Approximate Methods for Analyzing Queueing Network Models of Computing Systems

K. MANI CHANDY
CHARLES H. SAUER

Computer Sciences Department, University of Texas at Austin, Austin, Texas 78712

The two primary issues in choosing a computing system model are credibility of the model and cost of developing and solving the model Credibility is determined by 1) the experience and biases of the persons using the model, 2) the extent to which the model represents system features, and 3) the accuracy of the solution technique. Queueing network models are widely used because they have proven effective and are inexpensive to solve. However, most queueing network models make strong assumptions to assure an exact numerical solution. When such assumptions severely affect credibility, simulation or other approaches are used, in spite of their relatively high cost. It is the contention of this paper that queueing network models with credible assumptions can be solved approximately to provide credible performance estimates at low cost This contention is supported by examples of approximate solutions of queueing network models. Two major approaches to approximate solution, aggregation (decomposition) and diffusion, are discussed

Keywords and Phrases: performance evaluation, queueing networks, approximate solutions, hierarchical modeling

CR Categories: 3.81, 3.89, 4.32, 4 6, 6.20, 8 1

INTRODUCTION

Computer systems analysts use models to gain insight into the behavior of systems and to aid in systems design. The analyst has several tools (programs) at his disposal to aid in estimating the performance of systems. These include queueing models, random-number or trace-driven simulations, and statistical models. The specific tool that an analyst uses depends upon the amount of error he can tolerate, the speed with which he wishes to get results, and perhaps most importantly, the amount of faith he has in the tool. In our experience computer center managers and systems analysts generally rank tools in increasing order of credibility in the following sequence: queueing models, discrete-event random number simulations, trace-driven simulations, monitoring systems running synthetic jobs, and measurements of a real workload on a real system. Unfortunately, the above ranking is generally also a ranking of tools in increasing order of cost (i.e., time used by the analyst and time used by the solution technique).

We will frame our discussion of our approximations in the context of analysts choosing between competing predictive tools, focusing attention on the three criteria of: 1) time used by the solution technique (speed), 2) credibility, and 3) degree of accuracy required for the problem at hand. Credibility is a subjective criterion; it varies from analyst to analyst. Yet it is crucial to the understanding of how tools are used. In particular, it is fundamental to

CONTENTS

the basic issue of this paper: How can approximation methods be commonly used by systems analysts in the field?

In keeping with most of the queueing network literature, we adopt the stochastic approach to approximation methods. All the systems considered in this paper are assumed to attain equilibrium; furthermore, we restrict attention (with a few exceptions) to predicting the performance of systems at equilibrium. In our experience systems analysts rarely use tools to predict transient behavior; however, this situation may change as tools become increasingly flexible.

By *analyzing a model* we mean running a computer program that estimates queue-length distributions for one or more queues in the model. Device utilizations, mean response times, and throughputs may be derived from the queue length distribution. We usually ignore the problem of computing response time distributions. The problems of estimating response time distributions and their moments other than the mean are very difficult. The successful solutions have been limited to systems consisting of a single queue or a very restricted network of queues.

We generally ignore the software aspects of queueing network modeling, even though software is of critical importance. Rarely will an analyst implement a special purpose program for a specific model. Rather, the analyst will rely on existing software, perhaps adding modifications or a specialized user interface. Reiser and Sauer discuss queueing network software with emphasis on solution techniques [REIS78]. Sauer and MacNair survey significant queueing software with a comprehensive view of the requirements specifically associated with queueing software [SAUE78]. In this paper we only discuss software of special significance.

We put queueing network models into three categories:

1) Tractable: those which can be analyzed to give exact (as opposed to approximate) solutions in an "adequately" short time. ("Adequately" is a subjective term that will be defined later in this section.) It should be em-

phasized that tractable models often do not represent reality faithfully; the analyst should not be lulled into false security by the exactness of the solution.

2) Intolerably slow: those which can be analyzed to get exact results but which cannot be analyzed in an adequately short time.
3) Unsolved: those for which there is no known method of analysis guaranteed to give exact results.

The vast majority of queueing network models used for estimating and predicting computer system performance are of the tractable category. Rather than attempting to use a queueing network model of the second or third categories, most analysts would resort to simulation. However, constructing a simulation model with a conventional language may require a large amount of effort, and running a simulation may be computationally expensive. Thus there is a significant cost gap between queueing models and simulation models in the credibility sequence described above.

There have been three major approaches to bridging this gap:

1) Using approximate solution techniques;
2) Using a simple, tractable model to obtain bounds for performance measures of a more complex model; and
3) Using simulation tools *specifically designed* for solution of complex queueing models.

The following example will help to clarify these terms and approaches. Consider a model of a CDC 6000 series machine; this model consists of a central server with a queue for Peripheral Processors (PPs) ahead of the disk queues (Figure 1). To carry out an I/O operation, a job needs both a peripheral processor and a disk. When a job completes a CPU service it joins the PP queue. Only after securing a PP does the job enter one of the disk queues. The PP is held while the job is waiting in the disk queue and while the job is being serviced by the disk. After completion of disk service the job relinquishes its PP; the relinquished PP is immediately assigned to a job in the PP queue, or, if the PP queue is empty, the relinquished PP joins the pool of available PPs.

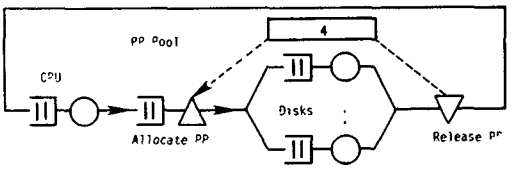


FIGURE 1. Central server model with peripheral processors.

Consider a system with five disks, four PPs, and a level of multiprogramming of five. Assume that the PP and disk queues have a first come first served (FCFS) queueing discipline and that the CPU has a processor sharing (PS) discipline. (PS is defined as the limiting case of a no overhead round-robin discipline as the quantum goes to zero. It is used in queueing models because it is much more mathematically tractable than round-robin.) Assume the CPU service time distribution is hyperexponential and that the I/O service time distributions are exponential. (Service time distributions have a key impact on the tractability of queueing models. In Section 1 we will discuss distribution representations for those readers unfamiliar with this subject.)

This model can be represented as a Markov process and an exact numerical solution can be obtained [KELL76]. (Markov processes are essential to the stochastic approach to queueing networks. This is especially true of exact solutions and specialized simulation techniques and less true of the aspects of approximation we discuss here. Section 1 gives a brief introduction to Markov processes.) However, the time required to obtain this solution on a CDC 6600 is in the order of several minutes. We must conclude that this model falls into the category of *intolerably slow* models because most analysts would prefer to use alternative methods or models. If the service time distributions were arbitrary, and thus less tractable than hyperexponential and exponential forms, then this model would fall into the category of *unsolved* models.

Approximate solution methods can be applied to the 4 PP model [KELL76]. There is no assurance, besides empirical results, that an approximation will give satisfactory

answers, but an approximation may give values very close to the exact values in less than a second of processor time. Thus the approximate solution is very attractive. Approximations usually use *heuristic* solution techniques. The philosophy is very similar to that of artificial intelligence search strategies: The methods are reasonable but, in general, it cannot be rigorously proven that the methods provide the desired solutions.

Consider this model with five rather than four PPs. Since the degree of multiprogramming is equal to the number of PPs, a job will never wait for a PP. Hence we can remove PPs from the model. The resulting model can be analyzed in less than a second on a CDC 6600, since the model satisfies product form (Section 1). This model falls into the *tractable* category.

It is possible to approximate the 4 PP system by a 5 PP tractable model because we know from experience that the 5 PP model provides a tight upper bound to the throughput of the 4 PP system. This is an instance of a complex model being analyzed by obtaining bounds for performance measures from a simpler model [SEVC77a]. Such an approach is extremely useful. However, we do not consider such approaches in this paper because a *general* theory underlying such approaches is yet to be developed; cases have to be treated individually. A general theory in this area would be a breakthrough in systems modeling. One must proceed cautiously; for instance, it is tempting to hypothesize that systems with more regular arrival processes (i.e., processes with smaller coefficients of variation of interarrival times, where the coefficient of variation is the standard deviation divided by the mean) may be analyzed to obtain lower bounds on the mean wait times of systems with more variable arrival processes, but this hypothesis is invalid [WOLF77].

Queueing network tools such as QSIM [FOST74] and RESQ [REIS78, SAUE77c] allow the user to easily specify and simulate complex queueing models of the intolerably slow and unsolved categories. (RESQ also provides numerical solutions for most models of the tractable category.) A simulation of the 4 PP model above would require considerably less than a minute to obtain satisfactory results. Thus simulation would be a more reasonable method than exact numerical solution of that model.

Thus one can conclude that both approximation and simulation are viable approaches to solving this model. Though the approximation has a definite computational advantage, one must consider factors such as credibility and availability of software. For other models we may not have a choice; only one technique is viable. Our object here is to survey approximation methods for intolerably slow and unsolved models. There is no known rule for choosing between simulation and approximation methods, nor is there one for choosing among approximations. However, we feel it is important to bring approximations to the attention of a larger audience.

Before proceeding with approximations we should say a little about simulation. Simulations could be considered an approximation technique, since simulations do *not* provide exact results, but this is an unusual view. To clarify this point, consider a simulation of a queueing network that has a tractable numerical solution. The numerical solution is *exact*—if a mean response time is estimated, then that value is correct for the queueing network (though not necessarily correct for the modeled computer system). A simulation estimate of mean response time will, hopefully, be *near* the correct value but will usually not be equal to the correct value. Two of the principal problems with simulation are: determining how close simulation estimates are to the correct values (for the model, not the modeled system), and determining how long to run the simulation in order to obtain estimates near the correct values. These problems are still difficult for arbitrary simulations, but much progress has been made toward their solution for *simulations of queueing networks.*

A general approach to the first problem is the estimation of confidence intervals. If a simulation run produces an estimate R for *mean* response time, then we produce a confidence interval estimate $[R - \delta, R + \delta]$ and say that the correct *mean* response time is in this interval with a certain

level of confidence, e.g., 90%. Suppose we run the same simulation model many times with different random number streams for each run, and estimate confidence intervals for mean response time on each run. Informally, we could expect the fraction of runs with confidence interval including the correct value, to be equal to the level of confidence. Note that we are *not* saying that 90% (or whatever the level of confidence) of the response times are in the interval; this is a common misinterpretation of the confidence interval.

Though estimation of confidence intervals is difficult for arbitrary simulations, much progress has been made in rigorous estimation of confidence intervals for stochastic systems in equilibrium, e.g., queueing networks. The most rigorous approach is the *regenerative method.* Lavenberg and Slutz give an excellent introduction to this method [LAVE75], Iglehart gives a thorough survey of it [IGLE78], and Sauer and MacNair illustrate application of the method to general queueing networks [SAUE77c]. The regenerative method is not always practical; Kobayashi [KOBA78] and Sauer [SAUE77a] discuss alternative methods for queueing models.

The second problem can be handled by *sequential stopping rules* applied in conjunction with confidence interval estimation [LAVE77]. In brief, a sequential stopping rule allows the simulation to run for a while (a "sampling period") and then examines the confidence intervals. If the intervals are "sufficiently narrow" then the simulation stops. Otherwise, additional sampling periods are run, with confidence intervals estimated after each period, until the intervals are sufficiently narrow. APLOMB, the simulation component of RESQ, provides several confidence interval methods and a sequential stopping rule.

The remainder of the paper discusses approximations. Section 1 informally describes the tractable category of models. Section 2 give examples of system models that are not in the tractable category but have been solved by approximations. Section 3 describes approximations based on aggregation of submodel results. Section 4 considers techniques for improving the accuracy of aggregation approximations. Section 5 briefly discusses use of aggregation in the management of modeling projects. Section 6 describes diffusion approximations. Section 7 is an overview of approximation techniques.

1. TRACTABLE QUEUEING NETWORK MODELS

Before giving a characterization of tractable models we informally review the concepts of arrival processes, service time distributions, and Markov processes. A more thorough introduction is given by Drake [DRAK67].

Mathematicians have tried to make the representation of queueing systems as simple as possible for the sake of tractability. They then studied more complex queueing systems because they are more representative of actual systems, and because they are more mathematically interesting. To keep our discussion simple we will avoid unnecessary mathematics and therefore sacrifice some rigor and general applicability.

First consider the arrival of jobs at a queue, a queue not connected to other queues. The simplest representation of the arrivals assumes that the arrivals are strictly independent of each other. To be more precise, we define a *Poisson process* for the arrivals: 1) observations of the arrivals during non-overlapping intervals of time are mutually independent; and 2) if we look at small enough intervals of time there will be at most one arrival per interval, and the probability of an arrival during the interval is equal to the overall rate of arrival times the length of the interval. The advantage of the Poisson representation of arrivals is that it is *memoryless* because of the first part of the definition; the probability of an arrival during a sufficiently small interval of time is totally independent of arrivals or lack of arrivals in the preceding intervals. The Poisson process is often a realistic representation of actual arrival processes but is usually chosen for queueing models because of the resulting tractability. Corresponding to the process of arrivals during a time interval, there is a distribu-

tion of times between arrivals. For the Poisson process the corresponding interarrival time distribution is the exponential distribution; "Poisson arrivals" and "independent identically distributed exponential arrivals" are equivalent. The exponential distribution also has a memoryless property; the probability distribution for the time until the next arrival is independent of the time elapsed since the last arrival. Statements about arrivals and interarrival times can be applied to service completions and interdeparture times, respectively.

Now consider a single isolated queue with FCFS discipline. It is not reasonable to describe the queue as a whole as memoryless; even if the arrival and service processes are memoryless, the response time a job experiences will be dependent on the number of other jobs ahead of it. A *Markov process* has a limited amount of memory. This memory consists of distinct disjoint states. The time the process spends in a state is exponential; the memory consists only of the state distinctions. (We are considering continuous time processes only, as is usually the case. Analogous statements can be made if time is represented as discrete units.)

For the queue we have just described, the Markov states are uniquely determined by the number of jobs at the queue. The queue changes from one state to another when a job arrives or departs. To obtain a solution for this process means to solve for the equilibrium probabilities of each state. This is done by solving a set of linear equations (balance equations) equating the rate at which the process leaves each state to the rate at which it enters the state. (A distinction is sometimes made between *global* and *local* balance equations. The equations just described are the global balance equations. Local balance equations ignore some of the terms of the global balance equations without affecting the solution. This is only possible for special kinds of Markov processes.)

Before returning to queueing networks, let us consider generalization of service time distributions. The same approach may be used with interarrival time distributions, but this is not often done with queueing networks. In addition to the memoryless property, the exponential distribution also has the characteristic that the standard deviation is equal to the mean. If we want to represent a distribution with standard deviation greater than the mean but still retain as little memory as possible, we can use the hyperexponential form of Figure 2. An individual service time will be exponentially distributed with mean S_1 with probability p_1 and exponentially distributed with mean S_2 with probability p_2. The figure represents a special case of the hyperexponential distribution; it will have coefficient of variation greater than one if $p_1 \neq p_2$. S is the mean service time.

On the other hand, if we want to represent a distribution with coefficient of variation less than one, we can use the hypoexponential form of Figure 3. In this case a service time consists of the sum of several exponential service times. Both the hyperexponential and hypoexponential forms are special cases of the *method of* (exponential) *stages* [Cox55]. The method of stages allows us to closely represent most service time distributions with the only memory introduced being the distribution stage of a service time in progress. For example, if we substitute a hyperexponential service time in our single queue example, then we distinguish Markov states of the process by the distribution stage of the job in service as well as by the number of jobs at the queue. For a discussion of equivalence of various forms of the method of stages, problems with low coefficient of variation, and a heu-

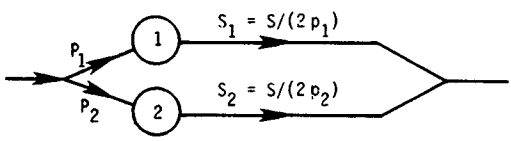


FIGURE 2. Hyperexponential form.

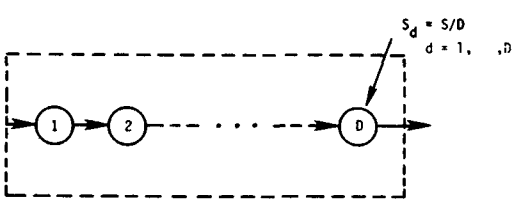


FIGURE 3. Hypoexponential form.

ristic solution to those problems see [SAUE75a]. For most of the remainder of the paper we assume time distributions represented by the method of stages.

Having discussed the necessary background, we now characterize tractable queueing networks. A queueing network will have a tractable solution if one or more of the following conditions are met:

1) State Space Size: The balance equations can be mechanically generated and numerically solved in an adequately short amount of time.
2) State Transition Structure: The state transitions are such that recursive techniques may be used to obtain the probabilities of a few states, and then the queue length distributions can be expressed in terms of these states [Herz75, SAUE75a].
3) Product Form: The equilibrium state probability distribution consists of factors representing the states of the individual queues, i.e., $P(S_1, \ldots, S_K) = (1/G)\, P(S_1) \ldots P(S_K)$ where $P(S_i)$ is the probability that the ith queue is in state S_i and $P(S_1, \ldots, S_K)$ is the probability that the network is in state $(S_1, \ldots, S_K)$. (G is a normalizing constant chosen so that the probabilities sum to one.)

Though in all three cases we can obtain performance measures such as throughput from the state probabilities, this is usually computationally inappropriate. For example, throughput can be obtained directly from normalizing constants without explicit consideration of state probabilities [DENN78a]. (See pp. 225-261, this issue.) With each of these conditions one can associate programs written to solve networks meeting that condition.

The best known program corresponding to the first condition is RQA [WALL66]. Experience with this program indicates that it is useful with models having not more than a few thousand states. Unfortunately, many models will have much larger, perhaps infinite, state spaces. Though there have been recent results on generating balance equations [GAVE76] and on numerical methods for this problem [STEW78], there is no indication that significantly larger state spaces will be accommodated. Note that the generality of the technique is limited only by algorithm complexity and machine resources.

The method of [HERZ75] corresponding to the second condition has only been applied to cyclic queue models (Figure 4) consisting of two queues with a fixed population of jobs. This technique is attractive because exact answers may be obtained rapidly for a variety of interesting queueing disciplines [SAUE75a]. A drawback is that a different procedure is required for each combination of queueing disciplines in the recursive method (as opposed to RQA). On the whole, this method is beneficial and is the basis of some of the approximations in Section 3. It has also been used directly in the parametric analysis of multiprocessing systems [SAUE77b].

The most important tractable models are those corresponding to the third condition, i.e., those having a product form solution. The exponential networks of [GORD67 and JACK63] have such a solution. Recent efforts [BASK75, CHAN72, DENN78, REIS75] have shown that many other networks have a product form solution. Programs such as ASQ [KELL73] and RESQ [REIS78] can analyze large product form networks in an adequately short time. In the remainder of this section we describe an important subset of the networks with product form solutions. The reader should be aware that this description is incomplete and should refer to the cited original works for formal and complete descriptions.

A queueing network of this subset consists of a set of jobs, a set of queues, an infinite source of jobs, a sink, and a set of routing rules. The jobs are partitioned into groups called "routing chains," or simply, "chains," by the routing rules. All jobs of a

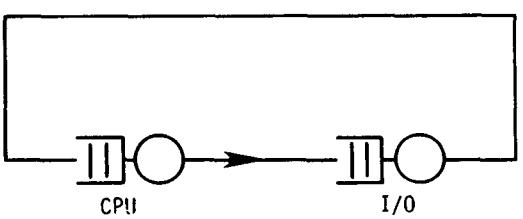


FIGURE 4. Cyclic queue model.

given chain are identical in their behavior. A chain is *open* if it is connected to the source and sink, and *closed* if it is not connected to these. (Thus a closed chain has a fixed population of jobs.) A network is *mixed* if it has both kinds of chains. The arrival process of jobs from the source is Poisson.

A queue consists of a set of devices, a queueing discipline, and a set of "local" classes. Two alternate, functionally equivalent representations of job classes have appeared in the literature. One considers classes to be global in the sense that a job belongs to that class unless it explicitly changes class. The other approach, the one we adopt, consists of classes local to queues and explicitly partitioned by routing chains. This representation facilitates implementation of solution packages [REIS78] and is often more convenient for model formulation. Global classes must also be partitioned into chains for non-simulation solutions; this is often ignored.

We will use Figure 5 to illustrate the concept of chains, local classes, and global classes. This figure shows three queues and four local classes. The four classes form a single routing chain because a job of one class will eventually join any other class. If we represented this network using global classes, we would need two global classes. For example, let global class A consist of local classes 1 and 2 and let global class B consist of local classes 3 and 4. When a job leaves device 1 it changes from (global) class A to class B; a job leaving device 2 changes from class B to class A.

If a job of a certain chain is to be routed to a given queue, then that queue must have one or more classes associated with the chain. The different classes of a queue may have distinct routing and, under conditions described below, may have distinct service time distributions. However, different priorities may not be assigned to the classes if a network is to retain a product form solution.

Consider the example of Figures 6, 7, and 8. They represent a computer system serving a set of terminals and also supporting a

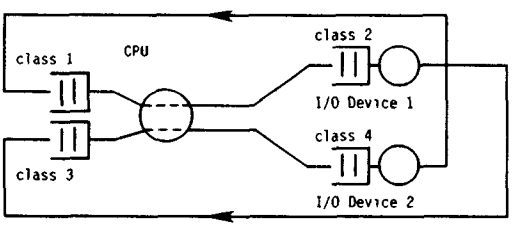


FIGURE 5. Single routing chain with four classes.

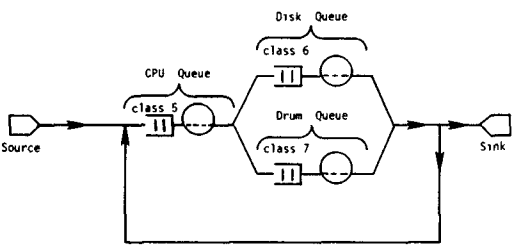


FIGURE 7 Open chain (batch load).

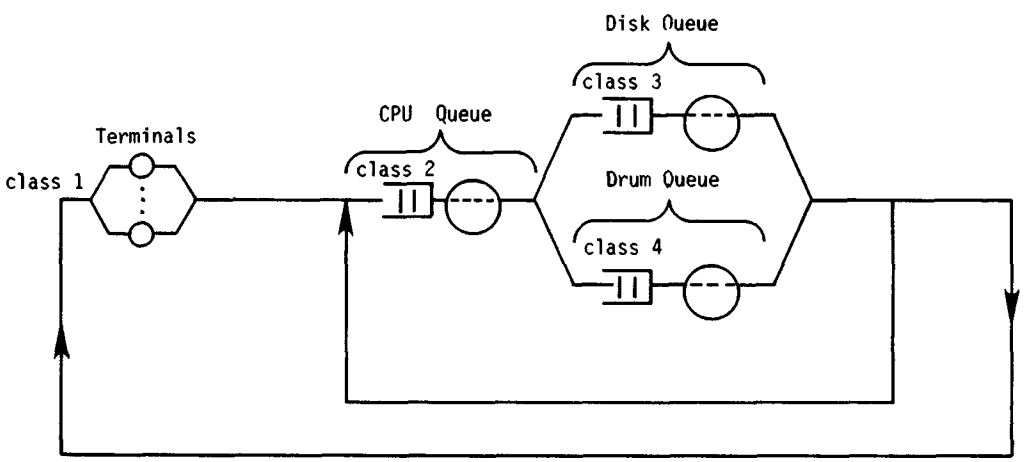


FIGURE 6. Closed chain (terminal load).

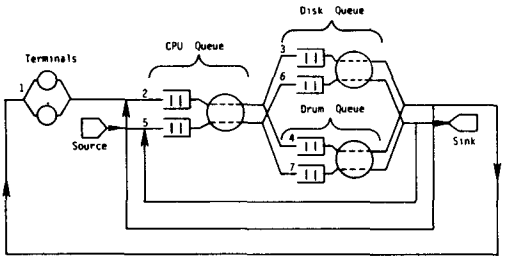


FIGURE 8. Central server model with terminal and batch loads.

batch workload. There are two routing chains: a closed chain representing the interactive load and consisting of classes 1 through 4, and an open chain representing the batch load and consisting of classes 5, 6, and 7. Assume the closed chain population is equal to the number of terminals. As this example illustrates, we consider the routing to be from class to class rather than from queue to queue. Where alternate routings are possible we must specify the relative frequency associated with each path.

For a certain class of queueing disciplines, which we shall call *product form disciplines* [CHAN77], different classes of a queue may have different service time distributions. The two most common instances of product form disciplines are 1) processor sharing, and 2) when the number of jobs at the device never exceeds the number of devices (the "infinite server" queue). The service time distributions at queues with product form disciplines may be non-exponential.

If a queue has a FCFS queueing discipline (and fewer devices than the possible queue length) then all classes of the queue must have the same exponential service time distribution. The service times may be "load (queue length) dependent.

2. EXAMPLES OF SYSTEMS THAT ARE ANALYZED VIA APPROXIMATION

We have attempted to motivate the reader to consider the use of approximations (Introduction) and we have outlined the class of models that can be analyzed without resorting to approximations. Models that require the use of approximation techniques usually represent reality more faithfully than models that do not. The question is—which is preferable: approximate solutions to relatively faithful models, or exact solutions to relatively unfaithful (tractable) models? The question cannot be answered without empirical evidence. However, if a model ignores the existence of a resource, that model cannot be used to design or schedule the ignored resource. For example, if peripheral processors are ignored in a central server model, we cannot use that model to determine the effect of the number of PPs on performance, and then approximations, simulation, or measurement are the only alternatives.

In this section we discuss *problems that force the use of approximations,* because tractable models fail to represent key aspects of these problems. Our goal in this section is to introduce the reader to a wide variety of such problems and to the literature describing successful approximate solutions to these problems.

Distributions and Disciplines

If service time distributions in a model have high coefficients of variation, and if the corresponding queueing disciplines are not product form disciplines (for example, FCFS), then the values of performance measures obtained from the model may be substantially different from values obtained assuming product form [SAUE75a]. For example, consider a cyclic queue model (Figure 4) with a single CPU, a single I/O device, and three jobs. If the mean CPU service is 40 ms., the coefficient of variation of CPU service time 5, and the mean I/O service time 40 ms. with an exponential distribution, then the CPU utilization with FCFS queueing at the CPU will be 64.9%. If we assume a product form CPU queueing discipline, e.g., PS, then the utilization will be 75.0%—a relative error of 15.6%.

Priority queueing disciplines usually give significantly different results than FCFS and PS. Networks of queues with priority disciplines have been solved approximately [CHAN75b, REIS76, SAUE75b, SEVC77a].

Multiple Resource Holding

A job may hold more than one resource at a time as in the peripheral processor ex-

ample. In the example in the Introduction, a PP does not have a service time associated with it; the length of time that a PP is held depends upon other devices such as disks. Main memory is another example of an important resource that does not have a service time associated with it, though jobs need main memory to use other devices such as the CPU. Resources that do not have service times associated with them, but limit the population of jobs that may utilize other devices, are called *passive resources* [FOST74]. Busses and switches in multiprocessing architectures are examples of critical resources that are passive. Models with passive resources have been analyzed approximately [BROW77, BROW75, DENN76, KELL76, SEKI71].

Blocking

Most models with passive resources allow an arbitrarily long queue for the passive resource. Thus, the passive resource serves only to limit the population of jobs in certain systems; for example PPs limit the population of jobs in the disk subsystem. It is usually assumed that a device will serve jobs whenever there are jobs in that device's queue. However, in models of packet or message-switching systems, a device may be *blocked* (Figure 9), i.e., prevented from serving "jobs" in its queue because a queue elsewhere in the network is full to its capacity and cannot accept any more jobs. In some communication systems a job attempting to enter a queue that is filled to capacity may be lost, i.e., disappear from the system. In some systems a device may expect an acknowledgment that a job (packet) has safely entered the next queue; if it does not get an acknowledgment in a specified period of time it may re-serve the job, i.e., retransmit the message. Queueing network models with blocking, lost messages, or acknowledgments do not satisfy product form. However, such models have been analyzed approximately [IRLA75, LAM76].

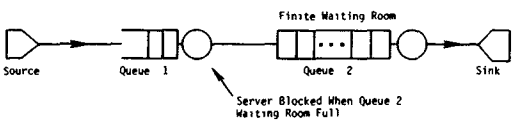


FIGURE 9 Blocking.

Schedulers

In many systems the processes that act as schedulers are not always active. Thus a job waiting for memory may have to wait, after memory becomes available, until the memory scheduler is activated.

The length of time that a job must wait for schedulers to become activated to allocate resources to it may comprise a significant part of the overall delay experienced by the job, especially in systems with a significant degree of process communication. Schedulers are clearly a scarce resource in this context. However, they are different from other resources such as CPUs and disks because 1) a scheduler is a program that requires the CPU to run, and hence a job needs two resources to get service: the CPU and the scheduler, and 2) a scheduler, once it is activated, may serve several jobs in a relatively short time, in which case it is incorrect to model the service time of a scheduler as the length of time between scheduler activations, whereas it is equally incorrect to model the service time as the length of time required for the scheduler to handle a single job after it has been activated. Systems with schedulers have been modeled approximately [BROW77].

Parallelism: Forks and Joins

Some systems allow jobs to fork, spawning new tasks that can be processed in parallel. For instance, CPU:I/O overlapped processing has been modeled by means of precedence graphs showing forks and joins [TOWS75]. In the CPU:I/O overlap model, a job is assumed to make several cycles where each cycle is modeled as a precedence graph (Figure 10). Each cycle in this example consists of three tasks; at the start of the cycle a CPU task must be performed. This task might set up the buffers for an I/O operation. After this task has been completed the job forks, creating two tasks that may proceed in parallel. One of the tasks requires an I/O device while the other requires the CPU. Only after all the tasks

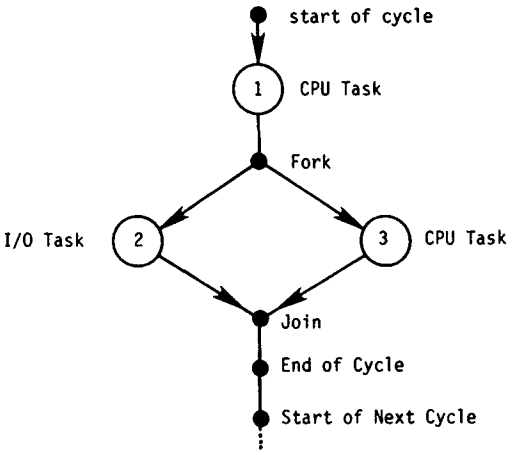


FIGURE 10. Precedence graph for CPU. I/O overlap model.

in a cycle have been completed can the next cycle be initiated. This model does not satisfy product form. Relatively complex overlap models have been solved by approximation methods [TOWS75]. Similar problems appear in communication networks [SAUE77c].

Routing

Most queueing network models assume that the frequency at which a job will join class j after leaving class i is a constant q_{ij}, independent of the state of the system. However, there are systems in which the route that a job takes through a network of queues is designed to depend upon the state of the system. For instance, a job requiring the use of a computing system in a multicomputer network may be allowed to use any one of a pool of computers. In this case a reasonable scheduling policy is to direct the job towards that computer with the least expected delay; thus the job's path depends upon the relative congestion at different computers. Such load balancing models do not generally satisfy product form.

However, Towsley has defined a class of load balancing strategies for closed networks that do satisfy product form [TOWS75]. Foschini has studied load balancing (also called dynamic routing) by means of diffusion approximations [FOSC77]. A general approach to approximation methods for dynamic job routing is yet to be devised.

Summary

The above problems are some of the characteristics that preclude tractable solutions. They are discussed in roughly descending order of importance, based on the amount of attention each has received in the computing literature. We do not believe a strict ordering can be given; importance of these characteristics varies from system to system. For example, it is generally assumed that CPU:I/O overlap is not significant, but individual systems may have a high degree of overlap, sufficient to make overlap more important than distributions, disciplines, or multiple resource holding.

3. AGGREGATION (DECOMPOSITION) APPROXIMATIONS

Perhaps the most important approximation strategy in queueing networks is that of *aggregation*: one solves portions of the model in isolation ("offline") and gathers the results together ("online") to produce a solution of the whole model. This approach has also been referred to as *decomposition*. One can view the strategy alternately as decomposition of the whole model or as aggregation of portions of the model. The choice of terminology is primarily a matter of emphasis and personal taste. Similar methods have been used for a long time in combinatorics, systems theory, and artificial intelligence. Indeed, the motivation for such methods in queueing networks came from similar approaches in general systems theory and in electrical networks.

These methods can be shown to give exact solutions for queueing networks with product form solutions [CHAN75a]. Though aggregation is computationally advantageous in parametric analysis of product form networks [CHAN75a], our interest will be in networks without tractable solutions. In these cases the strategy *will cause some error* because the aggregation process does not totally capture the interaction between the individual portions. The bulk of this section will be concerned with the representation of interfaces between portions.

There is another very important justification for the aggregation strategy [COUR75, COUR77]: loose coupling of subnetworks. This is in addition to the justification based on results for product form networks. We will discuss the loosely coupled justification later in this section.

Flow-Equivalent Methods

A basic approach is to replace a subnetwork of queues by a single ("composite") queue which is *flow-equivalent* to the subnetwork, i.e., the job flow through the composite queue is equal to the job flow through the subnetwork. This can be done repeatedly, replacing subnetworks (including those with composite queues) by flow-equivalents until the solution of the resulting network is tractable. There may be additional constraints on the aggregation process, e.g., if we need to find performances measures for a particular queue, this queue should appear explicitly in the network after aggregation.

Determination of the flow-equivalent is not obvious. In this section we will discuss exact flow-equivalents and approximations based on variations on exact flow-equivalents proposed for other networks. In Section 4 we discuss methods for estimating the amount of error introduced and taking appropriate corrective action.

We assume, for the time being, that the subnetwork has a single routing chain and that there is a single input path to the subnetwork and a single output path from the subnetwork. We primarily discuss closed networks since most of the research and application of flow-equivalent methods has concerned closed networks. (It should be pointed out that flow-equivalents also apply to open and mixed networks.)

First we must determine the mean service time for the composite queue. One obvious estimate would be the expected time spent in service in the subnetwork. However, this estimate may be much too low if all jobs in the composite queue receive parallel service; if only one composite queue job receives service the estimate is high if several jobs may be in service simultaneously in the subnetwork. Thus we must let the composite queue service times be *load dependent* (dependent on the queue length of the composite queue).

A method for determining composite queue service times that gives exact results for product form networks [CHAN75a] is the following: Consider the subnetwork with the output fed back to the input, i.e., consider the subnetwork offline. Determine the throughput, $X(n)$, along this feedback path for each possible job population size n in the subnetwork. Set $S(n)$, the mean service time of the composite queue given n jobs in the queue, to $1/X(n)$. In other words, if the network satisfies product form, then the online behavior of the subnetwork is identical to its offline behavior.

Passive Queue Flow-Equivalents

To illustrate an *approximation* using a flow-equivalent method consider the 4 PP example of the Introduction. Let the subnetwork be the PP and disk queues (Figure 11). From [CHAN75a, DENN78] we know that $X(n)$, $n = 1,2,3,4$ will be $G(n\text{-}1)/G(n)$, ignoring the PP queue. Further $X(5)$ must be the same as $X(4)$ since there will never be more than 4 jobs in the disk queues. Replacing a subnetwork with the load-dependent composite queue (Figure 12) results in a network that satisfies product form and is easily solved. Since the 4 PP model violates product form conditions, we

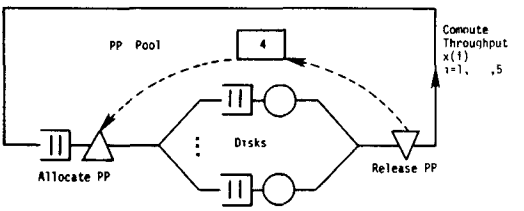


FIGURE 11 PP/disk subnetwork with output connected to input.

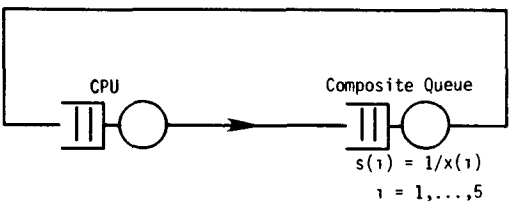


FIGURE 12. 4PP model with subnetwork replaced by flow-equivalent.

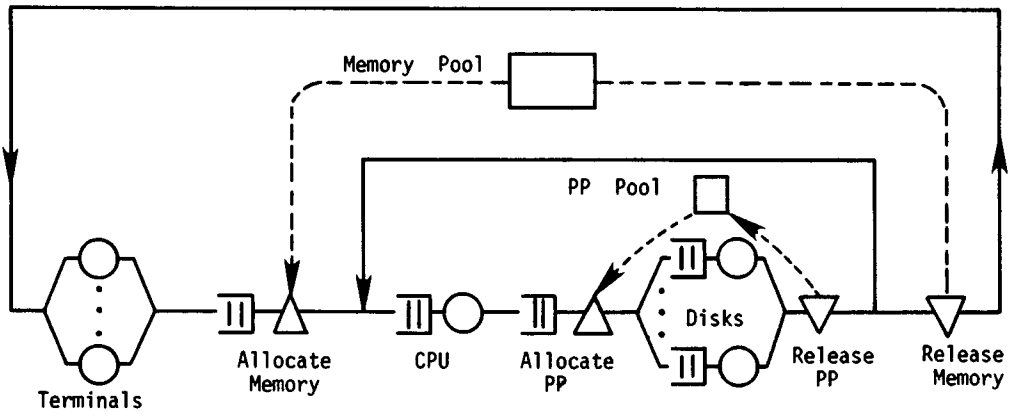


FIGURE 13 PP models with memory and terminals.

cannot expect performance measures of the network of Figure 12 to be the same as the corresponding measures of the original network. But the difference is very small [KELL76], and the aggregation is both reasonable and economical.

As a numerical example, consider a PP model with parameters chosen for ease of solution and exposition. Let there be a degree of multiprogramming of 3, 2 PPs, and 2 disks. Let the mean service time at the CPU and each disk be exponential with mean 40 ms. Assume all queueing disciplines are FCFS. The routing frequencies to each disk are equal. (Remember that this network does not satisfy product form.)

The exact value for CPU utilization is 82.6% [KELL76]. If we consider the PP-disk subnetwork offline with the output fed back to the input, we find $X(1)$, the completion rate with 1 job in the offline system, to be 25 jobs per second. $X(2)$ is 33.3 jobs per second and $X(3)$ is also 33.3 since at most 2 jobs can be in the disk queues. (Without this restriction $X(3)$ would be 37.5.) Substituting this offline behavior in the CPU-composite queue network we give the composite queue a mean service time of 40 ms. with load (queue length) 1 and 30 ms. with load 2 or 3. Solving this two queue (product form) network for CPU utilization we get a value of 83.0%. Thus the error in this case is small. If we solve a similar model with 3 PPs to get an upper bound on CPU utilization, this bound would be 84.6%.

Note that this replacement of subnetworks by flow-equivalents may be repeated.

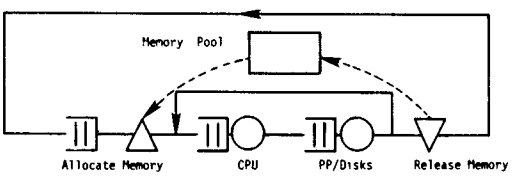


FIGURE 14 Network to be replaced with flow-equivalent

Consider the model of Figure 13. Here the 4 PP model is embedded in a network also representing memory contention and terminal think times. After making the above aggregation of the PP and disk queues, we can construct a flow-equivalent of the memory, CPU, and composite PP/disk queue in a similar manner (Figure 14). We are assuming that each job has the same fixed memory requirement. The more realistic case in which jobs require variable amounts of memory has been treated using flow-equivalents [BROW77].

Flow-Equivalents with Non-Exponential Distributions

As mentioned earlier, another situation where approximations succeed is the case of FCFS queues with non-exponential distributions. The 4 PP model had processor sharing as the CPU queueing discipline, but now suppose it was FCFS. Since the CPU service time distribution was given as hyperexponential, the model corresponding to Figure 12 would not be solvable by product form methods. However, the recursive method of [HERZ75] (the second condition in Section 1) is well suited to this new two

queue model, and again we get results close to the correct solution with little computational effort [KELL76].

Let us now consider a traditional central server model (without peripheral processors) in which all service times may be non-exponential and all queueing disciplines are FCFS. Again a reasonable approach is to isolate the I/O queues and try to find a flow-equivalent composite queue representation. Pictorially this would correspond to Figures 1, 11, and 12 without the passive queues. As we explain below, the solution of the I/O subnetwork and the composite queue representation are more difficult issues than with the 4 PP network that was similar to a product form network. Though this problem has received considerable attention, the known approaches must be considered heuristic, since they are supported primarily by intuition and empirical studies rather than rigorous proofs.

Presumably we still want to use a load-dependent queue as an approximate flow-equivalent of the I/O subnetwork because it represents reality more realistically. Three key issues must be faced:

1) How do we estimate the mean service times $S(n)$ for the (approximately) flow-equivalent system? This seems to be the most crucial issue. There are several ways of estimating the throughputs in the subnetwork; unfortunately, the better the estimation, the more expensive the method.
2) How do we choose the service time distributions in the flow-equivalent system? The emphasis in this work has been on selecting variances [SAUE75b, SEVC77b], though there has been an increasing realization that percentiles are very important [BUX77, LAZO77].
3) How do we select the queueing discipline in the flow equivalent system?

Issue 1: Selecting mean service times $S(n)$ for the flow-equivalent system. (All of the following methods extend to general networks.)

Method 1: Figure 15 shows the network obtained by feeding the output of the I/O system back on itself. This corresponds to taking the I/O system "offline." If any one of the I/Os has non-exponential service times, it does not satisfy the conditions of product form, and therefore the system may be very time-consuming to analyze. However, the most accurate solution is to model this subnetwork as a discrete-state Markov process and to then determine steady-state probabilities numerically, and thus compute the exact mean service times of the flow-equivalent.

Method 2: [ZAHO77] If there are many I/Os then the number of states in the Markov model of the subnetwork may be so large as to make the problem intractable. Suppose we have a technique to determine, relatively rapidly, the steady state probabilities for a network with J I/O devices. We may determine the approximate flow-equivalent composite I/O in several steps.

Step 1: We may group together the first J I/O devices and represent this group by a flow-equivalent system with *exact* rather than approximate service rates. We may then group together the next J devices and determine the flow-equivalent system for this group and so on. We may choose to have exponential or non-exponential service times for the flow-equivalent system.

Step 2: If the flow-equivalent systems in the first step are given non-exponential service times, in the second step we may determine the flow-equivalent for a group of J flow-equivalent systems obtained in the first step; thus in the second step we determine the approximate flow-equivalent for J^2 devices, and we proceed in this fashion until we have an approximate flow-equivalent for the entire I/O system. This process takes $\log_J M$ steps, where M is the number of I/Os.

If the flow-equivalents obtained in the first step are given exponential service

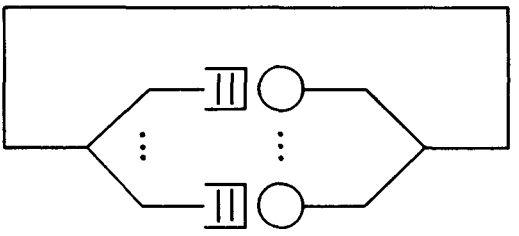


FIGURE 15 I/O subnetwork.

times, in the second step we have a network of systems each of which satisfies product form conditions, and hence we may determine the approximate flow-equivalent for the entire I/O system in two steps. This method will usually be less accurate than the previous method.

Method 3: We may make the assumption that all I/O devices have exponential service times, in which case the subnetwork has product form, allowing us to compute the subnetwork throughputs and the mean service rates in the approximate flow-equivalent directly. This method is very quick but probably not very accurate.

Issue 2: Selecting service time distributions for the flow-equivalent system.

The emphasis has been on selecting the most appropriate second moment. There has been no work on the potentially valuable method of using percentiles of the individual service distributions to determine the percentiles of the service distribution of the composite system.

The simplest solution is to assume that the service times of the composite system are exponential (as in Method 3 above). However, this solution can result in considerable error. A more realistic solution is to estimate the second moment of the composite system from the second moments of its component parts.

Let CV be the coefficient of variation of the service time, and let $\alpha = CV^2 - 1$. Thus α is negative for hypoexponentials, positive for hyperexponentials, and zero for exponentials. Use the subscript C for the composite system and the subscript i for the ith I/O device. Let p_i be the routing frequency for a job entering the I/O system going to the ith I/O device. Sauer and Chandy propose the following heuristic formula for the coefficient of variation of the composite system [SAUE75b]:

$$CV_C = \sum_i p_i CV_i.$$

Sevcik, et al. propose a more accurate formula [SEVC77b]:

$$\alpha_C = \sum_i p_i^2 \alpha_i$$

for heavy load conditions and also the following complex, but more accurate, equation:

$$\alpha_C = \sum_i p_i^4 \left(\frac{M_i}{M_C}\right)^2 \alpha_i$$

where M_i is the mean service time of the ith I/O device and M_C the mean time a job is in service in the subnetwork.

The general problem of estimating second moments of job inter-arrival times at different points in general networks has received considerable attention, though the focus has been on networks with a single chain of jobs, one job class per queue, and FCFS queueing disciplines. Considerable work remains to be done in the area of networks with priority and other multiclass disciplines. Sevcik, et al. [SEVC77b] has the most comprehensive treatment of this issue, based in part, on earlier work [DISN74, GELE76, KOBA74, REIS74].

A summary of the results in [SEVC77b] is presented in Figures 16, 17, and 18. Once

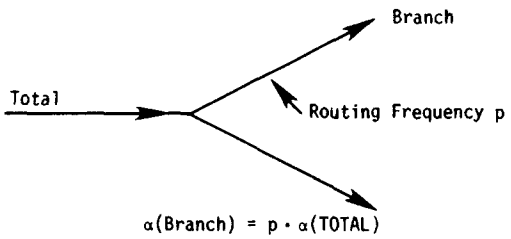
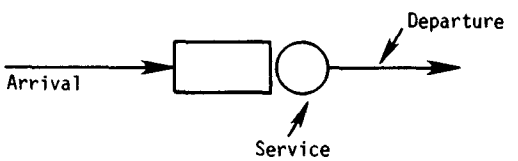
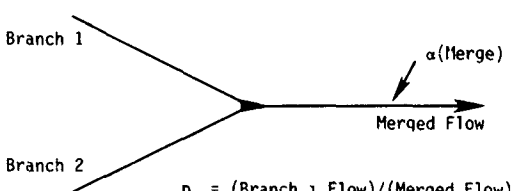


FIGURE 16. Estimating second moment of split flow.



α(Departure) = Utilization^2 · α(Service)
+ (1-Utilization^2) · α(Arrival)

FIGURE 17 Estimating second moment of interdeparture times.



p_i = (Branch i Flow)/(Merged Flow)

α(Merge) = $\sum p_i^2$ α(Branch i)

FIGURE 18. Estimating second moment of merged flow

the coefficient of variation is computed, the conventional approach is to represent the service time as a sequence of exponential stages (see Section 1).

Issue 3: Queueing disciplines.

The queueing discipline of the composite system does affect performance measures in the CPU-composite I/O network. Unfortunately, there has been very little work in the area of selecting queueing disciplines. Queueing disciplines have been selected more to reduce computational complexity than to better model the composite system. If there are many chains in the network, the assumption of processor sharing at each queue results in the simplest state-transition diagram. Usually, PS gives the same results for non-exponential distributions as exponentials, thus discarding efforts to characterize distribution *form*. Processor sharing allows a job to enter the I/O subsystem after another job B, but to leave before B leaves. For this reason we might choose a FCFS discipline for the flow-equivalent system for a sequence of FCFS queues in series. However, it must be emphasized that such arguments are merely educated guesses, and much empirical work needs to be done in this area.

In summary, the specification of flow-equivalent systems is still an art. However, empirical simulation work and case studies of real systems are leading to a better understanding of this problem.

General Closed Networks (Multiple Chains)

So far we have been assuming a single routing chain. The approximation techniques can be extended to multiple chains, but relatively little has been done in this area. The ease or difficulty of the extension depends on the similarity between the chains and on the number of chains.

Suppose we have a central server model with two chains. The chains are parallel in their routing (Figure 19), and the only queue that violates product form conditions is the CPU queue. We can determine a flow-equivalent of the I/O subnetwork in a manner analogous to the single chain case by connecting the output of each chain to the input of that chain, i.e., once again studying offline behavior. From considerations similar to the single chain case we determine the throughput $X_c(n_1, n_2)$ for each chain ($c = 1, 2$) for $n_c = 1, \ldots, N_c$ and $n_{3-c} = 0, \ldots, N_{3-c}$. This computation is similar to the single chain case, e.g., $X_1(n_1, n_2) = G(n_1 - 1, n_2)/G(n_1, n_2)$ [CHAN75a]. However, the choice of queueing discipline for the composite queue encounters the same sort of problems as those discussed in the previous subsection. One possible approach to specifying the queueing discipline is given in [SAUE75b]: Each chain has a dedicated device in the composite queue. Jobs of a chain are serviced FCFS by the dedicated device with load dependent service times $S_c(n_1, n_2) = 1/X_c(n_1, n_2)$. This discipline was chosen as being generally reasonable, but there are cases where it would be unrealistic, e.g., when the I/O subsystem consists of a single FCFS queue.

Even after replacement of the I/O subsystem by the approximate flow equivalent, the network consisting of the CPU queue and composite queue may be difficult to solve. For example, the CPU queue may violate product form by having a FCFS queueing discipline with different service time distributions for the classes of the different chains. The solution of the CPU and composite queue model must take into consideration the possible orderings of jobs of the different chains, and this increases the state space size drastically. Thus the solution quickly becomes unmanageable with even moderate chain populations.

Other important cases violating product form allow priorities among the different classes at the CPU queue. Because priorities eliminate many of the possible orderings of jobs, the FCFS ordering problem will be reduced by priorities, possibly even eliminated, depending on preemption rules.

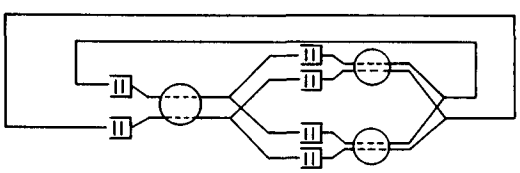


FIGURE 19. Central server model with two chains.

But these cases also become unmanageable with moderate numbers of chains (e.g., five).

When the CPU-composite queue network becomes unmanageable because of chain populations and/or numbers of chains, one can attempt aggregation of chains as well as aggregation of queues. In aggregation of queues one replaces several queues by a composite queue that is approximately flow-equivalent as far as the remaining jobs are concerned. In aggregation of chains one replaces several chains by a single chain that is approximately equivalent as far as the remaining jobs are concerned. The approach used in [SAUE75b] is first to solve a central server model identical to the given model, except that the CPU queue with disciplines violating product form is treated as a processor-sharing queue. The service times at the I/O queues are the same for all chains, assuming FCFS disciplines at those queues. For the chain-dependent values (the CPU service time distributions and the routing frequencies) the "composite chain" values are determined as a weighted sum of those values for the component chains. The weights used in [SAUE75b] are the throughputs of the component chains divided by the sum of the throughputs of the component chains. Other weights may be used, e.g., the number of jobs of the component chains divided by the total population of the component chains [MACN75].

These techniques extend to networks with more complex routing structures than the central server model; the appendix contains an example.

Flow-Equivalents in Simulation

Some of the subsystem flow-equivalents could be constructed from simulations, i.e., by simulating a subnetwork. We could also use simulation to analyze the model with the flow-equivalents. Note that flow-equivalent methods are totally general because the technique used in constructing flow-equivalents (e.g., simulation, queueing theory, regression analysis) is left unspecified. See [CHIU78, SAUE76, and SCHW78] for further discussion of this approach.

Justification of Aggregation

We informally described the class of product form networks in Section 1. At the beginning of this section we stated that aggregation was exact for product form networks, i.e., one can determine the flow-equivalent of a product form subnetwork and use the flow-equivalent in a product form network as if it were the given subnetwork. This should be plausible to the reader, since the state of a product form network factors into components representing the states of the queues of the network. Informally, aggregation of the queues corresponds to association of the factors. For a more formal discussion see [CHAN75a].

The primary justification we have implicitly used for aggregation approximation is the exact aggregation of product form networks. This justification becomes less credible as the network to be solved becomes "less similar" to a product form network. Examples include the number of queues violating product form conditions (e.g., distributions and disciplines) increasing, an individual queue tending to deviate greatly from product form conditions (e.g., service times at an FCFS queue having very small or very large coefficients of variation), and conditions such as multiple resource holding becoming dominant. There is another justification, loosely coupled subnetworks [COUR75, COUR77], which is independent of product form conditions. The product form justification holds regardless of the degree of coupling of subnetworks; the loosely coupled justification holds regardless of product form conditions.

We illustrate degree of coupling by an example. Consider a closed queueing network with two subnetworks A_1 and A_2 (Figure 20). A job leaving a subnetwork is fed back to the input of the same subnetwork with frequency p, and goes to the other

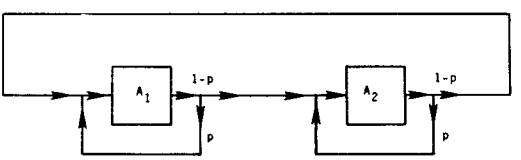


FIGURE 20. Two loosely coupled subnetworks, $p = 1$.

subnetwork with frequency $1 - p$. Consider the limiting behavior of this network as p tends to 1, but $p < 1$. The rate of flow of jobs between the subnetworks decreases as p increases, which implies that the times between subnetwork interactions (job transitions) increase. In the limiting case of p arbitrarily close to but less than 1, each subnetwork may be assumed to reach equilibrium between subnetwork interactions. This key idea will be developed further.

If $p = 1$ the network decouples into two closed networks B_1 *and* B_2 (Figure 21) where B_i, $i = 1,2$ is the network obtained by feeding the output of A_i back to its input (Figure 20). Let there be C chains in the network and let $X_c(n_1, \ldots, n_C)$ be the equilibrium throughput of chain c jobs in B_2 when the population of B_2 is $n_1, \ldots, n_C$ for chains $1, \ldots, C$, respectively. Suppose we wish to study the detailed behavior of jobs within A_1. Then we may replace A_2 by a composite queue that has a mean service time of $1/((1 - p)X_c(n_1, \ldots, n_C))$ for class c jobs (at a device dedicated to class c) when the queue population is $n_1, \ldots, n_C$. (The service distributions may be set to the interdeparture distribution in B_2. It can be shown that as p approaches one the interdeparture distribution becomes exponential.) Decoupling works as p tends to 1.

Summary

The key concept in aggregation is to replace a subnetwork by a simpler one, ignoring the interactions of subnetwork components not of interest to us. In the terminology of Denning and Buzen [DENN78], we are assuming that the subnetwork's offline behavior is identical to its online behavior. The basic problem is to 1) develop a framework for characterizing the offline behavior, 2) solve the subnetwork to estimate these characteristics, and 3) aggregate the behavior of subnetworks until we obtain a network with a tractable solution. We have assumed the concept of flow-equivalence as a solution to the first part of the problem and have illustrated approaches to the second and third parts by examples.

FIGURE 21. Decoupled subnetworks.

4. IMPROVEMENTS OF FLOW-EQUIVALENCE

The goal of flow-equivalence approximations is to simplify network analysis by replacing a subnetwork by a single composite queue that behaves in approximately the same way as the subnetwork. Furthermore, the composite queue must have simple queueing disciplines and distributions to keep the computation manageable. In this section we shall survey methods that attempt to reduce the inaccuracies resulting from simplistic representations of subnetworks. Rather than attempting to represent a subnetwork accurately, these methods carry out the computation assuming simplistic subnetwork representation and later attempt to correct for the inaccuracy in subnetwork representation. We survey two such methods: iteration and product form. We next discuss the common aspects of these methods and then discuss the methods in detail. The key concepts used in the two methods can be combined.

An Overview of the Approaches

We partition a network A into subsystems $S_1, \ldots, S_M$ in some suitable manner. Since we analyze each subsystem in turn, we would like to keep the number M of subsystems small. On the other hand, we do not want a subsystem to be so complex that analyzing it by itself becomes an unmanageable problem. We analyze each subsystem S_i, $i = 1, \ldots, M$ in the following way. Define C_i, the complement of S_i, to be the subnetwork obtained by removing S_i from network A. C_i is the system "seen" by S_i. Represent each system C_i by a single composite queue F_i. Analyze the network consisting of S_i and F_i by some suitable method; the specific method selected depends upon the complexity of S_i and F_i. If a simple service rate structure (such as

rates proportional to the number of jobs in the system) is assumed for the flow-equivalent system, it may be possible to get exact closed form formulas for the performance measures of the $F_i - S_i$ network; indeed, such simple solutions are the motivation for simple (approximate) rate structures for F_i. For general service rate structures, the $F_i - S_i$ network may be modeled as a Markov process and analyzed by a recursive method, by sparse matrix techniques, or by simulation. Since the F_i are approximations to the C_i, the results of the analyses of the $F_i - S_i$ networks will generally be inaccurate; informally speaking, the less accurate the representation of the C_i by the single queues F_i, the greater the error.

One consequence of this error is that the estimates of the performance measures for $S_1, \ldots, S_M$ computed from the $F_i - S_i$ analyses may not be compatible with each other. For instance, assume that we compute the rate of flow of jobs through each subsystem S_i by analyzing the $F_i - S_i$ network. By multiplying the flow rate through S_i by the probability that a job leaving S_i will enter S_j, we compute the rate of job flow from S_i to S_j. We may now compute the total flow entering S_j by summing the flows over all paths into S_j. If the analysis were free from error the flow rate into S_j must equal the flow rate through S_j. However due to the approximations inherent in the analysis of each subsystem, the computed flow rates into S_j may not equal the computed flow rates through S_j.

We can use several rules to check the compatibility of the computed performance measures for $S_1, \ldots, S_M$. The more checks we use, the greater will be our faith in the compatibility of the results. However we may not wish to spend a great deal of computing time on the checking process. It must be emphasized that even though the computed subsystem performance measures satisfy some compatibility tests, there may still be errors in the computed measures.

The fundamental problem with computing performance measures for $S_1, \ldots, S_M$ by the method discussed above is that the independent subsystem analyses emphasize the *local* view; i.e., the subsystem itself is modeled in detail while everything else is modeled in a gross fashion. The compatibility checks take a *global* network view. Product form methods attempt to correct for the local views emphasized in the independent subsystem analyses by taking the specific global network view discussed next. Let $p_i(s_i)$ be the value computed for the equilibrium probability that subsystem S_i is in state s_i by analyzing the $F_i - S_i$ network; note that the $p_i(s_i)$ are probabilities *local* to subsystem S_i — the rest of the network is ignored. Let $p(s_1, \ldots, s_M)$ be the equilibrium that subsystem S_i is in state s_i for $i = 1, \ldots, M$; note that $p(s_1, \ldots, s_M)$ takes a *global* network view because it is concerned with all the subsystems in the network. Product form methods assume that

$$p(s_1, \ldots, s_M) = \frac{1}{G} \prod_i p_i(s_i) \quad \text{all } s_1, \ldots, s_M$$

where G is a normalizing constant. Thus the local subsystem analyses are forced into the global mold of the product form. Performance measures are now computed from the network probabilities, $p(s_1, \ldots, s_M)$ rather than from the subsystem probabilities $p_i(s_i)$.

Product form methods are often combined with iterative methods; examples are provided later.

Iteration methods take the global view by making compatibility checks of the results of subsystem analyses. If the checks are not satisfied, the single queue models (F_i) of the complements (C_i) are modified; the specific method of modification depends upon the specific iteration method used. In the next iteration, the improved single queue models are used in the $F_i - S_i$ analyses. The process of making compatibility checks and modifying composite queue models is repeated until the compatibility checks are satisfied.

There are two difficulties with this method. First, satisfying the compatibility checks is no guarantee of the correctness of the results. Second, there is no guarantee that the iterations will terminate. Yet, (perhaps surprisingly) iteration methods seem to work satisfactorily much of the time [CHAN75b, LAM76]. We discuss some specific iteration methods next.

Iteration Methods

The questions we shall focus attention on are

- What are the subsystems that a network is decomposed into?
- What laws are used to check whether the subsystem analyses are compatible?
- What flow-equivalent models are used?
- How are the independent subsystem analyses adjusted if the results from these analyses are incompatible?

The laws that we use in checking the compatibility of the computed subsystem measures are called *invariants*. We now discuss three different cases where iterative methods have been used.

General Closed Queueing Networks

We summarize the method described in [CHAN75b]. Two invariants are used in this method.

1) The rate of flow of chain c jobs into a subsystem must equal the rate of flow of chain c jobs out of that subsystem.
2) If chain c is a closed chain, the sum of the mean number of chain c jobs in subsystem S_i, summed over all i, must equal the total number of chain c jobs in the network.

We now present some details of the algorithm assuming a closed network with a single closed chain of jobs. Extending the method to multiple job chains is straightforward. (Examples of a closed network with two job chains and of an open network are presented later in this section.) Let $\bar{n}_i$ be the mean queue length of queue i. Compute a set of numbers y_i, which we call relative throughputs, where

$$y_j = \sum_{i=1}^{M} y_i q_{ij}$$

Note that the y_i are unique up to a normalizing constant. Let t_j be the throughput through the jth queue. Since the flow into queue j is equal to the flow out of queue j,

$$t_j = \sum_i t_i q_{ij}. \quad \text{Invariant I; Flow Invariant}$$

Hence we have

$$t_j = \alpha y_j \quad \text{for all } j,$$

where α is a constant. Set

$$t_j' = t_j / y_j;$$

we refer to the t_j' as the *normalized* throughputs. An equivalent formula for the first invariant is

$$t_1' = t_2' = \ldots = t_M' = t' \quad \text{Flow Invariant}$$

where

$$t' = \sum_i t_i'/M$$

is the average value of the normalized throughput computed by analyzing all the queues independently.

The second invariant is

$$\sum_i \bar{n}_i = N \quad \text{Invariant II; Population Invariant}$$

where N is the population of jobs.

The Composite Queue F_i

We construct F_i in the following way. Ideally, we would like F_i to be the flow-equivalent of C_i. However, to determine the flow-equivalent for C_i we have to analyze C_i. Analyzing C_i may be a very expensive computational procedure if C_i does not satisfy product form. Hence, we are forced to settle for making F_i an *approximation* to the flow-equivalent of C_i. This approximation is obtained by computing the flow-equivalent for C_i *assuming* (generally incorrectly) that C_i satisfies product form.

Note that we use two levels of approximation here. First, the F_i are not the true flow-equivalents. Second, even if the F_i are the true flow-equivalents for C_i, this heuristic may not yield correct answers.

Iteration

On the kth iteration the service rates of F_i are set equal to the service rates of the flow-equivalent of the complement of queue i in an auxiliary network $S^{(k)}$. The auxiliary network is identical to the given network except that:

1) The service rates in the auxiliary network are different; and
2) The queueing disciplines in the auxiliary network are such that the network satisfies product form. Hence the flow-equivalents for the complement of any queue in the network can be readily computed.

The goal of the iteration is to adjust the

flow rates of the auxiliary network (which results in adjustments to the F_i and consequent changes in queue statistics), so as to force the individual queue statistics towards satisfying the invariants. The details of this adjustment are not presented here; they may be found in [CHAN75b]. If the invariants are not satisfied the results of the iterative method are incorrect; however, if the invariants are satisfied it does not necessarily mean that the solutions are correct. Nevertheless, this approach has been validated by comparing the results obtained by this method with results obtained from simulations for several cases. Networks with several chains of jobs and a variety of queueing disciplines (including priority disciplines) were analyzed by this method. For details regarding the algorithm, its accuracy, and execution times, the reader is referred to [CHAN75b].

A Case Study for VM/370 Using Iterative Methods

We next discuss an iterative scheme proposed for modeling scheduling strategies for interactive computers [BARD77]. This work is important for several reasons. The scheme itself is very simple, and therefore appealing. Furthermore, the method has been used to model the effects of workload and configuration changes for a widely used complex interactive operating system, VM/370. It has been incorporated into a performance package for configuring VM/370 systems as discussed in [BARD78] (See pp. 333–342, this issue).

We shall summarize the results in [BARD77], restricting attention to the principles of constructing an iterative method.

A user is assumed to make transitions between states in a cyclic fashion: THINK - MEMORY-WAIT - ACTIVE - THINK-..., or equivalently 1, 2, 3, 1, ...

For purposes of exposition assume that there are only two job chains:

Chain 1: trivial jobs

Chain 2: non-trivial jobs.

A trivial job is assumed to be immediately admitted to service, i.e., it is assumed to spend no time in the MEMORY-WAIT state. Thus its state transitions are THINK-ACTIVE-THINK, ..., or 1,2,1, ... A non-trivial job may have to spend time in the MEMORY-WAIT state. The diagram for the overall system is shown in Figure 22.

Model inputs: We are given the number of jobs N_i in chain i; the mean main memory requirement (resident set size) W_i for jobs in chain i; $T_{i,1}$, the mean think time (i.e., time in state 1 for chain i jobs), and S, the total amount of main memory available. We are also given details regarding the visit counts and mean service times at the CPU and I/O devices for both chains.

Model outputs: We are required to compute $T_{i,2}$ and $T_{i,3}$, the mean times spent by chain i jobs in state 2 (MEMORY-WAIT) and state 3 (ACTIVE). We shall also compute $N_{i,k}$, the average number of chain i jobs in state k, $k = 1, 2, 3$, and S_i the mean amount of main store occupied by all the chain i jobs together.

Aggregation: The model is analyzed into two stages. In the first stage the computer subsystems representing jobs in the active state (Figure 23) is analyzed. The mean time $T_{i,3}$ spent by chain i jobs in the active state is computed from this submodel (see below).

The overall model is analyzed in the second stage (Figure 24). Note that in this stage, the computer subsystem is represented as a flow-equivalent server with infinitely many servers, which is equivalent to a delay of $T_{i,3}$ time units.

Invariants: The probability P_{ik} that a random chain i job ($i = 1, 2$ for trivial, non-trivial) is in state k ($k = 1, 2, 3$ for think, memory-wait, and active) is proportional to the time spent by that job in that state, i.e.,

$$P_{ik} = \frac{T_{i,k}}{T_{i,1} + T_{i,2} + T_{i,3}} \quad \text{for all } i, k$$

Hence the mean number of chain i jobs in state k is

Invariant 1

$$N_{i,k} = N_i \, P_{ik} = N_i \, \frac{T_{i,k}}{T_{i,1} + T_{i,2} + T_{i,3}}.$$

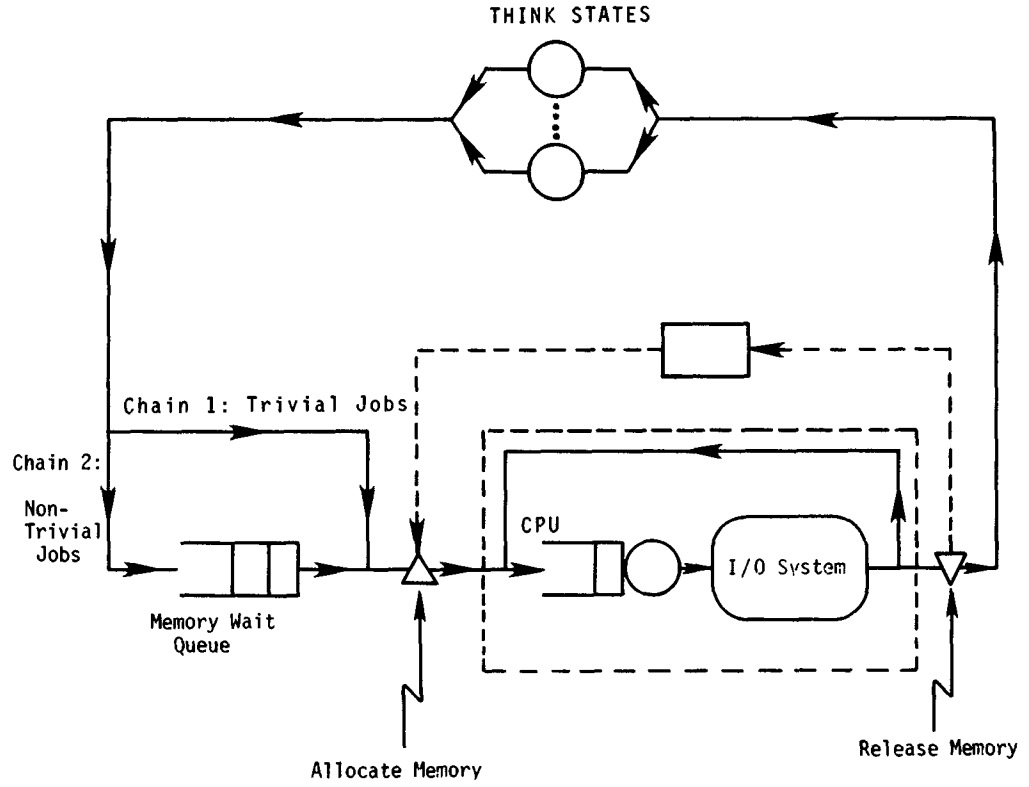


FIGURE 22. Interactive system.

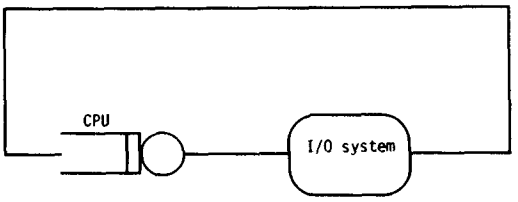


FIGURE 23. Computer subsystem.

Invariant 2

The mean amount of main storage used by chain i jobs (S_i) is the product of the average number of chain i jobs in main storage ($N_{i,3}$) and the average amount of main storage required per job (W_i).

$$S_i = N_{i,3} \cdot W_i.$$

Note that $T_{1,2}$, the time spent by a trivial job in the memory-wait state, is 0 (zero). Hence

$$S_1 = W_1 \; N_1 \cdot \frac{T_{1,3}}{T_{1,1} + T_{1,3}}.$$

Note that we are given N_1, W_1 and $T_{1,1}$. If we are given $T_{1,3}$ then we may compute S_1 from the above equation. If $S_1 \leq S$ then there is enough main store to accommodate trivial jobs, and our assumption of $T_{1,2} = 0$ is reasonable; if $S_1 > S$ the system is saturated by trivial jobs, and we must stop our analysis.

The above arguments are not strictly correct because they are based entirely on mean values, rather than distributions. Strictly speaking, a trivial job will experience memory-wait if all of main storage is filled with other trivial jobs; the only case when this situation will *never* occur is when

$$N_1 \; W_1 \leq S$$

rather than

$$N_{1,3} \; W_1 \leq S.$$

However, the goal here is not to come up with a mathematical model but rather to develop a heuristic program that seems to work most of the time. The only test for such heuristics is validation.

We may develop a similar invariant for chain 2 jobs. If we assume $T_{2,2} = 0$, i.e., the

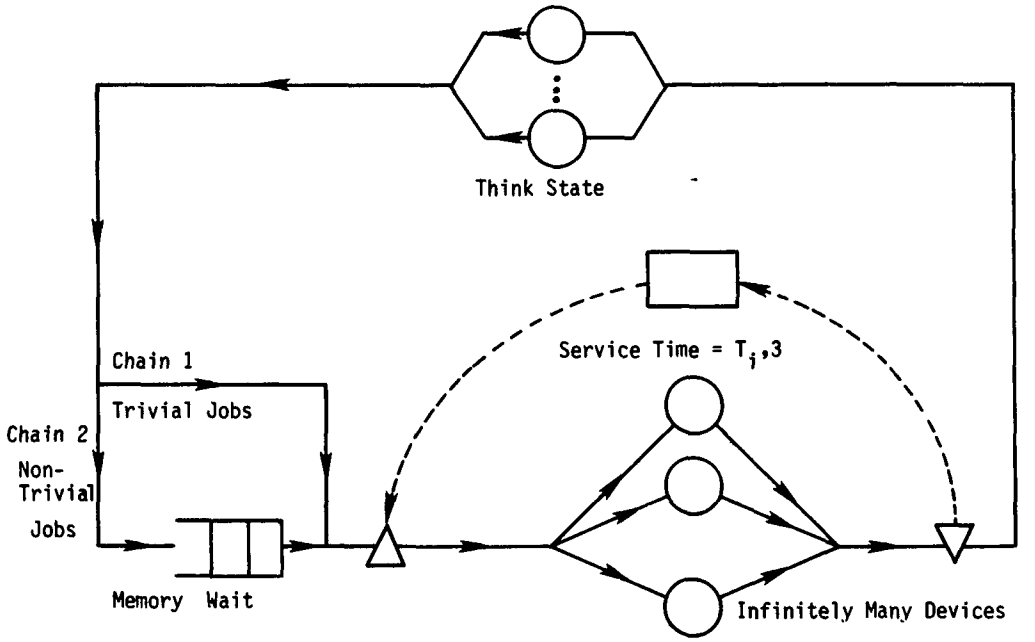


FIGURE 24. Overall model B.

time spent by a non-trivial job in memory-wait is 0, then the average amount of main store occupied by non-trivial jobs is

$$W_2 \cdot N_{2,3} = W_2 \cdot N_2 \frac{T_{2,3}}{T_{2,1} + T_{2,3}}$$

If this value does not exceed $S - S_1$ (the average amount of main store not occupied by trivial jobs) then our assumption that $T_{2,2} = 0$ is reasonable (though not strictly correct). If this value exceeds $S - S_1$ then $T_{2,2}$ must be positive, and we have

$$\begin{aligned} S - S_1 &= W_2 \; N_{2,3} \\ &= W_2 \cdot N_2 \frac{T_{2,3}}{T_{2,1} + T_{2,2} + T_{2,3}} \end{aligned}$$

or equivalently

$$T_{2,2} = \frac{W_2 \; N_2 \; T_{2,3}}{S - S_1} - T_{2,1} - T_{2,3}.$$

Invariant 3

The computing system (consisting of CPU and I/Os, see Figure 22) is modeled by a queueing network that is assumed to satisfy product form. Given the population $N_{i,3}$ of chain i jobs in the computing system, the visit rates at each device and mean service times at each device, we compute the mean wait times at each device. Multiplying the mean wait times at each device by the average number of visits made to that device by a chain i job on each interaction and summing over all devices, we compute the mean time $T_{i,3}$ spent by a chain i job in the computing system. Thus, the mean time spent by a chain i job in the active state is related to the mean number of jobs in the active state; this relation between $T_{i,3}$ and $N_{i,3}$ is invariant 3.

The Flow-Equivalent System——Since the analysis is carried out in terms of mean values, the computer subsystem is represented by mean delays $T_{i,3}$. This is equivalent to representing the computing subsystem by an infinite number of servers, each of which has a mean service time of $T_{i,3}$ for chain i jobs. The choice of an infinite server queue for representing a queueing network is popular because what it lacks in accuracy, it makes up for in simplicity and consequent computational speed.

The Iteration——Initialization: Assume values for $T_{i,2}$ and $T_{i,3}$.

Step 1: Use the assumed values for $T_{i,2}$ and $T_{i,3}$ and invariant 1 to com-

pute $N_{t,3}$, which is the job population assumed in the computer subsystem.

Step 2: Use the $N_{t,3}$ obtained in step 1 with invariant 3 to recompute $T_{t,3}$.

Step 3: Use $T_{t,3}$ obtained in step 2 with invariant 2 to recompute $T_{t,2}$.

Step 4: Return to step 1 with new values for $T_{t,2}$ and $T_{t,3}$, if there is a significant difference between the old and new values. Otherwise, stop.

This method has been validated and is in use. It cannot be overemphasized that the credibility of a heuristic technique rests on empirical validation rather than on logical rigor. Bard's method [BARD77] is credible because it has worked for several practical cases, even though there is no guarantee of convergence and the analysis is informal, being based on mean values. Systems analysts are also more likely to have faith in a model, such as Bard's, that considers key parameters, such as amount of main memory, memory scheduling, different kinds of users, and the complex queueing structure of I/O channels. The method has a computation time favorable to that of other methods (such as simulation) and *satisfactory accuracy*. There is no point in developing an algorithm that invests a great deal of computer time in obtaining a degree of accuracy that is greater than is required for making the decisions at hand.

An Iterative Scheme for Open Network Models of Packet-Switching Systems

We now discuss the use of iteration techniques in the analysis of communication networks [LAM76]. This discussion is important for three reasons. First, communication systems are important in themselves. Second, this section illustrates the use of heuristic approaches in modeling finite waiting rooms. Third, this section provides an example of approximation methods used to analyze *open* queueing network models. (See also [WONG78] pp. 343-351, this issue.)

A job in this context is a "packet," which is a basic unit of communication. The system accepts packets at sources, routes them through communication lines and switches, and delivers packets to destinations. The resources in the system are communication lines, the switches which route packets along appropriate paths, and the memory (store and forward buffer) at each switch in which packets are stored until they can be shipped out along appropriate communication lines. The servers in the queueing network model are the communication lines and the switches. However, since the actual service time of a switch is small compared to the service time at a communication line, we shall ignore switch service times.

Associated with each packet is a source node and a destination node. A packet moves through the network from the source node to the destination node along a fixed route. The ith node of the network has a finite number N_i of store and forward buffers, i.e., it has a finite waiting room. If there are N_i packets in the ith node when another node k tries to send one more packet J to the ith node, then the ith node refuses to accept J, and J must be retransmitted by node k. In other words, a server at node k serves J, and if at the instant at which J completes service the waiting room for node i is full, then the server repeats J's service, until eventually there is a vacant position in i's waiting room.

Figure 25 is a queueing model of *one* node of the network (node i). This node is assumed to communicate directly with I other nodes, $i_1, \ldots, i_I$. We also use one "dummy" node, i_0, to represent users who send and receive packets directly from node i. We assume that all lines are full-duplex, i.e., that packets can be transmitted in both directions simultaneously. There are $I + 1$ parallel servers associated with node i, and the jth such server, $j = 0, 1, \ldots, I$ models the communication line *from* node i *to* node i_j; this server serves those packets in the node i waiting room that have to go to node i_j. Even if all service times are assumed to be independent exponential random variables, this queueing network does not satisfy product form because of the finite waiting room and the consequent retransmission (re-serving) of jobs. Hence we shall resort to iterative approximation methods.

The network is partitioned into subsystems, one subsystem for each node. We

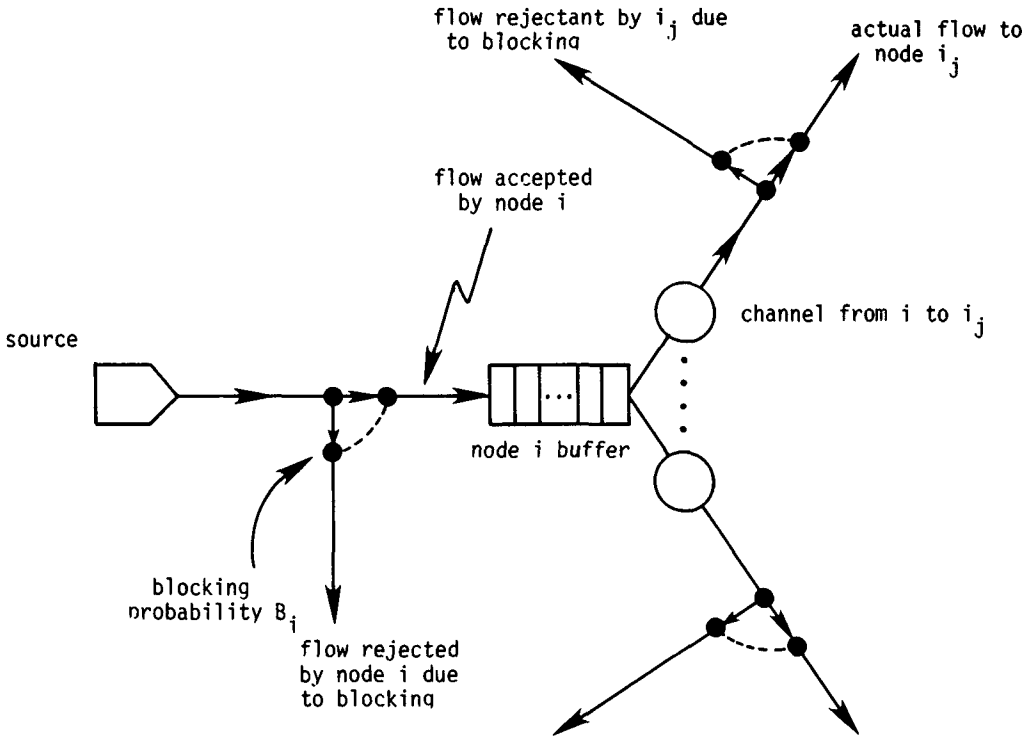


FIGURE 25. Network model for node ι.

next discuss the key questions which arise in iterative methods.

Invariants: Let λ_{ij} be the apparent flow rate of packets (i.e., not including retransmissions) from node i to node i_j. We can compute the λ_{ij} from the rate at which packets enter the network and the routes taken by packets as they travel from the source to the destination. Let r_{ij} be the actual flow rate of packets (i.e., the apparent flow plus retransmissions) from node i to node i_j Let B_j be the probability that the waiting room at node i_j is full. Then

r_{ij}. (fraction of time the node j waiting room is not full) $= \lambda_{ij}$

i.e.,

$$r_{ij} = \lambda_{ij}/(1 - B_j) \qquad \textit{Invariant 1}$$

Let μ_{ij} be the service rate of the channel from node i to node i_j. Assume product form for the queue length distribution of each channel. This assumption is invalid due to the finite waiting rooms. However, we make this assumption to make the computation manageable. The probability that there are q_{ij} packets at node i waiting to be shipped to node j is:

$$p_{ij}(q_{ij}) = p_{ij}(0) \cdot (r_{ij}/\mu_{ij})q_{ij}$$

Now assume product form for the entire ιth node. This is also an invalid assumption that is made to make the problem manageable. Then, the equilibrium probability that there are q_{ij} packets at node ι going to node $i_j, j = 0, 1, \ldots, I$ is

$$P_\iota(q_{\iota 0}, \ldots, q_{\iota I}) = \prod_j p_{ij}(q_{ij}) \qquad \textit{Invariant 2}$$

Thus Invariant 2 is a relationship between actual flow rates and node state probabilities.

Invariant 3

The probability B_ι that the node i waiting room is full is the sum of the probabilities $P_\iota\ (q_{\iota,0}, \ldots, q_{\iota,I})$ over all states with $q_{\iota,0} + \ldots + q_{\iota,I} = N_\iota$.

Thus the blocking probability B_ι depends upon (Invariant 3) the state probabilities p_{ij}, which depend upon (Invariant 2) the

actual flow rates r_{ij}, which depend upon (Invariant 1) the blocking probabilities B_j. In this formulation, the flow-equivalent of node i is defined in terms of the blocking probability B_i and the flows λ_{ji} and λ_{ij} into and out of i.

Iteration: A simple iterative scheme is to assume a set of blocking probabilities B_j and to then use Invariants 1, 2, and 3 in that order to get a new set of blocking probabilities. This iteration is repeated until successive iterations do not produce a significant change in blocking probabilities.

The networks described here may encounter deadlocks. In this case the equilibrium throughput is 0 (zero) and the above analysis is wrong. However, the results computed by the iterative method proposed here are meaningful (though not mathematically correct) if the probability of deadlock is low. In this case, the performance measures predicted by the model should be taken to be the values that occur after the network has been in deadlock-free operation for a long time.

Product Form Methods

Reiser and Kobayashi [REIS74] use product form approximations; their method is discussed in Section 6 in connection with diffusion approximations.

Shum and Buzen [SHUM77] couple the product form method with an iterative scheme in an interesting fashion. Their goal is to analyze closed queueing networks with general service time distributions and FCFS queueing disciplines. We shall discuss their technique in the same format as other iterative techniques.

Aggregation: Each queue is analyzed independently. The complement C_i of the ith queue is represented by a single queue F_i with a load independent service rate and exponential service times. This is obviously a very simplistic model for the complement. However, the product form and iterative prodecures attempt to correct for the inaccuracy in subsystem representation.

As in all product form methods, the equilibrium state probabilities for the entire network are computed by taking the product of the state probabilities for each queue that are computed from analysis of the $S_i - F_i$ networks. Performance measures are computed from the state probabilities of the entire network (rather than from subsystem state probabilities).

Invariants: The first invariant used in [CHAN75b] is also used here: flow rate into a queue must equal flow rate out of that queue. This scheme uses a single invariant.

Iteration: The algorithm iteratively adjusts service rates of the composite queues F_i until the flow invariant is satisfied. The flow invariant states that all queues must have the same normalized throughput (see 4, General Closed Queueing Networks). Let t'_i be the normalized throughput of queue i. The flow invariant is satisfied when

$$t'_1 = t'_2 = \cdots = t'_K.$$

To measure how closely the normalized throughputs computed at any point in the algorithm satisfy the flow invariant, the following error function is used.

$$L = \sum_{i=1}^{K} (y_i/\mu_i) \cdot (t'_i - t^*)^2$$

where y_i is the relative visit rate into queue i. μ_i is the service rate of queue i and hence y_i/μ_i is proportional to the utilization of queue i. t^* is the normalized throughput of the queue with the highest utilization. The above error function is used because the "bottleneck queue" (i.e., the one with the highest utilization) plays a crucial part in calculating queue statistics. The error function also gives greater weight to queues with greater utilization.

Let λ_i be the service rate of the composite queue F_i. Let

$$\lambda_i = ky_i \quad \text{all } i$$

where k is an (unknown) constant. Assume an initial value for k and at each step of the iteration adjust k to minimize the error function L.

Summary

Two questions must be asked about these iterative methods:

1) Do these methods give more accurate estimates of key performance measures than straightforward flow-equivalent approaches?

2) If these methods do give more accurate results, is the increased accuracy worth the additional computational effort?

These questions cannot be answered categorically. There are cases where a relatively straightforward flow-equivalent approach is more accurate than the iterative methods, and cases where the iterative methods are more accurate. However, it is our *opinion* that the methods discussed in this section generally improve upon the results of direct flow-equivalent methods; this opinion is based upon case studies, including those surveyed in this section.

The ubiquitous tradeoff of improved accuracy versus greater computational effort can only be resolved by studying the specific requirements of the problem at hand and by comparison with alternate methods such as simulation.

5. FLOW-EQUIVALENT SYSTEMS AND THE MANAGEMENT OF MODELING PROJECTS

Performance modeling projects of actual or proposed systems are usually carried out under intense time pressure. The largest portion of time by far is spent on understanding the system to be modeled: poring through documentation and talking to system designers. It is crucial to divide the work to allow several modelers to work together with minimum conflict. A solution used by Browne, et al. is based on the chief programmer-team approach used successfully in the development of large programs [BROW75]. This discussion is a summary of their solution.

The chief modeler (in analogy to the chief programmer) partitions the overall system model into logically consistent subsystem models and assigns each subsystem to a different member of the modeling team. Examples of subsystems in the study of the Air Force Logistics computing system include the CPU subsystem, database management, disk subsystem, and the tape subsystem. Each member of the modeling team is expected to become an expert on his subsystem, interact with the appropriate system designers, and eventually construct a flow-equivalent system model (or a range of such models) for his subsystem.

The chief modeler should partition the overall model into subsystems so that, as far as possible, the subsystems are independent of each other. For reasons of clarity it is helpful to focus attention first on subsystem disciplines (e.g., priorities) and later consider details about the job mix (e.g., map chains and job classes to priorities). The chief modeler is responsible for checking that the flow-equivalent models of the subsystems are reasonable, putting the subsystem models together, and evaluating the overall model. Browne, et al., refer to the models required to produce flow-equivalents as *micro*-models and to the overall model consisting of a network of flow-equivalents as a *macro*-model. Note that the complexity of the micro-model may be greater than that of the macro-model.

A large simulation program was also constructed in this parallel fashion by the chief modeler-team. Browne, et al. report good results from this management technique. Though they use a two-level hierarchy (chief modeler and his team) there is no reason that the method of flow-equivalents could not be used with a deeper hierarchy. The general question of how much error is introduced by this hierarchical technique is still open.

The problem of managing large-scale modeling projects has received very little attention; this problem is likely to become acute. (See COUR77.)

6. DIFFUSION APPROXIMATION

Diffusion approximation methods use the theory of diffusion processes to analyze queueing problems. The reader need not be scared by the wealth of literature on diffusion processes, or by the apparently difficult mathematics involved in the development of the theory of diffusion. Most users merely use formulas from the diffusion approximation literature without understanding the detailed development of the formulas. Indeed, it is not necessary to understand the development of the formulas, provided that empirical studies show that the formulas fit experience. The only cases in which the systems analyst needs a thorough understanding of the mathematics of diffusion is when attempting either to develop

better approximations [GELE75] or to extend the approximations to new problems [FOSC77]. The following section is directed toward the large majority of analysts who need some understanding of diffusion approximations, but who are unlikely to attempt research in the area.

The organization of this section is as follows: we 1) discuss diffusion processes from a very informal point of view; 2) discuss the problem of mapping queueing processes onto diffusion processes; 3) review studies on the accuracy of the diffusion process; and 4) finally attempt to see where diffusion approximation techniques fit into the general scheme of queueing network analysis.

Why Bother with Diffusion Processes?

To illustrate the need for diffusion approximations consider a $GI/G/1$ queue, i.e., a single server queue fed by a job source where service and interarrival times are independent random variables having arbitrary general distributions. (Note that diffusion approximation methods are not restricted to $GI/G/1$ queues.) Let $n(t)$ be the number of jobs in the queue at time t. $n(t)$ can take on the values 0, 1, 2, ... We may think of $n(t)$ as the position of a particle d (for discrete) that makes a jump (of + 1) to the right when a job arrives and a jump (of − 1) to the left when a job departs (Figure 26). In general, the probability that d will make a jump of +1 or −1 in the next incremental time interval depends upon d's past behavior. For example, the length of time that d will stay in its current place (in general) depends upon the time since the last arrival or departure.

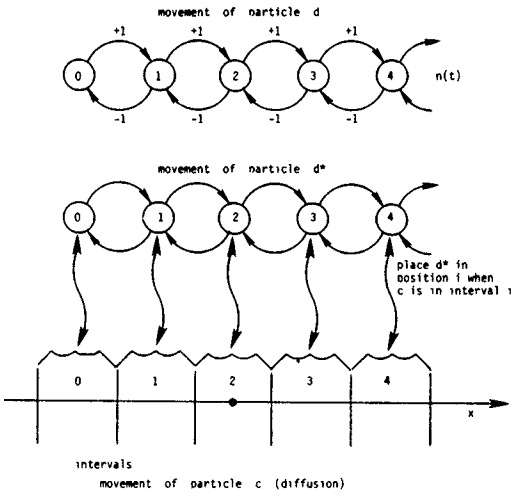


FIGURE 26. Modeling particle d by particle d^*.

Our goal is to deduce the probability of d's future behavior given its past behavior. A conventional approach is to represent (possibly approximately) the service time and the interarrival time by a finite collection of exponential stages (Section 1). All the relevant information in a queue's past behavior that is required to predict the queue's future behavior is captured by the "state" of the queue. The state-transition (differential) equations may be solved to obtain state probabilities as a function of time from which we can compute the probability that particle d is in position n at time t. Since there are an infinite number of states, this computation is not carried out numerically; if it can be carried out at all the computation is done algebraically. If we are dealing with a finite job population it may be possible to carry out the computation numerically, but we may prefer a less expensive approach such as the diffusion approximation. *Thus we attempt to use diffusion approximations only when numerical and algebraic methods fail.*

The difficulty with predicting the behavior of particle d is that it *has memory* in addition to its current position, i.e., the state of the system is not merely the particle's current position. (The analogy to the $GI/G/1$ queue tells us we must remember the residual service time.) In the diffusion approximation, we model the behavior of particle d approximately by the behavior of a particle c (for continuous) where c *has no memory.*

Whereas d can only take on values 0, 1, 2, ..., we let c take on all values on the nonnegative real line. Let us now decide how c should move along the real line. Since we are used to dealing with discrete state spaces we shall treat the continuous state space of c as the limiting case of a discrete state space. The discussion here is an informal treatment of material in COX65.

Assume that particle c can only move at times 0, T, $2T$, $3T$, ..., where T is some

small constant time period. c can only take (small) steps of magnitude M. Thus if we take the limit as T and M approach 0, we see that c moves continuously along the real time. In each time interval T we assume that the particle takes a step z where $z = +M$ with probability p, and $z = -M$ with probability $1-p$. In time nT, the total displacement of the particle will be the sum of n independent, identically distributed random variables, each with the same distribution as z. As n gets large, the distribution of the displacement approaches that of a *normal* random variable. We want the displacement per unit time to have a *finite* mean and variance so as to have a useful approximation to what actually happens with queue sizes—so we must take limits (as M and T approach 0) very carefully. Taking limits to ensure finiteness of mean and variance, we find that the particle c obeys the diffusion equation; this equation is discussed later in greater detail. We now discuss the mathematics of the diffusion equation in an informal manner.

Consider a particle c moving along a straight line. Let its position at time t be $x(t)$ (see Figure 27). Let the displacement of the particle in the interval t, $t + dt$ be, $dx(t)$, where

$$dx(t) = x(t + dt) - x(t)$$

Assume that $dx(t)$ is normally distributed with mean $\beta \cdot dt$ and variance $\alpha \cdot dt$. It is helpful to picture the process in one's mind; towards this end, imagine an arbitrarily large number of independent particles that move about according to the above assumptions. Suppose all the particles are at a point y at time t (Figure 28). Then at time $t + dt$ the particles would have moved, some one way and some in the opposite direction. The function showing the density of particles around a given point at time $t + dt$ has the familiar bell shape with a mean at $y + \beta \cdot dt$ and a variance of $\alpha \cdot dt$ (Figure 29). Our particle is *memoryless* in the sense that its future displacement, relative to its current position, is independent of the past.

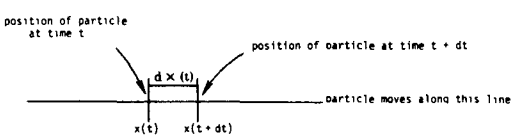


FIGURE 27. A particle moving along a straight line.

Let $p(x_0, x; t)$ be the density function for the process $x(t)$, given that $x(0) = x_0$. We may picture $p(x_0, x; t)$ in the following way: consider an arbitrarily large number of particles all of which start out at position x_0 at $t = 0$. The particles will be scattered along the real line at time $t > 0$. $p(x_0, x; t)$ is a function in x and t that shows the density of particles around a point x at time t. This density function $p(x_0, x; t)$ obeys the Fokker-Planck diffusion equation [Cox65]. A great deal of work has been done on the diffusion equation. Our goal is to use this process to model a queueing network.

To summarize our introduction to diffusion processes: particle d's position represents the number of jobs in the $GI/G/1$ queue. Particle d takes *discrete* jumps (of $+1$ or -1) and (in general) *d has memory* in addition to its current position. There is no method (at this time) to compute the probability distribution of $n(t)$, which is d's position at time t. We wish to deduce the

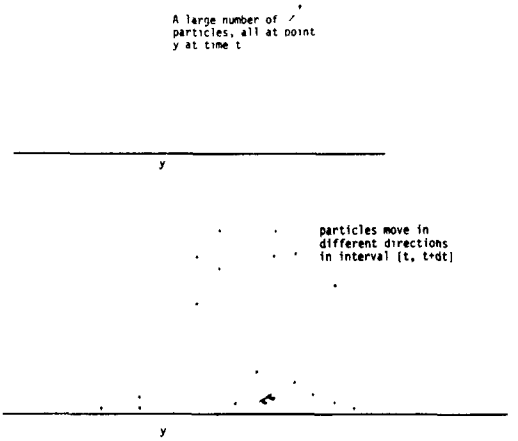


FIGURE 28.

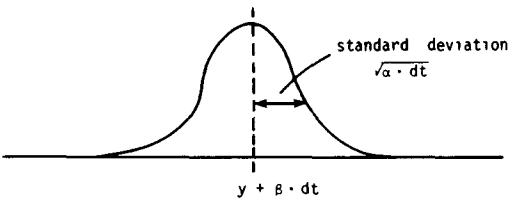


FIGURE 29.

behavior of d from the behavior of particle c that has simpler properties. Particle c moves along the *continuous* real line. Furthermore, c *has no memory* (except its position relative to the boundary). The position $x(t)$ of c satisfies the diffusion equation, which has been studied in depth; methods exist to compute the probability distribution of $x(t)$. Note also that *transient* (time dependent) analysis of the behavior of c is tractable.

Mapping Between Queueing and Diffusion Processes

We wish to deduce the behavior of particle d from the behavior of particle c. To do this, we shall simulate the behavior of d by a particle d^* that also jumps between points 0, 1, 2, ... but whose movement is driven by the movement of c, in the following way. Partition the real line into intervals; place d^* in position i when c is in the ith interval. When c moves to the $i-1$th (or $i+1$th) interval, move d^* to position $i-1$ (or $i+1$). Statistics regarding d^* are said to be diffusion approximations of the corresponding statistics regarding d.

The accuracy with which d^* models d depends upon:

1) How values are assigned to the parameters α and β that characterize the diffusion process (and hence characterize the movement of particle c).
2) How the real line is partitioned into intervals (recollect d^* is placed in the ith position when c is in the ith interval).
3) How we place a boundary condition on the diffusion process; typically we want c to move on the non-negative real line just as d does. There are different conditions we might place at the boundary $x = 0$ so as to ensure $x(t) \geq 0$ for all t. These boundary conditions affect the behavior of c and thus also the behavior of d^*.

We shall analyze each of these issues in turn.

Setting α and β

To make the computation of α and β tractable we shall make the (invalid) assumption that the queue is never empty. This assumption is more reasonable in "heavy traffic" conditions when the server's utilization approaches 1 (one), and the diffusion approximation tends to give better results under heavy traffic conditions.

Consider an incremental time interval $[t, t + dt]$. Let $n(t)$ be the queue length at time t. During this interval the expected number of jobs to arrive is $\lambda \cdot dt$ where λ is the arrival rate and the expected number of departures is $\mu \cdot dt$, where μ is the service rate. Hence

$$E[n(t + dt) - n(t)] = (\lambda - \mu) \cdot dt$$

We want the position $x(t)$ of particle c to reflect the queue length $n(t)$. Note from the earlier discussion that the incremental displacement $x(t + dt) - x(t)$ is a random variable with a mean of $\beta \cdot dt$. Hence, it is reasonable to set:

$$\beta = \lambda - \mu$$

By similar (though more complex) arguments we set:

$$\alpha = c_a\lambda + c_s\mu$$

where c_a and c_s are the squared coefficients of variation of the interarrival and service times respectively.

Selecting Intervals

A reasonable heuristic is to place d^* in the ith position when c is between i and $i + 1$ (Figure 26). Using this method of selecting intervals,

$$P^*(n_0, n;t) = \int_n^{n+1} P(x_0, x;t)dx$$

Boundary Conditions

The *reflective barrier* is the boundary condition normally used [Cox65]. This boundary condition states that the particle c must always be on the non-negative portion of the real line.

$$\int_0^{+\infty} p(x_0, x; t)dx = 1$$

The diffusion equation with this boundary condition has been solved and we can compute $p^*(n_0, n; t)$. We are primarily interested in the equilibrium queue length distribution:

$$p^*(n) = p^*(n_0, n, \infty)$$

Using the methods discussed previously for setting α and β, the boundary condition, and for selecting intervals, we get ([GAVE68])

$$p^*(n) = (1 - \hat{\rho}) \cdot \hat{\rho}^n, n = 0, 1, 2, \ldots \quad (6.1)$$

where $\hat{\rho} = \exp\{-2(1-\rho)/(c_s + c_a\rho)\}$ and ρ is the utilization. Let $p(n)$ be the equilibrium probability of n jobs in queue. We know that the fraction of time the server is idle is $p(0) = 1-\rho$, whereas we have $p^*(0) = 1-\hat{\rho}$, which is erroneous. Hence a good heuristic is to use another approximation:

$$\hat{p}(n) = \begin{cases} 1 - \rho & \text{if } n = 0 \\ \rho \cdot (1 - \hat{\rho})\, \hat{\rho}^{n-1} & \text{if } n \geq 1 \end{cases} \quad (6.2)$$

Gelenbe uses a more reasonable boundary condition [GELE75]. He assumes that when particle c hits the boundary $x = 0$, it sticks there for an exponentially distributed time after which it jumps back instantaneously into the region $x > 0$ with some probability density function. For instance, we may assume that c makes an instantaneous jump from $x = 0$ to the point $x = 1$, representing the arrival of a new job. We may assume that the mean time the process remains at the boundary $x = 0$ is $1/\lambda$, where λ is the arrival rate. With these assumptions Gelenbe reports improved results.

Networks

Consider a network with a single chain of jobs and K queues. Let n_i be the number of jobs in queue i, $i = 1, \ldots, K$, and let $p(n_1, \ldots, n_K)$ be the equilibrium probability obtained by the diffusion approximation that there are n_i jobs in queue i, $i = 1, \ldots, K$. In the pioneering work on networks by Kobayashi it was shown that given the usual assumptions for setting up the diffusion process,

$$\hat{p}(n_1, \ldots, n_K) = \frac{1}{G} \prod_{i=1}^{K} \hat{p}_i(n_i) \quad (6.3)$$

where G is a normalization constant for closed networks, and $G = 1$ open networks [KOBA74]. Note the product form of the network state probabilities.

For open networks we may estimate the coefficients of variation of the arrival process into each queue by the methods discussed earlier in Section 3, and then analyze each queue independently assuming independent interarrival times with the estimated coefficient of variation and the specified arrival rate. The flow-equivalent of the complement is represented by a job source that has independent identically distributed service times.

For closed networks, Reiser and Kobayashi [REIS74] use the product form method discussed in Section 4. With this method we analyze each queue independently assuming a (simple) model for the complement of the queue, compute $\hat{p}_i(n_i)$, and then use equation (6.2) to compute steady-state probabilities. We then compute network state probabilities assuming product form. Performance estimates are computed from the state probabilities $\hat{p}(n_1, \ldots, n_K)$ rather than directly from the $\hat{p}_i(n_i)$. For closed networks (unlike open networks), it is not possible to determine the arrival rate into a queue without carrying out a detailed analysis of the entire network. One solution is to estimate arrival rates (or, equivalently, throughputs) assuming exponential service times, and hence product form, and to then use the formula (6.2) for single server queues.

An alternative solution to estimating the arrival rates is to assume that the server with the highest utilization in the network is busy all the time (i.e., its utilization is 1) and then the throughput (arrival rate) of that queue is equal to its service rate; the throughputs for all other queues can be computed from any one throughput [DENN78]. This alternative method for estimating arrival rates is appropriate when the number of jobs in the network is very large.

Other Applications of the Diffusion Approximation: an Example

Foschini uses the diffusion approximation to solve routing problems [FOSC77]. Consider a system with a single source and two or more parallel queues (Figure 30). An arriving job joins the shorter queue. This system has no simple analytic solutions. Foschini has an ingenious application of the diffusion approximation to this problem;

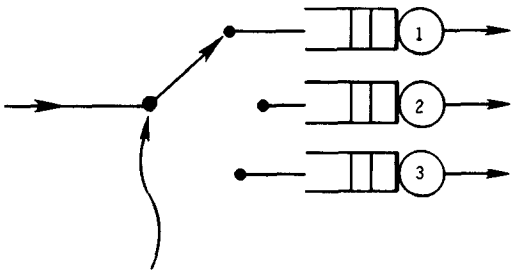


FIGURE 30 Dynamic routing

his paper is required reading for those who want to apply diffusion approximations to atypical problems.

The diffusion approximation is being applied to an increasing variety of problems. Its greatest value is in analyzing open networks with nonexponential service times in heavy traffic conditions because it is often difficult to use numerical techniques, other approximations, or simulations in those cases. A great deal of computing time is required in simulations to estimate equilibrium performance measures in open networks under heavy traffic conditions [LAVE75]. The other approximations are either not applicable to open networks, or assume exponential service times when used on open networks.

Diffusion approximation formulas developed for single queues will continue to be used in submodels within larger network models. The greatest barrier to increased usage of diffusion approximations is the difficulty in developing the mathematics of *new* problems. Sophisticated mathematics is not required for other techniques, such as flow-equivalence or iteration; any systems analyst can try these techniques on a new problem and then find out empirically whether they work. New results on diffusion approximations will continue to be generated by a small number of researchers and these results will later be used by the modeling community. The greatest emphasis in research is placed on models in which other methods fail. The solution of new problems based on diffusion approximations is still research; new applications of the other techniques are becoming commonplace.

7. CONCLUSION

A number of approximation methods for analysis of computing systems have been discussed here. New methods are being developed at many research centers. Despite the wealth of proposed techniques, there are currently only a few fundamental ideas in this area; let us review them.

Core Ideas

1) *Characteristics of networks with tractable solutions.* What types of networks have tractable solutions? We answered this in terms of state space size, state transition structure, and product form conditions. We made a passing reference to simulation.
2) *Flow-equivalent systems.* The concept of flow-equivalence plays a key role in approximations. The idea was defined and several examples were presented to illustrate the idea.
3) *The use of flow-equivalence in obtaining exact solutions.* If a network satisfies product form then any subnetwork can be replaced by its flow-equivalent without altering equilibrium conditions.
4) *The use of flow-equivalence in approximations.* Several subsystems may be replaced by a single flow-equivalent to reduce the complexity of analyzing a given network. Critical questions are how to determine a) the service times, b) the service distributions, and c) the queueing disciplines of the flow-equivalent. The literature was surveyed with regard to these questions.
5) *Iteration.* This method attempts to improve upon flow-equivalent schemes. Each queue in a network is analyzed independently by studying a simple 2-queue model consisting of the queue in question and a composite queue that represents the rest of the network. If the independent analyses mesh with each other, as determined by checking certain invariant properties of the networks, then the iteration stops. Otherwise, the composite

queues are modified to ensure a better fit.

6) *Product form methods.* This method also attempts to improve upon flow-equivalent schemes. The method works in three stages. In the first, each queue is analyzed independently, as above. In the second, the steady state probabilities of the entire network are computed by multiplying the state probabilities of the individual queues. In the third stage, the steady state probabilities of the network are summed appropriately to get estimates of the required performance measures. Product form and iterative methods may be combined.
7) *Diffusion approximations.* In some cases it is helpful to model a non-Markovian discrete queueing system by a continuous Markov (diffusion) process. The key points in this method are a) obtaining the parameters of the diffusion process, b) setting appropriate boundary conditions, and c) mapping the continuous process back onto the discrete process.

A Comparison of Analysis Techniques

The analyst should have a preferred approach for every problem encountered. Unfortunately, a set of recipes for cooking up the best solution to a problem does not now exist. The approach first attempted on a problem is a *matter of personal preference*; it also depends upon:

1) The frequency with which the problem is likely to recur,
2) The number of configurations that need to be analyzed,
3) The level of sophistication of the client (i.e., the person who *uses* the design—a designer or manager) in analysis methodology. (This is often the key factor),
4) The computer-aided design tools (programming packages) available to the analyst,
5) The time available to carry out the analysis,
6) The number of analysts available, and
7) The available data.

In our experience, the exigencies of the situation, rather than scientific methodology, dominate the selection of the approach. Consider each of the seven factors in turn.

Frequency of Occurrence

A performance analysis package may be designed to aid in the configuration of a specific family of computing systems. This package may be used several times; for instance, if it is developed by the vendor, it may be used by systems engineers each time a bid is made. In such cases, new methods or variants of the methods proposed here are developed specifically for the family of interest. There is generally sufficient time to develop new techniques suited for the class of problems of concern, and a relatively sophisticated modeling study, in which several techniques are compared, can be undertaken.

A systems analyst may be asked to analyze a single installation. In such cases we favor the use of existing modeling programming packages. If the problem is such that existing packages cannot be used, we recommend the direct use of flow-equivalence approximations, perhaps incorporated in the product form method. Flow-equivalence is an intuitive notion and is easily used. Product form methods are straightforward, do not require much computer time, and may improve accuracy. Iterative methods are less straightforward and so take longer to develop; hence we do not recommend them in such cases.

Number of Design Alternatives Considered

Several (possibly thousands of) alternatives may have to be compared for a given design. Very fast (and therefore relatively inaccurate) techniques must be used if the number of alternatives is large. The goal here is not to predict performance accurately, but to discard a large proportion of poor designs. When the number of options is small, simulation or measurement, if possible, are the preferred approaches. Several methods may be used in evaluating a particular design: relatively fast, inaccurate methods initially and slower, accurate methods later.

Client's Sophistication in Analysis Methods

A systems analyst is effective only when he can convince the designer or manager of a computing system that the results of his models are correct. There are cases of computing systems installations where results of modeling studies have been ignored to the detriment of the installations. An analyst's job does not stop with the analysis of a model; he must persuade the managers of the installation to make the decisions suggested by his analysis. A manager who has not been exposed to analytic models may not be convinced by analytic models, especially if the results of the models suggest that the manager has made wrong decisions. The analyst may be forced to use expensive simulations to help persuade management of the correctness of his results, even when the analyst himself would prefer to use other techniques.

Analysis Tools

The availability of programming packages plays a key role in the selection of methods, especially for problems that are not likely to recur. There are several packages for analyzing product form networks and two (RESQ [REIS78] and ASQ [INFO75] that couple product form methods with flow-equivalent approximations. Simulation languages, such as RESQ, based on queueing network structural models are particularly helpful in modeling computing systems [SAUE77c].

Time

Most one-of-a-kind analysis problems that arise in systems design have to be carried out under intense time pressure. Furthermore, the problem itself is generally a moving target, since most designs evolve and design variables change. There is insufficient time to develop special purpose analytic methods. In such cases the analyst has to resort to existing programming packages. The authors have found flow-equivalent methods to be particularly useful in this environment, since changes in the design of one subsystem usually affect only the flow-equivalent of that subsystem. If the entire analysis is one monolithic block, then changes in the design of a single subsystem can ripple throughout the analysis.

Number of Analysts

In one-of-a-kind design problems the urgency of the problem may result in several analysts being assigned to solve a complex problem, with the expectation that the problem will be solved in short time. It is vitally important to partition the problem so as to allow many analysts to work in parallel. Flow-equivalence provides a natural vehicle for partitioning.

Available Data

An analyst generally uses complex approximation methods or simulations because simpler models ignore some aspects of real systems. For instance, an analyst may decide that simple models that ignore service time distributions are inadequate. However, there are cases where measurements that yield service time *distributions* are not available, though mean values are measured. In these cases the only point in using more sophisticated models is to determine the sensitivity of the results to variations in the unknown service distribution.

The Authors' Biases

Ultimately, the selection of analysis techniques is a matter of personal preference; therefore, the reader should be aware of the authors' biases. In our opinion, analysis plays a role in the design process, but it is only one of the many aspects of design. In the natural sciences one of the goals of theory is to predict natural phenomena very accurately. In computer systems analysis, the goal is not to obtain a high degree of precision in prediciting phenomena, but rather to use analysis to recognize and discard poor design choices. The systems analyst must often rely on intuition, use methods that are not rigorous, give priority to pragmatic and political considerations (rather than to scientific method) in approaching problems, and realize that systems analysis (and the development of approximation methods in particular) is an art, not a science.

APPENDIX: An Example of a Complex Flow-Equivalent Queue

Consider a network with 4 queues Q_1, Q_2, Q_3, Q_4, 8 classes, and 2 closed chains, one consisting of classes {1, 2, 3, 4} and the other of classes {5, 6, 7, 8}. Assume 1 job in each chain. Assume that Q_1 and Q_2 have processor sharing disciplines. Assume mean service times for classes 1 and 2 are each 1 second, and for classes 5 and 6 are each 2 seconds; others are arbitrary. The class transition diagrams are shown in Figure A.1. The classes accepted and output by each queue are shown in Table A.I. Note that the classes output must match the classes input in the class transition diagram.

Now let us construct a queue Q that is flow-equivalent to the queue obtained by combining Q_1 and Q_2. The network obtained by feeding the output of Q_1 and Q_2 back to itself is shown in Figure A.3. Any queue Q' that accepts classes 1, 5, and 6, and outputs classes 3, 7, and 8 at the same throughputs as Q in the same network (Figure 13) is flow-equivalent to Q. For example consider a system Q' that accepts a class 1 job, services it, and then outputs it as a class 3 job. Q' also accepts class 5 and 6 jobs, services them, and outputs them as class 7 or 8 jobs with equal probability. Let $S_i(n_1, n_2)$ denote the mean service time to class i jobs, $i = 1, 5, 6$, in Q' when the number of jobs in chain $j, j = 1, 2$, is n_j. Then set

$$S_1(n_1, n_2) = 1/x_3(n_1, n_2)$$
$$S_5(n_1, n_2) = S_6(n_1, n_2)$$
$$= 1/((x_7(n_1, n_2) + x_8(n_1, n_2)))$$

Let p_{ik} be the frequency that a class i job becomes a class k job after receiving service from Q'. Then

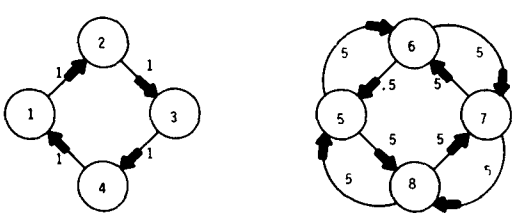


FIGURE A.1 Class transitions.

TABLE A I.

Queue No.	*Classes Accepted*	*Classes Output*
1	1,5	2,8,6
2	2,6	3,5,7
3	3,7	4,6,8
4	4,8	1,7,5

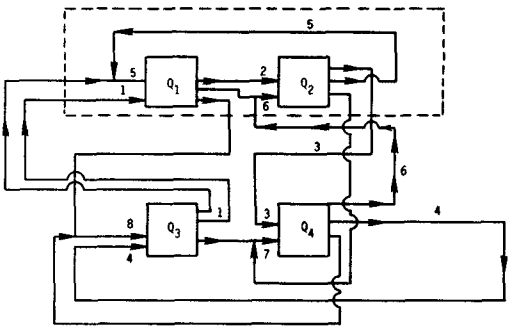


FIGURE A.2. Given network.

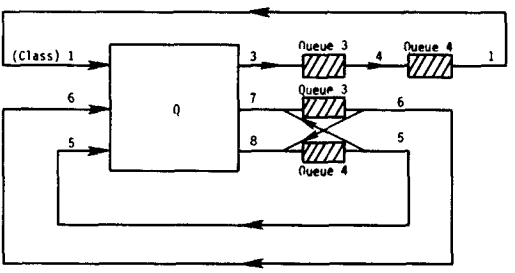


FIGURE A.3. Offline system.

$$P_{13} = 1$$
$$P_{57} = P_{67} = 1/2 \text{ and}$$
$$P_{58} = P_{68} = 1/2$$

In a similar manner, flow-equivalence applies to arbitrary complex networks.

REFERENCES

BARD77 BARD, Y. "The modeling of some scheduling strategies for an interactive computer system," in *Computer performance*, K. M Chandy and M. Reiser (Eds.), Elsevier North-Holland, Inc., New York, 1977, pp. 113-138.

BARD78 BARD, Y "The UM/370 performance predictor," *Comput. Surv.* **10,** 3 (Sept. 1978), 333-342.

BASK75 BASKETT, F., CHANDY, K.M., MUNTZ, R.R.; AND PALACIOS-GOMEZ, F. "Open, closed and mixed networks of queues with different classes of customers," *J. ACM* **22,** 2 (April 1975), 248-260

BROW75 BROWNE, J. C.; CHANDY, K. M.; BROWN, R. M.; KELLER, T. W., TOWSLEY, D. F., AND DISSLEY, C W. "Hierarchical techniques for development of realistic models of complex computer systems," *Proc. IEEE* **63,** (June 1975), 966-975

BROW77 BROWN, R. M.; BROWNE, J. C.; AND CHANDY, K. M. "Memory management and response time," *Commun. ACM* **20,** 3 (March 1977), 153-165.

BUX77 BUX, W.; AND HERZOG, U. "The phase concept: approximation of measured data

and performance analysis," in *Computer performance*, K.M. Chandy and M. Reiser (Eds.), Elsevier North-Holland, Inc., New York, 1977, pp. 23–38.

CHAN72 CHANDY, K. M. "The analysis and solutions for general queueing networks," in *Proc. 6th Annual Princeton Conf Information Science and Systems*, 1972, pp. 224–228.

CHAN75a CHANDY, K. M.; HERZOG, U.; AND WOO, L S. "Parametric analysis of queueing networks," *IBM J. Res Dev* **19**, (Jan. 1975), 36–42.

CHAN75b CHANDY, K. M , HERZOG, U , AND WOO, L. S. "Approximate analysis of general queueing networks," *IBM J. Res Dev.* **19**, (Jan. 1975), 43–49.

CHAN77 CHANDY, K M.; HOWARD, J H.; AND TOWSLEY, D F. "Product form and local balance in queueing networks," *J ACM* **24**, 2 (April 1977), 250–263.

CHIU78 CHIU, W. W.; AND CHOW, W. M. *A hybrid hierarchical model of a multiple virtual storage (MVS) operating system*, RC-6947, IBM Research, Yorktown Heights, N.Y., Jan. 1978.

COUR75 COURTOIS, P. J. "Decomposability, instabilities and saturation in multiprogramming systems," *Commun. ACM* **18**, 7 (July 1975), 371–376

COUR77 COURTOIS, P J. *Decomposability queueing and computer system applications*, Academic Press, Inc , New York, 1977.

COX55 COX, D. R. "A use of complex probabilities in the theory of stochastic processes," *Proc. Cambridge Philos. Soc.* **51**, (1955), 313–319

COX65 COX, D R., AND MILLER, H. D. *The theory of stochastic processes*, John Wiley and Sons, Inc., New York, 1965.

DENN76 DENNING, P. J., AND KAHN, K. C. "An L = S criterion for optimal multiprogramming," in *Proc Int Symp. Computer Performance Modeling, Measurement and Evaluation*, 1976, P.P.S. Chen and M. Franklin (Eds), ACM, New York, and IFIP, Geneva, pp. 219–229.

DENN78 DENNING, P J., AND BUZEN, J. P. "The operational analysis of queueing network models," *Comput. Surv.* **10**, 3 (Sept 1978), 225–261.

DISN74 DISNEY, R. L ; AND CHERRY, W. P "Some topics in queueing network theory," in *Mathematical methods in queueing theory*, A. B. Clarke (Ed.), Springer-Verlag New York, Inc., New York, 1974.

DRAK67 DRAKE, A W. *Fundamentals of applied probability theory*, McGraw-Hill, Inc , New York, 1967

FOSC77 FOSCHINI, G. J. "On heavy traffic diffusion analysis and dynamic routing in packet switched networks," in *Computer performance*, K.M. Chandy and M. Reiser (Eds.), Elsevier North-Holland, Inc., New York, 1977, pp. 419–514.

FOST74 FOSTER, D. V.; MCGEHEARTY, P. F., SAUER, C. H.; AND WAGGONER, C. N. "A language for analysis of queueing models," in *Proc. Fifth Annual Pittsburgh Modeling and Simulation Conf.*, 1974, pp. 381–386.

GAVE68 GAVER, D. P. "Diffusion approximations and models for certain congestion problems," *J. Appl. Probab.* **5**, (1968), 607–623.

GAVE76 GAVER, D. P.; AND HUMFELD, G. "Multitype multiprogramming probability models and numerical procedures," *Computer performance*, Elsevier North-Holland, Inc., New York 1976, pp. 38–43

GELE75 GELENBE, E. "On approximate computer system models," *J. ACM* **22**, 2 (April 1975), 261–269.

GELE76 GELENBE, E.; AND PUJOLLE, G "The behavior of a single queue in a general queueing network," *Acta Inf* **7**, (1976), 123–136.

GORD67 GORDON, W. J.; AND NEWELL, G F. "Closed queueing networks with exponential servers," *Oper. Res.* **15**, (1967), 254–265.

HERZ75 HERZOG, U., WOO, L S.; AND CHANDY, K. M "Solution of queueing problems by a recursive technique," *IBM J. Res Dev.* **19**, (May 1975), 295–300.

IGLE78 IGLEHART, D. L "The regenerative method for simulation analysis," in *Current trends in programming methodology, Vol. III software modeling and its impact on performance*, K. M. Chandy and R. T. Yeh (Eds.), Prentice-Hall, Inc., Englewood Cliffs, N J., 1978, pp 52–71.

INFO75 INFORMATION RESEARCH ASSOCIATES, *User's manual for the ASQ system*, I R A , Austin, Texas, 1975.

IRLA75 IRLAND, M. "Queueing analysis of a buffer allocation scheme for a packet switch," in *Proc. National Telecommunications Conf., IEEE*, New York, 1975, p 24

JACK63 JACKSON, J R "Jobshop-like queueing systems," *Manage. Sci* **10**, (1963), 131–142

KELL73 KELLER, T W. *ASQ user's manual*, TR-27, Computer Science Dept., Univ Texas at Austin, Texas, 1973.

KELL76 KELLER, T. W "Computer systems models with passive resources," PhD Thesis, Univ. Texas at Austin, 1976.

KOBA74 KOBAYASHI, H "Application of the diffusion approximation to queueing networks I: equilibrium queue distributions," *J. ACM* **21**, 2 (April 1974), 316–328.

KOBA78 KOBAYASHI, H. *Modelling and analysis an introduction to system performance evaluation methodology*, Addison-Wesley Publ Co , Reading, Mass , 1978, Ch 4

LAM76 LAM, S. S. "Store-and-forward buffer requirements in a packet switching network," *IEEE Trans Commun* **24**, (April 1976), 394–403.

LAVE75 LAVENBERG, S. S , AND SLUTZ, D R. "Introduction to regenerative simulation," *IBM J. Res. Dev* **19**, (Sept 1975), 458–463.

LAVE77 LAVENBERG, S S , AND SAUER, C H. "Sequential stopping rules for the regenerative method of simulation," *IBM J Res Dev.* **21**, (Nov. 1977), 545–558

LAZO77 LAZOWSKA, E D. "The use of percentiles in modeling CPU service time distributions," in *Computer performance*, K.M. Chandy and M Reiser (Eds.), Elsevier North-Holland, Inc., New York, 1977, pp. 53–66.

MACN75 MACNAIR, E. A ; AND WOO, L S. Private communication, 1975.

REIS74 REISER, M., AND KOBAYASHI, H. "Accuracy of the diffusion approximation for some queueing systems," *IBM J. Res Dev.* **18,** (1974).

REIS75 REISER, M., AND KOBAYASHI, H. "Queueing networks with multiple closed chains theory and computational algorithms," *IBM J Res. Dev* **19,** 3 (May 1975), 283-294.

REIS76 REISER, M "Interactive modeling of computer systems," *IBM Syst J* **15,** (1976), 309-327.

REIS78 REISER, M ; AND SAUER, C H. "Queueing network models methods of solution and their program implementation," in *Current trends in programming methodology, Vol. III software modeling and its impact on performance,* K. M Chandy and R. T. Yeh (Eds.), Prentice-Hall, Inc., Englewood Cliffs, N J , 1978, pp. 115-167.

SAUE75a SAUER, C. H. "Configuration of computing systems: an approach using queueing network models," PhD Thesis, Univ Texas at Austin, May 1975.

SAUE75b SAUER, C. H , AND CHANDY, K. M "Approximate analysis of central server models," *IBM J. Res. Dev* **19,** (May 1975), 301-313

SAUE76 SAUER, C H., WOO, L S , AND CHANG, W *Hybrid analysis/simulation· distributed networks,* RC-6341, IBM Research, Yorktown Heights, N Y., June 1976

SAUE77a SAUER, C H. *Confidence intervals for queueing simulations of computer systems,* RC-6669, IBM Research, Yorktown Heights, N.Y , July 1977

SAUE77b SAUER, C. H , AND CHANDY, K M. *The impact of distributions and disciplines on multiple processor systems,* RC-6621, IBM Research, Yorktown Heights, N Y , July 1977. Also to appear in *Commun. ACM*

SAUE77c SAUER, C H., AND MACNAIR, E A. *Computer/communication system modeling with extended queueing networks,* RC-6654, IBM Research, Yorktown Heights, N Y., July 1977.

SAUE78 SAUER, C H , AND MACNAIR, E A *Queueing network software of systems modeling,* RC-7143, IBM Research, Yorktown Heights, N.Y , May 1978.

SCHW78 SCHWETMANN, H. D "Hybrid simulation models of computer systems," to appear in *Commun. ACM,* 1978.

SEKI71 SEKINO, A *Performance evaluation of multiprogrammed time-shared computer systems,* Proj. MAC TR-103, MIT, Cambridge, Mass., Sept 1971

SEVC77a SEVCIK, K C "Priority scheduling disciplines in queueing network models of computer systems," in *Proc IFIP Congress 77,* North-Holland Publ. Co., Amsterdam, pp. 565-570

SEVC77b SEVCIK, K C., LEVY, A I., TRIPATHI, S. K., AND ZAHORJAN, J L. "Improving approximations of aggregated queueing network subsystems," in *Computer performance,* K.M Chandy and M. Reiser (Eds.), Elsevier North-Holland Inc., New York, 1977, pp. 1-22

SHUM77 SHUM, A., AND BUZEN, J. P. "The EPF technique. a method for obtaining approximate solutions to closed queueing networks with general service times," in *Measuring modeling & evaluating computer systems,* H. Beilner and E. Gelenbe (Eds.), North-Holland Publ. Co , Amsterdam, 1977, pp. 201-220.

STEW78 STEWART, W. J "A comparison of numerical techniques in Markov modeling," *Commun. ACM* **21,** (Feb. 1978), 144-151.

TOWS75 TOWSLEY, D. F. "Local balance models of computer systems," PhD Thesis, Univ Texas at Austin, Dec. 1975.

WALL66 WALLACE, V. L., AND ROSENBERG, R. S "Markovian models and numerical analysis of computer system behavior," in *Proc. 1966 AFIPS Spring Jt. Computer Conf.,* Vol 28, Spartan Books, Washington, D C., pp 141-148

WOLF77 WOLFF, R W. "The effect of service time regularity on system performance," in *Computer performance,* K.M. Chandy and M. Reiser (Eds.), Elsevier North-Holland, Inc , New York, 1977, pp. 297-304.

WONG78 WONG, J W. Queueing network modeling of computer communication networks," *Comput. Surv* **10,** 3 (Sept. 1978), 343-351.

ZAHO77 ZAHORJAN, J. L. "Iterative aggregation with global balance," Project SAM Notes, Univ. of Toronto, Toronto, Ont , Canada, Feb 1977.

RECEIVED FEBRUARY 6, 1978; FINAL REVISION ACCEPTED JUNE 7, 1978