Yonathan Bard
Charles H. Sauer

IBM Contributions to Computer Performance Modeling

Performance modeling can be used throughout the life of a computer system, from initial design, through implementation, configuration (and reconfiguration) and even tuning. Performance models are usually solved by numerical techniques, where possible, and by simulation, otherwise. This paper summarizes IBM's contributions to performance modeling and the solution of performance models.

Introduction

There are many reasons for wishing to model the performance of a computer system. When a brand-new system is being designed or implemented, modeling is the only way to determine whether performance specifications will be met. When a system is being configured for a specific workload, modeling is the economical way to narrow down the search among the myriad possible configurations. Modeling is required for capacity planning to meet anticipated increases in an installation's workload. Even in the day-to-day management of a computer installation, modeling may indicate which of many possible tuning actions can best alleviate performance bottlenecks.

Performance modeling consists of estimating the values of system performance parameters, given descriptions of the system's configuration and workload. The performance parameters most commonly modeled are the response times to service requests of various types (transactions, jobs, messages), throughputs (number of requests completed per unit time), utilizations (percent of time system components are busy), and queue lengths (number of requests waiting for service at various system components). Depending on the modeling method employed, one obtains anything from mere average values of these quantities to full probability distributions.

In the following sections we shall first enumerate the general classes of modeling methods. We shall then give a brief summary of queueing network theory, which is the basis of most computer performance models. We then summarize the contributions of IBM to analytic and simulation models of computer systems.

It is impossible to do justice here to all IBM contributions to this area. We have chosen to concentrate our attention on the "main line" area of queueing network models, thereby excluding much important work on subjects like priority queueing, program behavior, and teleprocessing system modeling. We have also excluded the extensive work on aspects of performance analysis not directly related to modeling. We apologize to all authors whose work should have been listed but was not, due to our oversight or ignorance.

Apart from making specific research contributions, IBM authors have also published some books covering aspects of modeling [1-4]. These books contain many case studies which demonstrate the practical value of the modeling technology—another subject which is beyond the scope of this paper. The reader not familiar with queueing network models and their applications may also wish to consult special issues of *Computing Surveys* (Vol.

10, September 1978) and *Computer* (Vol. 13, April 1980) for general discussion.

Modeling methods

Computer performance models may be of the empirical, analytic, or simulation types, with various hybrids possible. An *empirical* model is constructed by fitting relatively simple equations, usually linear, to measured performance data. The usefulness of this method is limited, since such equations can generally be applied only under conditions not too different from those previously observed. Such models have been used, for example, to describe the relation between system overhead and the functions that the system performs [5] and to evaluate the effectiveness of system modifications [6]. Methods for constructing and fitting such models are discussed in [7].

An *analytic* model consists of a set of equations which are thought to capture, at least approximately, the relevant aspects of the system's behavior. Such equations can sometimes be solved explicitly, but more often than not some numeric methods are required to evaluate the solutions. Analytic models, which will be described in some detail below, have proven extremely useful for computer performance analysis: they are fast, relatively easy to program, and they produce more than acceptable accuracy in many cases. However, they generally yield only equilibrium average values of the performance measures, and there are still many problems which are analytically intractable. Even when good approximations are available, these must be continually validated, either against real data or against simulation models.

A *simulation* model is a computer program, designed to imitate the behavior of the real system in some detail. A computer system simulator tracks the progress of each job or transaction through the various system components, accumulating performance statistics as it goes along. Simulators have found wide application in the modeling of computer system performance, as well as in the validation of analytic models. They are capable of almost unlimited accuracy, depending on the amount of detail that is built into them. They can produce estimates of entire distributions of performance measures, not just averages. They can be used to study transient, as well as equilibrium, system behavior. However, they may be difficult to implement and validate, and their running may consume large amounts of computing resources. By and large, simulation should be considered only when a suitable analytic model is not available.

Computers as networks of queues

A computer system may be viewed as a set of components, each of which is capable of providing certain services. For instance, the CPU executes instructions, the channels transmit data, and the I/O devices both store and transmit data. A job or transaction submitted to the system may be viewed as a set of service requests, which must be executed by these components in a certain order. When the transaction requires service from a system component, it may find the latter busy serving some other transaction. Our transaction must then queue up and wait for its turn. The performance of the system is determined primarily by the length of service and queueing times at each component, and by the rules controlling the sequence of transitions from component to component. For performance modeling purposes, the system may therefore be viewed as a *network of queues*. It is not surprising that most performance models, both analytic and simulation, are basically queueing network models.

Analytic models

As implied by the foregoing, analytic computer performance models are based primarily on *queueing network theory*. This theory originated in the work of Jackson [8] and Gordon and Newell [9]. The simplest nontrivial queueing network model, the so-called "machine repairman model," was applied to computer system modeling by Scherr [10]. The first full-fledged applications of the theory to our field are due to Buzen [11] and Arora and Gallo [12]. This theory deals with a fixed number of customers (*e.g.*, jobs or transactions) circulating among a set of queues, constituting a so-called *closed* network. An *open* network is one where customers are allowed to arrive from and depart to the outside world. Subsequent work, by Baskett *et al.* [13], by Reiser and Kobayashi at IBM [14, 15], by Chandy *et al.* [16], by Lam at IBM [17], and by Towsley [18], has greatly expanded the scope of the theory, so that it now encompasses networks with

- Several customer classes, each with its own service and routing requirements.
- Mixed networks, *i.e.*, networks where some customers are fixed within the network and others are allowed to arrive and depart.
- Servers of several types, including processor sharing, infinite server, last come first served preemptive resume, and first come first served (in the last case, all customer classes must have identical exponentially distributed service times).
- Servers with variable service rates, depending on queue length and subnetwork population.
- Routing probabilities depending on queue length and subnetwork population.
- Routing probabilities depending on previous routing of a job.
- Arrival rates and loss functions dependent on network population.

However, some restrictions still apply, among which are the following:

- A customer can request service only from one server at a time.
- Priority scheduling is not allowed.
- Successive transaction interarrival and service times must be statistically independent.

The main result of the theory is that, for those networks satisfying the restrictions, the probability of each network state is proportional to the product of terms, each involving the state of a single queue. By "state" we mean the number of customers of each class receiving or waiting for service at each queue. Each of these terms consists of a simple function of the state of its queue, and of the total average service time required by each customer class at that queue. Because of the form of the state probabilities, it is referred to as the *product form solution*, and networks satisfying the required conditions are called *product form networks*. Once the state probabilities have been computed, all the required performance measures can be easily calculated.

To explicitly evaluate the product form solution, one requires a proportionality constant, whose function is to ensure that all state probabilities sum to unity. The evaluation of this so-called *normalizing constant* is quite trivial for open networks, but can become quite burdensome when a closed network has many customer classes. In fact, from a computational point of view, analyzing the performance of a closed (or mixed) network is more or less equivalent to evaluation of the normalizing constant (although some algorithms do not evaluate that constant explicitly).

The main threads in the development of queueing network applications are

- Finding efficient algorithms for obtaining the exact product form solution.
- Finding approximate solutions for non-product form networks.
- Finding fast algorithms for obtaining approximate solutions to a large variety of networks, both product form and not.

IBM has made significant contributions to all three areas. These are described in the following sections.

- *Exact algorithms for product form networks*

Buzen [19] originated what later became known as the *convolution algorithm* for computing the normalizing constant. If $G(n, m)$ is the normalizing constant for a network with n servers and m jobs, then the algorithm uses a convolution formula of the form

$$G(n, m) = \sum_i G(n - 1, m - i)A(n, i) ,$$

where $A(n, i)$ is a factor related to the processing rate of the nth server when its queue length is i. The formula permits calculation of the normalizing constant for progressively larger networks. The original version applied only to closed single-class networks. This was generalized at the IBM Thomas J. Watson Research Center by Chandy, Herzog, and Woo [20], by Reiser and Kobayashi [14], and by Sauer [21] to encompass the entire range of product form networks. This algorithm has remained the standard one for several years, but is currently receiving competition from several new algorithms, all developed (at least partly) at IBM: Reiser and Lavenberg's *mean value analysis* [21–23] and Chandy and Sauer's *local balance algorithm for normalizing constants* [21, 24]. Also noteworthy are some algorithms not applicable to all product form networks, namely Moore's *partial fraction algorithm* [25], Kobayashi's *Polya enumeration* [26], and Chandy and Sauer's *coalesce computation of normalizing constants* [24].

The computational efforts required for the various full-range algorithms are comparable, but they differ somewhat in their numerical properties. Algorithms which compute the normalizing constant explicitly are subject to floating point overflow when solving grossly unbalanced networks. Mean value analysis, which avoids the normalizing constant calculation, is free from that problem, but is subject to loss of accuracy when dealing with variable-rate servers. For networks with only fixed-rate and infinite servers, mean value analysis appears to be the algorithm of choice, and recent work by Reiser [23] may overcome some of the method's deficiencies. The mean value algorithm works directly with the mean values of queue lengths and response times at each queue. It computes these quantities for a given network by relating them to values for networks with, in turn, one fewer customer in each class. Specifically, in the case of a network with fixed-rate servers, it applies the following formulas alternately:

$$T(n, m) = t(n)[1 + N(n, m - 1)] , \quad (1)$$

$$N(n, m) = \frac{mT(n, m)}{\sum_j T(j, m)} ,$$

where $T(n, m)$ and $N(n, m)$ are, respectively, the total time a job spends at the nth server, and the average queue length of the nth server, when there is a total of m users in the system, and $t(j)$ is the total service time per job at the

*n*th server. This method has spawned several approximate methods which will be discussed in the following sections.

- *Approximations for non-product form networks*

Many methods have been proposed for obtaining approximate solutions to queueing networks which do not have a product form solution. Among those contributed by workers at IBM we list the following:

- The diffusion approximation [27] replaces the discrete transitions of customers between queues by means of a continuous diffusion process. The method provides a convenient way of modeling nonexponential service times by taking into account the variance, as well as the mean, of the service times.
- Many approximations are based on decomposing the network into several parts, replacing each part with a single more or less equivalent server, and then solving the resulting reduced network [4, 28]. Many of these decompositions [29–32] are based on a queueing-theoretic analogue of Norton's theorem [20], which permits replacing all but one of the queues in a product form network with a single equivalent queue. Another interesting method which uses a hierarchical decomposition principle is due to Florkowski [33].
- In decomposition approximations one must solve a reduced network (or networks) which does not satisfy product form conditions. The method of Herzog, Chandy, and Woo [34] provides an efficient algorithm for solving reduced networks which arise in decomposition approximations.
- In mean value analysis one uses a "delay equation," *e.g.*, Eq. (1), relating the total average delay suffered by a customer in a given queue to the average length of that queue for a network with one fewer customer. It is a straightforward matter to adapt these equations to non-product form queues, *e.g.* first come first served with different average service times for different classes, as well as more complex cases involving blocking and parallelism. The results [22, 35] are approximate, but often quite serviceable.

- *Fast approximate algorithms*

The algorithms described in the previous sections work well and reasonably quickly in many cases. However, when the number of closed customer classes reaches ten or so, the algorithms become too slow and storage-consuming. Computational requirements can also become excessive when the number of customers and/or servers becomes very large. IBM workers have contributed some algorithms which are faster and more storage-conserving than the previously described ones yet produce accurate answers, typically within 5% of their slower counterparts.

Pittel [36] has investigated the asymptotic behavior of the product form solution as the number of customers in each class increases beyond bounds, while preserving fixed ratios between different types. This led to a very fast and simple iterative algorithm, valid for large populations (documented in [37]). It then turned out that the same algorithm is derivable from mean value analysis [22] if one neglects the difference between networks whose number of customers differs by one. This method was then applied to many non-product form networks [35]. A general approach, valid for both small and large populations, was then developed [22, 38] as documented in [39]. The approach consists of approximating the properties of a network with $m - 1$ customers using the properties of a network with m customers, and then applying the mean value equations. The latter now contain only quantities relating to an m customer network, and are usually solvable by means of a very simple iterative procedure.

- *Implementations*

A general-purpose queueing network solver, such as QNET4 [40], is very useful for *ad hoc* computer system modeling. For routine use, however, it presents various difficulties, most notably the problem of expressing a computer system modeling problem in queueing theoretic terms. Therefore, many special-purpose models, tailored to specific systems, have been constructed. Among these are models of MVS [41], VM/370 [37, 42], and CICS [43]. Similar models exist for IMS, Systems 34 and 38, and others. Such a model contains in its core a suitable queueing network algorithm, but this is surrounded by layers of software which

- prompt for inputs describing system configuration and workload in terms meaningful to the end user. For instance, the configuration description may consist of the CPU model, main storage size, I/O device types, and their channel connections. The workload description may consist of the number of users or transaction rate for each class, and average resource demands (instructions executed, I/O and other service requests made, storage occupied) per transaction of each class.
- translate these inputs into quantities required by the queueing network algorithm, *i.e.* service times and routing probabilities at each queue. This translation is accomplished by using built-in tables of hardware and software characteristics, such as CPU and device speeds, and operating system path lengths.

Many such models are available to IBM system engineers for use in configuring systems to meet customers' performance requirements.

These models are typically accurate to within 5% in the estimation of utilizations and throughputs, and within

20% for average queue lengths and response times. The accuracy of the performance predictions is determined more by the accuracy of the workload characterization fed into the model than by the quality of the model itself. For this reason, it is most desirable to have means of deriving model inputs directly from measurements taken on real systems [37, 44].

Special-purpose models have also been devised for various system components. Of particular interest is the I/O subsystem, consisting of channels, control units, strings, and devices. The contention for channel and control unit time cannot be modeled adequately by means of queueing network theory. A successful approach to this problem is to regard each device as a single server queue, whose service time is the sum of seek, latency, rotational delay, search, and data transfer times. The rotational delay time, in turn, is a function of the contention for various path components (channel, control unit, and head of string). Models due to Seaman *et al.* [45] and Bard [46, 39] have successively tackled the following cases: single path to each device; single path with rotational position sensing; and multiple path with devices shared among several CPUs.

Simulation

Simulation is a popular approach to the solution of computer system models because of its generality and because system details can be represented very accurately. However, there are a number of potentially overwhelming problems with simulation: 1) Constructing a simulation program and verifying that the program is logically correct and properly represents the simulated system can require a great amount of human effort. 2) It may be very difficult to characterize the system workloads and features which have the greatest impact on performance. 3) Running a simulation program should be considered a statistical experiment; the performance measures obtained from a simulation must be viewed skeptically unless appropriate statistical methods are used. 4) Detailed simulation programs may require large amounts of computer time to provide accurate performance estimates. IBM has made significant contributions which alleviate these problems.

- *Construction of simulation models*

GPSS Though introduced two decades ago, the General Purpose Simulation System (GPSS) remains one of the most popular languages for computing system simulations [47]. A principal reason for the success of GPSS is that it provides convenient abstractions which may be used to describe systems without actually writing a simulation program; GPSS constructs a simulation program based on the user's description. The abstract elements of GPSS are called "blocks." Current versions of GPSS have roughly 50 block types. Each block type has a unique pictorial symbol. Usually one will construct a diagram showing the flow of "transactions" through various blocks. (The transactions may literally represent transactions in a data base system, or may represent commands or batch jobs in a general-purpose computer system, messages in a computer communication system, etc.) Once such a diagram has been constructed, the computer implementation of the model is a mechanical translation of the diagram.

CSS and SNAP/SHOT Closely related to GPSS is the Computer System Simulator (CSS) [48]. Rather than the abstract blocks of GPSS, CSS blocks are predefined characterizations of IBM hardware components. Construction of the simulation model in CSS is thus primarily a matter of describing the software (operating system, application programs, etc.) which runs on the hardware. A further step in this direction of alleviating the need for simulation programming is the Systems Network Analysis Program/Simulated Host Overview Technique (SNAP/SHOT) [49]. In addition to characterizations of hardware components, SNAP/SHOT provides corresponding characterizations of many of IBM's software products, especially those associated with the Systems Network Architecture.

Research Queueing Package An alternate approach to alleviating the need for simulation programming is the high-level modeling language provided by the Research Queueing Package (RESQ) [50–52]. The three principal contributions of RESQ are that 1) Several solution methods, numerical, approximate, and simulation, are brought together in one software package. 2) Systems are described in terms of very high-level abstract elements, based on queueing networks. 3) Several user interfaces provide both novice and experienced users productive means to describe systems. We discuss each of these contributions in turn.

In the past many (most?) modeling practitioners have restricted themselves to one solution method, either analytic (including numerical and approximate methods) or simulation. RESQ includes the previously mentioned QNET4 package, approximate solution components, and a simulation component for the solution of models. Thus the RESQ user is strongly encouraged to avoid arbitrary restrictions on solution methods and to use a method most appropriate to the problem at hand. The presence of several solution methods also makes feasible the mechanical use of the hybrid solution methods to be described.

There are a number of assumptions required for exact analytic or numerical solution of a significantly sized

queueing network model to be feasible. In addition to specific limitations on particular types of queues, there are general assumptions usually left implicit, *e.g.*, that a job in the queueing network contends for only one resource at a time and/or that a job may not be involved in simultaneous synchronous activities. However, characteristics such as these are important in actual systems, where several resources (*e.g.*, memory, channel, controller and device) may be necessary for particular activities and messages may be transmitted as packets across different communication paths, to be reassembled at their destination. RESQ provides extensions of traditional queueing networks so that such characteristics may be included in a model to be solved by approximation or simulation. The most important of these extensions is the "passive" queue, first proposed by Foster, McGehearty, Sauer, and Waggoner [53] and redefined in RESQ [50, 51, 54]. Traditional queues are referred to as "active" because a job holding a resource of the queue is actively using the resource. Resources of passive queues are held so that the job may use a resource of primary importance. Passive queues have been demonstrated to provide compact representations of complex contention situations and protocols [51, 55].

The first RESQ interface was an interactive prompter, with built-in tutorial facilities, so that a novice could easily learn RESQ terminology and characteristics. Recently, a second version of RESQ has been developed, compatible with the first [56]. This version incorporates a much more sophisticated user interface, a modeling language analogous to a programming language. The RESQ2 language has been designed to encourage the user to produce well-structured models, in the sense of structured programming, so that modelers can effectively cope with large systems. The "submodel" facility of RESQ2 is designed so that modelers can cooperate in constructing a model and build upon the previous work of other modelers.

- *Workload characterization*

Trace-driven modeling In simulation models, the workload has traditionally been described by means of probability distributions. These, however, may not capture important interdependencies of workload characteristics. This problem is overcome in trace-driven modeling, first proposed by Cheng [57] and later popularized by others [58–60]. With trace-driven modeling, the simulator "executes" the same sequence of transactions that was actually traced on a real system. If used properly, trace-driven models can reproduce measurement results very closely. One can then make modifications in the model and have confidence that the model results are very close to the performance that would be observed if corresponding modifications were made to the actual system.

Characterization of paging In both distribution-driven and trace-driven simulations of virtual memory systems, a compact and accurate representation of paging activity is desirable. The obvious representation, a complete history of page references, is usually impractical. Of particular impact has been the characterization in terms of distance strings in stack replacement algorithms [61]. Other related work includes the lifetime functions discussed by Belady and Kuehner [62] and the semi-Markov characterizations of Lewis and Shedler [63]. Other compact representations of paging behavior have been developed at IBM for inclusion in various models. These include the macro-instructions of Boksenbaum *et al.* [64], the page survival index [65], the paging index [44], and the global LRU analysis of Chiu and Chow [60].

- *Statistical aspects of simulation*

Random number generators When one is characterizing systems by probability distributions, one must have generators for producing samples from the distributions. Nearly all practical generators for general distributions require a generator for the uniform distribution on the interval [0, 1]. In designing such a uniform generator there are a number of pitfalls which can only be avoided by careful use of number theory to propose a generator and of rigorous statistical tests to verify that the generator has the desired properties [66]. The generator proposed by Lewis, Goodman, and Miller [67] has been shown to have very good properties. In fact, its properties are so highly regarded that this generator has been subsequently incorporated in highly regarded software for other vendors' machines [68]. This is surprising because random number generators are usually designed for specific arithmetic characteristics, such as word size, and are generally not easily transported. IBM research has also contributed methods for simulation of processes not described by stationary distributions, notably nonhomogeneous Poisson processes [69].

Output analysis Another difficult problem in probabilistic simulations is the analysis of output. The running of the simulation is a statistical experiment; the results of the simulation program may not be accurate estimates of model performance measures. The most important recent contribution to this problem is the Regenerative Method for confidence intervals [70]. IBM has contributed a number of improvements and extensions for the Regenerative Method, including stopping rules [71], extension to response time distributions [3], and variance reduction techniques [72]. IBM authors have also applied variance reduction techniques to previous methods for confidence

intervals [73]. The regenerative method is incorporated in RESQ, and the practical applicability of the method has been demonstrated by a number of RESQ models [51]. A recently proposed spectral method [74] for confidence intervals may prove to be more useful than previous methods, including the regenerative method.

- *Computational expense*

Hybrid simulation One factor in the computational expense of simulation is the disparity in event rates in different parts of the system, causing unnecessarily long simulation of one part in order to have a long enough simulation of another part with low event rates. An important method to avoid this expense is hierarchical solution, where these different parts of the model are solved separately. A special case of hierarchical solution is hybrid simulation, where part of the model is solved numerically. Examples of hybrid simulation are found in [60] and [75].

Design of experiments and validation In addition to reducing the expense of individual simulations, one can reduce the number of simulations required to cover a parameter space by appropriate design of experiments [76]. Rather than run a simulation for each combination of parameters, one can run simulations for a small subset of the combinations and estimate results for the other combinations. This approach is also useful for validating the model. For this purpose, identical sets of experiments are run on the real system and on the model. The statistical effects of various system parameters are evaluated in both cases, and the two sets of computed effects are tested for lack of significant differences. If the test is passed, the model is considered to be validated. Model validation can also be combined with estimation of the unknown model parameters [77].

References

1. A. O. Allen, *Probability, Statistics, and Queueing Theory: With Computer Science Applications*, Academic Press, Inc., New York, 1978.
2. H. Kobayashi, *Modeling and Analysis*, Addison-Wesley Publishing Co., Reading, MA, 1978.
3. D. L. Iglehart and G. S. Shedler, *Regenerative Simulation of Response Times in Networks of Queues*, Springer-Verlag, New York, 1980.
4. C. H. Sauer and K. M. Chandy, *Computer Systems Performance Modeling*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.
5. Y. Bard, "Performance Criteria and Measurement for a Time Sharing System," *IBM Syst. J.* **10,** 193-216 (1971).
6. H. P. Friedman and G. Waldbaum, "Evaluating System Changes Under Uncontrolled Workloads: A Case Study," *IBM Syst. J.* **14,** 340-352 (1975).
7. Y. Bard and M. Schatzoff, "Statistical Methods in Computer Performance Analysis," *Current Trends in Programming Methodology, Volume III: Software Modeling and Its Impact on Performance*, K. M. Chandy and R. T. Yeh, Eds., Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978, pp. 1-51.
8. J. R. Jackson, "Jobshop-like Queueing Systems," *Manage. Sci.* **10,** 131-142 (1963).
9. J. Gordon and G. F. Newell, "Closed Queueing Systems with Exponential Servers," *Oper. Res.* **15,** 254-265 (1967).
10. A. L. Scherr, *An Analysis of Time-Shared Computer Systems*, MIT Press, Cambridge, MA, 1967.
11. J. P. Buzen, "Analysis of System Bottlenecks Using a Queueing Network Model," *Proceedings of the ACM-SIGOPS Workshop on System Performance Evaluation*, Cambridge, MA, 1971, pp. 82-103.
12. S. R. Arora and A. Gallo, "The Optimal Organization of Multiprogrammed Multi-level Memory," *Proceedings of the ACM-SIGOPS Workshop on System Performance Evaluation*, Cambridge, MA, 1971, pp. 104-141.
13. F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *J. ACM* **22,** 248-260 (1975).
14. M. Reiser and H. Kobayashi, "Queuing Networks with Multiple Closed Chains: Theory and Computational Algorithms," *IBM J. Res. Develop.* **19,** 283-294 (1975).
15. H. Kobayashi and M. Reiser, "On Generalization of Job Routing Behavior in a Queueing Network Model," *Research Report RC-5252*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1975.
16. K. M. Chandy, J. H. Howard, and D. F. Towsley, "Product Form and Local Balance in Queueing Networks," *J. ACM* **24,** 250-263 (1977).
17. S. S. Lam, "Queuing Networks with Population Size Constraints," *IBM J. Res. Develop.* **21,** 370-378 (1977).
18. D. F. Towsley, "Queueing Network Models with State-Dependent Routing," *J. ACM* **27,** 323-337 (1980).
19. J. P. Buzen, "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *Commun. ACM* **16,** 527-531 (1973).
20. K. M. Chandy, U. Herzog, and L. Woo, "Parametric Analysis of Queuing Networks," *IBM J. Res. Develop.* **19,** 36-42 (1975).
21. C. H. Sauer, "Computational Algorithms for State-Dependent Queueing Networks," *Research Report RC-8698*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1981.
22. M. Reiser and S. Lavenberg, "Mean Value Analysis of Closed Multichain Queueing Networks," *J. ACM* **27,** 313-322 (1980).
23. M. Reiser, "Mean Value Analysis and Convolution Method for Queue-Dependent Servers in Closed Queueing Networks," *Performance Eval.* **1,** 7-18 (1981).
24. K. M. Chandy and C. H. Sauer, "Computational Algorithms for Product Form Queueing Networks," *Commun. ACM* **23,** 573-583 (1980).
25. F. R. Moore, "Computational Model of a Closed Queuing Network with Exponential Servers," *IBM J. Res. Develop.* **16,** 567-572 (1972).
26. H. Kobayashi, "A Computational Algorithm for Queue Distributions via the Polya Theory of Enumeration," *Performance of Computer Systems*, M. Arato, A. Butrimenko, and E. Gelenbe, Eds. North-Holland Publishing Co., Amsterdam, 1979, pp. 79-88.
27. H. Kobayashi, "Application of the Diffusion Approximation to Queueing Networks," *J. ACM* **21,** 316-328 (1974).
28. C. H. Sauer and K. M. Chandy, "Approximate Solution of Queueing Models," *Computer* **13,** No. 4, 25-32 (1980).
29. K. M. Chandy, U. Herzog, and L. Woo, "Approximate Analysis of General Queuing Networks," *IBM J. Res. Develop.* **19,** 43-49 (1975).
30. C. H. Sauer and K. M. Chandy, "Approximate Analysis of Central Server Models," *IBM J. Res. Develop.* **19,** 301-303 (1975).
31. W.-M. Chow and P. S. Yu, "An Approximation Technique for Central Server Queueing Models with a Priority Dispatching Rule," *Research Report RC-8163*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1980.

32. C. H. Sauer, "Approximate Solution of Queueing Networks with Simultaneous Resource Possession," *Research Report RC-8679*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1981.
33. J. H. Florkowski, "Extended Analytic Models for System Evaluation," *Technical Report TR00.2549*, IBM Data Systems Division, Poughkeepsie, NY, 1974.
34. U. Herzog, L. Woo, and K. M. Chandy, "Solution of Queuing Problems by a Recursive Technique," *IBM J. Res. Develop.* **19,** 295–300 (1975).
35. Y. Bard, "Some Extensions to Multiclass Queueing Network Analysis," *Performance of Computer Systems*, M. Arato, A. Butrimenko, and E. Gelenbe, Eds., North-Holland Publishing Co., Amsterdam, 1979, pp. 51–61.
36. B. Pittel, "Closed Exponential Networks of Queues with Blocking," *Research Report RC-6174*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1976.
37. Y. Bard, "An Analytic Model of the VM/370 System," *IBM J. Res. Develop.* **22,** 498–508 (1978).
38. P. Schweitzer, unpublished notes (1977).
39. Y. Bard, "A Model of Shared DASD and Multipathing," *Commun. ACM* **23,** 564–572 (1980).
40. M. Reiser, "Interactive Modeling of Computer Systems," *IBM Syst. J.* **15,** 309–327 (1976).
41. D. C. Schiller, "System Capacity and Performance Evaluation," *IBM Syst. J.* **19,** 46–67 (1980).
42. Y. Bard, "The VM/370 Performance Predictor," *Computing Surv.* **10,** 333–342 (1978).
43. P. H. Seaman, "Modeling Considerations for Predicting Performance of CICS/VS Systems," *IBM Syst. J.* **19,** 68–80 (1980).
44. Y. Bard, "A Characterization of VM/370 Workloads," *Modeling and Performance Evaluation of Computer Systems*, H. Beilner and E. Gelenbe, Eds., North-Holland Publishing Co., Amsterdam, 1976, pp. 35–56.
45. P. H. Seaman, R. A. Lind, and T. L. Wilson, "On Teleprocessing System Design, Part IV: An Analysis of Auxiliary Storage Activity," *IBM Syst. J.* **5,** 158–170 (1966).
46. Y. Bard, "Task Queueing in Auxiliary Storage Devices with Rotational Position Sensing," *Technical Report G320-2070*, IBM Cambridge Scientific Center, Cambridge, MA, 1971.
47. G. Gordon, "The Development of the General Purpose Simulation System," *Proceedings of the ACM SIGPLAN History of Programming Languages Conference*, Los Angeles, 1978, pp. 183–198.
48. P. H. Seaman and R. C. Soucy, "Simulating Operating Systems," *IBM Syst. J.* **8,** 264–279 (1969).
49. H. M. Stewart, "Performance Analysis of Complex Communications Systems," *IBM Syst. J.* **18,** 356–373 (1979).
50. C. H. Sauer, M. Reiser, and E. A. MacNair, "RESQ—A Package for Solution of Generalized Queueing Networks," *Proceedings*, National Computer Conference, Dallas, TX, 1977, pp. 977–986.
51. C. H. Sauer and E. A. MacNair, "Computer/Communication System Modeling with Extended Queueing Networks," *Research Report RC-6654*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1977.
52. C. H. Sauer and E. A. MacNair, "Queueing Network Software for Systems Modeling," *Research Report RC-7143*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1978. See also *Software—Practice and Experience* **9,** 5 (1979).
53. D. V. Foster, P. F. McGehearty, C. H. Sauer, and C. N. Waggoner, "A Language for Analysis of Queueing Models," *Proceedings of the Fifth Annual Pittsburgh Modeling and Simulation Conference*, 1974, pp. 381–386.
54. M. Reiser and C. H. Sauer, "Queueing Network Models: Methods of Solution and their Program Implementation," *Current Trends in Programming Methodology, Volume III: Software Modeling and Its Impact on Performance*, K. M. Chandy and R. T. Yeh, Eds., Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978, pp. 115–167.
55. C. H. Sauer, "Passive Queue Models of Computer Networks," Computer Networking Symposium, Gaithersburg, MD, 1978.
56. Charles H. Sauer, Edward A. MacNair, and Silvio Salza, "A Language for Extended Queuing Network Models," *IBM J. Res. Develop.* **24,** 747–755 (1980).
57. P. S. Cheng, "Trace-Driven System Modeling," *IBM Syst. J.* **8,** 280–289 (1969).
58. S. W. Sherman, F. Baskett, and J. C. Browne, "Trace Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System," *Commun. ACM* **15,** 1063–1069 (1972).
59. S. W. Sherman, "Trace-Driven Modeling: An Update," *Proceedings of Symposium on Simulation of Computer Systems*, Boulder, CO, 1972, pp. 87–91.
60. W. W. Chiu and W.-M. Chow, "A Performance Model of MVS," *IBM Syst. J.* **17,** 444–462 (1978).
61. R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM Syst. J.* **9,** 78–117 (1970).
62. L. Belady and C. J. Kuehner, "Dynamic Space Sharing in Computer Systems," *Commun. ACM* **12,** 282–288 (1969).
63. P. A. W. Lewis and G. S. Shedler, "Empirically Derived Micromodels for Sequences of Page Exceptions," *IBM J. Res. Develop.* **17,** 86–100 (1973).
64. C. Boksenbaum, S. Greenberg, and C. Tillman, "Simulation of CP-67," *Technical Report G320-2093*, IBM Cambridge Scientific Center, Cambridge, MA, 1973.
65. Y. Bard, "Characterization of Program Paging in a Timesharing Environment," *IBM J. Res. Develop.* **17,** 387–393 (1973).
66. D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Addison-Wesley Publishing Co., Inc., Reading, MA, 1969.
67. P. A. W. Lewis, A. S. Goodman, and J. M. Miller, "A Pseudo-Random Number Generator for the System/360," *IBM Syst. J.* **8,** 136–146 (1969).
68. *Computer Subroutine Libraries in Mathematics and Statistics*, International Mathematical & Statistical Libraries, Inc., Houston, TX.
69. P. A. W. Lewis and G. S. Shedler, "Simulation of Nonhomogeneous Poisson Processes with Log-linear Rate Function," *Biometrika* **63,** 501–506 (1976).
70. D. L. Iglehart, "The Regenerative Method for Simulation Analysis," *Current Trends in Programming Methodology, Volume III: Software Modeling and Its Impact on Performance*, K. M. Chandy and R. T. Yeh, Eds., Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.
71. S. S. Lavenberg and C. H. Sauer, "Sequential Stopping Rules for the Regenerative Method of Simulation," *IBM J. Res. Develop.* **21,** 545–558 (1977).
72. S. S. Lavenberg, T. L. Moeller, and C. H. Sauer, "Concomitant Control Variables Applied to the Regenerative Simulation of Queueing Systems," *Oper. Res.* **27,** 134–160 (1979).
73. S. S. Lavenberg, T. L. Moeller, and P. D. Welch, "Statistical Results on Multiple Control Variables with Application to Variance Reduction in Queueing Network Simulation," *Research Report RC-7423*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1978.
74. T. L. Moeller and P. D. Welch, "A Spectral Based Technique for Generating Confidence Intervals from Simulation Outputs," *Proceedings of the 1977 Winter Simulation Conference*, 1977, pp. 177–184.
75. C. H. Sauer, L. S. Woo, and W. Chang, "Hybrid Analysis/Simulation: Distributed Networks," *Research Report RC-6341*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1976.
76. M. Schatzoff and C. C. Tillman, "Design of Experiments in Simulator Validation," *IBM J. Res. Develop.* **19,** 252–262 (1975).
77. H. Beilner and G. Waldbaum, "Statistical Methodology for Calibrating a Trace-Driven Simulator of a Batch Computer

System," *Statistical Computer Performance Evaluation*, W. Freiberger, Ed., Academic Press, Inc., New York, 1972, pp. 423–460.

Received July 15, 1980; revised February 11, 1981

Yonathan Bard is located at the IBM Cambridge Scientific Center, 545 Technology Square, Cambridge, Massachusetts 02139. Charles H. Sauer is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.