Edgar H. Sibley
Panel Editor

Information system design can be influenced and ultimate performance parameters accurately forecast through the coordinated use of five performance analysis tools. The complementary methodology is used to first predict and then validate system performance throughout its life cycle.

THE COORDINATED USE OF FIVE PERFORMANCE EVALUATION METHODOLOGIES

GORDON E. ANDERSON

During the past twelve years, there has been a rapid evolution of performance evaluation from hit-and-miss measurement to experimental computer science [4]. The progress in the development of algorithms for solving queueing network models, for example, has been remarkable. However, the many recent papers dealing with performance evaluation usually have either presented a new algorithm for solving queueing networks or dealt with only one performance evaluation methodology. This paper describes the *coordinated use* of *five* performance evaluation methodologies to determine system performance feasibility, influence system design, predict the performance attributes of proposed systems, identify performance bottlenecks, and provide upward migration paths for a real-time point-of-sale system. The methodologies applied are work-load characterization, queueing network modeling, load simulation, measurement with a hardware monitor, and simulation modeling.

THE SYSTEM TO BE EVALUATED

The performance of a proposed retail store controller and attached point-of-sale (POS) terminals was to be evaluated. The architecture of this system, shown in Figure 1, consists of a main CPU with 128K bytes of main memory, a sub-CPU with 4K bytes of memory for communication processing, up to 512K bytes of bubble memory, an 8-inch floppy disk unit, and up to 32 POS terminals multidropped on a 4800-bps communication line. Both the main CPU and the sub-CPU are Fujitsu M6800 microprocessors that run at 768 kHZ.

The system must perform three functions. The first is providing price data on merchandise items in real time. A request for price data originates at a POS terminal and is passed over the communication line to the controller. The controller looks up the price data in bubble memory and returns the data to the POS terminal over the communication line. The second is maintaining a transaction log of all sales activity. The POS terminals send messages to the controller at the end of each sales transaction and the controller records these data on a floppy disk. The third is keeping sales totals by employee number, sales department, etc. Again, this is accomplished by message interchanges between the POS terminals and the store controller.

Identification of Performance Requirements and Objectives

Performance requirements were a mean interactive response time of less than 2 seconds and a mean end-of-sale transaction response time (which involves the ex-

© 1984 ACM 0001-0782/84/0200-0119 75¢

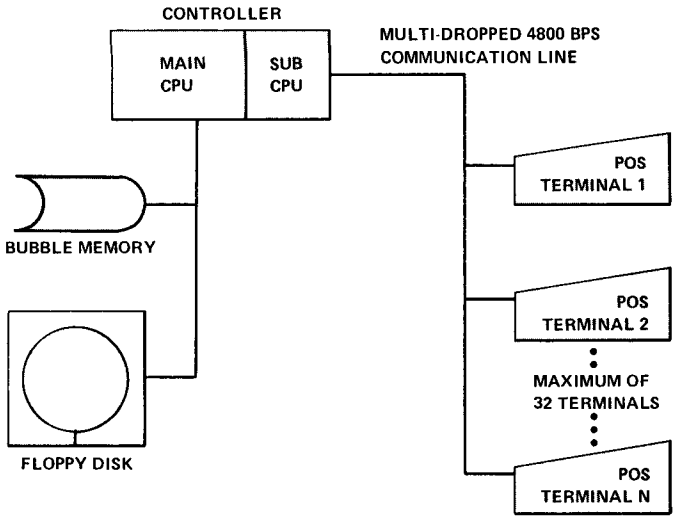

FIGURE 1. System Hardware Configuration

change of multiple messages) of less than 8 seconds. These response time requirements were based on a survey of potential users, experience, and response time studies such as those by Miller [9].

Software Performance Engineering

Conducting performance analysis during (rather than after) design and development is known as software performance engineering, the advantages of which are numerous, as pointed out by Smith [12]. Performance analysis was concurrent with the design and development of the controller software. This permitted us to identify design trade-offs and roughly predict the number of POS terminals that could be connected to the controller while maintaining the response time requirements.

One clear advantage of software performance engineering is that the participation of performance analysts in system design usually results in an efficiently designed and implemented system. Designing and implementing a system before finding out that it does not meet performance objectives often results in postdevelopment performance tuning that is difficult to achieve while preserving the original design. Another advantage is that the *cost* of achieving performance objectives can be identified during the design phase. Putting a price tag on performance and functional requirements forces reevaluation of these requirements early in the system development cycle and helps separate mandatory from optional requirements.

Development of a Performance Evaluation Plan

We developed a performance evaluation plan as follows. The first step in the analysis would be to characterize the work load of the proposed system. Second would be the creation of an approximate queueing network model of the system. Third, special hardware would be built to interface a hardware monitor with the microprocessor-based system, permitting measurement of the controller as soon as it was up and running. Fourth, design and coding of a load simulator would proceed concurrently with performance analysis and the development of the actual system. This would allow measurement of the system under controlled loading. Fifth, a detailed simulation model of the system would be constructed, validated, and used to identify bottlenecks and answer "what if" questions.

METHODOLOGY 1: WORK-LOAD CHARACTERIZATION

A study of retail store work-load volumes showed that peak loading occurs during the Christmas holiday season. Several days in that period, such as the Saturday before Christmas, are peak volume days. Further work-load studies from these busiest days showed that volume varied from hour to hour and that peak hour volume was approximately twice the mean peak day volume. This peak hour of peak day volume was used as the design goal for the system. That is, the system's performance under these conditions had to be rapid enough to avoid congestion and customer dissatisfaction in retail stores.

METHODOLOGY 2: A QUEUEING NETWORK MODEL

While the actual system was still in the design stage, we created an approximate queueing network model to provide preliminary performance data. In the system, there are six components of response time: communication buffer delay, read poll delay, output queueing delay, line input time, line output time, and controller processing time. Four different methodologies were used to calculate the six components.

Communication buffer delay is a problem of simultaneous resource possession. Solutions to this type of problem may be found by simulation, and several analytical techniques, such as those by Chandy, Herzog, and Woo [2] and Jacobson and Lazowska [6], have been developed. However, little was known of the software processing profiles and device service times at the time the queueing network model was created because the software had not yet been written. We believed that any resulting error would more likely be caused by lack of knowledge of the system than by computational or modeling technique. So, for the approximate model, communication buffer delay was simply estimated to be 100 milliseconds. This estimate was based on experience with similar systems.

Read poll delay and output queueing delay were calculated using queueing equations such as those presented in Kleinrock [7] and Martin [8]. Since communication line speed, protocol, and the number of bytes to be transmitted were known, the calculations were straightforward.

Line input and output times may be calculated easily from knowledge of line speed, the communication protocol used, line turnaround times, etc. From the proto-

col and data throughput requirements, the number of characters to be transmitted can be derived. This number is then multiplied by line character time and added to the sum of all line turnaround delays to get line input and output times.

The remaining component of response time is ***controller response time***. Figure 2 shows a queueing network model of the controller, consisting of a main CPU, bubble memory unit, and floppy disk. Note that the sub-CPU is not modeled here because it controls the communication line and Direct Memory Access (DMA) transfers data into the main CPU's memory. Consequently, any delays caused by the sub-CPU are accounted for in the line turnaround delays. In the queueing network model of the controller, there were three job classes, each of which had different branching probabilities through the network of queues and different service times at each server. Furthermore, the network was an open one in which the number of jobs was allowed to vary. Queueing for each resource was on a first-come, first-served (FCFS) basis.

Currently, there is no tractable exact solution algorithm for open networks of queues with FCFS queueing and different service times by job class [1]. Some approximate solution algorithms for these networks, such as those presented by Reiser and Lavenberg [11] and Chandy and Neuse [3], have been proposed. However, these solutions are limited to non-FCFS systems when service times depend on job class or are computationally expensive [10]. Furthermore, these approximate techniques are limited to closed networks. Other than simulation, there are two straightforward approaches to the problem. The first is to use the processor-sharing queueing discipline at each server. The second is to average the several job classes into one. We chose the second approach because the three job classes had similar service times. Also, we believed that any resulting error was more likely to be caused by errors in estimating service times than by analytical approach. Consequently, the controller component of response time was calculated using the method first described by Jackson [5], which has the added advantage of computational simplicity.

Queueing Network Model Results

The analytical model produced predicted interactive response times that varied from 0.96 second for 16 connected terminals to 2.18 seconds for 32 connected terminals. At the time our subjective opinion, based on experience, was that the analytical model results were within ±50 percent of the performance of the yet-to-be-built system. Consequently, we concluded that performance of the actual system would be acceptable for its intended purpose.

Later measurement of the running system showed that the queueing network model had produced response time predictions that were 25–30 percent longer than the measured times. We considered this accuracy excellent for an unvalidated, predictive model. In fact, the error was actually less than 30 percent when one considers that the queueing network model added 100-millisecond delay for communication buffer queueing, whereas the actual system had no communication buffer delay: there was enough memory to provide a communication buffer for each of the POS terminals. Queueing for communication buffers had been identified by performance analysts as a potential bottleneck during design of the system. The accuracy of the unvalidated analytical model was excellent because a great deal of effort was spent analyzing potential processing profiles and job branching probabilities in an attempt to obtain very accurate model service time and job branching parameters.

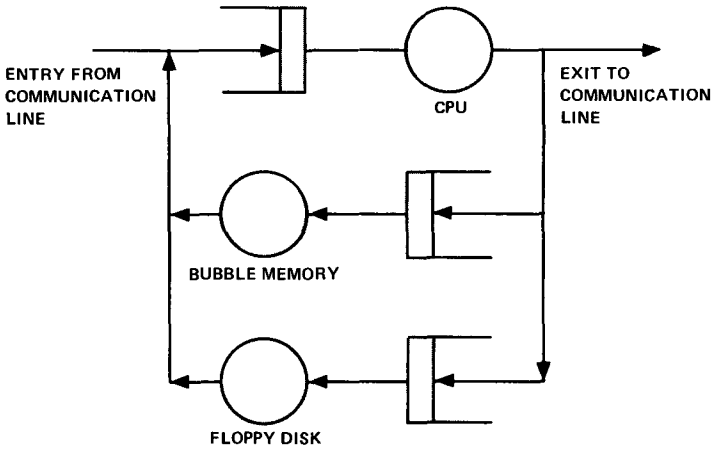

FIGURE 2. Open Queueing Network Model of System

METHODOLOGIES 3 AND 4: LOAD SIMULATION AND MEASUREMENT

Creating a validated model of the system under study necessitated obtaining or creating measurement tools, load simulation tools, and data analysis and reduction tools. We chose a general-purpose hardware monitor because measurement without the interference found in software monitors was desired. This choice required that special devices be built to interface an available hardware monitor with the measured system. Most hardware monitors have clip-on probes designed to interface with large computer backplanes, but since the system to be measured was based on microprocessors, special interfaces were needed to physically access measured data signals. The primary use of the hardware monitor and interfaces was to directly measure device utilizations.

A load simulation tool that could drive the actual system under controlled load was required. To this end, we developed a communication load simulator that could simulate between 1 and 32 terminals multidropped on the communication line that interfaced with the real system. The load simulator was capable of both creating Poisson or fixed-time interarrivals and translating these interarrivals into the actual communi-

cation line message interchanges found in the real environment. In other words, the load simulator was able to simulate a retail store with up to 32 POS terminals. The load simulator could also time-stamp the response messages (with 10-millisecond resolution) for later data reduction.

The data reduction and analysis capability was satisfied by an existing program that analyzed magnetic tapes containing time-tagged communication messages and produced a wide variety of statistics such as means, standard deviations, and histogram plots of response times. This was achieved easily and cheaply by having the load simulator write its data onto magnetic tape in a format compatible with the data reduction and analysis program.

The coordinated use of the above tools produced measured response times over a wide range of work loads and functionality. The work loads, obtained from studies of actual systems, were recreated using the load simulator, which was used to drive the system under test in a controlled manner.

METHODOLOGY 5: SIMULATION MODELING

Model Requirements

We required a validated model of the system that could produce accurate results over a wide range of workload volume, number of terminals attached to the system, and system functionality. A wide range of system functionality means that different functional processing profiles would be modeled; for example, some sales transactions required interactive price look-up and credit authorization processing, whereas others did not. The model would be used to simulate all these types of processing.

In order to obtain a wide domain of validity for the model, significant detail and fidelity were required. For example, the actual system contained a network of servers with multiple job classes in which service times varied with job class. At most servers, the queueing discipline was FCFS, but other service disciplines included preemptive priority and polling. Furthermore, parallel resource acquisition needed to be modeled. As tractable analytic solutions to such models do not exist, we decided that simulation modeling would be used.

A major disadvantage of simulation modeling is its great cost in both time and effort. (There have been simulation models that have required as much time and effort to construct as the actual systems they modeled.) In the search for a simulation tool that would provide the greatest possible modeler productivity, we studied available simulation languages and modeling tools and chose the Performance Analysts Workbench System (PAWS) as the most powerful for our particular modeling requirements.

Information Processing Graphs

Modeling with PAWS begins with the creation of an information processing graph (IPG). IPGs were developed by Chandy and others [13] for the purpose of viewing the performance attributes of a computer or computer/communication system at a high level of abstraction. IPGs contain all the information found in queueing network diagrams plus other crucial functions such as memory management, parallel resource acquisition, interrupt processing, and parallel processing. IPGs are similar to directed graphs and consist of nodes connected by edges. Transactions (information to be processed, such as jobs or tasks) flow from node to node along the edges. At each node the information is processed in some way before the transaction leaves that node.

Figure 3 shows an IPG of the real-time system modeled. In the figure, a Poisson arrival process is modeled at node ARRSOURCE where transactions are spawned. The transactions then proceed to CLOSEGATE where the number of transactions in the system is held to n, where n is the number of terminals in the system. If n or more transactions are already in the system, the new transaction must queue at CLOSEGATE. As each transaction leaves the system, a new transaction, if one is queued, is permitted to enter the system. This is done in PAWS using tokens that are returned to CLOSEGATE from node OPENGATE, through which every transaction passes upon leaving the system. From CLOSEGATE, transactions travel to TERMINAL1 through TERMINALn with equal probability. A transaction's polling priority on the multidropped communication line is controlled by the terminal with which the transaction is associated. From its respective terminal, a transaction proceeds to PHSECOMP, where its phase is incremented. In PAWS, the user can specify that a transaction pass through several behavioral phases. The behavior of a transaction at each node (e.g., the service time at a service node) can be specified for each phase. In this way, multiple job classes with different (and even dynamically changing) service times by class are modeled.

Other nodes in the IPG perform a number of functions: PLUTIME permits the gathering of price lookup response times; CREDTIME permits the gathering of credit response times; RCVLINE, OPENTGAT, RCVDELAY, RCVGATE, and CLOSEGATE model the behavior of the receive side of a half-duplex multidropped communication line with a polling queueing discipline. In the actual system, transmissions have priority over receptions; however, once a read poll sequence begins, it cannot be interrupted by a transmission. This behavior is explicitly modeled by coordination with the transmit side of the line (nodes XMTCOMP1, XMTGATE, SETCLOSE, XMTLINE, XMTCOMP2, XMTTIME, and SETNOPEN). As can be seen, a great deal of fidelity with the actual system is possible.

Simulation Modeling With PAWS

IPGs are easily translatable into an executing simulation model using PAWS. Each node in an IPG can be

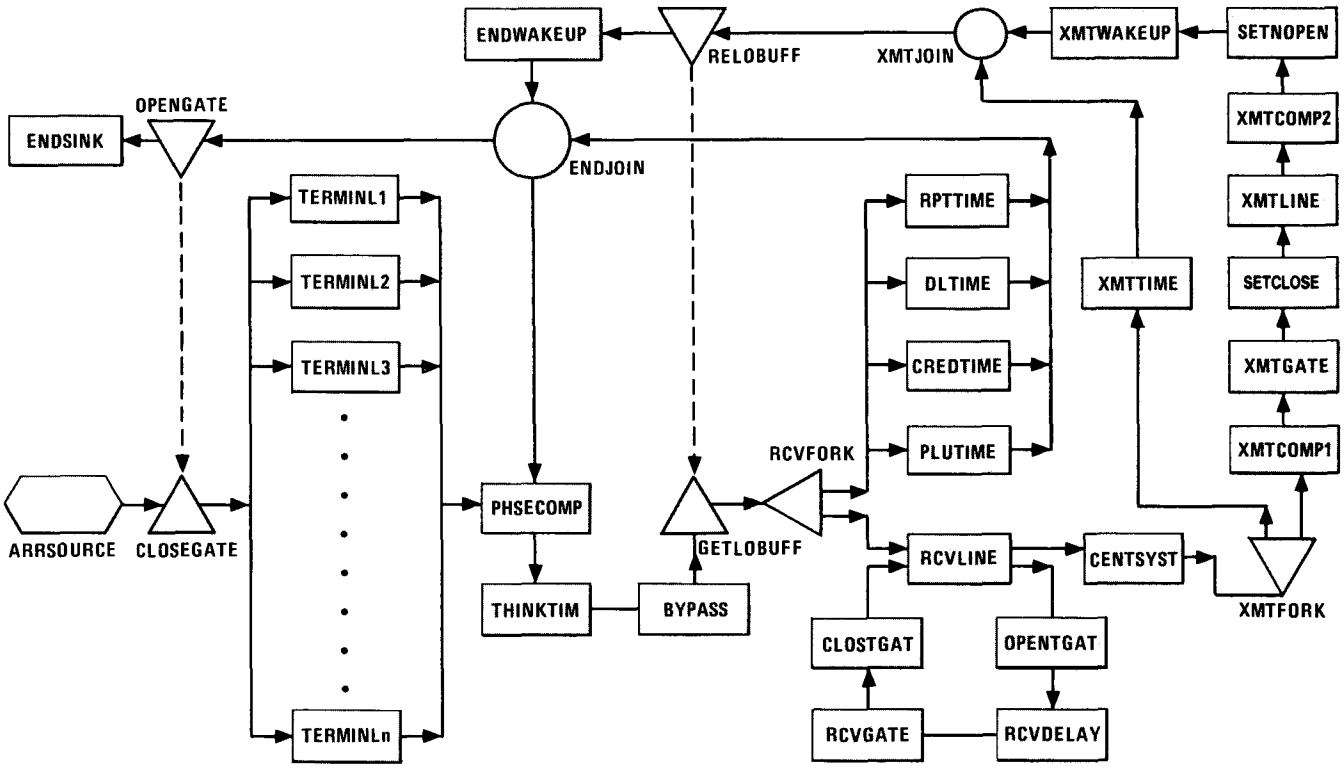

FIGURE 3. Information Processing Graph

implemented in PAWS in from 2 to 15 or so lines of PAWS source code. For example, the Poisson arrival process modeled at ARRSOURCE is implemented in 4 lines of source code:

```
ARRSOURCE
  TYPE SOURCE
    REQUEST
      ⟨POSTRANS, INITPHASE⟩
        EXPO (INTARTIME)
```

After coding, a PAWS program is compiled like any other high-level language. If the compilation is successful, the simulation model is executed for a user-specified length of time and performance statistics are generated. Typical performance statistics generated are histograms, means, standard deviations, and variances for data such as queue lengths, response times, queueing times, and device utilizations.

Our PAWS simulation model of the real-time, POS system was completed at about the same time the actual system became available for measurement. All that was required to validate the model was adjustment of service time parameters. By adjusting the service time parameters, a model was created that produced response time statistics and CPU utilization that agreed with measured values over a wide range of work-load volume, number of terminals, and processing functionality.

Model and Measurement Results

Figure 4 presents the response time statistics generated by the preliminary queueing network model, the validated PAWS simulation model, and the actual measured response times. The figure shows agreement between the PAWS simulation model and measured values for three types of interactions: price lookup interactivity, data log messages, and report messages. In all cases, the modeled response times are within 11.5 percent of measured response times, and the agreement holds over a wide range of work-load volume. For the preliminary queueing network model, modeled response times are within 30 percent of measured values. Recall that results of the queueing network model represent an average of the three types of response times, whereas the PAWS simulation model and measured values differentiate response times by job class.

Figure 5 shows main CPU utilization values for the preliminary queueing network model, the PAWS simulation model, and measured values. Agreement between the simulation model and measured values is within 2 percent.

In addition to the values shown in Figures 4 and 5,

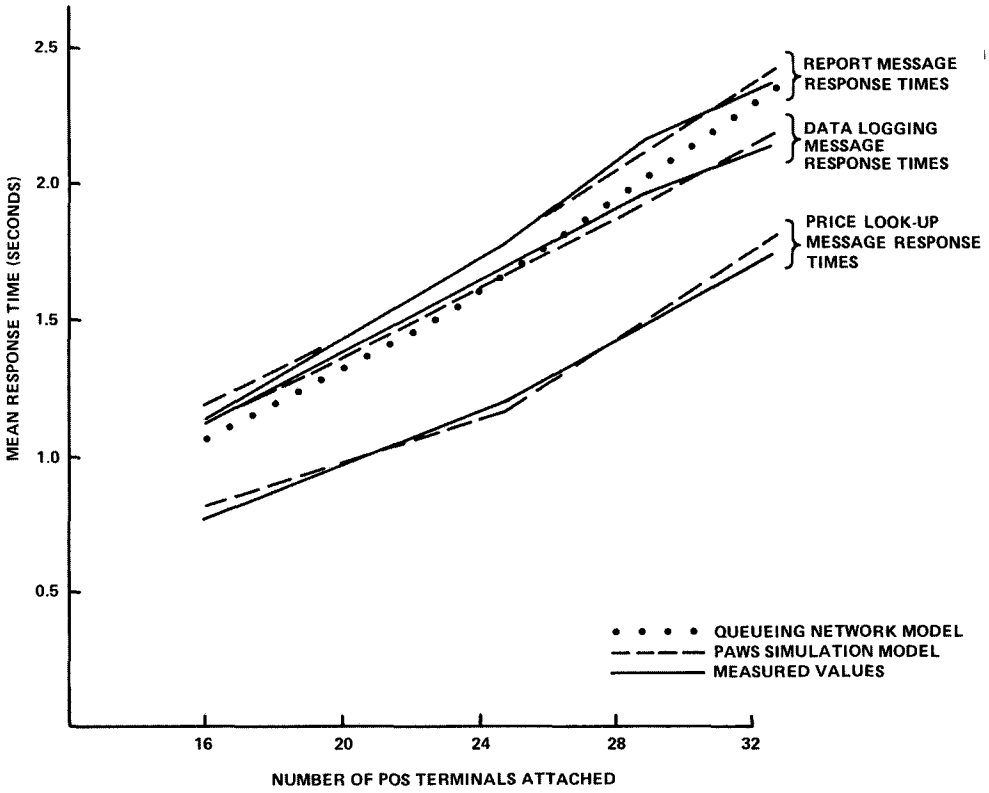

FIGURE 4.
Modeled and Measured Response Times

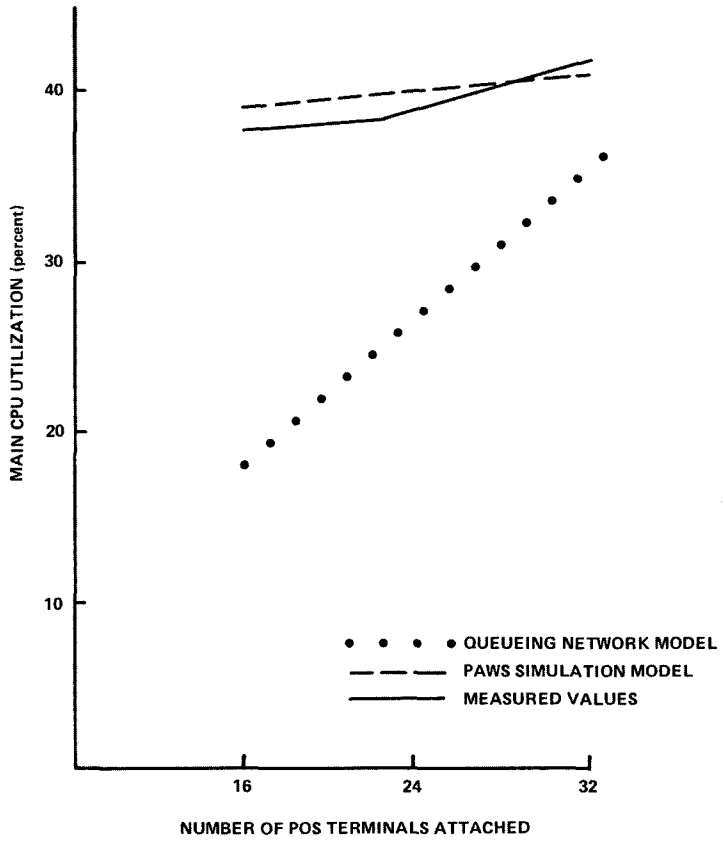

FIGURE 5.
Main CPU Utilization

several other simulation model runs were made using different functional processing. Simulated response times remained within 11.5 percent and CPU utilization within 2 percent of measured values.

USES OF THE VALIDATED MODEL

After the model was validated, it was used to answer several "what if" questions. For example, since 74 percent of the response time delay was in the multidropped communication line, the question arose as to how much performance improvement could be gained by increasing line speed from 4800 to 9600 bps. We used the model to show that a 43 percent improvement in response time could be gained for the 32-terminal system if line speed were increased from 4800 to 9600 bps and interpoll delay were reduced from the measured 21.7 to 13.3 milliseconds. When these changes were made, performance improvements agreed with model predictions.

Since CPU utilization was low, the model was used to assess the feasibility of adding a second multidropped communication line with the idea that more than 32 POS terminals could be supported. The model showed that addition of a line was feasible, with CPU utilization remaining below 65 percent.

In addition to its value in extending the performance and applications of the controller, the simulation model is now being used to evaluate potential customer installations. If a potential customer is considering a store controller, sales personnel can complete a questionnaire that captures work-load and functional options. This questionnaire is then translated into a PAWS simulation model run, and an accurate performance assessment can be returned to the sales office within a few working days.

CONCLUSIONS

The coordinated use of five performance evaluation methodologies as demonstrated here illustrates how they may be applied to complement each other and achieve highly desirable results. These results include the determination of performance feasibility prior to system development, the identification of performance trade-offs during design and development accurately, the ability to assess the performance of proposed installations accurately, and the identification of economical upward migration paths which expand potential uses of the evaluated system.

Acknowledgments. The author would like to thank Doug Neuse, Kenneth Shumate, and John Hooper for their helpful suggestions.

REFERENCES

1. Baskett, F., Chandy, K.M., Muntz, R.R., and Palacios, F.G. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM 22,* 2 (Apr. 1975), 248–260. One of the first papers to describe exact, product-form solutions for a wide class of multiple job class queueing network problems.
2. Chandy, K.M., Herzog, U., and Woo, L. Approximate analysis of general queueing networks. *IBM J. Res. Develop. 19,* 1 (Jan. 1975). An early milestone in the development of approximate, rather than exact, solution methods for performance metrics of networks of queues.
3. Chandy, K.M., and Neuse, D. Linearizer: A heuristic algorithm for queueing network models of computing systems. *Commun. ACM 25,* 2 (Feb. 1982), 126–134. Chandy and Neuse present an approximate solution method for queueing networks that yields accurate results in stress tests where other approximation techniques produce errors of up to 35 percent.
4. Denning, P. Performance analysis: Experimental computer science at its best. *Commun. ACM 24,* 11 (Nov. 1981), 725–727. Denning summarizes progress made in solution methods for queueing networks and discusses the subsequent importance to experimental computer science.
5. Jackson, J.R. Job shop-like queueing systems. *Management Sci. 10,* 1 (Jan. 1963). Jackson was the first to show that an open network of n queues behaves as n independent queues.
6. Jacobson, P.A., and Lazowska, E.D. Analyzing queueing networks with simultaneous resource possession. *Commun. ACM 25,* 2 (Feb. 1982), 142–151. This paper gives an iterative solution method (called the Method of Surrogate Delays) for solving problems with simultaneous resource possession.
7. Kleinrock, L. *Queueing Systems.* Vol. 2, *Computer Applications.* Wiley, New York, 1976. Kleinrock's seminal work describes the application of queueing theory to computer and communication systems.
8. Martin, J. *Systems Analysis for Data Transmission.* Prentice-Hall, Englewood Cliffs, N.J., 1972. A classic work widely used by systems analysts to calculate queueing delays, the number of channels required, and other performance metrics when designing or configuring computer and communication systems.
9. Miller, R.B. Response time in man-computer conversational transactions. In *AFIPS Conference Proceedings for Joint Computer Conference* (San Francisco, Calif., Dec. 9–11), vol. 33, pt. 1. Thompson, Washington, D.C., 1968, pp. 267–277. This paper discusses human productivity as a function of response time in interactive computer systems.
10. Neuse, D. Approximate analysis of large and general queueing networks. Ph.D. dissertation, Univ. of Texas, Dec., 1982. Neuse's thesis discusses both exact and approximate solution methods for networks of queues and presents three new approximation algorithms that give excellent results: SCAT, Linearizer, and HAM.
11. Reiser, M., and Lavenberg, S. Mean value analysis of closed multichain queueing networks. *J. ACM 27,* 2 (Apr. 1980), 313–322. Reiser and Lavenberg present a simple, intuitive, and widely used algorithm for obtaining solutions for performance metrics of product-form networks of queues.
12. Smith, C.U. Increasing productivity by software performance engineering. In *Proceedings of Computer Measurement Group XII* (New Orleans, Dec.). The Computer Measurement Group, Phoenix, Arizona, 1981. Smith describes the integration of performance analysis and software engineering to achieve well-designed, efficient computer systems.
13. *The Performance Analyst's Workbench System: Modeling Methodology and User's Manual.* Information Research Associates, Austin, Texas, 1981. A user's manual for PAWS: introduces information processing graphs, a powerful and succinct way to view the performance attributes of computer and communication systems.

CR Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Design Studies; C.4 [**Performance of Systems**]: Measurement Techniques; C.4 [**Performance of Systems**]: Modeling Techniques; I.6.2 [**Simulation and Modeling**]: Simulation Languages; I.6.3 [**Simulation and Modeling**]: Applications; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms: Measurement, Performance

Additional Key Words and Phrases: hardware monitor, load simulator, model validation, queueing network model, simulation model, software performance engineering, work-load characterization

Received 4/83; revised 8/83; accepted 8/83

Author's Present Address: Gordon E. Anderson, 2125 Mountain Vista Drive, Encinitas, CA 92024

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.