begin{cases} p_{cache} & \text{if } t = t(cache) \\ p_{main} & \text{if } t = t(main) \\ p_{disks} & \text{if } t = t(disks) \\ p_{tape} & \text{if } t = t(tape) \\ 0 & \text{otherwise} \end{cases} \tag{1.17}$$

The *expectation*, $E[X]$, of a random variable, X, is its average or mean. If the random variable takes discrete values from the set $A = \{a_1, a_2, \cdots\}$, it has a pmf, and we have

$$E[x] = \sum_{i \in A} a_i m_X(a_i) \tag{1.18}$$

If X has a pdf, we have

$$E[X] = \int_{x=-\infty}^{\infty} x f_X(x)\, dx \tag{1.19}$$

If the expectation of X is finite, we can also determine it by the expression

$$E[X] = \int_{x=0}^{\infty} (1 - F_X(x))\, dx - \int_{x=-\infty}^{0} F_x(x)\, dx \tag{1.20}$$

If you remember elementary techniques of integration, you might try deriving Equation 1.20 from Equation 1.19.

Expectation is a linear operator. That is a fancy way of saying that

$$E[X_1 + X_2 + \cdots + X_n] = E[X_1] + E[X_2] + \cdots + E[X_n] \tag{1.21}$$

The n'th moment of a random variable X is $E[X^n]$. The first moment is, of course, the mean. The *variance* of X is given by

$$V[X] = E[X^2] - (E[X])^2 \tag{1.22}$$

The *standard deviation* of a random variable is the square root of its variance.

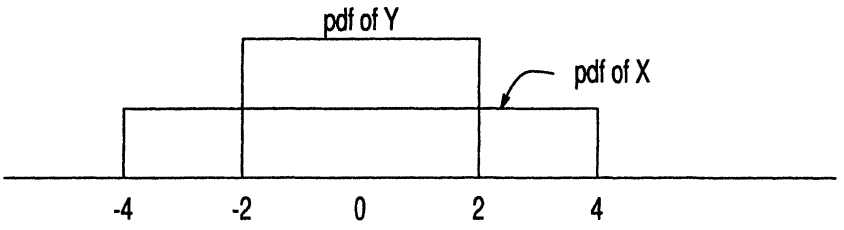

Figure 1.2. Illustrating Variance

While the first moment gives us the average value, the variance tells us how much variation or spread we can expect. For example, consider the random variables X and Y with pdf's shown in Figure 1.2. We have

$$f_X(x) = \begin{cases} 0.125 & \text{if } -4 \leq x \leq 4 \\ 0 & \text{otherwise} \end{cases} ; \; f_Y(x) = \begin{cases} 0.250 & \text{if } -2 \leq x \leq 2 \\ 0 & \text{otherwise} \end{cases} \tag{1.23}$$

While random variables both have a mean of 0, it is clear that X is more "spread out." This is reflected in the variances: $V[X] = 8$; $V[Y] = 2$.

Let us now turn to the distribution of the sum of independent random variables. This will also give us an opportunity to demonstrate the usefulness of Bayes' law. If X, Y are independent random variables with pdf's, then

$$\begin{aligned} \text{Prob}\,(X + Y < w) &= \int_{x=-\infty}^{\infty} \text{Prob}\,(X + Y < w \cap X = x)\, dx \\ &= \int_{x=-\infty}^{\infty} \text{Prob}\,(X + Y < w | X = x) f_X(x)\, dx \\ &= \int_{x=-\infty}^{\infty} \text{Prob}\,(Y < w - x | X = x) f_X(x)\, dx \\ &= \int_{x=-\infty}^{\infty} f_Y(w - x) f_X(x)\, dx \end{aligned} \tag{1.24}$$

We can apply this expression recursively to the sum of more than two variables. For example,

$$\text{Prob}\,(X+Y+Z<w)=\int_{x=-\infty}^{\infty} f_{Y+Z}(w-x)f_X(x)\,dx \tag{1.25}$$

where f_{Y+Z} is the pdf of the random variable $Y+Z$.

Let us now look at two important PDFs. Perhaps the simplest is the *uniform* distribution, which we have already encountered in Figure 1.2. For a continuous random variable, the PDF and pdf are

$$F_X(x)=\begin{cases} 0 & \text{if } x<a_1 \\ \dfrac{x-a_1}{a_2-a_1} & \text{if } a_1\le x\le a_2 \\ 1 & \text{otherwise} \end{cases} \;;\; f_X(x)=\begin{cases} 1/(a_2-a_1) & \text{if } a_1\le x\le a_2 \\ 0 & \text{otherwise} \end{cases} \tag{1.26}$$

The exponential distribution and density functions are given by

$$F_X(x)=\begin{cases} 1-e^{-\mu x} & \text{if } x>0 \\ 0 & \text{otherwise} \end{cases} \;;\; f_X(x)=\begin{cases} \mu e^{-\mu x} & \text{if } x>0 \\ 0 & \text{otherwise} \end{cases} \tag{1.27}$$

A random variable which is exponentially distributed is said to be *memoryless*. The reason for this lies in the following computation:

$$\begin{aligned} \text{Prob}\,(X>x\,|\,X>a) &= \frac{\text{Prob}\,(X>x\cap X>a)}{\text{Prob}\,(X>a)} \\ &= \begin{cases} \dfrac{\text{Prob}\,(X>a)}{\text{Prob}\,(X>a)} & \text{if } a\ge x \\ \dfrac{\text{Prob}\,(X>x)}{\text{Prob}\,(X>a)} & \text{if } a<x \end{cases} \end{aligned} \tag{1.28}$$

Since $\text{Prob}\,(X>t)=e^{-\mu t}$, we have

$$\text{Prob}\,(X>x\,|\,X>a)=\frac{\text{Prob}\,(X>x\cap X>a)}{\text{Prob}\,(X>a)}=\begin{cases} 1 & \text{if } a\ge x \\ e^{-\mu(x-a)} & \text{if } a<x \end{cases} \tag{1.29}$$

If $a < x$, we have from Equation 1.29 that

$$\text{Prob}\,(X > x \mid X > a) = \text{Prob}\,(X > x - a) \tag{1.30}$$

which is a function of the *difference* between x and a. That is, for every $\delta > -a$,

$$\text{Prob}\,(X > x \mid X > a) = \text{Prob}\,(X > x + \delta \mid X > a + \delta) \tag{1.31}$$

If, for example, a light bulb has an exponentially-distributed lifetime, the probability that it will burn out over the next hour is not a function of how old it is, but only of whether it has yet burned out or not. That is why this distribution is called memoryless. This property is very important in mathematical modeling.

Associated with the exponential distribution is the *Poisson process.* Consider some events, such as memory requests, that occur over a period of time. Let $N(t)$ denote the number of such events over the interval of time $[0, t]$. The event-arrival process is called *Poisson* with rate $\lambda(t)$ if:

- The probability of one or more events occurring in an interval $[a, b]$ is unaffected by what happened outside this interval.
- The probability of an event occurring in an interval $[t, t + dt]$ is $\lambda(t)\,dt$.
- The probability of two events occurring in an interval of length dt is of the order of $(dt)^2$ or less.

If $\lambda(t) = \lambda$ for all t, we have a *homogeneous* Poisson process. We can show that

$$\text{Prob}\,(N(t) = n) = e^{-\int_{x=0}^{t} \lambda(x)\,dx} \frac{\left\{\int_{x=0}^{t} \lambda(x)\,dx\right\}^n}{n!} \tag{1.32}$$

If $\lambda(t) = \lambda$ for all t, we have

$$\text{Prob}\,(N(t) = n) = e^{-\lambda t} \frac{\{\lambda t\}^n}{n!} \tag{1.33}$$

That there is a relationship between the Poisson process and the exponential distribution is demonstrated as follows. Denote by τ the time between two successive event occurrences. We have

$$\begin{aligned} \text{Prob}\,(\tau > t) &= \text{Prob}\,(N(t) = 0) \\ &= e^{-\lambda t} \end{aligned} \tag{1.34}$$

Thus, the interarrival time (that is, the time between successive event arrivals) of a Poisson process is exponentially distributed.

Another important process is the *Bernoulli process.* Consider a set of random variables, $X_1, X_2, \cdots, X_n, \cdots$, which can take only two values: 0 and 1. The sum $S_n = X_1 + X_2 + \cdots + X_n$, $n = 1, 2, \cdots$, is called a Bernoulli process.

Suppose $\text{Prob}\,(X_i = 1) = p$, and $\text{Prob}\,(X_i = 0) = 1 - p$ for all $i = 1, 2, \cdots$. Then,

$$\text{Prob}\,(S_1 = k) = \begin{cases} p & \text{if } k = 1 \\ 1 - p & \text{if } k = 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.35}$$

$$\begin{aligned} \text{Prob}\,(S_2 = k) &= \text{Prob}\,(S_1 + X_2 = k) \\ &= \text{Prob}\,([S_1 = k - 1 \cap X_2 = 1] \cup [S_1 = k \cap X_2 = 0]) \\ &= \text{Prob}\,(S_1 = k - 1)p + \text{Prob}\,(S_1 = k)(1 - p) \end{aligned} \tag{1.36}$$

$$\vdots$$

$$\text{Prob}\,(S_n = k) = \text{Prob}\,(S_{n-1} = k - 1)p + \text{Prob}\,(S_{n-1} = k)(1 - p) \tag{1.37}$$

It is easy to show (try it) that this series of equations yields

$$\text{Prob}\,(S_n = k) = \binom{n}{k} p^k (1 - p)^{n-k} \tag{1.38}$$

where $\binom{n}{k}$ is the number of combinations of n things, taken k at a time. You will no doubt remember from elementary algebra that

$$\binom{n}{k} = \begin{cases} \dfrac{n!}{k!(n-k)!} & \text{if } k \leq n \\ 0 & \text{otherwise} \end{cases} \tag{1.39}$$

As an aside, perhaps the best way to compute these functions is to use the recursion

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \tag{1.40}$$

As an elementary exercise, try proving this result.

1.3 Markov Chains

Markov chains are perhaps the most important tool in the development of mathematical performance models. In this section, we will provide an informal (well, almost informal) treatment. We will make no claims of rigor, restricting ourselves to some common sense observations and basic mathematics.

Everyone is familiar with the idea of a finite-state machine (FSM). It consists of a finite set of *states* and *transition rules*. At any time, the system must be in exactly one state. The transition rules govern how the system moves from state to state, usually in response to a clock and other inputs. The next state thus depends only on

- The present state;
- The input(s), if any; and
- The transition rules.

An FSM is deterministic (if it weren't, computing would be impossible!). In other words, if you have two identical FSMs, start them in the same initial state, and provide them with identical inputs, you will get identical outputs. Markov chains are very similar to FSMs, except in two crucial respects:

- They may have a finite or a countably infinite number of states.[1]
- Their state transitions are usually not deterministic: it is possible to have *probabilistic* transition rules. That is, we can specify that the system will move from, say, state 1 to state 2, with probability $p_{1,2}(i)$ in response to an input, i.

[1]By "countably infinite," we mean that it is possible to set up a one-to-one mapping between the states of the Markov chain and the set of integers. For example, the set of rational numbers is countably infinite, while the set of real numbers is not. Consult any book on real analysis for further information.

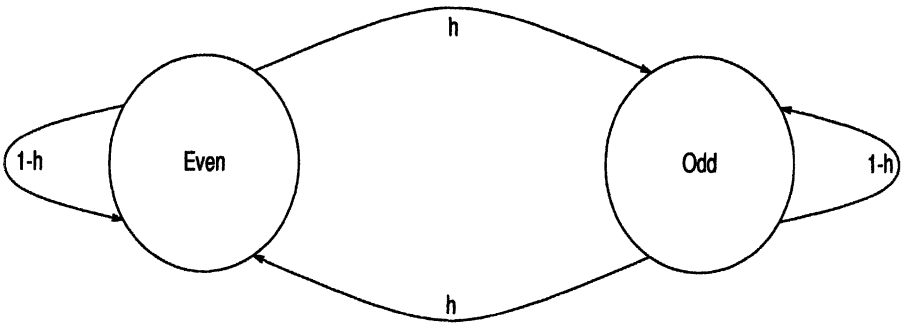

Figure 1.3. Markov Chain for First Coin-Tossing Example

Because their state transitions can be probabilistic, it is possible to take two identical Markov chains, start them in the same initial state, apply identical inputs, and yet end up in different states. When dealing with Markov chains, we are interested in finding the probability of the chain being in a particular state.

A Markov chain may be either *discrete-* or *continuous-*time. A discrete-time chain only undergoes state changes at integral multiples of some time granule (that is, a clock), while continuous-time chains can undergo state changes at any time.

Let us consider a few toy examples of discrete-time chains. Consider a situation where we toss an unfair coin once every clock period. We are not concerned with the total number of heads and tails that result: only with whether that total number is odd or even. Given that we start with even parity at time 0, what is the probability of having even parity at time n, for any $n = 1, 2, \cdots$? Let the probability of having a head be h; that of a tail is $1 - h$.

There are only two states: odd and even. At any time, the next state that the system goes to depends only on the present state, and the outcome of the present coin toss. It is therefore a Markov chain.

Figure 1.3 shows the Markov chain associated with this example. The arcs are labelled with the transition probabilities, which are calculated from the following table:

Present State	Event	Next State	Probability
Odd	Head	Even	h
Odd	Tail	Odd	1-h
Even	Head	Odd	h
Even	Tail	Even	1-h

We can write the probability of the system being in a state at time i as a function of its state at time $i-1$. That is,

$$p_{odd}(i) = p_{odd}(i-1) \cdot (1-h) + p_{even}(i-1) \cdot h \quad (1.41)$$

$$p_{even}(i) = p_{odd}(i-1) \cdot h + p_{even}(i-1) \cdot (1-h) \quad (1.42)$$

Note that $p_{even}(i) + p_{odd}(i) = 1$, since the chain must be in one of its states at any one time. Thus, one of these equations is redundant. Let us drop Equation 1.42 from consideration, and limit ourselves to Equation 1.41. We therefore have:

$$\begin{aligned} p_{odd}(i) &= p_{odd}(i-1) \cdot (1-h) + (1 - p_{odd}(i-1)) \cdot h \\ &= p_{odd}(i-1) \cdot (1-2h) + h, \quad i > 0 \end{aligned} \quad (1.43)$$

If $h = 0$, every toss of the coin will turn up tails, and there will be no state change, that is, the system will be frozen at its initial state. If $h = 1$, every toss will turn up heads and there will be a state change every clock period. In both cases, the memory of the initial state will propagate to eternity. If $h = 0$, we will have $\lim_{i\to\infty} p_{odd}(i) = p_{odd}(0)$, and if $h = 1$, $\lim_{i\to\infty} p_{odd}(i)$ does not exist. Now, if $0 < h < 1$, the limit $\lim_{i\to\infty} p_{odd}(i)$ exists, and is independent of the initial state (that is, $p_{odd}(0)$): we can see this by inspection, and will not show this formally. How can we obtain this limit? The easiest way of doing this (once we are assured that such a limit exists) is to take limits in Equation 1.43 as follows:

$$\lim_{i\to\infty} p_{odd}(i) = \lim_{i\to\infty} p_{odd}(i-1)(1-2h) + h \quad (1.44)$$

But, $\lim_{i\to\infty} p_{odd}(i) = \lim_{i\to\infty} p_{odd}(i-1)$. For brevity, write $\lim_{i\to\infty} p_{odd}(i) = \pi_{odd}$. We therefore have from Equation 1.44,

$$\pi_{odd} = \pi_{odd} \cdot (1-2h) + h \quad (1.45)$$

$$\Rightarrow \pi_{odd} = 1/2 \quad (1.46)$$

Equations such as Equation 1.46 are commonly referred to as *balance equations*. They can usually be written down by inspection of the Markov chain, by balancing the "flow" out of a state with the "flow" into it. The intuitive argument is that if the probability of being in a state does not change with time (which is the case here in the limit as time goes to infinity), there must be an equality of flow into, and out of, each state. In our example, the "flow" out of state **odd** was given by $\pi_{odd}h$, and the flow into state **even** was given by $(1-\pi_{odd})h$. Equating, we obtain

$$\pi_{odd}h = (1-\pi_{odd})h \quad (1.47)$$

which yields the result $\pi_{odd} = 1/2$.

In general, the flow out of a state in a discrete-time chain is the product of the probability of being in that state and the transition probability.

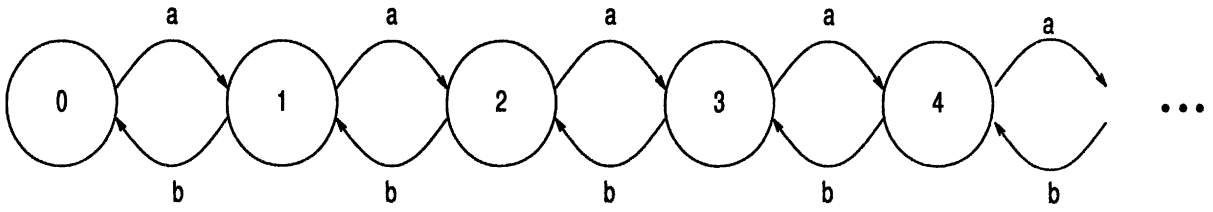

Figure 1.4. Markov Chain for Second Example: Birth-Death Process

Probabilities of the form $\lim_{i\to\infty} p_{odd}(i)$ are called *steady-state* probabilities for obvious reasons.

As a second example, consider the discrete-time chain in Figure 1.4. If the system is in state i, at each clock tick, the probability of a transition to state $i+1$ is a, and for all $i > 0$ the probability of a transition to $i-1$ is b. This Markov chain represents a *birth-death* process: the term arose from considering each move to the right a birth, and each move to the left a death. The balance equations can be written down by inspection. Once again, denote by π_i the steady-state probability of being in state i.

$$\begin{aligned} a\pi_0 &= b\pi_1 \\ (a+b)\pi_i &= a\pi_{i-1} + b\pi_{i+1}, \quad i > 0 \end{aligned} \tag{1.48}$$

We also have the boundary condition that all the probabilities must add to one:

$$\pi_0 + \pi_1 + \cdots + \pi_n + \cdots = 1 \tag{1.49}$$

To obtain the value of π_i, $i = 0, 1, \cdots$, we use Equation 1.48 to express all the π_i in terms of π_0, and then use Equation 1.49 to solve for π_0. We have from Equation 1.48 and some algebra,

$$\begin{aligned} \pi_1 &= (b/a)\pi_0 \\ \pi_2 &= (b/a)^2\pi_0 \\ &\vdots \\ \pi_n &= (b/a)^n\pi_0 \\ &\vdots \end{aligned} \tag{1.50}$$

From Equations 1.49 and 1.50, we have

$$\begin{aligned} (1 + (b/a) + (b/a)^2 + \cdots + (b/a)^n + \cdots)\pi_0 &= 1 \\ \Rightarrow \pi_0 &= \frac{1}{1-(b/a)} \end{aligned} \tag{1.51}$$

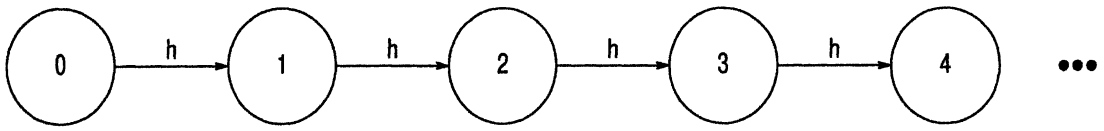

Figure 1.5. Markov Chain for Third Example: Coin Tossing

Note that we have $\pi_i > 0$ only if $b/a > 1$, that is, if $b > a$. It is only if $b > a$ that steady-state exists.

Let us now construct a third example of Markov chains. This time, let us count the number of heads that are generated by n coin tosses, for $n = 1, 2, \cdots$. The Markov chain for this system is shown in Figure 1.5. Note that this chain has an infinite number of states. State i represents the situation where i heads have been obtained. Let $p_i(n)$ denote the probability of obtaining i heads after n tosses. We have

$$p_i(n) = hp_{i-1}(n-1) + (1-h)p_i(n-1) \tag{1.52}$$

Note that unless $h = 0$, the limit $\lim_{n\to\infty} p_i(n) = 0$, for all $i = 0, 1, 2, \cdots$. A state whose probability goes to zero in the limit as time goes to infinity is called a *transient* state. In this chain, every state is transient.

1.4 Queues

A *queue* is a waiting area where jobs are held, awaiting service. They are served in first-come-first-served, or some other prespecified order. There is a vast literature on queues, and we restrict ourselves here to providing the bare minimum needed to understand the papers in this volume.

Let us begin with some notation. An A/B/C/D/E queue means the following:

- A refers to the arrival process.
- B refers to the service process.
- C is the number of servers.
- D is the maximum number of jobs for which the queue has room. The default value is ∞.
- E is the maximum number of jobs. The default value is ∞.

For example, the queue A/B/1 refers to a queue with arrival process denoted by A, service process denoted by B, and having a single server. Since the fourth and fifth fields are not specified, the default values are assumed for the waiting room and the maximum number of jobs.

We denote by M a Poisson arrival process or an exponentially distributed service process; by D a deterministic arrival or service process, and by G a general arrival or service process. For example, the queue M/M/1 means that:

- Jobs arrive according to a Poisson process.
- The service time of each job is exponentially distributed.
- There is one server.
- There is infinite waiting room and no limit on the number of jobs.

Similarly, the queue M/G/1 means that:

- Jobs arrive according to a Poisson process.
- The service time of each job is generally distributed (that is, there is no restriction on the form the service-time distribution may take).
- There is one server.
- There is infinite waiting room and no limit on the number of jobs.

and the queue M/D/1 means that:

- Jobs arrive according to a Poisson process.
- The service time of each job is *deterministic*, that is, each job takes exactly the same service time.
- There is one server.
- There is infinite waiting room and no limit on the number of jobs.

We shall not go into how to analyze these queues, but content ourselves with pointing out some basic formulas. If the mean arrival rate into a queue is λ, the expected waiting time in

the queue, W, and the mean number of jobs in the queue, L, are related by Little's Law[2]:

$$L = \lambda W \tag{1.53}$$

The Pollaczek-Khinchine (PK) formulas apply to M/G/1 queues. Let $W^*(s)$ and $B^*(s)$ denote the Laplace transforms of the pdfs of the job waiting and service times, let the arrival rate be λ, and let the mean job service time be τ. Further, let $N(z) = \sum_{i=0}^{\infty} \pi_i z^i$, where π_i is the probability of i jobs in the queue. $N(z)$ is called the z-transform of π_i, $i = 0, 1, cdots$. The PK formulas are as follows:

$$N(z) = B^*(\lambda - \lambda z)\frac{(1-\tau\lambda)(1-z)}{B^*(\lambda - \lambda z) - z} \tag{1.54}$$

$$W^*(s) = \frac{s(1-\rho)}{s - \lambda + \lambda B^*(s)} \tag{1.55}$$

The Laplace and z transforms can be used to find the moments of the waiting time and number in the queue. It is easy to show that if $A^*(s)$ is the Laplace transform of a pdf $a(x)$, the n'th moment of that pdf is given by

$$(-1)^n \left.\frac{d^n A^*(s)}{ds^n}\right|_{s=0} \tag{1.56}$$

and if $N(z)$ is the z-transform of the pmf of random variable X, given by

$$E[X] = \left.\frac{dN(z)}{dz}\right|_{z=1} \tag{1.57}$$

$$E[X(X-1)] = \left.\frac{d^2N(z)}{dz^2}\right|_{z=1} \tag{1.58}$$

Equations 1.56 and 1.58 are easy to verify. We can use them to find an expression for the mean waiting time in an M/G/1 queue:

$$W = \frac{\lambda}{2(1-\lambda\tau)}\left\{\tau^2 + \sigma_s^2\right\} \tag{1.59}$$

[2]There are some types of queues which do not follow Little's law, but these will not be encountered in this volume.

where σ_s is the standard deviation of the service time. The mean number in the queue is given by

$$L = \lambda\tau + \frac{\lambda^2 + \sigma_s^2}{2(1 - \lambda\tau)} \tag{1.60}$$

Equations 1.59 and 1.60 are also sometimes referred to as the Pollaczek-Khinchine formulas.

1.5 The Role of Analytical Performance Models

It is important to understand where analytical models can, and cannot, be used. To ensure that they are tractable, almost all analytical performance models are approximate. Also, there is often no way to tightly bound the accuracy of such models. That is, one cannot guarantee that the real performance measure is within $x\%$ of that predicted by the model, for some finite $x\%$. Usually, the only way to assess the accuracy of the model is to run a few simulations and compare the simulation and model outputs.

The approximate nature of performance models is often acceptable for two reasons. First, the models themselves might be used to explore design alternatives, and it is sufficient to have approximate estimates to correctly rank the alternatives. Second, it may be impossible to accurately estimate the input parameters for the model. For example, we may have only an approximate idea of the workload. In such cases, approximate estimates are all that is theoretically possible.

If more accurate performance characterization is needed (and we have sufficiently accurate workload information to make it possible), the designer must turn to simulation or experiments on a prototype. Of course, one has to pay for the additional accuracy: writing simulation models and developing prototypes are neither easy nor inexpensive tasks.

One should also realize that the quality of the output of a performance model depends also on the quality of the input data, and on the appropriateness of the chosen performance measure. No matter how good the model may be, it cannot be expected to give accurate results if the input data are wrong or not representative of the workload that the system will be subjected to in practice. Collecting representative workload data is critical to accurate performance prediction.

The appropriateness of the performance measure is a more subtle factor than the accuracy of the model or the representative nature of the input data, but is no less important. A good performance measure will have the following characteristics:

(a) The measure will be relevant or meaningful in the context of the application.

(b) The measure will allow an unambiguous comparison to be made between machines.

(c) It will be possible to develop models to estimate this performance measure.

(d) The model to estimate this performance measure will not be very difficult to collect or estimate.

It is usually impossible to meet all four requirements. (c) and (d) are essential if the measure is to be practically meaningful, and we try to do the best we can with respect to (a) and (b).

This introductory chapter contains one paper. This is an excellent survey of performance evaluation techniques. In addition, the authors also discuss the gathering of workload data and simulation techniques.

1.6 Suggestions for Further Reading

1. L. Kleinrock, *Queuing Systems, Vols. 1 and 2,* New York: Wiley, 1975 and 1976.

   This is an excellent, if slightly dated, introduction to performance evaluation using queuing theory. It is meant for readers with a fair background in probability theory, and is highly recommended.

2. K.S. Trivedi, *Probability & Statistics with Reliability, Queuing, and Computer Science Applications,* Englewood Cliffs: Prentice-Hall, 1982.

   This is a very good introduction to probability techniques used in performance evaluation. It requires no prior knowledge of probability.

3. W. Feller, *An Introduction to Probability Theory and its Applications,* (2 vols.), New York: John Wiley, 1968, 1971.

   This book is a classic. Many regard this as the best book on probability theory they have ever read. Not only does it provide an excellent introduction to probability theory, but it includes a wealth of examples which illustrate the applicability of probability to a wide range of fields.

4. D.E. Knuth, R.L. Graham, and O. Patasnik, *Concrete Mathematics,* Reading: Addison-Wesley, 1989.

   This is a well-written book containing much of the mathematics that computer scientists should know.

Other useful sources include:

- A.O. Allen, *Probability, Statistics, and Queuing Theory with Computer Science Applications,* New York: Academic Press, 1978.
- D. Ferrari, *Computer Systems Performance Evaluation,* Englewood Cliffs: Prentice-Hall, 1978.
- H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology,* Reading: Addison-Wesley, 1978.
- E.A. MacNair and C.H. Sauer, *Elements of Practical Performance Modeling,* Englewood Cliffs: Prentice-Hall, 1985.

Computer Performance Evaluation Methodology

PHILIP HEIDELBERGER, MEMBER, IEEE, and STEPHEN S. LAVENBERG, MEMBER, IEEE

(Invited Paper)

Abstract—**The quantitative evaluation of computer performance is needed during the entire life cycle of a computer system. We survey the major quantitative methods used in computer performance evaluation, focusing on post-1970 developments and emphasizing trends and challenges. We divide the methods used into three main areas, namely performance measurement, analytic performance modeling, and simulation performance modeling, which we survey in the three main sections of the paper. Although we concentrate on the methods per se, rather than on the results of applying the methods, numerous application examples are cited. The methods to be covered have been applied across the entire spectrum of computer systems from personal computers to large mainframes and supercomputers, including both centralized and distributed systems. The application of these methods has certainly not decreased over the years and we anticipate their continued use as well as their enhancement when needed to evaluate future systems.**

Index Terms—**Computer performance measurement, computer performance modeling, computer workload characterization, discrete event simulation, queueing networks.**

I. Introduction

PERFORMANCE is one of the key factors that needs to be taken into account in the design, development, configuration, and tuning of a computer system. Hence, the quantitative evaluation of computer performance is required during the entire life cycle of a system. (The evaluation of a computer system can also involve such factors as function, ease of use, cost, availability, reliability, serviceability, and security but we will not consider these factors here.) In this paper we will survey the major quantitative methods used in computer performance evaluation. We will focus on post-1970 developments and emphasize trends and challenges for the future. We will concentrate on the methods per se, rather than on the results of applying the methods, but will also cite numerous application examples. The methods to be covered have been applied across the entire spectrum of computer systems from personal computers to large mainframes and supercomputers, including both centralized and distributed systems. (Although the methods have also been applied in evaluating the performance of communication networks, we will not consider network performance evaluation.) The main challenge to be faced in computer performance evaluation is that the development of the required performance evaluation methods keep pace with the explosion of new system designs brought on by rapid technological advances.

Manuscript received March 7, 1984; revised August 8, 1984.

The authors are with the IBM T. J. Watson Research Center, Yorktown Heights, NY 10598.

As we will see, a broad spectrum of skills is needed in computer performance evaluation. The skills range from designing and implementing measurement instrumentation to mathematically analyzing queueing models of computer performance. While computer performance evaluation is of great practical importance to computer manufacturers and computer installation managers, it is also an area of considerable research activity both in industry and universities. The number of recent books which deal with aspects of computer performance evaluation attests to the interest in this area. Recent books include [57], [61], [69], [110], [117], [127], [163], and [197]. Computer performance evaluation papers regularly appear in computer science journals as well as in more practically oriented publications and there are journals that are devoted exclusively to the topic (e.g., *Performance Evaluation*). In addition there are regularly held conferences devoted exclusively to computer performance evaluation including the highly practical conferences sponsored by the Computer Measurement Group, Inc. and the more research oriented conferences sponsored by ACM SIGMETRICS and by IFIP.

We have divided computer performance evaluation methods into three main areas, namely performance measurement, analytic performance modeling, and simulation performance modeling. We will survey these areas in the remaining three sections of the paper. Performance measurement is possible once a system is built, has been instrumented, and is running. However, modeling is required in order to otherwise predict performance. Performance modeling is widely used not only during design and development, but also for configuration and capacity planning purposes. Performance models span the range from simple analytically tractable queueing models to very detailed trace driven simulation models. One of the principle benefits of performance modeling, in addition to the quantitative predictions obtained, is the insight into the structure and behavior of a system that is obtained by developing a model. This can be particularly valuable during system design and can result in the early discovery and correction of design flaws. Finally, it is common that performance measurement and both analytic and simulation performance models are used during the life cycle of a system. As more information about the design of a system becomes available, more detailed models can be developed. Once the system can be measured, previously developed models can be validated and modified if necessary. The models can then be used with greater confidence to investigate the performance effects of design enhancements and configuration changes.

Reprinted from *IEEE Trans. Computers*, Vol. C-33, No. 12, Dec. 1984, pp. 1195–1220. Copyright © 1984 by The Institute of Electrical and Electronics Engineers, Inc. All rights reserved.

II. Performance Measurement

A. Introduction

The measurement of system performance is of great practical importance to computer installation managers and to computer manufacturers. For example, in order to effectively manage a computer installation performance measurement is required to do the following.

1) Identify current performance problems and correct them, e.g., by tuning or workload balancing.

2) To identify potential future performance problems and prevent them, e.g., by upgrading system resources in a timely manner.

These measurement activities are typically carried out in an uncontrolled live user environment. Computer manufacturers typically measure performance in a controlled environment using benchmarks which may be real workloads or synthetic executable workloads. For interactive systems remote terminal emulators are commonly used to provide a reasonable approximation of an interactive environment. The purposes of a computer manufacturer's measurement activities include assessing the performance of a new system as soon as a prototype is running, providing performance data for competitive bidding, and helping customers configure their systems to meet performance objectives.

Performance measurement is also a fundamental part of research activities conducted at universities and elsewhere in which new system designs are not simply proposed or studied on paper but are implemented, tested, and studied empirically. An early example of a system that was heavily instrumented for performance measurement as part of research activities is the Multics system, an advanced (for its time) time sharing operating system developed at the Massachusetts Institute of Technology in the 1960's [157]. Other early examples are the C.mmp multiprocessor system developed at Carnegie-Mellon University in the 1970's [65], [100], and the PRIME multiprocessor system developed at Berkeley in the 1970's [56]. More recent examples are the Cm* multiprocessor system developed at Carnegie-Mellon University starting in the 1970's [67], [173], and the Erlangen General Purpose Array currently being developed at the University of Erlangen-Nurnberg [64]. The increasing support for experimental computer science at universities will increase the research use of performance measurement.

Another use of performance measurement is to obtain input parameter values for and to validate performance models. A discussion of such performance measurement in the context of analytical queueing network models can be found in [156] and in the context of a trace driven simulation model of C.mmp in [134].

The advantage of performance measurement over performance modeling is, of course, that the performance of the real system is obtained rather than the performance of a model of the system. Interactions may be present in a system that affect performance and are difficult to capture in a model. If they can be captured, say in a very detailed simulation model, the model may take extremely long to program and run. An example illustrating this is a recent paper on cache performance [42] where measured cache hit ratios differed considerably from those in comparable trace driven simulation studies due to such effects as operating system references, task switching, and instruction prefetching which are not normally represented in simulation studies of cache performance. Among the disadvantages of performance measurement are the need for a running system, not just a design, the measurement instrumentation required, the need for a dedicated system if controlled measurements are to be taken, the time consumed to set up and make a measurement run, and the difficulty of modifying the system so that the effect of system changes can be studied.

A method that has recently been used to evaluate performance that lies somewhere between system measurement and detailed simulation modeling is virtual machine emulation. A virtual machine is an execution environment that is functionally the same as a target system other than the actual physical system on which the environment runs. One use of virtual machines is to do functional prototyping. Although the functional properties of the target system are maintained, real-time properties and hence performance are not. IBM's VM/370 control program supports multiple virtual machines on a single physical system. Canon *et al.* [31] describe an enhancement to VM/370 that adds timing simulation via a virtual clock in order to closely approximate the real-time properties of a target system. The user specifies the processor and I/O device timing characteristics of the target system. Executable workloads can be run on the emulated system and performance measured. Virtual machine emulation does not exactly reproduce the performance of the target machine since the timing characteristics of the target machine's devices are only approximated. For more detail and validation results see [31] and for a discussion in the context of emulating distributed systems and networks see [202]. Emulation capabilities are not widely available and this approach, while interesting, does not appear to be widely used.

Numerous measurement studies have been reported on in the literature although their number is dwarfed by the number of analytic or simulation modeling studies. For example, Schwetman and Browne [172] reported on experiments on a large multiprogrammed computer system at the University of Texas. The experiments were conducted in a controlled environment using an artificial batch workload and had the purpose of studying the variations in performance produced by changes in resource availability and scheduling. Recent examples include measurement studies of the speedup achieved when running algorithms on a multiprocessor [67], performance enhancements to a relational database system [188], cache performance of a minicomputer [42], the performance of a new virtual memory management technique [146], the computational speed of supercomputers [26], the degree of parallelism achieved by a processor array [64], and the paging characteristics of a virtual memory system for an object oriented personal computer [15]. In addition to measurements aimed at evaluating some aspect of system performance, the measurement of program performance has received considerable attention. The importance of measuring program performance was demonstrated in [108]. It is

particularly important for programs written in very high level languages where there is no simple mapping between source code and machine operations so that performance is difficult to predict. This point is discussed and illustrated in detail in [46]. Our concern, however, will be aspects of system performance rather than program performance.

The focus of the remainder of this section is not on the applications of performance measurement but rather on the methods and tools that are used in performance measurement. The measurement of new systems often requires new tools and techniques as we will see. The topics we will cover are measurement instrumentation, workload characterization, and statistical aspects of performance measurement including design of experiments and analysis of results.

B. Instrumentation

We will review some of the principles of measurement instrumentation, present an early important example, and then several recent examples that illustrate trends and challenges. A thorough discussion of measurement instrumentation principles can be found in [57, chapter 2] and in [61, chapter 5].

The basic means of instrumenting a system for performance measurement purposes are hardware probes and software (or microcode) probes. Hardware probes are high-impedance electrical probes that are connected to the hardware device being measured. They can be used to sense the state of hardware components of the system, e.g., registers, memory locations, and data transfer paths. The term hardware monitor refers to a measurement device that uses hardware probes. A hardware monitor is typically external to the measured system and does not interfere with the measured system or alter its performance. The sensed signals can be combined in order to sense more complex states than those measured directly. The resulting signals together with the output of a real-time clock can be processed to detect events (state changes) of interest which can then be counted or recorded as a trace (time stamped sequence of events). In addition, times between events can be obtained by counting clock pulses between events. Hardware monitors are commonly implemented using both high-speed hardwired logic and slower speed stored program logic. A mini- or microcomputer may interface with the monitor to set up and control a measurement session and reduce, analyze, and display the collected data. Hardware monitors usually do not have access to software related information such as which process caused an event, although there are exceptions as discussed later in the examples. This is a disadvantage of hardware monitors along with their cost and often their lack of ease of use.

Software probes are instructions added to the measured system (i.e., to the operating system or to application programs) to gather performance data. They may gather data by reading memory locations or otherwise sensing status. A measurement device that uses software probes is called a software monitor. (Microcode probes may also be used.) Since the monitor's instructions run on the measured system and hence use system resources they alter, possibly significantly, the performance of the system. In some cases this effect is straightforward to compensate for, e.g., by subtracting out the CPU utilization and other resource usage due to the monitor, but in other cases it may not be. Thus, software monitors produce performance estimates that may differ from the true performance of the system running without the monitor.

A software monitor can be either event driven or timer driven or both. In event driven monitoring the probes detect events, e.g., instruction executions, storage accesses, I/O interrupts, and then collect data. In timer driven monitoring data are collected (sampled) at specified time instants. This sampling is typically accomplished by generating interrupts based on a hardware clock or interval timer and then passing control to a data collection routine. Sampling typically requires less code and that code is executed less frequently than event driven monitoring. Hence, it interferes less with the measured system. However, sampling introduces additional errors in the performance estimates since status is only sampled periodically. These sampling errors can be decreased by increasing the sampling frequency, and hence the interference. Other errors can occur using sampling if care is not taken. For example, if certain system routines are not interruptable then their contribution to CPU utilization will not be measurable by sampling as described above. Hardware monitors need not produce true performance values either, e.g., due to the resolution of the real-time clock (see [61, chapter 5]). However, they are typically much more accurate than software monitors. It is therefore important that the accuracy of a software monitor be tested, perhaps by comparing its measurements to those of a hardware monitor. One such study of accuracy and methods for correcting the software measurements can be found in [19].

It is possible to combine the advantages of hardware monitors (speed and accuracy) with those of software monitors (flexibility and easy access to software related data) by judiciously combining hardware and software probes in a so-called hybrid monitor, examples of which will be given below. The software causes of hardware events can be readily measured with a hybrid monitor.

An early important example of measurement instrumentation was that done for the Multics system at the Massachusetts Institute of Technology [157]. Multics was an innovative multiprogrammed time sharing operating system that supported multiprocessing, demand paging, and sharing, among other features. Measurement instrumentation was integrated into the system at the early design phase and software probes were an integral part of the operating system. Measurement was directed towards a detailed understanding of operating system performance and was very successful in revealing unsuspected performance problems. The hardware (GE 645) on which Multics ran provided features that were exploited by the measurement facilities. These hardware features were a program readable clock and program loadable clock comparison register for generating timer interrupts, a memory cycle counter for each processor, and an externally drivable I/O channel via which a separate computer could externally monitor memory contents. The many software monitor facilities that were implemented are discussed in detail in [157].

Included were timer and counter facilities for selectable operating system modules and limited tracing facilities. The performance interference caused by these facilities was found to be acceptably small. A remote terminal emulator was implemented to simulate interactive users but due to physical limitations the number of simulated users was small. Therefore, an internal driver was also implemented to generate heavier loads. Real-time graphical displays of performance were generated using the separate computer that served as an external monitor.

We next briefly discuss recent examples of monitors that illustrate trends in this area. Each monitor is described in detail in the references that are given. DIAMOND [90] is a hybrid monitor developed by DEC for internal use. Hardware probes sense the program counter, the CPU mode, channel and I/O device activity, and a system assigned task id which is contained in a special register. A software probe senses the user's id which is also contained in a special purpose register. All sensed signals are buffered in a digital interface and then analyzed by a microcoded machine to obtain traces and histograms of various kinds. A separate minicomputer controls the measurement. Emphasis was placed on ease of use. There is a natural language interface with interactive dialogs for new users. The interface facilitates the set up of a measurement session, replication of experiments, maintenance of a measurement log, and report preparation.

XRAY [14] is a low overhead event and timer driven software monitor developed by TANDEM for networks of TANDEM/16 computer systems. It is integrated into the GUARDIAN network operating system. Networkwide measurements are controlled from a single node with data collected and analyzed locally at each node. The emphasis is on measuring hardware utilizations and access rates, both in total and the contributions by specified processes. Counters are employed that are incremented using microcoded instructions in order to keep the overhead low. The counters are periodically written to files. The primary use of the monitor is for bottleneck detection. A language is provided for the selection of hardware components and processes to be measured and this selection can be changed online while data are being collected. The language also facilitates browsing through the data and real-time displays are provided. There is no time synchronization between nodes so that typically no attempt is made to correlate data from different nodes.

A hybrid monitor was developed by Olivetti for its S6000, a single-bus architecture minicomputer, and similar machines [60]. Data are captured by hardware probing of the bus's address and data lines. Additional general hardware probes were provided as well as a clock pulse signal from the measured system. Low overhead software probes were inserted in the operating system. Event traces, event counts, and times between events were obtained from the measured signals using hardwired logic. However, this processing is controlled via the contents of registers that are loadable by the user via a controlling minicomputer. This separate computer both controls the measurements and analyzes the results.

The Erlangen General Purpose Array is a tightly coupled hierarchically organized multiprocessor being developed at the University of Erlangen-Nurnberg. It is an extensible array of elementary "pyramids," each pyramid consisting of a top control processor and four bottom working processors. The control processor is multiprogrammed and the working processors can execute in parallel on behalf of one program at a time. Both hardware and software monitors were implemented to study in detail the dynamic behavior of the parallelism achieved by the system as well as more traditional performance measures [64], [86]. The hardware monitor, Zahlmonitor III, records event traces as well as event counts and elapsed times. Software events can be measured by probing a register that contains system assigned process numbers, as well as by other hardware means. The trace for a single processor consists of a time ordered sequence of the pairs (process executing, execution time). The asynchronous traces from the different processors are merged by the hardware into a single well-ordered trace. (So far measurements have been reported for only a single pyramid.) Such detailed trace information has been used, for example, to compare different methods of process synchronization [64]. Traces of I/O activity are also obtained and can be related to the processor traces. In addition to the hardwired logic used to implement the above functions a minicomputer is used to control the measurements, analyze the traces, and provide graphical displays. A software monitor for tracing software events on each processor was also implemented and graphical methods were developed for dynamically displaying parallel activities.

A general trend is illustrated by these examples. In terms of applications the measurement of multiple processor systems, ranging from tightly coupled to geographically distributed systems, is of increasing interest. In order to understand the complex interactions that occur in such systems when they run parallel or distributed programs and the effect of these interactions on performance, special instrumentation will be required. Flexible and easy to use monitors, either hardware monitors that can access software related data, hybrid monitors, or low overhead software monitors, are fundamental tools that will aid in gaining this understanding. Hardware and hybrid monitors are being made flexible and easy to use by the use of a controlling computer that provides a user friendly interface including real-time graphics capabilities for displaying measurement results. A further discussion of using a separate computer for controlling distributed system experiments can be found in [63], which includes a description of a distributed system experimental testbed developed at Honeywell.

C. Workload Characterization

The performance of a system obviously depends heavily on the demand for hardware and (application and system) software resources of the workload being processed. Therefore, the quantitative characterization of the resource demands of workloads is an important part of computer performance evaluation studies. This is true for both measurement and

modeling studies. Workload characterization provides a basis for constructing representative synthetic executable workloads to drive a system being measured or for obtaining representative parameter values for analytic or simulation performance models. A comprehensive discussion of workload characterization can be found in [61, chapter 2]. We will focus on methods used to characterize workloads for performance measurement studies. However, much of what we will say is also applicable to workload characterization for analytic or simulation models.

We can distinguish three types of workloads suitable for performance measurement studies, namely, live workloads, executable workloads consisting of portions of real workloads, and synthetic executable workloads. By live workloads we mean real workloads generated and executed in a live user environment. Live workloads are not suitable for controlled and reproducible measurement experiments and will not be considered here. The other two types of workloads are not executed in a live user environment. They are generated and submitted for execution using a program called a driver that simulates a live user environment. A driver that runs external to the system being measured and simulates interactive users is called a remote terminal emulator. Remote terminal emulators have been used for many years in performance measurement studies. A portion of a real interactive workload that is suitable for driving an interactive system using a remote terminal emulator can be obtained by tracing the sequences of think times and commands generated by each user logged on to a system as the system executes. However, traces can be expensive to obtain and store and are not flexible in terms of the workload characteristics they represent. Synthetic executable workloads have the advantage that they can be made parametric and hence flexible in representing workload characteristics. For example, parameters can control the amount of computation a program does and the number of files and records it reads or writes. They have the disadvantage of possible lack of realism, i.e., they may not adequately represent features of real workloads that can significantly affect system performance.

We next summarize the steps that comprise a common approach to characterizing an existing real workload. Examples of workload characterization studies in which several, if not all, of these steps are applied can be found in [1], [3], [18], [61, chapter 2], [75], [76], [130], [174], and [184]. If a system has a mixed workload, e.g., batch, interactive, and database, then this approach can be applied separately to each such distinct part of the workload. The five steps that comprise this approach are as follows.

1) Selection of the workload component to be characterized. For example, when characterizing a batch workload the component may be a job or a job step, when characterizing an interactive workload it may be a session, a sequence of commands or a single command, and when characterizing a database workload it may be a transaction.

2) Selection of the features (parameters) used to characterize a component. The features may be hardware resource demands [1], [3], e.g., processor instructions or time, memory space used, number of I/O's to various devices, or software resource demands [75], [76], e.g., number of calls to compilers, editors, file handlers. An advantage of using hardware resource features is that hardware resource demands directly impact system performance. An advantage of using software resource features is that the resulting characterization may be more system independent.

3) Workload measurement. The real workload is measured while executing to obtain the feature values for each measured workload component. Typically, data are obtained for a large number of components. For example, measurements collected over a period of a month or more yielded data on over 10 000 job steps in the studies reported on in [3], [174]. The result is a large collection of multivariate data.

4) Exploratory data analysis. In this step empirical distributions and sample moments of each of the features may be obtained. In order to obtain comparable ranges for feature values, features with highly skewed distributions may be transformed, e.g., by taking the log of the values. Components having feature values that are outliers may be deleted and different features may be scaled to lie in a common interval. The deletion of outliers requires great care since outliers may have very large resource demands and hence strongly influence system performance. Transformations and/or scaling are often applied to the data if the next step is performed.

5) Cluster analysis. The measured components (perhaps after transformation and/or scaling), or a random sample of the measured components, are partitioned into clusters such that components in the same cluster have similar feature values. The purpose is to treat all components in a cluster as being effectively identical so that a compact workload characterization can be obtained. For example, in studying workloads from over 100 systems Artis [3] found that in each case many thousands of job steps could be partitioned into 15–20 clusters based on eight hardware resource oriented features. Cluster analysis was originally developed in the context of biological taxonomy and has been applied in a wide variety of disciplines. A large number of clustering algorithms exist (e.g., [78], [143]). The algorithms most commonly applied to workload characterization are variants of the K-means (also called nearest centroid) algorithm. The basic algorithm and some of its variants are presented in [78, chapter 4]. The basic algorithm partitions the components into a specified number of clusters in order to locally minimize the partition error, defined to be the sum over all components of the Euclidean distance between a component and the centroid (center of mass) of the cluster to which it has been assigned. An initial partition is chosen and components are moved one at time between clusters in order to reduce the partition error until a local minimum is achieved. Criteria for choosing the number of clusters are presented in [78]. Variants applied to workload characterization can be found in [1], [3], [174]. Issues in applying cluster analysis to workload characterization are discussed in [2], [61, chapter 2].

It is possible to construct synthetic executable workloads based on the above type of workload characterization. The components of the synthetic workload can be parameterized

and the parameter values chosen to match the feature values of a cluster, specified for example by the cluster's centroid. An interesting example of this is given in [3] where the clusters also provide a basis for workload forecasting. Reports on the construction and use of parametric workloads can also be found in [172], [184]. (Cluster analysis was not used in these studies and representative feature values were obtained by sampling.) Unfortunately, most papers on workload characterization using cluster analysis do not report on the actual construction and use of parametric workloads and it is not clear how widely this is done.

Most workload characterization studies have used hardware oriented features. In [75] and [76], software resource demands were used instead. The workload component was an interactive job (sequence of tasks) and seven software resource demands were used to characterize each component. After clustering was performed the sequence of software resources used by each job in a cluster was modeled by an absorbing Markov chain. The purpose was to obtain a more realistic workload model than one which assumes that successive resource demands are statistically independent. The data indicated that the Markov chain had to be either nonhomogeneous or higher than first order (i.e., the current resource demand depends on the past several resource demands) in order to provide a statistically adequate model. (Details of the statistical tests used are given in [76].)

The type of workload characterization we have described also provides a basis for determining representative parameter values for analytic and simulation performance models. For example, cluster analysis can be used to determine the job classes to be used in a product form queueing network model (see Section III) and to assign representative values to the mean service demands in the model. (Only the means of the service demands affect the performance of such models.) The number of jobs in each class in the model can be chosen to be proportional to the number of jobs in each cluster. If service demand distributions were needed, e.g., for nonproduct form models or simulation models, they could be obtained from the empirical distributions of the feature values within each cluster.

One of the main challenges in workload characterization is to develop synthetic executable workloads that yield approximately the same system performance as the real workloads they represent. Despite considerable activity in workload characterization there is little published evidence of success in this regard. An approach in which workload components are directly characterized by the system performance they yield rather then by their resource demands is described in [58], [61, chapter 2]. Although this approach may prove successful in meeting the above challenge it yields a highly system dependent workload characterization. Issues related to the adequacy of resource demand oriented workload characterizations are discussed in [59] where queueing network models are used to show that simple characterizations can yield the same model performance as more complex ones. Most workload characterization has involved batch and to a lesser extent interactive workloads. Therefore, a key area that needs to be addressed is characterization of database workloads. Another challenge is to develop synthetic executable workloads to drive new systems when relevant real workloads are nonexistent. This is the case for multiprocessor systems that support parallel processing and for distributed systems. A facility to generate synthetic executable workloads for the Cm* multiprocessor system is described in [178]. A high-level language is provided for representing a synthetic parallel program as a type of data flow graph. The processing activities specified by the nodes of the graph are realized using a library of system specific routines. There is clearly a continuing need for synthetic workload generation in the performance evaluation of new systems.

D. Statistical Methods

It is important that sound experimental methods be employed in performance measurement studies. The purpose of a study should be clearly formulated, the measurement experiments should be carefully designed, and the resulting data should be carefully analyzed so that meaningful conclusions can be made. This is true whether measurement is done in a live user environment or in a controlled environment. While common sense can play an important role in this regard, so can statistical methods. Statistical methods have been discussed in some detail in performance evaluation books, e.g., [57, chapter 2], [110, chapter 5], and articles written for the performance evaluation community, e.g., [9], [166]. Application examples are discussed in these books and articles. We will first discuss the random nature of performance measurement data and will advocate that confidence intervals be obtained in order to account for this randomness when producing performance estimates. We will then discuss regression analysis and statistical design of experiments and representative applications of these methods. We will conclude with a discussion of why these two methods have rarely been used in performance measurement studies.

1) The Random Nature of Measurement Data: Random fluctuations are often present in performance measurement data. This is the case when synthetic executable workloads are at least in part probabilistically generated. For example, successive think times and successive command types may be probabilistically chosen when generating a synthetic interactive workload. Even without this obvious source of randomness, repetitions of a measurement session can yield nonidentical data due to factors that are uncontrollable or too difficult to control from session to session. Such data can also be considered to fluctuate randomly. This is the case when measurement is conducted in a live user environment due to the uncontrolled nature of the workload. Therefore, a measured data sequence, e.g., a sequence of measured response times, should be viewed as a random sequence. Any performance estimate produced from such a sequence, e.g., the sample average, should be viewed as a random variable. The same situation arises in discrete event simulation when a probabilistic model is simulated. Many methods have been developed for statistically analyzing simulation outputs, e.g., see Section IV-B of this paper and [117, chapter 6]. Typically, the methods are used to obtain confidence in-

tervals in addition to point estimates. (The definition of a confidence interval is given in Section IV-B.) Confidence intervals should also be obtained when dealing with system measurements, particularly when synthetic probabilistic workloads are used. In practice they very rarely are.

An application to a performance measurement study of a broadly applicable method for obtaining confidence intervals that was originally developed for simulation output analysis can be found in [85]. The method was applied to estimating steady state characteristics of transaction response time sequences in a database system. Measurement was done in a controlled environment using synthetic workloads with probabilistically selected transaction types and transaction arrival times. Confidence intervals for the mean response time and for quantiles of the response time distribution were obtained at several transaction rates for two system variants and used to compare the two variants. The method used, called the spectral method (see Section IV-B), obtains a confidence interval for a steady state parameter from a single output sequence, i.e., repeated measurement sessions are not necessary. The method is broadly applicable since it makes only weak probabilistic assumptions about an output sequence. For example, it does not assume members of the sequence are independent or normally distributed. An alternative broadly applicable method of obtaining confidence intervals is independent replications which requires statistically independent and identical repetitions of a measurement session. While this is feasible when synthetic probabilistically generated workloads are used, it may not be possible in a live user environment.

2) Regression Analysis: Regression analysis can be used to approximate the functional dependence of one or more variables (called dependent variables) on another collection of variables (called independent variables). The approximate functional relationship can then be used to predict values of the dependent variables from values of the independent variables. Each dependent variable is expressed as a postulated function of the independent variables and a collection of parameters plus a random error term. While the form of the function is assumed known, the parameter values are not. Usually, a linear relationship is postulated (linear regression), i.e.,

$$y = a_0 + a_1 x_1 + \cdots + a_k x_k + \varepsilon \qquad (2.1)$$

where y is a dependent variable, x_j's are the independent variables, a_j's are the parameters, and ε is the random error. The parameter values are estimated from n observed values of all the variables, i.e., from $y_i, x_{1i}, \cdots, x_{ki}, i = 1, \cdots, n$, and an informal measure of goodness of fit is obtained. If the errors for different observations are assumed to be independent and normally distributed with zero mean and identical variances then confidence intervals for the parameters can be obtained and formal statistical tests of goodness of fit can be applied. Details and further discussion can be found in the concise presentations in [9], [110, chapter 5], and in standard texts on regression analysis, e.g., [50]. In an example due to Bard [6], which is discussed in [9], the CPU time consumed by an operating system (the dependent variable) is linearly related to the number of calls to certain operating system services (the independent variables). The parameters have a physical interpretation, i.e., they are the CPU times per call for each of the services represented. These overhead parameters could not readily be measured but the number of operating system calls of each type and the total operating system CPU time could be measured. Estimates for the overheads were obtained using linear regression. A subsequent study [10] revealed inadequacies in this linear model which were corrected to some degree. This illustrates that the application of statistical methods may not be straightforward.

3) Design of Experiments: Statistical design of experiments is used to design experiments whose purpose is to estimate the effects of multiple controllable factors on measured responses. Separate sets of measurements are made with the factors set at different specified levels. A key aspect of the designs is that the factors are varied simultaneously rather than one at a time in order to facilitate estimating the effects of interactions between the factors. Typically, a linear model with additive random error is used to approximate the relationship between a measured response and the effects of the factors. For example, for a two-factor experiment where factor 1 has I levels and factor 2 has J levels, the linear model would be

$$y_{ij} = m + a_i + b_j + c_{ij} + \varepsilon_{ij} \qquad (2.2)$$

where m is the overall mean, and for factor 1 at level i and factor 2 at level j, y_{ij} is the measured response, a_i is the main effect of factor 1, b_j is the main effect of factor 2, c_{ij} is the interaction effect of factors 1 and 2, and ε_{ij} is the error term. The errors for measurements at different levels are assumed to be independent random variables with zero mean and identical but unknown variances. If measurements are obtained for all combinations of the factor levels the experiment is called a full factorial experiment. The measured responses at the different combinations of factor levels (including, if possible, replicated measurements at each combination) are used to estimate the overall mean and the main and interaction effects in the linear model. A technique called analysis of variance is used to determine the significance of the effects. If the errors are assumed to be normally distributed then formal statistical tests can be applied. If the number of factors and levels is so large that a full factorial experiment is too costly then a fractional factorial experiment can be conducted. In such an experiment measurements are obtained for only certain combinations of factor levels. Although all interaction effects cannot be estimated with fractional experiments, all main effects and some interactive effects can be. More detail on design of experiments can be found in [110, chapter 5], [166], [197, chapter 11], and in standard texts on design of experiments, e.g., [20], [43].

An often cited application of design of experiments to performance measurement studies can be found in [198], [199]. (The first paper reports on the application and conclusions while the second describes the statistical methods used.) The effects of four factors (e.g., paging algorithm,

main memory size) on various measures of paging performance were estimated by varying each factor at three levels using a full factorial design resulting in 81 different combinations of factor levels. For one performance measure the conclusion was that three main effects and one interaction effect predominated. Bard and Schatzoff [9] discussed this example and showed that the same conclusions could have been obtained with a fractional factorial design using only 16 combinations instead of 81.

The above experiments were performed in a controlled environment. An application in a live environment due to Margolin *et al.* [131] is discussed in [9]. The purpose was to compare the effects on performance of two different free storage management algorithms. Measurements were taken on eight days in two consecutive weeks (Mondays–Thursdays only). Rather than run algorithm *A* the first week and algorithm *B* the second week the algorithms were assigned to days so that each algorithm was run twice each week and once on each of the corresponding days of the weeks. The purpose was to eliminate as much as possible the otherwise uncontrollable effects of day of the week and week-to-week workload variations. Clearly, this is good common sense and formal methods are not necessary to arrive at this design. Bard and Schatzoff [9] give a formal description of the design as well as the analysis of variance results. It turned out that the effects of day and week were so small and of algorithm so large that the careful design was not required. However, in other cases it might be.

An interesting application of design of experiments to simulation model validation is given in [167]. Identical multifactor experiments were carried out for performance measurements of a system and for a trace driven simulation model of the system. The criterion for model validity was that the same significant effects be identified from both experiments.

4) Conclusion: The application of regression analysis and statistical design of experiments to performance measurement studies has been rare. This was noted by Grenander and Tsao [74], who tried to motivate their increased use, and it remains true today. While their lack of use may be due to lack of familiarity with these methods by performance analysts, it is also true that these methods are not straightforward to apply in performance measurement studies. This is partly because they are based on assumptions about the data, e.g., independent errors with common means and variances, that may be far from true for performance data. Also, there is typically a large number of variables that can affect performance and their effect may be more complex than can be explained by standard statistical models. Published work on the application of regression analysis and statistical design of experiments in performance measurement studies indicates that successful application of the methods requires the involvement of both experienced applied statisticians and experienced computer performance analysts. It is rare that both skills reside in one person. Nonetheless, by carefully applying these methods more meaningful conclusions can be drawn from performance measurement studies than would otherwise be possible. Therefore, we recommend their increased use in such studies.

III. Analytic Performance Modeling

Computer systems can generally be characterized as consisting of a set of both hardware and software resources and a set of tasks, or jobs, competing for and accessing those resources. Examples of hardware resources include main memory and devices such as CPU's, channels, disks, tape drives, control units, and terminals. An example of a software resource is a lock for a database item. Because there are multiple jobs competing for a limited number of resources, queues for the resources are inevitable and with these queues come delays.

It is therefore natural to represent, or model, the system by a network of interconnected queues. The purpose of the model is to predict the performance of the system by estimating characteristics of the resource utilizations, the queue lengths and the queueing delays. Analytic performance models are queueing network models for which these characteristics may be found mathematically (or analytically). Therefore, research in performance modeling methodology has essentially been research in queueing theory. Key advances in computer performance modeling have also been seen as fundamental breakthroughs in queueing theory. Queueing theory has attained new relevance because of the computer performance modeling application. Furthermore, to a great extent, the direction of queueing theory has been influenced and driven by this application.

Queues are also inevitable in communications systems and a closely related topic is performance evaluation of communications systems. Indeed, the telephone system provided motivation for the earliest work on queueing theory [54]. Communications systems consist of messages accessing hardware resources such as switches, channels, buffers, and computers. Software resources in a communications system result from the system's communications protocols. Such a software resource might be a message passing token in a ring network or a limit on the number of messages on a route imposed by a window flow control scheme. The distinction between computer and communications systems is diminishing. However, the focus of this paper is on computer systems. Analytic models have also had substantial impact in performance evaluation of communications systems (e.g., [106], [107], [170]).

In this section we will give an overview of the role of analytic modeling in computer performance evaluation and highlight the major methodological advances that have taken place over the last decade. These advances are threefold.

1) Identification of a broad class of models, called product form queueing networks, having a mathematically tractable solution.

2) Development of computationally efficient and numerically stable algorithms for product form queueing networks.

3) Development of accurate and computationally efficient algorithms to approximate the solution of large product form queueing networks and queueing networks that do not fall into the product form class.

We will close the section by indicating what we believe are the key challenges which performance modeling and queueing theory must meet in order to maintain relevance in com-

puter science throughout the next decade.

A. The Role of Analytic Models

Analytic performance modeling has become widely accepted as being a cost effective evaluation technique for estimating the performance of computer systems. Analytic models are cost effective because they are based on efficient solutions to mathematical equations. However, in order for these equations to have a tractable solution, certain simplifying assumptions must be made regarding the structure and behavior of the queueing network model. As a result, analytic models cannot capture all of the detail that can be built into simulation models. Nevertheless, for many types of systems the key resources and workload requirements can be analytically modeled with sufficient realism to provide insight into the bottlenecks and key parameters affecting system performance. It is generally thought that carefully constructed analytic models can provide estimates of average job throughputs and device utilizations to within 10 percent accuracy and estimates of average response time to within 30 percent accuracy [127, p. 14]. We cite three areas where this level of accuracy is usually considered sufficent and where analytic models have had substantial impact, namely capacity planning, I/O subsystem performance evaluation, and as a preliminary design aid in development of new systems.

1) Capacity Planning: Analytic models play a key role in capacity planning. Capacity planning is the process of determining future computing system hardware needs based on projections of the growth in the workload and hence in the demand for processing power, memory space, and I/O activity. Capacity planning generally consists of three steps.

1) Measurement and data reduction of the current system.

2) Construction and validation of a queueing model of the current system.

3) Extending the model to incorporate new devices and running the model against projected future workloads.

The system is parameterized by key factors such as the speeds and numbers of the CPU's, disks, and channels and the size of main memory (which affects paging rates). The rate of transactions arriving to the system (or the number of users and the users' think times) and the transaction workload requirements are also parameters of the model. The key parameters are varied until a configuration meeting both the cost and performance objectives of the organization is determined. Because the queueing models can be solved efficiently, a large number of model runs can be made allowing a thorough search of the parameter space. Due to the uncertainties in future workloads, the accuracy of the queueing model is more than sufficient for this purpose.

There are a number of commercial capacity planning packages currently available including BEST/1 (from BGS Systems) for IBM MVS systems [29] and the VM Performance Planning Facility (from IBM) for IBM VM systems (based on a model described in [7]). In addition to a queueing component, these packages both contain interfaces to measurement facilities and data reduction capabilities.

2) I/O Subsystem Modeling: Because of the vast difference between I/O access times and main memory access times, I/O subsystem performance has become a dominant factor affecting overall system performance. This is expected to continue in the future as processors and main memories become faster whereas access times for mechanically activated disks are not expected to decline much further. Memory hierarchies have been constructed to mask this speed difference and I/O subsystems have been constructed to get the best possible performance out of the memory hierarchy.

Analytic models are widely used to predict the performance of I/O subsystems. They have been successfully used to analyze the performance of proposed changes in I/O subsystem architecture. Examples include modeling buffered, or cached, disk units and dynamic path selection both of which attempt to reduce I/O service times. Buffered disks attempt to reduce the probability of a seek while dynamic path selection tries to reduce the probability of the additional rotation that occurs because the transfer path is busy when a disk attempts to reconnect to a channel for data transfer [21], [23]. More detailed discussions of I/O subsystem models, with additional references, may be found in [117], [127].

3) Preliminary Design Aid: Analytic models have been successfully applied to study the performance of proposed future computer architectures or systems. An analytic model can provide insight into the key factors affecting performance of a proposed system, and determine the sensitivity of performance to parameter changes. Such a model can provide guidance into the overall design of the system and also be useful in the development of more detailed simulation models as the design matures. For example, the analytic model could determine where effort should placed in building the simulation model; if performance is not a problem in some subsystem, then that subsystem need not be modeled in great detail. The analytic model could also be used to limit the range of parameters to be used in the more expensive runs of the simulation model.

An example of such a study can be found in [73]. This model studies the overall design considerations and technology tradeoffs for the memory interconnection structure of several hypothetical future multiprocessor systems. One of the systems consists of a few high performance processors, each with its own local memory. There is also a large semiconductor memory that is shared by all processors. Upon a page fault to the local memory, a page must be transferred from the shared memory. Contention occurs for both the modules of the shared memory and for transfer buses between the shared and local memories. The system is parameterized by the numbers of processors, memory modules, and buses, the memory access and bus transfer times, and the page fault rate. The purpose of the model is to determine the bus bandwidth required to support the processors at a reasonable performance level.

B. Product Form Queueing Networks

Queueing theory has a long history beginning with the work of Erlang [54]. It is not our intention to give a history

of queueing theory, but rather to highlight recent methodological advances that have had a significant impact on computer performance evaluation. We start with a simple example.

Fig. 1 represents a simple model of a computer system, called a central server model with terminals. There is a fixed number N of users, each with his own terminal, submitting transactions to the system. Each user is represented by a job in the network. When a transaction is submitted it goes through a number of CPU–I/O cycles; it executes on the CPU, performs I/O on one of the I/O devices, and returns to the CPU, repeating this process until the transaction is completed. Between transactions the user is in the think state. Let the node in Fig. 1 representing the terminals be service center 1, the CPU be service center 2, and the I/O devices be service centers $3, \cdots, M$. In order to completely define the model, the following must be specified.

1) The queueing disciplines at each of the centers.
2) The service requirements of jobs at the centers.
3) The routing of jobs between centers.

When the above are appropriately defined, the evolution of the system can frequently be modeled by a continuous time Markov chain. For example, suppose that the service disciplines at the devices are all first come first served (FCFS), that the think times and the service times at each device are independent random variables with a common exponential distribution, and that when a job leaves center i, it proceeds to center j with probability p_{ij}. Let $Q_i(t)$ denote the queue length at center i at time t. Then $Q = \{Q(t) = (Q_1(t), \cdots, Q_M(t)), t \geq 0\}$ forms a continuous time Markov chain. Let $n = (n_1, \cdots, n_M)$ and define $p(n, t) = \text{Prob}\{Q(t) = n\}$.

Finding $p(n, t)$ requires solving a system of linear differential equations with constant coefficients. However, the limiting, or stationary, distribution $p(n) = \lim_{t \to \infty} p(n, t)$ can be found by solving a system of linear equations, called the global balance equations, which, for each state, equates the rate of flow into the state to the rate of flow out of the state. In this application, we are usually interested in the stationary distribution not only for reasons of mathematical convenience but also because a time average converges to its stationary value. There is one linear equation for each state of the system. In the above example there are

$$\binom{N + M - 1}{M - 1} \tag{3.1}$$

equations since that is the size of the set $\{n : n_1 + \cdots + n_M = N\}$. Thus, the number of states becomes unmanageably large as N and M increase.

For a restricted class of networks called product form networks, including the above example, the solution to the global balance equations can be shown to be a product of terms where the form of each term is explicitly given. More specifically,

$$p(n) = (1/G(N)) \prod_{i=1}^{M} p_i(n_i) \tag{3.2}$$

where $G(N)$ is a normalization constant chosen to make the probabilities sum to one. Performance measures such as the mean response time, mean queue lengths, and device utilizations can be computed once $G(N)$ is known. The importance of the product form solution is not only that it gives the form of the stationary distribution, but also that efficient algorithms exist to compute $G(N)$ and the relevant performance metrics (in order $M(N + 1)$ operations). The existence of computationally efficient algorithms for a broad class of models makes analytic queueing models an attractive tool for applied performance modeling studies of computer systems.

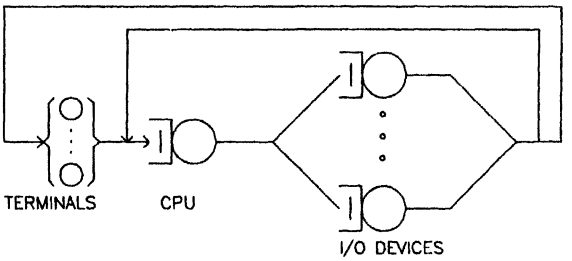

Fig. 1. Queueing diagram of the central server model with terminals.

Jackson [97] was the first to show a product form solution in a general network of queues. He was motivated by job-shop, or manufacturing, applications. Jackson considered an open network, meaning a network in which jobs arrive from an external source, pass through some sequence of service centers, and eventually depart from the system. The class of Jackson networks allows only a single type, or chain, of jobs with a Markovian arrival process dependent on the total population of the network. Service disciplines are FCFS, service demands are exponential with queue length dependent service rates, and routing is Markovian. The routing probabilities appear in the solution only through terms relating the relative number of visits jobs make to the centers, called visit ratios. Gordon and Newell [72] extended Jackson's results to cover closed networks; networks such as the central server model with terminals in which there are neither external arrivals nor departures but rather a fixed number jobs circulating indefinitely.

1) The Convolution Algorithm and BCMP Networks: Buzen [27], [28] was the first to develop a computationally efficient algorithm, the convolution algorithm, for Gordon and Newell's class of closed networks. Let $G_j(n)$ denote the normalization constant for a network with n jobs and centers $1, \cdots, j$. In this notation $G(N) = G_M(N)$. Buzen's algorithm computes $G(N)$ by convolving arrays according to the recursion

$$G_j(n) = \sum_{i=0}^{n} G_{j-1}(i) X_j(n - i) \tag{3.3}$$

where $X_j(i)$ is defined in terms of the relative visit ratio and the service rates of center j and $G_1(i) = X_1(i)$.

Baskett, Chandy, Muntz, and Palacios-Gomez [11] greatly extended the class of product form networks. Their generalization allowed for the following.

1) Multiple types, or chains, of jobs. Jobs in different chains can have different routing probabilities and different service demand distributions (at non-FCFS service centers).

2) Mixed networks, meaning networks with both open and closed chains.

3) New service disciplines including infinite servers (IS), processor sharing (PS, a limiting case of round robin as the quantum size goes to zero), last come first served preemptive resume (LCFSPR, of little importance in practice), in addition to FCFS.

4) General service demand distributions at IS, PS, and LCFSPR service centers.

The service demand distribution at any FCFS center must be exponential and all jobs receiving service at that center must have the same mean service demand regardless of type. These networks have come to be called BCMP networks.

For a network consisting of only closed chains, the product form is as follows. Define n_{ji} to be the number of chain j jobs at center i, $\boldsymbol{n}_i = (n_{1i}, \cdots, n_{Mi})$, $|\boldsymbol{n}_i| = n_{1i} + n_{2i} + \cdots + n_{Mi}$ and let ρ_{ji} be the relative visit ratio of chain j jobs to center i times the mean service demand of chain j jobs at center i. Let $\mu_i(n)$ be the service rate at center i when there are n jobs at center i and let $A_i(n) = \mu_i(1)\mu_i(2)\cdots\mu_i(n)$. The stationary distribution is given by

$$p(\boldsymbol{n}_1, \cdots, \boldsymbol{n}_M) = (1/G(N)) \prod_{i=1}^{M} p_i(\boldsymbol{n}_i) \tag{3.4}$$

where

$$p_i(\boldsymbol{n}_i) = A_i(|\boldsymbol{n}_i|)\,|\boldsymbol{n}_i|\,! \prod_{j=1}^{K} (\rho_{ji}^{n_{ji}}/n_{ji}!) \tag{3.5}$$

where M is the number of sevice centers and K is the number of closed chains. In the above, only those states for which $n_{j1} + \cdots + n_{jM} = N_j$ for each chain j have nonzero probability where $N = (N_1, \cdots, N_K)$, and N_j is the population of closed chain j.

Kelly [102] has a somewhat different, but in many ways equivalent, formulation of queueing networks yielding product form. The BCMP formulation has become the most widely used in practice. Although the class of product form networks has been somewhat extended beyond the BCMP networks, e.g., certain types of state dependent routing in [196], these extensions have not been found to be particularly useful in applications. It is now generally thought that the class of product form networks will not be significantly extended beyond the BCMP networks.

Reiser and Kobayashi [154] developed a generalization of the convolution algorithm for BCMP networks. The computational complexity of this and other algorithms depends essentially on the number of service centers, the number of closed chains, and the populations of the closed chains. This assumes that the extent of any population dependent arrival rates for open chains is limited [117], [160]. For closed networks with queue length independent service rates the convolution algorithm requires on the order of

$$MK \prod_{k=1}^{K} (1 + N_k) \tag{3.6}$$

operations to compute $G(N)$. This is again a considerable savings compared to the computational cost of solving the set of linear balance equations of the underlying continuous time Markov chain; there would be at least

$$\prod_{k=1}^{K} \binom{N_k + M - 1}{M - 1} \tag{3.7}$$

equations.

2) The Mean Value Analysis Algorithm: Reiser and Lavenberg [155] developed a new algorithm, the Mean Value Analysis (MVA) algorithm, for product form networks. This algorithm computes mean performance measures such as utilizations, throughputs, mean queue lengths, and mean response times directly without explicit computation of the normalization constant $G(N)$. The algorithm was first developed for closed networks with fixed rate and queue length dependent rate servers only, but has been extended to cover a broader range of product form networks including state dependent routing and more general forms of state dependent service rates [153], [160]. For networks with either IS or fixed rate service centers, MVA has an intuitively appealing justification, making it easier to teach than the convolution algorithm. MVA also provides a basis for approximations for either large product form networks or nonproduct form networks. It is also easier to program than the convolution algorithm and it avoids certain numerical instabilities present in the convolution algorithm. MVA is based on two simple principles.

1) Little's formula $L = \lambda W$ which is a generally applicable theorem relating the mean queue length L to the throughput λ and the mean waiting time W.

2) The "Arrival Theorem" [120], [175] which states that in a stationary product form network, the state distribution that a job sees upon arrival to a service center is equal to the stationary distribution of the network with that job removed.

In this paper we will describe MVA for a closed product form network with either IS or fixed rate service centers. The key equation of the MVA algorithm relates, according to the Arrival Theorem, the mean response time of a job at a service center to the mean queue length of a network with one job removed. For a network with fixed rate service centers define μ_i to be the rate at service center i, S_{ji} to be the mean service demand of chain j jobs at center i, y_{ji} be the relative visit ratio of chain j jobs at center i, and let $R_{ji}(\boldsymbol{n})$, $L_{ji}(\boldsymbol{n})$, and $\Lambda_{ji}(\boldsymbol{n})$ be the mean response time, mean queue length, and throughput, respectively, of chain j jobs at center i in a network having $\boldsymbol{n} = (n_1, \cdots, n_K)$ jobs. For fixed rate service centers, the key MVA equation is

$$R_{ji}(\boldsymbol{n}) = (S_{ji}/\mu_i)\left(1 + \sum_{k=1}^{K} L_{ki}(\boldsymbol{n} - \boldsymbol{e}_j)\right) \tag{3.8}$$

where $\boldsymbol{e}_j$ is a vector of zeros except for a one in position j. By the Arrival Theorem, $L_{ki}(\boldsymbol{n} - \boldsymbol{e}_j)$ is the mean chain k queue length when a chain j job in a network with population $\boldsymbol{n}$ arrives at center i. For FCFS service centers (3.8) says that a job's response time consists of waiting for those jobs in front of it on arrival plus its own service time. For IS service centers

$$R_{ji}(\boldsymbol{n}) = S_{ji}/\mu_i. \tag{3.9}$$

An application of Little's formula to the mean cycle time (time between arrivals of the same job) of center i yields

$$\Lambda_{ji}(\boldsymbol{n}) = \frac{n_j}{\sum_{m=1}^{M} (y_{jm}/y_{ji}) R_{jm}(\boldsymbol{n})} \tag{3.10}$$

and an application of Little's formula to the mean queue length at center i yields

$$L_{ji}(\boldsymbol{n}) = \Lambda_{ji}(\boldsymbol{n}) R_{ji}(\boldsymbol{n}). \tag{3.11}$$

Equations (3.8)–(3.11) define a recursion allowing one to proceed from populations $\boldsymbol{n} - \boldsymbol{e}_j$ for $j = 1, \cdots, K$ to population $\boldsymbol{n}$.

For the above networks, the computational complexity for MVA is comparable to that of convolution for both single and multiple chain networks. For these networks MVA requires on the order of $MN_1N_2 \cdots N_K$ storage locations as opposed to $2N_1N_2 \cdots N_K$ storage locations for the convolution algorithm. For networks with queue length dependent rate service centers, the computational and storage requirements of both algorithms are greater than those listed above; see [117], [160], [204] for more complete discussions of the computational complexity of these two major algorithms for product form networks. Lavenberg [117], Sauer [160], Reiser [153], and Chandy and Sauer [39] describe several other algorithms for solving product form networks.

C. Algorithms for Large Product Form Networks

The computational complexity given in (3.6) and the storage requirements for solving product form networks become prohibitive for multiple chain networks having a large number of closed chains or a few closed chains with large populations. Such networks are becoming increasingly important with the advent of distributed processing. For example, Goldberg *et al.* [70] consider a model of the LOCUS local area distributed operating system. Their model contains sites connected by a communications network. Each site is essentially a central server model with terminals and the users logged on at each site are modeled by a separate closed chain. In today's environment there could easily be 50 sites with, say, 5 users per site. The operations count to exactly solve such a product form network would be larger than 6^{50}. A number of algorithms has been developed to solve, either exactly or approximately, networks of this size.

1) The Tree Convolution Algorithm: Lam and Lien [113] developed the Tree Convolution algorithm to exactly solve networks in which each chain visits only a few centers (sparseness) that are clustered in certain parts of the network (locality). Such networks are common in models of communications systems. Because of the sparseness and locality properties, many states have probability zero. The Tree Convolution algorithm tries to avoid summing over states with probability zero. The algorithm builds trees of centers representing the order in which subnetworks are convolved together. The trees are carefully chosen in an attempt to minimize a cost function capturing space–time complexity. Spectacular savings can occur. Lam and Lien considered a model of window flow control in a communications system. The model had 64 channels (service centers), 32 virtual routes (chains), and a widow size of 3 for each route ($N_k = 3$). They report a decrease from 10^{22} operations for both Convolution and MVA to 10^6 operations for the Tree Convolution algorithm. Similar savings in storage were obtained. A tree version of MVA has also been developed [200].

2) Bounds for Product Form Networks: A number of algorithms that produce upper and lower bounds on the performance measures in product form networks has been developed. The bounds are produced with much less computational effort than would be required to solve the model exactly. A discussion of simple bounds for queueing networks may be found in [127]. We next discuss two recently developed methods that allow a tradeoff between the computational effort and the tightness of the bounds. The limiting case (in a sense to be made precise below) of both of these methods yields an exact solution.

The first method, based on a multiple integral representation and asymptotic expansion of $G(N)$, has been developed by McKenna *et al.* [137], McKenna and Mitra [135], Ramakrishnan and Mitra [150], and McKenna and Mitra [136]. Currently, the method applies to mixed networks with at least one IS center visited by each closed chain, single-server fixed rate service centers, and an assumption of "normal usage" which states roughly that the utilizations are not too close to one. The requirement for an IS center will frequently be met in practice since a set of users at terminals is modeled by an IS center.

The integral representation of $G(N)$ comes about by using the integral representation for the factorial terms in (3.5)

$$n! = \int_0^\infty e^{-x} x^n \, dx, \tag{3.12}$$

and by applying the multinomial theorem to simplify terms in the sum over all possible states. The multiple integral is then expressed in terms of a "large parameter" N, i.e., $G(N) = I(N)$. It is suggested to choose N to be the maximum over i and j of the ratio of the sum over k of the ρ_{jk}'s for IS centers visited by chain j to ρ_{ji} for non-IS centers visited by chain j. An asymptotic expansion for $I(N)$ is developed

$$I(N) \sim \sum_{n=1}^{\infty} A_n / N^n. \tag{3.13}$$

The coefficients A_n turn out to be related to the normalization constants of certain product form networks, called pseudo-networks, with small populations. $I(N)$ is estimated by taking the first m terms in (3.13). Bounds on the difference between $I(N)$ and its m term estimate are also obtained. These bounds become tighter as m increases and converge to the exact result as $m \to \infty$.

Ramakrishnan and Mitra [150] report that, in practice, usually less than four terms are required in the expansion to obtain satisfactory results. They report solving some very large networks, including a one-term expansion of a network with 23 service centers, 17 chains, and 1000 jobs in each chain. Different expansions are required for "heavy usage"; these have been worked out completely for some particular

networks [137] and are anticipated for general networks.

The second bounding technique is the Performance Bound Hierarchy (PBH) developed in [51], [52]. We first consider a network with a single chain and single server fixed rated centers. The level i bounds are produced by obtaining optimistic and pessimistic bounds on the performance measures of a network with $N - i$ jobs, and then applying the MVA equations (or a slight modification of them for the optimistic bounds) until population N is reached. The initial bounds at population $N - i$ are trivially obtained from simple asymptotic bounds (e.g., [127]). Letting $\lambda^i_{opt}(N)$ and $\lambda^i_{pess}(N)$ denote the optimistic and pessimistic level i bounds on the throughput at some center in a network with population N, then the PBH produces a nested hierarchy of bounds in the sense that

$$\lambda^{i-1}_{pess}(N) \leq \lambda^i_{pess}(N) \leq \lambda(N) \leq \lambda^i_{opt}(N) \leq \lambda^{i-1}_{opt}(N) \qquad (3.14)$$

where $\lambda(N)$ is the exact throughput. Nested bounds for mean queue lengths and mean response times are also obtained. The level N bounds correspond to exact application of MVA. For single-chain networks, the bounds are of theoretical interest only since it is not very costly to solve such networks exactly.

The methodology has been extended to multiple chain networks in [51], although the procedure is significantly more complicated. A nested hierarchy of bounds is also obtained in this case. The computational complexity to obtain the level i bounds is order

$$MK\binom{K+i}{i}. \qquad (3.15)$$

Algorithms, called looping algorithms, to improve the initial bounds are also described; it was found empirically that the level i looping bounds are usually about as tight as the ordinary level $i+2$ bounds.

In practice, the bounds produced by both methods tend to loosen as congestion in the network increases. A limited comparison between the PBH bounds and the integral representation bounds showed that, for comparable computational effort, neither method dominated the other [51].

3) Approximation Algorithms for Large Product Form Networks: Approximation methods provide a still cheaper alternative to solving product form networks consisting of single-server fixed rate and IS centers. The decreased cost is offset by the fact that bounds on the errors introduced by the approximate solution method are usually not available. The analyst must rely on experience gained in validation studies to judge whether or not the approximation method is reliable. For a given network, it is impossible to judge the quality of the approximation other than to compare it to a result obtained by a more costly method (exact solution, bounding technique, or simulation).

There have been a number of methods suggested for approximately solving large product form networks. These methods typically rely on MVA for motivation. In a multichain product form network the key MVA equation for single-server fixed rate centers given in (3.8) implies a recursion over all possible populations $\boldsymbol{n}$ for which $\boldsymbol{0} \leq \boldsymbol{n} \leq \boldsymbol{N}$. The approximate MVA methods eliminate the need for this recursion by estimating $L_{ki}(\boldsymbol{n} - \boldsymbol{e}_j)$. Bard [8] and Schweitzer [171] proposed estimating

$$L_{ki}(\boldsymbol{n} - \boldsymbol{e}_j) = \begin{cases} L_{ki}(\boldsymbol{n}) & k \neq j \\ L_{ji}(\boldsymbol{n})(n_{ji} - 1)/n_{ji} & k = j. \end{cases} \qquad (3.16)$$

The MVA equations can then be reduced to a set of K nonlinear equations which are typically solved by successive substitution [117]. Existence of a physically meaningful solution has been shown in [48] but uniqueness and convergence have not been shown except for the case of single-chain networks [51]. The algorithm has been shown empirically to be fairly accurate for most networks (errors greater than 20 percent are rare) and is known to yield exact results in the limit as the chain populations increase to infinity.

Chandy and Neuse [37] describe the Linearizer approximation algorithm which is designed to improve the accuracy of the Bard–Schweitzer algorithm for smaller populations. Linearizer is basically an iterative technique to improve estimates of $F_{jki}(\boldsymbol{n})$ where

$$F_{jki}(\boldsymbol{n}) = \begin{cases} (L_{ki}(\boldsymbol{n} - \boldsymbol{e}_j)/n_{ki}) - (L_{ki}(\boldsymbol{n})/n_{ki}) & k \neq j \\ (L_{ji}(\boldsymbol{n} - \boldsymbol{e}_j)/(n_{ji} - 1)) - (L_{ji}(\boldsymbol{n})/n_{ji}) & k = j. \end{cases} \qquad (3.17)$$

and the Bard–Schweitzer algorithm assumes $F_{jki}(\boldsymbol{n}) = 0$. The iteration begins by applying Bard–Schweitzer at populations N and $(N - \boldsymbol{e}_k)$ for all k. At each iteration, values of $F_{jki}(N)$ obtained from the previous iteration are used to calculate new estimates of $F_{jki}(N)$. Neither existence, uniqueness, nor convergence has been shown, although this has not been a problem in practice and the method provides quite accurate results for most models (errors greater than 2 percent are extremely rare).

Another approach to dealing with large closed chain populations is to replace a closed chain by an open chain with an appropriately chosen arrival rate. In a closed product form network with an IS center having mean service times S_k, Lavenberg [116] has shown that as the populations N_k increase in such a way that N_k/S_k converges to a constant λ_k, then the stationary distribution converges to that of a network having open chains with Poisson arrival rates λ_k. He also gives adjustments to compensate for the finite populations N_k. In related work, Zahorjan [205] showed that replacing a closed chain by an open chain (in certain classes of single-chain product form networks) with the same throughput yields pessimistic results in the sense that mean response times are larger in the open chain representation and can, in fact, be substantially larger.

The model of LOCUS described earlier has been solved approximately by a method described in [48]. The method as applied to the LOCUS model assumes that most of the work done by a job is processed at its local site and only occasionally does a job visit a foreign site. The method loops through the sites representing the effect of foreign jobs at a local site by an open chain and representing the processing at foreign sites by an IS delay. This method was somewhat less accurate, but significantly faster than Linearizer on large LOCUS models.

D. Nonproduct Form Networks

Although the class of product form networks has proven quite useful, there are many important features which, when incorporated into a model, lead to queueing networks violating the product form assumptions. Such features include priority scheduling disciplines, general service time distributions at queues with FCFS disciplines, and certain blocking phenomena. In models of computer systems, the blocking frequently arises because a job requires more than one resource before it can be processed. Examples include the following.

1) Holding a channel and a disk drive before data transfer can occur.

2) Obtaining a memory partition before job processing can occur.

3) Obtaining a database lock before the data item can be read from disk.

Increasing model realism often leads to models without product form. There are three ways to solve such networks: exactly, approximately, or by simulation. Unless the model has very special structure, the computational cost of exact solution techniques quickly gets out of hand due to the explosion in the size of the state space. However, in general, such structure is hard to find and exact solution is, for all practical purposes, impossible. The main focus of this section is on approximation methods; simulation will be discussed in Section IV. However, before discussing approximations, we will describe several general classes of nonproduct form models for which special structures do exist along with relatively efficient computational algorithms.

1) Models Having a Matrix Geometric Solution: Neuts [145] has studied a general class of Markovian models, arising frequently in applications, for which computationally efficient algorithms can be constructed. Neuts calls this class "Models of the GI/M/1 Type" since they are a two-dimensional generalization of the GI/M/1 queue. The form of the solution to this class is called the "Matrix Geometric Solution." We discuss the solution for discrete time Markov chains; an analogous treatment for continuous time Markov chains is also given in [145]. The state space consists of the set of two-dimensional pairs (i,j) where $i \geq 0$ and $1 \leq j \leq B$ for some finite constant B. Transitions can occur from state (i,j) to states (i',j') where $0 \leq i' \leq i+1$. Although a somewhat more general structure can be accommodated, the basic structure of the transition matrix of the Markov chain is

$$P = \begin{bmatrix} B_0 & A_0 & 0 & 0 & 0 & \cdots \\ B_1 & A_1 & A_0 & 0 & 0 & \cdots \\ B_2 & A_2 & A_1 & A_0 & 0 & \cdots \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdots \end{bmatrix} \tag{3.18}$$

where A_k and B_k are square B-by-B matrices. Letting x_k be the stationary distribution of the kth block, i.e., states of the form $\{(k,j),\ 1 \leq j \leq B\}$, then, assuming irreducibility and a stability condition,

$$x_k = x_0 R^k \tag{3.19}$$

where R is the minimal nonnegative solution to the nonlinear matrix equation

$$R = \sum_{k=0}^{\infty} R^k A_k \tag{3.20}$$

and x_0 is a left eigenvector (corresponding to the eigenvalue one) of the matrix

$$B[R] = \sum_{k=0}^{\infty} R^k B_k, \tag{3.21}$$

i.e., $x_0 = x_0 B[R]$ with $x_0 (I - R)^{-1} e = 1$ where e is a vector of ones. Recursive algorithms to compute R exist and are particularly simple for quasi-birth and death processes in which transitions can only occur from (i,j) to (i',j') for $i' = i-1, i, i+1$. In general, the algorithms should be computationally efficient provided B is not too large, $A_k = 0$ for large values of k, or the spectral radius of R is not too close to one.

A computer performance evaluation example is given by Latouche [115] who considers a system consisting of C CPU's and J I/O devices. There is a single queue for the CPU's and another queue for the I/O devices. There is a total of M memory partitions and arriving jobs can only enter the system if a partition is free, otherwise it queues for the next available partition. This is an example of simultaneous resource possession since jobs must simulataneously own both a memory partition and a device before receiving service. Nelson and Iyer [144] applied the Matrix Geometric approach to study performance tradeoffs in a model of a replicated database. A variety of other applications can be found in [145].

2) Matrix Methods for Nearly Decomposable Models: A number of matrix iterative methods, called aggregation/disaggregation methods, which converge to the exact solution of nearly decomposable Markovian models has been recently studied by Cao and Stewart [32]. A Markovian model is termed nearly completely decomposable if the state space can be partitioned in such a way that the transition matrix P can be written as

$$P = \begin{bmatrix} P_{11} & P_{12} & P_{13} & \cdots \\ P_{21} & P_{22} & P_{32} & \cdots \\ P_{31} & P_{32} & P_{33} & \cdots \\ \cdot & \cdot & \cdot & \cdots \end{bmatrix} \tag{3.22}$$

where the off diagonal block matrices P_{ij} for $i \neq j$ are nearly zero. Thus, transitions are most likely to occur within blocks and transitions between blocks are infrequent. Models of this type arise frequently in applications as will be discussed in the section on approximations. In aggregation/disaggregation methods, the states within a block are lumped together to form a single aggregate state. These methods then compute the stationary probabilities of the aggregate states and the conditional state probabilities given the aggregate state. These quantities are computed exactly in the limit as the number of iterations tends to infinity. Cao and Stewart [32] showed that a number of previously described such proce-

dures can be placed in the same framework of being related to block matrix iterative methods. They also established convergence criteria and determined the convergence rates of the methods. At each step of the iteration, a set of linear equations must be solved for each of the aggregate states of the model. In addition, a set of equations describing the rate of flow between aggregate states must be solved. The methods avoid working directly with the original transition matrix P and their efficiency is thus related to the number of aggregate states and the number of states in each of the aggregate states. These methods appear promising for moderate sized problems, but we know of no reports in the literature in which these methods have been used to solve the kind of very large scale models that arise in performance modeling applications.

3) Approximation Methods: We will not give a comprehensive survey of approximation methods, but rather describe the general approaches that are used and comment on the validation of approximations. Surveys of approximation techniques along with more detailed descriptions can be found in [38], [48], [117], [162]. In addition, Lazowska *et al.* [127] describe a number of approximation techniques for modeling specific subsystems.

For product form networks, both exact solution techniques and bounding techniques have been developed as discussed earlier. Due to both the complexity and the explosion in size of the multidimensional state spaces that arise, neither exact solution techniques nor bounding techniques for general classes of nonproduct form networks have been forthcoming. The bounding techniques for product form networks have been based on the special nature of those networks, either through the convolution or MVA equations. Since these equations do not apply to nonproduct form networks, the bounding techniques cannot be applied. Approximation techniques (based on either limit theorems or heuristics) are thus the only viable alternative to simulation. Because error bounds are usually not available with an approximation technique, the accuracy of the method in any particular case can only be determined by comparison to simulation (or the exact solution if the state space is small enough). In order to develop generally applicable and reliable approximation techniques that can be incorporated into a package for, say, capacity planning, it is necessary to thoroughly validate the approximation over a wide range of parameter values by comparison to simulations. While it is relatively easy to suggest approximations, it is difficult, tedious, and computationally expensive to validate them.

Thus, although there is a plethora of methods for approximating the solutions to nonproduct form networks incorporating a variety of features, very few of these techniques have been thoroughly validated. An ideal validation study identifies a practically important class of models and key parameters of that class. Validations are then performed over the entire range of the parameter space and errors are quantified in terms of the parameters. A qualitative interpretation of the errors is then given, identifying where (in the parameter space) the approximation performs well and where it fails. Bryant *et al.* [25] performed such a validation for networks incorporating priority scheduling. They considered a set of two queue networks and parameterized the model in terms of the total utilization of the priority server ρ and the fraction of that utilization devoted to the high-priority jobs f. Contour plots of errors over the parameter space were given showing that several of the proposed methods were reliable as long as both ρ and f were not too close to one. The implication of the validation studies is that the approximation can be applied in practice with confidence provided these parameters are kept out of the extreme region where the method fails. There are few other such systematic validation studies reported in the literature. This type of comprehensive study becomes more difficult to design and expensive to perform in networks having more than two queues for which there may be more than two key parameters.

The major approaches to developing approximations are based on either limit theorems or heuristics. The limit theorems provide theoretical justification for an approximation whereas the heuristics are based on common sense and experience. There are two general classes of limit theorems used to justify approximations.

1) Heavy traffic limit theorems which lead to diffusion approximations.

2) "Nearly Completely Decomposable System" limit theorems which lead to a class of approximations called hierarchical decomposition, or aggregation, approximations.

Heavy traffic limits were originally obtained for classical queueing problems such as the waiting time process in a GI/G/1 queue as the traffic intensity approaches one [103]. They have been extended to both open networks of queues with high traffic intensities and general service times (e.g., [66], [94], [95], [151]) as well as to closed networks with large populations (e.g., [91]). The heavy traffic limit theorems show that as the traffic intensity approaches one, a properly scaled version of the queue length process converges in distribution to a diffusion process. Kobayashi [109] and Gelenbe [68] have applied diffusion approximations to queueing network models of computer systems. However, in practice, diffusion approximations have not been widely used because most systems do not operate with all service centers at or near full capacity. Furthermore, proper treatment of the boundary conditions of the multidimensional diffusions generally leads to an intractable system of partial differential equations [77].

The theory behind hierarchical decomposition is treated in Courtois [45]. A nearly completely decomposable Markov transition matrix Q can be rewritten in the form $Q = A + \varepsilon B$ where A is a stochastic block diagonal matrix with, say, M blocks and ε is small. For such systems, interactions between states within a block are much greater than those between blocks. This allows an accurate approximation to the stationary distribution of the Markov chain defined by Q to be computed by solving for the stationary distribution of each of the blocks of A and then computing the stationary distribution of an M-by-M transition matrix P which approximates the transition rates between blocks. This method is exact as $\varepsilon \rightarrow 0$. Whereas decomposition finds approximations for the stationary probabilities of the aggregate states and the

conditional state probabilities given the aggregate state, these quantities are computed exactly (in the limit as the number of iterations $\rightarrow \infty$) for the matrix iterative aggregation/disaggregation methods that were discussed in the previous section. However, the aggregation/disaggregation methods will generally involve substantially more computational effort than decomposition. In computer performance examples, if M is small and the blocks are chosen to correspond to product form networks, then a computationally efficient solution procedure results. Although error bounds are, in principle, possible to compute [45], [186], this does not seem to have been done for any very large scale applications.

We now consider an example of how decomposition has been used to model simultaneous resource possession in single chain networks [4], [44], [117], [159]. In the central server model with terminals pictured in Fig. 1, suppose that there is a limited number of memory partitions P. A job must acquire one of these partitions before its processing can begin and queueing is FCFS for the partitions. A block, or aggregate state, corresponds to the number of jobs using or waiting for partitions. This leads one to consider fixed multiprogramming levels in the (product form) CPU–I/O central server "inner" model. For each possible multiprogramming level, $n = 1, \cdots, P$, let $\alpha(n)$ be the throughput of jobs at the CPU divided by the average number of visits a job makes to the CPU before returning to the terminals; this corresponds to the rate of jobs leaving the CPU–I/O complex when the multiprogramming level is n. These throughputs are then used as queue length dependent rates for an aggregate server representing the CPU–I/O complex in a cyclic two service center "outer" model consisting of the terminals and the aggregate server. The aggregate server has rates $\mu(n)$ defined by

$$\mu(n) = \begin{cases} \alpha(n) & 0 \leq n \leq P \\ \alpha(P) & n \geq P. \end{cases} \tag{3.23}$$

Selection of these rates models limiting the maximum multiprogramming level to P. The two service center outer model also has product form and in this particular case reduces to a birth and death process. Thus, the approximate solution to the nonproduct form network is efficiently obtained by solving two related product form networks.

There is also a second justification for this method that is provided by a theorem due to Chandy, Herzog, and Woo [34] which states that the above procedure yields exact results in product form networks. In the above example, this corresponds to not having a limit on the multiprogramming level, i.e., P equals the total population of the network. The theorem states that a product form network can be solved exactly by replacing a subnetwork of service centers by a single service center with queue length dependent service rates. The rates are determined by solving, for each population, a network in which the service times of all service centers outside the subnetwork are set to zero, in effect "shorting" out the rest of the network. This theorem is sometimes called Norton's theorem since it is a queueing analog to Norton's theorem in analysis of electrical circuits. The theorem has been extended to multiple chain networks in [111].

Hierarchical decomposition has been used to model a variety of features in nonproduct form networks including the simultaneous resource possession example given above, priority scheduling disciplines [161] and certain types of parallelism [82]. In practice it has been found to be a highly accurate approximation technique and is a cornerstone of much applied modeling work.

However, decomposition has limitations for multiple chain models. For example, suppose the central server model with terminals has K chains and a limited number of memory partitions P_k for each chain k. Then throughput rates for each chain $\mu_k(n_1, \cdots, n_K)$ must be computed for each possible multiprogramming level $(n_1, \cdots, n_K)$ where $0 \leq n_k \leq P_k$ for $k = 1, \cdots, K$. Assuming the inner model still has product form, these could (at least in theory) be calculated in one pass of MVA. However, the outer model no longer satisfies product form and its solution requires solving the global balance equations for a K-dimensional birth and death process. The size of the state space is $(N_1 + 1) \cdots (N_K + 1)$ where N_k is the number of jobs in chain k. The transition matrix is sparse and can be solved by matrix iterative methods provided the product of the N_k's is not too large. Sauer [159] considered this model for two chains and $N_1 = 40$ and $N_2 = 4$ and found that decomposition provided quite accurate results. However, for models with more than two chains the method becomes computationally infeasible because of the explosion in the size of the state space. A method proposed independently by Brandwajn [22] and Lazowska and Zahorjan [126] attempts to circumvent this problem by iteratively solving a sequence of K one-dimensional birth and death processes rather than one K-dimensional birth and death process. The rates used for chain k are essentially obtained by setting the populations of the other chains to estimates of their mean values. Only limited validations of this approach have been done; it is generally less accurate than complete decomposition but is obtained at much lower cost.

It is sometimes possible to prove limit theorems for particular models allowing theoretically justified approximations. Salza and Lavenberg [158] prove a theorem showing that the sojourn time in a central server model converges to an exponential distribution as the feedback probability converges to one. This result is used, along with decomposition, to approximate the response time distribution in a central server model with terminals; the response time distribution in queueing networks is not analytically tractable except in special cases (e.g., [165], [201]). Other examples of special limit theorems are given by Mitra and Weinberger [142] and Lavenberg [118] who have used asymptotic expansions to estimate the probability of lock contention in a model of a database system as the size of the database increases.

Approximations based on heuristics have been used to model a wide variety of features in queueing networks. There are two major approaches upon which such heuristics are based.

1) The MVA equations and the Arrival Theorem.
2) Iterative methods.

Some approximation methods are based on a combination

of approaches, for example, iteration and decomposition [22], [126]. Still other approximations use specially tailored techniques.

A primary use of the MVA approach has been to model service disciplines and distributions other than those allowed in product form networks [8], [25], [152]. For some service disciplines or distributions, if the distribution of the number of jobs seen upon arrival of a job to a service center is known, then the mean response time at that service center can be calculated either exactly or approximately. In the MVA based approach, the distribution of jobs seen on arrival is assumed to be given by the Arrival Theorem; namely, it is the stationary distribution of a network with that one job removed. Because the Arrival Theorem is not valid except in product form networks, the method is an approximation. Reiser [152] considers an approximation to model the FCFS discipline for centers having exponential distributions with different means. If center i is FCFS, then the requirement for product form is that the mean service demands S_{ji} are independent of j. For distinct S_{ji}'s the suggested approximation is to modify (3.8) as follows:

$$R_{ji}(\boldsymbol{n}) = (1/\mu_i)(S_{ji} + \sum_{k=1}^{K} S_{ki} L_{ki}(\boldsymbol{n} - \boldsymbol{e}_j)), \quad (3.24)$$

which states that a job's average response time is its own average service time plus the mean queue length of jobs found on arrival times the average service time of each job. If all the service times are identical, (3.24) reduces to (3.8). A modification using the mean residual life (from renewal theory) for distributions other than exponential was also suggested by Reiser [152] and approximations for priority queueing disciplines have been proposed by Bryant *et al.* [25] and Chandy and Lakshmi [36].

A second general heuristic approach to analyzing non-product form networks is iteration. In this approach, a sequence of simplified networks is solved so that, upon convergence, the solution closely approximates the solution of the network of interest. If each network in the sequence has product form, the overall method is computationally efficient provided convergence is obtained in a reasonable number of iterations. This approach can often be shown to be equivalent to finding the fixed point of a multidimensional set of nonlinear equations $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x})$ [41], [48], [81], [82]. Existence of a solution can frequently be shown by applying the Brouwer fixed point theorem [147]. The equations are typically solved by successive substitutions, i.e., $\boldsymbol{x}^{n+1} = \boldsymbol{f}(\boldsymbol{x}^n)$. If the function $\boldsymbol{f}$ can be shown to be a contraction mapping, then convergence and uniqueness are guaranteed. However, in performance applications $\boldsymbol{f}$ usually cannot be shown to be such a mapping. The function $\boldsymbol{f}$ is generally a complicated function usually involving the solution of a product form network and the variables $\boldsymbol{x}$ represent performance measures of the network.

Heidelberger and Trivedi [81] consider a model of parallel processing in which a primary job spawns an asynchronous task whenever it passes through a particular node in the network. The primary jobs are represented by a closed chain. The overall model does not have product form but it is approximated by a product form network having an extra open chain representing the spawned tasks. The Poisson arrival rate of the spawned tasks is set equal to the throughput of the primary jobs at the spawning node. The throughput of primary jobs depends on the arrival rate of spawned jobs and the fixed point problem reduces to finding that arrival rate for which the primary job throughput equals the spawned task arrival rate. Uniqueness and convergence were proven for this example and an extensive validation study was performed.

Iterative methods have also been applied to analyzing networks incorporating a wide variety of features including simultaneous resource possession [98], [99], general service time distributions [132], database lock contention [194], and parallel processing systems in which tasks requiring synchronization are spawned [82]. Selection of the approximating simplified networks is still an art form.

E. Operational Analysis—An Alternative Viewpoint

We have presented the analysis of queueing networks from the stochastic process point of view. For example, using the theory of Markov processes, product form can be shown formally to hold provided certain assumptions on service time distributions, arrival processes, stochastic routing and queueing disciplines are met. A recently developed alternative viewpoint is Operational Analysis which relates measurable quantities in queueing networks [30], [47], [127]. The measurable quantities are not assumed to be random variables. The operational viewpoint is that if a system is measured during an interval $[0, T]$, say, then certain relationships between variables must hold provided the system satisfies certain operational assumptions during the interval. For example, if a queue length can only change by either plus or minus one job when there is an arrival or departure, then the number of arrivals must equal the number of departures provided T is chosen so that the queue length at time zero equals the queue length at time T. More subtle relationships can be shown as well including an operational analog of Little's result $L_T = \lambda_T W_T$ provided the queue is empty at times zero and T [53], [128]. In this equation $L_T = (1/T)\int_0^T L(s)\,ds$ where $L(s)$ is the queue length at time s, $\lambda_T = N(T)/T$ where $N(T)$ is the number of arrivals (and departures) from the queue in $[0, T]$, and $W_T = (1/N(T))\sum_{i=1}^{N(T)} W(i)$ where $W(i)$ is the waiting time of customer i. In fact, this sample path version of Little's result predates the first papers on operational analysis by about 15 years. In the operational framework, the fraction of time spent in a state is the analog of a state probability. Using this analog, product form can also be shown to be an operational law provided certain operational assumptions are satisfied during the measurement period. These include homogeneity assumptions on routing and devices. For example, device homogeneity assumes that the departure rate from a center depends only on the queue length of that center and not on the state of any other center. Operational versions of MVA and the Arrival Theorem have been derived as well.

The obvious question is how the operational assumptions are related to the stochastic assumptions. This has not been

investigated in much detail. However, there is an interesting analysis by Bryant [24] showing that, in the limit as the length of the measurement period $T \to \infty$, the only M/G/1 queue satisfying the operational assumption of homogeneous service times is the M/M/1 queue, i.e., the queue with exponential service times. The assumption of homogeneous service times is thus analogous to the assumption of a fixed (queue length independent) rate server with exponential service times. We conjecture that there are similar results for networks relating operational to stochastic assumptions. For example, one can show (using ergodic properties of finite state space continuous time Markov chains) that the operational assumptions necessary for product form hold in the limit with probability one (along an appropriate sequence of regeneration points) for the class of single chain closed product form networks with queue length dependent exponential centers. Furthermore, the homogeneous service time assumption can be shown to hold in the limit for such networks with queue length independent exponential centers. For such closed single chain models, no other general class of well-defined stochastic queueing models has been identified for which the operational assumptions hold, with probability one, in the limit. Whether or not such a class exists is an open question.

One claimed advantage of the operational viewpoint is that it allows one to test the assumptions and to provide bounds on errors induced when the operational assumptions are violated. Indeed, although it is possible to formally test in a statistical sense the stochastic assumptions, there are few quantitative sensitivity results in queueing theory. However, the sensitivity results for operational analysis can frequently be interpreted in the stochastic setting. For example, Suri [190] provides sensitivity results, in the form of gradients, for deviations from the homogeneous service time assumption. The gradients can be used to obtain error bounds for performance measures predicted by assuming homogeneous service times in networks in which this assumption is slightly violated (the bounds are valid only in an appropriately defined neighborhood about the homogeneous point). These results can also be interpreted as being gradients and bounds on stochastic queueing networks when service rates are not constant.

Operational analysis thus provides a different set of assumptions on which to base and interpret certain results primarily about product form queueing networks. It has broadened the appeal of queueing networks to include those who feel uncomfortable making stochastic assumptions such as independence, stationarity, and exponential distributions. A challenge for operational analysis is to provide major new results, rather than (as has been the case) to merely provide operational versions of existing theorems which were originally derived using stochastic analyses.

F. Future Challenges

Analytic performance modeling has been an extremely useful tool for evaluating the performance of computing systems of the 1970's and early 1980's. However, computing systems are rapidly advancing and analytic modeling techniques must advance with them in order to maintain relevance in the late 1980's and into the 1990's. Distributed processing, parallel processing, and radically new computer architectures present significant modeling challenges and opportunities.

Distributed processing systems will become commonplace. These systems will serve large numbers of users, consist of many devices, and will incorporate large databases, both centralized and distributed. High levels of performance will be key. Performance needs to be designed into these systems, not only in determining the number of devices and their speeds, but also in designing operating system algorithms to dynamically manage the system. Optimization of queueing networks will thus become more important both in terms of distributed system design and load balancing. Some optimal load balancing results for queueing networks are beginning to emerge [192], however, major new advances in this area are required. Further work is needed in database modeling where satisfactory methods for analyzing lock contention are also just starting to emerge [71], [118], [142], [148], [193], [195]. Particular attention needs to be paid to modeling distributed databases.

Parallel processing systems will also become widespread as hardware costs continue to decline. By parallel processing we mean multiple subtasks running concurrently and cooperating to solve some larger task. Parallel processing will come about as a result of new computer architectures, such as data flow and multiple microprocessor architectures, which are explicitly designed to take advantage of parallelism. Parallel processing will also arise as the result of distributed databases and the requirement for highly reliable and available systems. There have been very few techniques developed to analyze such systems [55], [81], [82], [87] and much work remains to be done.

In meeting these challenges approximations will play a key role. If possible, computable error bounds need to be developed, although this looks extremely difficult for nonproduct form networks. Improved methods for applying hierarchical decomposition in multiple chain models need to be developed. A consistent and comprehensive framework for validating approximations needs to be developed and applied. The validations need to include stress cases to identify when approximations fail as well as when they are succesful. Finally, there has been very little work on validating combinations of approximations. Most approximation studies have focused on introducing a single nonproduct form feature into an otherwise product form network. Exceptions to this are the previously mentioned capacity planning tools [7], [29] which incorporate a variety of approximations simultaneously and have been validated against measurements.

IV. Simulation Performance Modeling

Simulation is an extremely versatile and useful tool in computer performance evaluation. Whereas analytic techniques have limitations on the range of features that can be modeled, a simulation model can be constructed to an almost

arbitrary level of detail. This allows one to model extremely complex situations that are analytically intractable. Furthermore, whereas analytic models typically provide only mean values, simulations can provide estimates of distributions and higher moments. In addition, dynamic, or transient, behavior can be studied using simulation while analytic models can usually be used to study only steady state behavior. In fact, a major application of simulation is to validate analytic models.

There are two distinct types of simulation that have become widespread in computer performance evaluation.

1) Trace Driven Simulation: This is a simulation of a deterministic model that is driven by a sequence, or trace, obtained from measurements of an existing system. The model often does not have a queueing structure. Trace driven simulations have been primarily used to study the performance of storage hierarchies and of processor pipelines.

2) Stochastic Discrete Event Simulation: This is typically a simulation of a queueing model that is driven by sequences of random (or pseudorandom) numbers with user specified distributions. These random sequences are used for obtaining service times, routing, etc. Occasionally, trace data are used (possibly in conjunction with random sequences) to drive queueing model simulations, e.g., [177]. Discrete event simulations have extremely broad applicability and have been used extensively in computer performance evaluation.

In this section, we will discuss these two types of simulation. Since stochastic discrete event simulations are statistical experiments, their outputs are random and require careful statistical interpretation. We will describe some of the statistical problems that arise in analysis of simulation output data, including confidence interval generation and simulation run length control.

A. Trace Driven Simulation

In the design of storage hierarchies, trace driven simulation has been used to study the performance effects of paging algorithms (e.g., [5], [12]), cache management algorithms [182], database buffering strategies [179], buffered, or cached, disks [180], and long-term file migration policies [181].

An appropriate trace, or script, is obtained by measuring a system. For example, in studies of cache performance a job is run and the sequence of memory address (page and line) references is recorded. A software model of the cache organization and its management algorithms is constructed. These management algorithms determine how data are brought into and removed from the cache. The software cache model takes the address reference sequence as input and simulates the behavior of the cache. The model can be easily changed to reflect different cache organizations and management algorithms. The key performance measure of interest in such studies is usually the cache miss ratio, the fraction of references not found in the cache. The miss ratio is critically dependent on the size of the cache and it is common to plot the miss ratio as a function of the cache size. These curves can be efficiently generated in one pass through the address trace using a "stack processing algorithm" [133] for a broad class of demand paging algorithms, including the Least Recently Used replacement algorithm. Stack processing is applicable to performance evaluation of multilevel memory hierarchies, not just a single-level cache.

The combination of a flexible software model and repeatable workloads enables one to study a variety of cache organizations under identical workloads without having to build hardware. Using a set of representative measurement traces increases model realism and avoids the necessity of constructing possibly complicated stochastic workload models. However, obtaining representative traces for multiprogramming environments may be difficult in practice. Consider a page reference trace from a multiprogramming system which includes references from the operating system dispatcher which is invoked upon page faults. When such a trace is run against a model simulating different paging algorithms, a reference from the trace that caused a page fault in the measured system may not cause a page fault in the simulated system. However, the trace still consists of the references from the dispatcher which should not be invoked in the simulator. As a result of this type of consideration, many traces are taken from single jobs running in uniprogramming environments. Smith [182] simulated task switching by switching between traces of different jobs at periodic intervals. As noted in Section II-A, Clark [42] observed differences between measurement and trace driven simulation results for these reasons among others.

Trace driven simulation is also a powerful tool in the performance analysis of processor pipeline design. Instruction traces, obtained from measurements, are used to drive a software model of the pipeline. The model may be of arbitrary detail. For example, the model may or may not include the effects of a finite size cache. The key performance measure in pipeline studies is usually the instruction throughput, the number of instructions per machine cycle. Lang *et al.* [114] describe in more detail a general approach to modeling pipelines. Pipeline simulators, which tend to be built by manufacturers for use in machine development, are often proprietary. However, examples of pipeline simulators in the open literature include those by Kumar and Davidson [112] who modeled the IBM 360/91, Smith [183] who studied branch prediction strategies for the CDC CYBER 170 architecture, Srini and Asenjo [185] who considered a variety of changes in the Cray-1S architecture, and MacDougall [129] who considered the IBM 370 architecture.

B. Stochastic Discrete Event Simulation

Stochastic discrete event simulations are driven by sequences of pseudorandom numbers generated by the computer. In addition to specifying the model structure, the analyst must also specify the distributions of these sequences. The term discrete event refers to the fact that events in the simulation can only occur at a countable number of points in simulated time and not continuously. Discrete event simulations are much more widespread in computer performance evaluation than are continuous simulations and we not will discuss continuous simulation at all.

The development of simulation models is greatly facilitated by the use of general purpose simulation modeling lan-

guages, such as GPSS, SIMSCRIPT, GASP, and SLAM, overviews and examples of which are given in [125]. These and other simulation languages ease development of simulation models by providing high-level constructs and features common to all simulations, such as random number generation, event scheduling, queue management, and statistics gathering and reporting. A number of these and other simulation languages are beginning to appear on microcomputers [149]. Simulation modeling packages specifically designed for computer performance evaluation have also been developed, e.g., RESQ [164], PAWS [96], and QNAP [140]. These packages provide higher level modeling constructs particularly well suited for simulation of queueing network models of computer performance. Even higher level packages exist for modeling particular computer systems; e.g., SNAP/SHOT [187], used in capacity planning, includes built in models of certain IBM devices and control programs.

General texts on simulation include [62], [125]. These texts treat basic simulation modeling concepts, random number generation, statistical analysis of simulation output data, and give overviews of simulation languages. References [110] and [117] contain sections on various aspects of simulation as applied to computer performance evaluation.

1) Statistical Analysis of Simulation Output: Discrete event simulation is controlled statistical experimentation. Models are driven by random input sequences, such as service times, and produce random output sequences, such as response times, from which estimates of the response time distribution and/or its moments are obtained. If the simulation is rerun under identical conditions (but driven by different statistically independent input sequences), the output estimates will be different, often dramatically so. Thus, as in the case of measurements, sound statistical methods are required to interpret the results of simulation models. We will give an overview of some of the statistical aspects of simulation output analysis. More complete discussions are in [104], [105], [117, chapter 6], [123].

Because simulation outputs are random, it is important to assess the amount of variability in estimates that is due purely to random sampling effects. In addition to assessing statistical accuracy, it is important to be able to adjust the length(s) of the simulation run(s) so as to obtain estimates of specified accuracy. These two problems of accuracy assessment and run length control can be addressed through the use of confidence intervals. Suppose μ is some unknown output characteristic of the model to be estimated, e.g., the mean steady state response time. An interval (L, U) with random endpoints L and U is said to be a $100(1-\alpha)$ percent confidence interval for μ if Prob $\{L \leq \mu \leq U\} = 1 - \alpha$. For $(1-\alpha)$ close to one, there is a high probability that the unknown parameter μ is between L and U. The confidence interval is usually formed in such a way that a point estimate $\hat{\mu}$ of μ lies in the interval and is frequently the midpoint of the interval. Formation of such a confidence interval generally requires an estimate of the variance of $\hat{\mu}$. The width of the interval $U-L$ is then a measure of the accuracy of $\hat{\mu}$. The narrower the interval, the more confidence can be placed in the estimate. Simulation run length control algorithms have been developed that allow a model to be run until confidence intervals of suitable accuracy (as defined by the analyst) have been obtained or until a CPU time limit is exceeded [83], [121].

Classical statistical techniques can be applied when estimating transient characteristics since multiple independent replications can be performed thereby generating iid (independent and identically distributed) observations. However, simulation studies of queueing network models of computer performance often involve estimating steady state characteristics. There are a variety of procedures for generating confidence intervals for steady state characteristics. Standard statistical approaches based on iid observations cannot be directly applied since simulation output sequences are usually both autocorrelated and nonstationary. The autocorrelation arises because of queueing; if the waiting time of a job at a device is large, then it is likely that the waiting time of the next job will also be large. Nonstationarity is a consequence of the model's initial conditions; if all jobs in the model of Fig. 1 are started in the think state at the terminals, then the first job arriving to the CPU experiences no queueing whereas subsequent arrivals may require queueing.

Suppose we let $X_1, \cdots, X_N$ be the response time output sequence generated by the simulation and that we are interested in estimating the mean steady state response time $\mu = \lim_{n\to\infty} E(X_n)$. The usual estimate for μ is the sample average

$$\hat{\mu} = (1/N) \sum_{n=1}^{N} X_n. \tag{4.1}$$

The problem of nonstationarity is that for small n, $E(X_n) \neq \mu$ which in general means that $E(\hat{\mu}) \neq \mu$. The typical approach for dealing with this problem, which is also called the problem of the initial transient, is to determine an N_0 such that $E(X_n) \simeq \mu$ for $n \geq N_0$, delete the observations before N_0, and then estimate μ by

$$\hat{\mu} = (1/(N - N_0)) \sum_{n=N_0+1}^{N} X_n. \tag{4.2}$$

This will be an approximately unbiased estimate of μ. Schruben [169] has proposed statistical tests for stationarity that can be used to test the adequacy of an N_0.

The other difficult problem is dealing with the autocorrelation. Suppose that a satisfactory N_0 is found so that $\{X_n, n \geq N_0\}$ is approximately a (covariance) stationary sequence. Then the central limit theorem states that, for large N, $(\hat{\mu} - \mu)/\sigma(\hat{\mu})$ is approximately normally distributed with mean zero and variance one where $\sigma^2(\hat{\mu})$ is the variance of $\hat{\mu}$. If the X_n's are iid, then $\sigma^2(\hat{\mu}) = \sigma^2(X)/(N - N_0)$ where $\sigma^2(X)$ is the variance of X_n. However, this equation is not valid for correlated observations. For large sample sizes, the correct expression for the variance of correlated observations is

$$\sigma^2(\hat{\mu}) \simeq (\sigma^2(X)/(N - N_0)) \left(\sum_{k=-\infty}^{\infty} \rho_k \right) \tag{4.3}$$

where ρ_k is the autocorrelation between X_n and X_{n+k}. Thus, the variance of a correlated sequence equals the variance of

an independent sequence times an expansion factor (the sum of the autocorrelation function) which measures the amount of correlation in the sequence. This expansion factor is usually positive in simulations of queueing networks, and it can easily be much larger than one. For example, in the M/M/1 queue with traffic intensity 0.90, the expansion factor is 367 [16]. It cannot be ignored in generating confidence intervals since assuming positively correlated observations are independent leads to severe underestimation of the variance and confidence intervals which are much too narrow.

There are two general approaches to dealing with the correlation when generating confidence intervals.

1) Avoid it by organizing the simulation output into iid observations.

2) Estimate the correlation and compensate for it.

There are three methods to avoid the correlation. The first method is to use independent replications. This has the advantage of simplicity but the disadvantages of being sensitive to the effects of the initial transient and wasteful of data if the transient portion is discarded from the beginning of each replication. The second method is called batch means (e.g., [124], [138]) which operates on a single run. The sequence is divided into long blocks, or batches, of length B so that the means of the batches are approximately iid. If B is so chosen, then classical statistical methods can be applied. However, selection of an adequate B is a difficult statistical problem and there is currently no completely satisfactory method. The third method is called the regenerative method (e.g., [93]) which is based on the fact that some stochastic sequences, called regenerative processes, contain regeneration points which delimit the sequence into iid blocks of random length. However, the method is not generally applicable since most processes arising in computer performance evaluation are either not regenerative or contain too few regeneration points to produce valid confidence intervals unless the run is extremely long.

A single-run method, called the spectral method, that estimates the correlation in the sequence is described in [83]. This method uses the fact that $\sigma^2(\hat{\mu}) \simeq p(0)/(N - N_0)$ where $p(0)$ is the spectral density of the process at zero frequency. The spectral density is the Fourier cosine transform of the covariance function $\{\gamma_k\}$ of the process defined by

$$p(f) = \sum_{k=-\infty}^{\infty} \gamma_k \cos(2\pi f k). \tag{4.4}$$

Spectral estimation techniques are applied to estimate $p(0)$. This method has been shown empirically to produce satisfactory results for a variety of computer performance models and has been incorporated into a run length control procedure.

The problems of combining initial transient detection and deletion, confidence interval generation, and run length control into one automatic procedure have been investigated by Heidelberger and Welch [84]. Procedures combining the transient tests in [169] with the spectral method were tested. Although generally acceptable results were obtained, there were extreme cases of slowly developing transients in which the methods failed. Further work is needed in this area.

The above procedures are used to place a confidence interval on a single parameter which can be expressed as a mean. Both moments and points on a probability distribution can be expressed as means. Analagous techniques also exist for generating confidence intervals for quantiles [80], [92]. Multivariate statistical procedures can be applied to place simultaneous confidence intervals on more than one parameter (e.g., [125], [168]), although this is rarely done in practice. Regression and design of experiments are potentially useful techniques in simulation modeling. For example, Jones [101] used design of experiments and analysis of variance to design and analyze simulation experiments investigating factors affecting disk subsystem performance. However, because of the difficulties in identifying key parameters and specifying a functional form and the requirement for independent observations with identical variances, these techniques have not been widely used by simulation practitioners. These are essentially the same reasons that were cited in Section II-D for why such techniques have not been applied frequently in measurement studies.

The variance expansion due to the autocorrelation in the output sequences [see (4.3)] poses another difficulty besides variance estimation; it implies that quite a long simulation run may have to be performed before a reasonable level of accuracy is attained. Variance reduction techniques attempt to decrease the variance of an estimate by using known information about the system being simulated. For example, Lavenberg *et al.* [119] propose using known information about service times and job routing in a variance reduction technique known as control variables. While variance reduction techniques are an attractive possibility for cutting down on simulation run lengths and have been much studied in the literature, they have been rarely successful in real applications [122].

Ho *et al.* [89] and Suri [189] have proposed an interesting technique called perturbation analysis for estimating the gradient of a performance measure with respect to some input parameter of the simulation, e.g., a mean service time. The gradient estimate is obtained from a single run by carrying along certain ancillary information which measures the effect of an infinitesimal change in the parameter on the sample path. The gradient estimates can be used in sensitivity analysis or optimization procedures. Questions of the unbiasedness and ergodicity properties (convergence with probability one) of the gradients estimates have been addressed in some specific instances [88], [191]. General conditions for these properties to hold have been given in [33], although these conditions appear difficult to verify in practice. The method has also been proposed to study the effects of finite changes in parameters, e.g., increasing a buffer size. Although some empirical validations have been done, little is known about the statistical properties of such estimates. Perturbation analysis should also prove applicable in trace driven simulations and measurements of real systems.

C. Future Challenges

As computer systems become larger and more complicated, analytic modeling will become more difficult and simulation will play an increasingly important role in per-

formance evaluation. A key challenge is to be able to devise computationally efficient techniques to simulate models of increasingly complex systems. Effective generally applicable variance reduction techniques need to be developed, although this appears unlikely. Innovative approaches, such as perturbation analysis, need to be developed, refined, and applied in practice. Hierarchical modeling techniques should be applicable in simulation modeling as well as in analytic modeling. However, some preliminary studies [17] indicate that such an approach is not always computationally attractive and there are difficult statistical issues to be faced. However, hybrid modeling techniques that contain both analytic and simulation components, often in a hierarchical structure, have occasionally been successfully applied in practice (e.g., a model of IBM's MVS operating system by Chiu and Chow [40]) and should prove to be an effective approach in the future. Shanthikumar and Sargent [176] survey hybrid modeling techniques.

Parallel processing holds promise for increasing simulation efficiency. A potentially attractive approach is to coordinate the activities of multiple microprocessor systems. However, control of distributed simulation is analagous to ensuring consistency in distributed databases. Therefore, synchronization and deadlock detection algorithms are required (e.g., [35]) for distributed simulations just as concurrency control algorithms [13] are required for distributed databases. The challenge is to be able to effectively partition models in such a way that both the overhead for these algorithms is small and that high levels of parallelism can be achieved. Prototype systems and languages are beginning to appear (e.g., [203]).

Computer graphics will also play an increasingly important role in simulation. Graphics can be used to facilitate model input [49]. Data analysis, both in general and in simulation applications, will employ graphical techniques and these are starting to appear [79], [117, chapter 6]. Real-time graphic animation of simulation is an intriguing possibility [139], [141]. Such real-time graphics present exciting opportunities for interactive control and analysis of simulation models.

Acknowledgment

The authors are grateful to D. Ferrari, D. Mitra, R. Suri, and W. White for conversations that were helpful in the preparation of this paper. We thank K. Trivedi for inviting us to write this paper. We also thank E. Zoernack for her help in preparing the manuscript.

References

[1] A. K. Agrawala, R. M. Bryant, and J. M. Mohr, "An approach to the workload characterization problem," *Computer*, vol. 9, pp. 18–32, 1976.

[2] A. K. Agrawala and J. M. Mohr, "The relationship between the pattern recognition problem and the workload characterization problem," in *Proc. SIGMETRICS/CMG VIII*, 1977, pp. 131–139.

[3] H. P. Artis, "Capacity planning for MVS computer systems," in *Performance of Computers Installations*, D. Ferrari, Ed. Amsterdam, The Netherlands: North-Holland, 1978, pp. 25–35.

[4] B. Avi-Itzhak and D. P. Heyman, "Approximate queueing models for multiprogramming computer systems," *Oper. Res.*, vol. 21, pp. 1212–1229, 1973.

[5] Ö. Babaoğlu and D. Ferrari, "Two-level replacement decisions in paging stores," *IEEE Trans. Comput.*, vol. C-32, pp. 1151–1159, Dec. 1983.

[6] Y. Bard, "Performance criteria and measurement for a time-sharing system," *IBM Syst. J.*, vol. 10, pp. 193–216, 1971.

[7] ——, "The VM/370 performance predictor," *Comput. Surveys*, vol. 10, pp. 333–342, 1978.

[8] ——, "Some extensions to multiclass queueing network analysis," in *Performance of Computer Systems*, M. Arato, A. Butrimenko, and E. Gelenbe, Eds. Amsterdam, The Netherlands: North-Holland, 1979, pp. 51–61.

[9] Y. Bard and M. Schatzoff, "Statistical methods in computer performance analysis," in *Current Trends in Programming Methodology, Vol. III: Software Modeling*, K. M. Chandy and R. T. Yeh, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1978, pp. 1–51.

[10] Y. Bard and K. Suryanarayana, "On the structure of CP-67 overhead," in *Statistical Computer Performance Evaluation*, W. Freiberg, Ed. New York: Academic, 1972, pp. 329–346.

[11] F. Baskett, K. M. Chandy, R. R. Muntz, and F. Palacios-Gomez, "Open, closed, and mixed networks of queues with different classes of customers," *J. Assoc. Comput. Mach.*, vol. 22, pp. 248–260, 1975.

[12] L. A. Belady, "A study of replacement algorithms for a virtual storage computer," *IBM Syst. J.*, vol. 5, pp. 78–101, 1966.

[13] P. A. Bernstein and N. Goodman, "Concurrency control in distributed database systems," *Comput. Surveys*, vol. 13, pp. 185–221, 1981.

[14] R. Blake, "Instrumentation for multiple computers," in *Perform. Eval. Rev. 9, Proc. PERFORMANCE '80*, 1980, pp. 11–25.

[15] R. Blau, "Paging on an object-oriented personal computer," in *Perform. Eval. Rev., Special Issue, Proc. 1983 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 44–54.

[16] N. Blomqvist, "The covariance function of the M/G/1 queueing system," *Skand. Akt. Tidskr.*, vol. 50, pp. 157–174, 1967.

[17] A. Blum, L. Donatiello, P. Heidelberger, S. S. Lavenberg, and E. A. MacNair, "Experiments with decomposition of extended queueing network models," IBM, Yorktown Heights, NY, Res. Rep. RC 10213, 1983.

[18] M. L. Bolzoni, M. Calzarossa, P. Mapelli, and G. Serazzi, "A package for the implementation of static workload models," in *Perform. Eval. Rev. 11, Proc. 1982 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 58–67.

[19] P. Bourret and P. Cros, "Presentation and correction of errors in operating system measurements," *IEEE Trans. Software Eng.*, vol. SE-6, pp. 395–398, 1980.

[20] G. E. P. Box, W. G. Hunter, and J. S. Hunter, *Statistics for Experiments, An Introduction to Design, Data Analysis and Model Building*. New York: Wiley, 1978.

[21] A. Brandwajn, "Multiple paths versus memory for improving DASD subsystem performance," in *PERFORMANCE '81*, F. J. Kylstra, Ed. Amsterdam, The Netherlands: North-Holland, 1981, pp. 415–434.

[22] ——, "Fast approximate solution of multiprogramming models," in *Proc. 1982 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 141–149.

[23] ——, "Models of DASD subsystems with multiple access paths: A throughput-driven approach," *IEEE Trans. Comput.*, vol. C-32, pp. 451–463, 1983.

[24] R. M. Bryant, "On homogeneity and on line-off-line behavior in M/G/1 queueing systems," *IEEE Trans. Software Eng.*, vol. SE-7, pp. 291–299, 1981.

[25] R. M. Bryant, A. E. Krzesinski, and P. Teunissen, "The MVA pre-empt resume priority approximation," in *Perform. Eval. Rev., Special Issue, Proc. 1983 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 12–27.

[26] I. Y. Bucher, "The computational speed of supercomputers," in *Perform. Eval. Rev., Special Issue, Proc. 1983 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 151–165.

[27] J. P. Buzen, "Queueing network models of multiprogramming," Ph.D. dissertation, Div. Eng. Appl. Phys., Harvard Univ., Cambridge, MA, 1971.

[28] ——, "Computational algorithms for closed queueing networks with exponential servers," *Commun. ACM*, vol. 16, pp. 527–531, 1973.

[29] ——, "A queueing network model of MVS," *Comput. Surveys*, vol. 10, pp. 319–331, 1978.

[30] J. P. Buzen and P. J. Denning, "Measuring and calculating queue length distributions," *Computer*, vol. 13, pp. 33–44, 1980.

[31] M. D. Canon, D. H. Fritz, J. H. Howard, T. D. Howell, M. F. Mitoma, and J. Rodriquez-Rosell, "A virtual machine emulator for performance evaluation," *Commun. ACM*, vol. 23, pp. 71–80, 1980.
[32] W. Cao and W. J. Stewart, "Iterative aggregation/disaggregation techniques for nearly uncoupled markov chains," North Carolina State Univ., Tech. Rep., 1984; also *J. Assoc. Comput. Mach.*, to be published.
[33] X. R. Cao, "Convergence of parameter sensitivity estimates in a stochastic environment," in *Proc. 23rd IEEE Conf. Decision and Control*, Las Vegas, NV, to be published.
[34] K. M. Chandy, U. Herzog, and L. Woo, "Parametric analysis of queueing networks," *IBM J. Res. Develop.*, vol. 19, pp. 36–42, 1975.
[35] K. M. Chandy, V. Holmes, and J. Misra, "Distributed simulation of networks," *Comput. Networks*, vol. 3, pp. 105–113, 1979.
[36] K. Chandy and M. S. Lakshmi, "An approximation technique for queueing networks with preemptive priority queues," Dep. Comput. Sci., Univ. Texas, Austin, TX, Tech. Rep., 1983.
[37] K. M. Chandy and D. Neuse, "Linearizer: A heuristic algorithm for queueing network models of computer systems," *Commun. ACM*, vol. 25, pp. 126–134, 1982.
[38] K. M. Chandy and C. H. Sauer, "Approximate methods for analysis of queueing network models of computer systems," *Comput. Surveys*, vol. 10, pp. 263–280, 1978.
[39] ——, "Computational algorithms for product form queueing networks," *Commun. ACM*, vol. 23, pp. 573–583, 1980.
[40] W. W. Chiu and W. M Chow, "A performance model of MVS," *IBM Syst. J.*, vol. 17, pp. 444–462, 1978.
[41] W-M. Chow, "Approximations for large scale closed queueing networks," *Perform. Eval.*, vol. 3, pp. 1–12, 1983.
[42] D. W. Clark, "Cache performance in the VAX-11/780," *ACM Trans. Comput. Syst.*, vol. 1, pp. 24–37, 1983.
[43] W. G. Cochran and G. M. Cox, *Experimental Designs (second edition)*. New York: Wiley, 1957.
[44] P. J. Courtois, "Decomposability, instabilities and saturation in multiprogramming systems," *Commun. ACM*, vol. 18, pp. 371–377, 1975.
[45] ——, *Decomposability: Queueing and Computer System Applications*. New York: Academic, 1977.
[46] C. A. Coutant, R. E. Griswold, and D. R. Hanson, "Measuring the performance and behavior of icon programs," *IEEE Trans. Software Eng.*, vol. SE-9, pp. 93–103, 1983.
[47] P. J. Denning and J. P. Buzen, "The operational analysis of queueing network models," *Comput. Surveys*, vol. 10, pp. 225–261, 1978.
[48] E. de Souza e Silva, S. S. Lavenberg, and R. R. Muntz, "A perspective on iterative methods for the approximate analysis of closed queueing networks," in *Mathematical Computer Performance and Reliability*, G. Iazeolla, P. J. Courtois, and A. Hordijk, Eds. Amsterdam, The Netherlands: North-Holland, 1984, pp. 225–244.
[49] M. C. Dewsnup, "Using graphics to build simulation models," in *Proc. 1983 Winter Simulation Conf.*, pp. 279–280.
[50] N. R. Draper and H. Smith, *Applied Regression Analysis (second edition)*, New York: Wiley, 1981.
[51] D. L. Eager, "Bounding algorithms for queueing network models of computer systems," Ph.D. dissertation, Dep. Comput. Sci., Univ. Toronto, Toronto, Ont., Canada, 1984.
[52] D. L. Eager and K. C. Sevcik, "Performance bound hierarchies for queueing networks," *ACM Trans. Comput. Syst.*, vol. 1, pp. 99–115, 1983.
[53] S. Eilon, "A simpler proof of $L = \lambda W$," *Oper. Res.*, vol. 17, pp. 915–917, 1969.
[54] A. K. Erlang, "The theory of probabilities and telephone conversations," *Nyt Tidsskrift Matematik*, vol. 20, pp. 33–39, 1909.
[55] G. Fayolle, R. Iasnogorodski, and I. Mitrani, "The distribution of sojourn times in a queueing network with overtaking: Reduction to a boundary problem," in *PERFORMANCE '83*, A. K. Agrawala and S. K. Tripathi, Eds. Amsterdam, The Netherlands: North-Holland, 1983, pp. 477–486.
[56] D. Ferrari, "Architecture and instrumentation of a modular interactive system," *Computer*, vol. 6, pp. 25–29, 1973.
[57] ——, *Computer Systems Performance Evaluation*, Englewood Cliffs, NJ: Prentice-Hall, 1978.
[58] ——, "A performance-oriented procedure for modeling interactive workloads," in *Experimental Computer Performance Evaluation*, D. Ferrari and M. Spadoni, Eds. Amsterdam, The Netherlands: North-Holland, 1981, pp. 57–78.
[59] ——, "On the foundations of artificial workload design," in *Perform. Eval. Rev. 12, Proc. 1984 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 8–13.
[60] D. Ferrari and V. Minetti, "A hybrid measurement tool for minicomputers," in *Experimental Computer Performance Evaluation*, D. Ferrari and M. Spadoni, Eds. Amsterdam, The Netherlands: North-Holland, 1981, pp. 217–233.
[61] D. Ferrari, G. Serazzi, and A. Zeigner, *Measurement and Tuning of Computer Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1983.
[62] G. S. Fishman, *Principles of Discrete Event Simulation*. New York: Wiley, 1978.
[63] W. R. Franta, H. K. Berg, and W. T. Wood, "Issues and approaches to distributed testbed instrumentation," *Computer*, vol. 15, pp. 71–81, 1982.
[64] H. Fromm, U. Hercksen, U. Herzog, K.-H. John, R. Klar, and W. Kleinoder, "Experiences with performance measurement and modeling of a processor array," *IEEE Trans. Comput.*, vol. C-32, pp. 15–31, Jan. 1983.
[65] S. H. Fuller, R. J. Swan, and W. A. Wulf, "The instrumentation of C.mmp, a multi-(mini) processor," in *Proc. COMPCON 73*, pp. 173–176.
[66] D. P. Gaver, "Diffusion approximations and models for certain congestion problems," *J. Appl. Prob.*, vol. 5, pp. 607–623, 1968.
[67] E. F. Gehringer, A. K. Jones, and Z. Z. Segall, "The Cm* testbed," *Computer*, vol. 15, pp. 40–53, 1982.
[68] E. Gelenbe, "On approximate computer system models," *J. Assoc. Comput. Mach.*, vol. 22, pp. 261–269, 1975.
[69] E. Gelenbe and I. Mitrani, *Analysis and Synthesis of Computer Systems*. New York: Academic, 1980.
[70] A. Goldberg, G. Popek, and S. S. Lavenberg, "A validated distributed system performance model," in *PERFORMANCE '83*, A. K. Agrawala and S. K. Tripathi, Eds. Amsterdam, The Netherlands: North-Holland, 1983, pp. 251–268.
[71] N. Goodman, R. Suri, and Y. C. Tay, "A simple analytic model for performance of exclusive locking in database systems," in *Proc. 2nd ACM Symp. Database Syst.*, 1983, pp. 203–215.
[72] W. J. Gordon and G. F. Newell, "Closed queueing systems with exponential servers," *Oper. Res.*, vol. 15, pp. 254–265, 1967.
[73] A. Goyal and T. Agerwala, "Performance analysis of future shared storage systems," *IBM J. Res. Develop.*, vol. 28, pp. 95–108, 1984.
[74] U. Grenander and R. F. Tsao, "Quantitative methods for evaluating computer system performance: A review and proposals," in *Statistical Computer Performance Evaluation*, W. Freiberger, Ed. New York: Academic, 1972, pp. 3–24.
[75] G. Haring, "On state-dependent workload characterization by software resources," in *Perform. Eval. Rev. 11, Proc. 1982 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 51–57.
[76] ——, "On stochastic models of interactive workloads," in *PERFORMANCE '83*, A. K. Agrawala and S. K. Tripathi, Eds. Amsterdam, The Netherlands: North-Holland, 1983, pp. 133–152.
[77] J. M. Harrison and M. I. Reiman, "On the distribution of multidimensional reflected Brownian motion," *SIAM J. Appl. Math.*, vol. 41, pp. 345–361, 1981.
[78] J. A. Hartigan, *Clustering Algorithms*. New York: Wiley, 1975.
[79] P. Heidelberger and P. A. W. Lewis, "Regression-adjusted estimates for regenerative simulations, with graphics," *Commun. ACM*, vol. 24, pp. 260–273, 1981.
[80] ——, "Quantile estimation in dependent sequences," *Oper. Res.*, vol. 32, pp. 185–209, 1984.
[81] P. Heidelberger and K. S. Trivedi, "Queueing network models for parallel processing with asynchronous tasks," *IEEE Trans. Comput.*, vol. C-31, pp. 1099–1108, 1982.
[82] ——, "Analytic queueing models for programs with internal concurrency," *IEEE Trans. Comput.*, vol. C-32, pp. 73–82, Jan. 1983.
[83] P. Heidelberger and P. D. Welch, "A spectral method for confidence interval generation and run length control in simulations," *Commun. ACM*, vol. 24, pp. 233–245, 1981.
[84] ——, "Simulation run length control in the presence of an initial transient," *Oper. Res.*, vol. 31, pp. 1109–1144, 1983.
[85] P. Heidelberger, P. D. Welch, and P. C. Yue, "Statistical analysis of database systems measurements," in *PERFORMANCE '81*, F. J. Kylstra, Ed. Amsterdam, The Netherlands: North-Holland, 1981, pp. 335–344.
[86] U. Hercksen, R. Klar, W. Kleinoder, and F. Kneissl, "Measuring simultaneous events in a multiprocessor system," in *Perform. Eval. Rev. 11, Proc. 1982 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 77–88.
[87] U. Herzog, W. Hoffmann, and W. Kleinoder, "Performance modeling and evaluation for hierarchically organized multiprocessor computer systems," in *Proc. 1979 Int. Conf. Parallel Processing*, pp. 103–114.
[88] Y. C. Ho and X. Cao, "Perturbation analysis and optimization of queueing networks," *J. Optimization Theory Applications*, vol. 40,

pp. 559–582, 1983.
[89] Y. C. Ho, X. Cao, and C. Cassandras, "Infinitesimal and finite perturbation analysis for queueing networks," *Automatica*, vol. 4, pp. 439–445, 1983.
[90] J. H. Hughes, "DIAMOND—A digital analyzer and monitoring device," in *Perform. Eval. Rev. 9, Proc. PERFORMANCE '80*, 1980, pp. 27–34.
[91] D. L. Iglehart, "Limiting diffusion approximations for the many server queue and the repairman problem," *J. Appl. Prob.*, vol. 2, pp. 429–441, 1965.
[92] ——, "Simulating stable stochastic systems, VI. Quantile estimation," *J. Assoc. Comput. Mach.*, vol. 23, pp. 347–360, 1976.
[93] ——, "The regenerative method for simulation analysis," in *Current Trends in Programming Methodology, Vol. III: Software Modeling*, K. M. Chandy and R. T. Yeh, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1978, pp. 52–71.
[94] D. L. Iglehart and W. Whitt, "Multiple channel queues in heavy traffic. I.," *Adv. Appl. Prob.*, vol. 2, pp. 150–177, 1970.
[95] ——, "Multiple channel queues in heavy traffic. II: Sequences, networks, and batches," *Adv. Appl. Prob.*, vol. 2, pp. 355–369, 1970.
[96] *PAWS/A User Guide*, Information Research Associates, Austin, TX, 1983.
[97] J. R. Jackson, "Jobshop-like queueing systems," *Mgmt. Sci.*, vol. 10, pp. 131–142, 1963.
[98] P. A. Jacobson and E. D. Lazowska, "Analyzing queueing networks with simultaneous resource procession," *Commun. ACM*, vol. 25, pp. 142–151, 1982.
[99] ——, "A reduction technique for evaluating queueing networks with serialization delays," in *PERFORMANCE '83*, A. K. Agrawala and S. K. Tripathi, Eds. Amsterdam, The Netherlands: North-Holland, 1983, pp. 45–59.
[100] A. K. Jones and P. Schwarz, "Experience using multiprocessor systems—A status report," *Comput. Surveys*, vol. 12, pp. 121–165, 1980.
[101] D. A. Jones, "Statistical investigation of factors affecting filestore performance," in *PERFORMANCE '81*, F. J. Kylstra, Ed. Amsterdam, The Netherlands: North-Holland, 1981, pp. 315–333.
[102] F. P. Kelly, *Reversibility and Stochastic Networks*. New York: Wiley, 1980.
[103] J. F. C. Kingman, "The heavy traffic approximation in the theory of queues," in *Proc. Symp. Congestion Theory*, W. Smith and W. Wilkinson, Eds. Chapel Hill, NC: Univ. North Carolina Press, 1965, pp. 137–159.
[104] J. P. C. Kleijnen, *Statistical Techniques in Simulation, Part I*. New York: Marcel Dekker, 1974.
[105] ——, *Statistical Techniques in Simulation, Part II*. New York: Marcel Dekker, 1975.
[106] L. Kleinrock, *Queueing Systems, Volume I, Theory*. New York: Wiley, 1975.
[107] ——, *Queueing Systems, Volume 2, Computer Applications*. New York: Wiley, 1976.
[108] D. E. Knuth, "An empirical study of FORTRAN programs," *Software—Practice and Experience*, vol. 1, pp. 105–133, 1971.
[109] H. Kobayashi, "Application of the diffusion approximation to queueing networks I: Equilibrium queue distributions," *J. Assoc. Comput. Mach.*, vol. 21, pp. 316–328, 1974.
[110] ——, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology*. Reading, MA: Addison-Wesley, 1978.
[111] P. Kritzinger, S. van Wyk, and A. Krzesinski, "A generalization of Norton's theorem for multiclass queueing networks," *Perform. Eval.*, vol. 2, pp. 98–107, 1982.
[112] B. Kumar and E. S. Davidson, "Performance evaluation of highly concurrent computers by deterministic simulation," *Commun. ACM*, vol. 21, pp. 904–913, 1978.
[113] S. S. Lam and Y. L. Lien, "A tree convolution algorithm for the solution of queueing networks," *Commun. ACM*, vol. 26, pp. 203–215, 1983.
[114] D. E. Lang, T. K. Agerwala, and K. M. Chandy, "A modeling approach and design tool for pipelined central processors," in *Proc. 5th Annu. Symp. Comput. Arch.*, 1979, vol. 7, pp. 122–129.
[115] G. Latouche, "Algorithmic analysis of a multiprogramming-multiprocessor computer system," *J. Assoc. Comput. Mach.*, vol. 28, pp. 662–679, 1981.
[116] S. S. Lavenberg, "Closed multichain queueing networks with large population sizes," in *Applied Probability—Computer Science: The Interface, Volume I*, R. L. Disney and T. J. Ott, Eds. Boston, MA: Birkhauser, 1982, pp. 219–249.
[117] *Computer Performance Modeling Handbook*, S. S. Lavenberg, Ed. New York: Academic, 1983.
[118] S. S. Lavenberg, "A simple analysis of exclusive and shared lock contention in a database system," in *Perform. Eval. Rev. 12, Proc. 1984 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 143–148.
[119] S. S. Lavenberg, T. L. Moeller, and P. D. Welch, "Statistical results on control variables with application to queuing network simulation," *Oper. Res.*, vol. 30, pp. 182–202, 1982.
[120] S. S. Lavenberg and M. Reiser, "Stationary state probabilities at arrival instants for closed queueing networks with multiple types of customers," *J. Appl. Prob.*, vol. 17, pp. 1048–1061, 1980.
[121] S. S. Lavenberg and C. H. Sauer, "Sequential stopping rules for the regenerative method of simulation," *IBM J. Res. Develop.*, vol. 21, pp. 545–558, 1977.
[122] S. S. Lavenberg and P. D. Welch, "A perspective on the use of control variables to increase the efficiency of Monte Carlo simulations," *Mgmt. Sci.*, vol. 27, pp. 322–335, 1981.
[123] A. M. Law, "Statistical analysis of simulation output data," *Oper. Res.*, vol. 31, pp. 983–1029, 1983.
[124] A. M. Law and J. S. Carson, "A sequential procedure for determining the length of a steady-state simulation," *Oper. Res.*, vol. 27, pp. 1011–1025, 1979.
[125] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. New York: McGraw-Hill, 1982.
[126] E. D. Lazowska and J. Zahorjan, "Multiple class memory constrained queueing networks," in *Perform. Eval. Rev. 11, Proc. 1982 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 130–140.
[127] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance—Computer System Analysis Using Queueing Network Models*. Englewood Cliffs, NJ: Prentice-Hall, 1984.
[128] J. D. C. Little, "A proof of the queueing formula, $L = \lambda W$," *Oper. Res.*, vol. 9, pp. 383–387, 1961.
[129] M. H. MacDougall, "Instruction-level program and processor modeling," *Computer*, vol. 17, pp. 14–24, 1984.
[130] S. A. Mamrak and P. D. Amer, "A feature selection tool for workload characterization," in *Proc. SIGMETRICS/CMG VIII*, 1977, pp. 113–120.
[131] B. H. Margolin, R. P. Parmelee, and M. Schatzoff, "Analysis of free storage algorithms," *IBM Syst. J.*, vol. 10, pp. 283–304, 1971.
[132] R. A. Marie, "An approximate analytical method for general queueing networks," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 530–538, 1979.
[133] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Syst. J.*, vol. 9, pp. 78–117, 1970.
[134] P. F. McGehearty, "Performance evaluation of a multiprocessor under interactive workloads," Ph.D. dissertation, Dep. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, 1980.
[135] J. McKenna and D. Mitra, "Integral representations and asymptotic expansions for closed Markovian queueing networks: Normal usage," *Bell Syst. Tech. J.*, vol. 61, pp. 661–683, 1982.
[136] ——, "Asymptotic expansions and integral representations of moments of queue lengths in closed Markovian networks," *J. Assoc. Comput. Mach.*, to be published.
[137] J. McKenna, D. Mitra, and K. G. Ramakrishnan, "A class of closed Markovian queueing networks: Integral representations, asymptotic expansions, and generalizations," *Bell Syst. Tech. J.*, vol. 60, pp. 599–641, 1981.
[138] H. Mechanic and W. McKay, "Confidence intervals for averages of dependent data in simulations II," IBM Corp., Yorktown Heights, NY, Tech. Rep. ASDD 17-202, 1966.
[139] D. J. Medeiros and J. T. Larkin, "Animation of output applied to manufacturing capacity analysis," in *Proc. 1983 Winter Simulation Conf.*, S. Roberts, J. Banks, and B. Schmeiser, Eds., pp. 283–286.
[140] D. Merle, D. Potier, and M. Veran, "A tool for computer system performance analysis," in *Performance of Computer Installations*, D. Ferrari, Ed. Amsterdam, The Netherlands: North-Holland, 1978, pp. 195–213.
[141] R. R. Miller, "Simulation and graphics on microcomputers," *Byte*, vol. 9, pp. 194–200, 1984.
[142] D. Mitra and P. J. Weinberger, "Probabilistic models of database locking: Solutions, computational algorithms and asymptotics," Bell Labs., Tech. Rep., 1984; also *J. Assoc. Comput. Mach.*, to be published.
[143] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms," *Comput. J.*, vol. 26, pp. 354–359, 1983.
[144] R. D. Nelson and B. R. Iyer, "Analysis of a replicated data base," IBM Corp., Yorktown Heights, NY, Res. Rep. RC 10148, 1983.
[145] M. F. Neuts, *Matrix-Geometric Solutions in Stochastic Models, An Algorithmic Approach*. Baltimore, MD: The Johns Hopkins Univ. Press, 1981.

[146] T. Nishigaki, "Experiments on the knee criterion in a multiprogrammed computer system," *IEEE Trans. Software Eng.*, vol. SE-9, pp. 79–86, 1983.
[147] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. New York: Academic, 1970.
[148] D. Potier and P. Leblanc, "Analysis of locking policies in database management systems," *Commun. ACM*, vol. 23, pp. 584–593, 1980.
[149] C. A. Pratt, "Going further," *Byte*, vol. 9, pp. 204–208, 1984.
[150] K. G. Ramakrishnan and D. Mitra, "An overview of PANACEA, A software package for analyzing Markovian queueing networks," *Bell Syst. Tech. J.*, vol. 61, pp. 2849–2872, 1982.
[151] M. I. Reiman, "The heavy traffic diffusion approximation for sojourn times in Jackson networks," in *Applied Probability—Computer Science, The Interface, Vol. II*, R. L. Disney and T. J. Ott, Eds. Boston, MA: Birkhauser, 1982, pp. 409–422.
[152] M. Reiser, "A queueing network analysis of computer communication networks with window flow control," *IEEE Trans. Commun.*, vol. COM-27, pp. 1199–1209, 1979.
[153] ——, "Mean value analysis and convolutional method for queue-dependent servers in closed queueing networks," *Perform. Eval.*, vol. 1, pp. 7–18, 1981.
[154] M. Reiser and H. Kobayashi, "Queueing networks with multiple closed chains: Theory and computational algorithms," *IBM J. Res. Develop.*, vol. 19, pp. 283–294, 1975.
[155] M. Reiser and S. S. Lavenberg, "Mean-value analysis of closed multichain queueing networks," *J. Assoc. Comput. Mach.*, vol. 27, pp. 313–322, 1980.
[156] C. A. Rose, "A measurement procedure for queueing network models of computer systems," *Comput. Surveys*, vol. 10, pp. 263–280, 1978.
[157] J. H. Saltzer and J. W. Gintell, "The instrumentation of Multics," *Commun. ACM*, vol. 13, pp. 495–500, 1970.
[158] S. Salza and S. S. Lavenberg, "Approximating response time distributions in closed queueing network models of computer performance," in *PERFORMANCE '81*, F. J. Kylstra, Ed. Amsterdam, The Netherlands: North-Holland, 1981, pp. 133–145.
[159] C. H. Sauer, "Approximate solution of queueing networks with simultaneous resource possession," *IBM J. Res. Develop.*, vol. 25, pp. 894–903, 1981.
[160] ——, "Computational algorithms for state-dependent queueing networks," *ACM Trans. Comput. Syst.*, vol. 1, pp. 67–92, 1983.
[161] C. H. Sauer and K. M. Chandy, "Approximate analysis of central server models," *IBM J. Res. Develop.*, vol. 19, pp. 301–313, 1975.
[162] ——, "Approximate solution of queueing models," *Computer*, vol. 13, pp. 25–32, 1980.
[163] ——, *Computer Systems Performance Modeling*. Englewood Cliffs, NJ: Prentice-Hall, 1981.
[164] C. H. Sauer and E. A. MacNair, *Simulation of Computer Communication Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1983.
[165] R. Schassberger and H. Daduna, "The time for a round trip in a cycle of exponential queues," *J. Assoc. Comput. Mach.*, vol. 30, pp. 146–150, 1983.
[166] M. Schatzoff, "Design of experiments in computer performance evaluation," *IBM J. Res. Develop.*, vol. 25, pp. 848–859, 1981.
[167] M. Schatzoff and C. C. Tillman, "Design of experiments in simulator validation," *IBM J. Res. Develop.*, vol. 19, pp. 252–262, 1975.
[168] L. W. Schruben, "Control of initialization bias in multivariate simulation response," *Commun. ACM*, vol. 24, pp. 246–252, 1981.
[169] ——, "Detecting initialization bias in simulation output," *Oper. Res.*, vol. 30, pp. 569–590, 1982.
[170] M. Schwartz, *Computer Communication Network Design and Analysis*. Englewood Cliffs, NJ: Prentice-Hall, 1977.
[171] P. Schweitzer, "Approximate analysis of multiclass closed networks of queues," in *Proc. Int. Conf. Stochastic Control and Optimization*, Amsterdam, The Netherlands, 1979.
[172] H. D. Schwetman and J. C. Browne, "An experimental study of computer system performance," in *Proc. ACM Nat. Conf.*, 1972, pp. 693–703.
[173] Z. Segall, A. Singh, R. T. Snodgrass, A. K. Jones, and D. P. Siewiorek, "An integrated instrumentation environment for multiprocessors," *IEEE Trans. Comput.*, vol. C-32, pp. 4–14, Jan. 1983.
[174] G. Serazzi, "A functional and resource-oriented procedure for workload modeling," in *PERFORMANCE '81*, F. J. Kylstra, Ed. Amsterdam, The Netherlands: North-Holland, 1981, pp. 345–361.
[175] K. C. Sevcik and I. Mitrani, "The distribution of queueing network states at input and output instants," *J. Assoc. Comput. Mach.*, vol. 28, pp. 358–371, 1981.
[176] J. G. Shanthikumar and R. G. Sargent, "A unifying view of hybrid simulation/analytic models and modeling," *Oper. Res.*, vol. 31, pp. 1030–1052, 1983.
[177] S. W. Sherman, F. Baskett, and J. C. Browne, "Trace-driven modeling and analysis of CPU scheduling in a multiprogramming system," *Commun. ACM*, vol. 15, pp. 1063–1069, 1972.
[178] A. Singh and Z. Segall, "Synthetic workload generation for experimentation with multiprocessors," in *Proc. 3rd Int. Conf. Distributed Computing Systems*, 1982, pp. 778–785.
[179] A. J. Smith, "Sequentiality and prefetching in database systems," *ACM Trans. Database Syst.*, vol. 3, pp. 223–247, 1978.
[180] ——, "On the effectiveness of buffered and multiple arm disks," in *Proc. 5th Annu. Symp. Comput. Arch.*, 1978, vol. 6, pp. 242–248.
[181] ——, "Long term file migration: Development and evaluation of algorithms," *Commun. ACM*, vol. 24, pp. 521–532, 1981.
[182] ——, "Cache memories," *Comput. Surveys*, vol. 14, pp. 473–530, 1982.
[183] J. E. Smith, "A study of branch prediction strategies," in *Proc. 5th Annu. Symp. Comput. Arch.*, 1981, vol. 9, pp. 135–148.
[184] K. Sreenivasan and A. J. Kleinman, "On the construction of a representative synthetic workload," *Commun. ACM*, vol. 17, pp. 127–133, 1974.
[185] V. P. Srini and J. F. Asenjo, "Analysis of Cray-1S architecture," in *Proc. 5th Annu. Symp. Comput. Arch.*, 1983, vol. 11, pp. 194–206.
[186] G. W. Stewart, "Computable error bounds for aggregated Markov chains," *J. Assoc. Comput. Mach.*, vol. 30, pp. 271–285, 1983.
[187] H. M. Stewart, "Performance analysis of complex communications systems," *IBM Syst. J.*, vol. 18, pp. 356–373, 1979.
[188] M. Stonebraker, J. Woodfill, J. Ranstrom, M. Murphy, M. Meyer, and E. Allman, "Performance enhancements to a relational database system," *ACM Trans. Database Syst.*, vol. 8, pp. 167–185, 1983.
[189] R. Suri, "Infinitestimal perturbation analysis of discrete event dynamic systems: A general theory," in *Proc. 22nd IEEE Conf. Decision and Control*, San Antonio, TX, 1983.
[190] ——, "Robustness of queueing network formulas," *J. Assoc. Comput. Mach.*, vol. 30, pp. 564–594, 1983.
[191] R. Suri and M. Zazanis, "Perturbation analysis gives strongly consistent estimates for an M/G/1 queue," Harvard Univ., Cambridge, MA, Tech. Rep., 1984.
[192] A. N. Tantawi and D. Towsley, "Optimal load balancing in distributed computer systems," IBM Corp., Yorktown Heights, NY, Res. Rep. RC 10346, 1984.
[193] Y. C. Tay, "A mean value performance model for locking in databases," Ph.D. dissertation, Div. Appl. Sci., Harvard Univ., Cambridge, MA, TR-04-84, 1984.
[194] A. Thomasian, "Queueing network models to estimate serialization delays in computer systems," in *PERFORMANCE '83*, A. K. Agrawala and S. K. Tripathi, Eds. Amsterdam, The Netherlands: North-Holland, 1983, pp. 61–81.
[195] A. Thomasian and I. K. Ryu, "A decomposition solution to the queueing network model of the centralized DBMS with static locking," in *Perform. Eval. Rev., Special Issue, Proc. 1983 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 82–92.
[196] D. F. Towsley, "Queueing network models with state-dependent routing," *J. Assoc. Comput. Mach.*, vol. 27, pp. 323–337, 1980.
[197] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[198] R. F. Tsao, L. W. Comeau, and B. H. Margolin, "A multi-factor paging experiment: I. The experiment and the conclusions," in *Statistical Computer Performance Evaluation*, W. Freiberger, Ed. New York: Academic, 1972, pp. 103–134.
[199] R. F. Tsao and B. H. Margolin, "A multi-factor paging experiment: II. Statistical methodology," in *Statistical Computer Performance Evaluation*, W. Freiberger, Ed. New York: Academic, 1972, pp. 135–158.
[200] S. Tucci and C. H. Sauer, "The tree MVA algorithm," IBM Corp., Yorktown Heights, NY, Res. Rep. RC 9338, 1982.
[201] J. Walrand and P. Varaiya, "Sojourn times and the overtaking condition in Jackson networks," *Adv. Appl. Prob.*, vol. 12, pp. 1000–1018, 1980.
[202] R. T. Wang and J. C. Browne, "Virtual machine-based simulation of distributed computing and networking," in *Perform. Eval. Rev. 10, Proc. 1981 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 154–156.
[203] D. L. Wyatt, S. Sheppard, and R. E. Young, "An experiment in microprocessor-based distributed digital simulation," in *Proc. 1983 Winter Simulation Conf.*, S. Roberts, J. Banks, and B. Schmeiser, Eds. pp. 271–277.
[204] J. Zahorjan, "The approximate solution of large queueing network models," Ph.D. dissertation, Comput. Syst. Res. Group, Univ. Toronto, Toronto, Ont., Canada, Technical Report CSRG-122, 1980.

[205] ——, "Workload representations in queueing models of computer systems," in *Perform. Eval. Rev., Special Issue, Proc. 1983 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, pp. 70–81.