Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

SIMULATING DEDICATED UNIX PC-BASED APPLICATION SYSTEMS

Boleslaw K. Szymanski†

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180

Steven H. Azzaro

Engineering Systems Laboratory
GE Corporate Research & Development
Schenectady, New York 12301

ABSTRACT

An event-driven simulation model was developed to estimate and balance the hardware resources needed to support dedicated UNIX PC-based application systems. This paper describes the simulation approach, the software package developed, and the tools used to generate the model and collect data from it. Novel elements of this research include (1) simulation of software that reacts to changes in its external environment and (2) a methodology for using UNIX tools to estimate system characterization parameters. Results obtained using the model were validated by comparing them with data collected from an actual system.

1. INTRODUCTION

1.1 Why Model a Computer System?

Accurate prediction of a computer system's performance provides a means of estimating the computer hardware resources needed to administer an application software installation. This task is particularly important for dedicated computer systems, in which hardware is entirely devoted to servicing one application and unused capacity cannot be put to other uses. Modeling can provide the information needed to make cost-effective decisions about field hardware for initial and future system releases. When several installations of an application are planned, even a small cost saving or loss in each installation makes a substantial difference.

There are three major reasons for evaluating system performance: (1) for *hardware selection evaluation:* to make a cost-effective choice of hardware by establishing the initial system capacity needed, (2) for *performance projection*: to evaluate the effect that system software enhancements will have on a specific hardware configuration prior to software development, and (3) for *balancing* hardware/software resources [Lucas 1975]. *Hardware selection evaluation* uses system performance in configuring hardware for the given application. *Performance projection* involves estimating the performance of a system that does not yet exist, in order to estimate the feasibility and performance of the design. *Balancing* the hardware configuration involves identifying potential bottlenecks in system performance and removing them through the upgrade of selected parts of the system [Bard 1975].

1.2 Queuing Network and Simulation Modeling

For the analysis of the application system discussed here, a queuing network model was used as a first estimate of the hardware requirements of the projected system. In queuing network modeling, a computer-based system is represented as a network of queues that are amenable to closed-form solution. Separable queuing networks were applied; that is, networks in which each resource can be separated from the network and evaluated in isolation. Our experience as well as the experience of others shows that for the best case these models are accurate to within 5 to 10% for system resource utilizations and throughputs and to within 10 to 30% for response times [Anderson 1984; Lazowska et al. 1984]. Simulation modeling has been shown to be adequate in addressing selection evaluation, performance projection, and resource balancing [Lucas 1975]. In the past, a great deal of effort was required to construct a model, and a great deal of time was also required for its execution. Recent commercial offerings, however, provide high-level modeling elements in pictorial format, boosting productivity and enhancing maintainability. Simulation modeling was used in this project to enable assessment of maximum system requirements brought on by bursts of activity, rather than average values, typically derived from queuing network models.

2. EVENT-DRIVEN SIMULATION MODEL

In order to obtain reliable system performance predictions in various hardware and software environments, we developed software for simulating UNIX PC-based application computer systems. The software package models a CPU, a disk, and I/O activity. Parameters associated with each hardware device can be varied, permitting assessment of cost/performance tradeoffs. The variable parameters are processor speed, memory size, disk access speed, and latency. The system computational load is characterized by definition of a set of processes that represent system utilization. Processes are operated in one of four possible modes (described in Section 3.2.1, "Mode of Operation") to enable precise description of the simulated system behavior. Characteristics of each process—such as I/O, paging interrupt frequency, and required CPU time—are specified by the user.

The simulation technique used is event-driven. The simulated computer system is viewed as a collection of processes that share the same hardware in a multi-programming, virtual memory environment with a UNIX operating system. The hardware is typical for PC systems and consists of a single CPU and one or more disk devices. Each process starts its execution by entering the ready queue of the CPU. A free CPU is allocated to the process at the head of the ready queue, which runs until it (1) terminates, (2) times out, (3) needs a page that is not resident in the memory, or (4) needs to perform an I/O operation. Each process maintains four event-counters that measure the elapsed CPU time required for each of these events. The operating system maintains the working set of active processes in such a way as to equalize their paging ratios. The same disks are used for I/O operations as for paging.

2.1 Simulation Event Definitions

Five events are defined in our simulation: *Request-CPU, Release-CPU, Request-Disk, Release-Disk, and Terminate.* In the simulation package, each event is associated with a procedure that controls the processing of that event. In an actual system, the operating system processes events.

2.1.1 Request-CPU

The Request-CPU event procedure first checks the status of the CPU. If the CPU is not free, the serviced process is placed at the end of the ready queue. If the CPU is free, the process acquires the CPU. On the process's first CPU visit, a new set of counters is generated; on subsequent visits, the counters are updated. The smallest counter is selected and subtracted from all counters. The smallest counter determines the time of the next event—*Release-CPU*—for the serviced process.

† Consultant to General Electric Company

2.1.2 *Release-CPU*

The Release-CPU event is generated by either the Request-CPU event procedure that found the CPU free or by the Release-CPU event procedure in the previous call. The Release-CPU event procedure creates a new event for the serviced process that is determined by the process's event counter. If a process times out—because of the lack of paging or I/O requests—and no other processes are active at that point, it may release the CPU and then acquire it again immediately.

The Release-CPU event procedure selects the process at the head of the ready queue to acquire the CPU. On the process's first CPU visit, a new set of counters is generated; on subsequent visits, the counters are updated. The smallest counter is selected and subtracted from all counters. The smallest counter determines the time of the next event for the serviced process.

2.1.3 *Request-Disk*

The Request-Disk event procedure first checks the status of the disk. If the disk is idling, the event procedure causes the process to be placed at the end of the disk queue. If the queue is empty and the disk is idling, the event procedure creates a new event for the serviced process: *Release-Disk*. (The time of this event is selected according to the distribution of the Disk Service Time; see Section 3.1.4 "Average Disk Service Time.") Otherwise the serviced process is placed at the end of the disk queue.

2.1.4 *Release-Disk*

The Release-Disk event procedure releases the disk and produces a new event: *Request-CPU*. If the disk queue is not empty, the process at the head of the queue is set to acquire the disk.

2.1.5 *Terminate*

The Terminate event causes the process to become dormant for some period (based on its *Mode of Operation*, described in Section 3.2.1) and then generates a *Request-CPU* event.

2.2 Simulation Execution

As is typical in event-driven simulation, the driving routine is the scheduler, which maintains the global clock, lists of future events, and list of existing (active and dormant) processes. The future event list is ordered by the event times. The scheduler simply removes the first event (i.e., the event closest to the global clock in time) from the event list, advances the global clock to the event time, and then executes the corresponding event routine. The execution usually produces one or more new events. The simulation ends when the global clock exceeds the specified simulation time limit. Execution of each event updates statistical information gathered about the system (sizes of queues, utilization of devices, waiting times of processes, etc.). Although relatively simple, this system allows accurate simulation of complex applications.

2.3 Simulation Tool—The Extend Simulation Environment

Extend [1988] is a simulation tool commercially available for the Apple-Macintosh environment. It furnishes the basis for the implementation of the simulation package. This tool provides many helpful features, including a graphical interface, graphics for simulation of components and parameters, and a library of useful functions.

Extend assumes that the system is simulated at equal, discrete intervals. This assumption created a problem because getting an interval small enough to provide accurate results would introduce many intervals in which nothing happened. The solution was to define the simulation in terms of states of the processes of the modeled system. Each state of the system defines an allocation of processes to devices and queues. The transitions between states are driven by the events defined in Section 2.1, "Simulation Event Definitions." Thus, the ticks of Extend's standard clock were used to represent event count rather than elapsed time.

In our simulation, the currently processed event predefines the time of those future events that are enabled by its occurrence. For example, the event *Request-Disk* (with disk idling) predefines the time of the event *Release-Disk*. (Service time determination is discussed in Section 3.4, "Simulating Process Variability.") On the other hand, the event *Release-Disk* forces the event *Request-Disk* to occur immediately for the process at the head of the disk queue, with the predefined time equal to the current time. The event with a predefined time closest to the current time is the next one to be processed. The difference between the current time and the next event predefined time is a varying simulation-step time elapsing between adjacent clock ticks. It can vary from 0 to the maximum service time for the system. This approach allowed us to make effective use of the Extend package.

The typical simulation, shown in Figure 1, is composed of components that represent the CPU, disk, and processes that define the application. The monitoring components responsible for gathering and displaying the results are also shown. The connections between components represent the information flow about event time and the process generating the event.

3. THE MODEL

The modeled system is defined by a number of parameters that characterize its hardware and software components, as shown in Figure 2.

3.1 Hardware Parameters

The hardware parameters are *CPU Relative Speed, Time Slice, Memory Factor,* and *Average Disk Service Time.*

3.1.1 *CPU Relative Speed*

The model assumes a 25MHz processor. This parameter can be modified to investigate the effect of CPU speed on system performance and balance. Replacing the given CPU with a faster one increases the total cost of the hardware configuration, improves the system throughput, and decreases the probability of process delays due to system overload. At the same time, the higher paging rate and I/O interrupt frequency increase the load on the disk, which may not be economically justifiable. In selecting the CPU speed, the main concern is to provide sufficient processing capacity for the application to run without delays. The CPU speed can also be used to balance the CPU, memory, and disk by adjusting the I/O interrupt frequency to acceptable levels. Note that the paging rate can be adjusted by selecting the appropriate memory size.

A rough estimation of the processor speed is the clock speed expressed in MHz. In PC-based systems, however, the processor is usually faster than the memory, which creates so-called wait states in the microprocessor execution. Through appropriate design, the wait states can be limited. In general, the slower the processor, the fewer wait states it experiences. The following formula is used to estimate the effect of the wait state on the overall processor speed using data collected from an actual system:

$$s = pr*pf/Tu = pr*pf/(util*Tm) \quad (1)$$

In this formula, pr is the average number of instructions executed between page faults, pf is the average page fault counter reported in the monitored period, and Tu is the time during which the processor was running processes in the monitored period, Tm. Tu is equal to the average utilization, $util$, times the monitored period, Tm. Since pr is independent of the processor speed (it is a function of the memory size and page replacement algorithm), for two monitored runs with the same monitoring period, Tm, the ratio of speed is:

$$s1/s2 = (pf1/pf2)*(util2/util1) \quad (2)$$

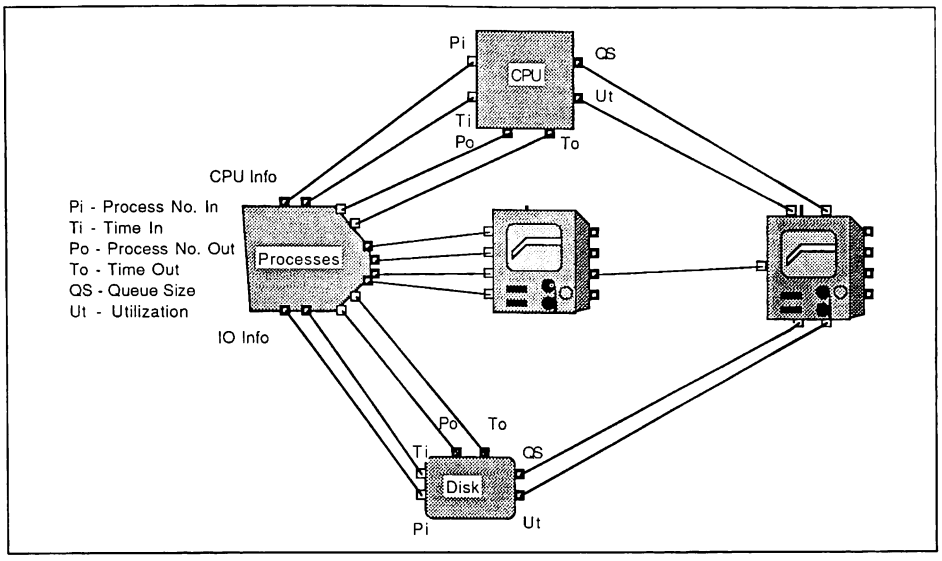

Figure 1. Graphical Representation of Simulation Components in Extend: Each Block Containing Simulation Code, Data, and User Screen Information; the Two Monitor Icons Representing Information Plotted on the Screen While the Simulation is in Process

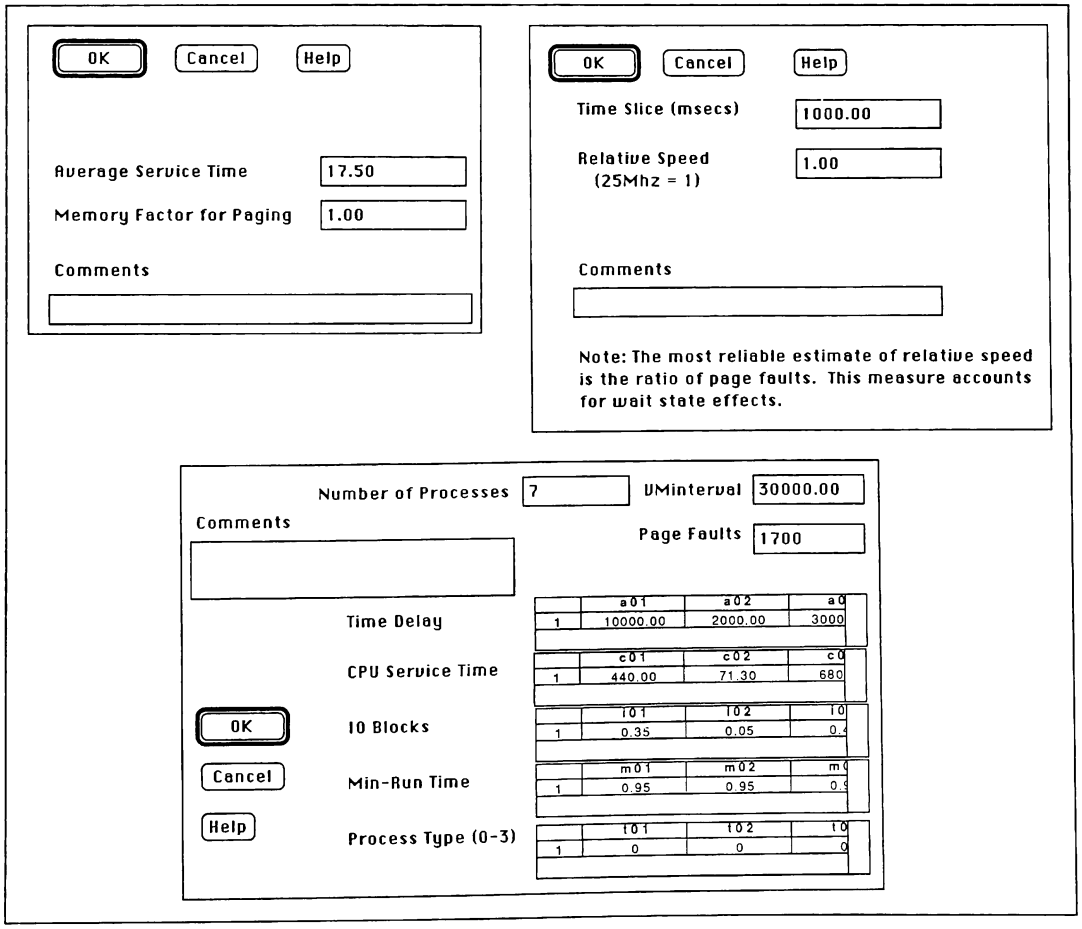

Figure 2. Forms Used to Elicit Parameters for Simulating a Specific Configuration: *top left*—Requests Parameters for Disk; *top right*—Requests Parameters for CPU; *bottom*—Requests Parameters for Each Process

and *pf1*, *util1* and *pf2*, *util2* are reported in system accounting reports (created by VMSTAT, described in Section 3.2.5, "Paging"). The measurements of speed are generally independent of the executed programs, and therefore can be used as a convenient yardstick in comparisons of different machine speeds.

3.1.2 Time Slice

The Time Slice value defines how long processes can run undisturbed on the CPU. Thus the time slice influences the turnover rate of the processes and, indirectly, their response time. Varying this parameter makes it possible to determine, through simulation, the value of the time slice that optimizes a particular application.

3.1.3 Memory Factor

The Memory Factor describes the paging rate relative to the standard memory of 4MB. Since the memory factor is a nonlinear function of the real memory size, it has to be evaluated by monitored runs on the actual system. To assess the impact of the memory size change on the paging rate, we need to adjust the reported number of page faults to the processor utilization. Thus the paging rate, p, of the process is:

$$p = pf/(util*Tm) \quad (3)$$

Note that the UNIX system keeps this rate the same for all processes running on the machine at any given time. The paging rates are algorithm- and load-dependent and need to be remeasured for each new software system implemented or designed for the machine.

The memory factor can be varied to estimate the most economical memory size. Note that in balanced systems the paging rate should be close to the I/O frequency. In a hardware configuration in which this match is achieved, an increase in memory would not be justified since the cost incurred by additional memory would not provide significant improvement in system performance.

3.1.4 Average Disk Service Time

Average Disk Service Time has three components: (a) *seek time*—time spent moving the read/write head to the appropriate cylinder on disk, (b) *latency time*—time spent waiting for the appropriate part of the disk to rotate under the head, usually assumed to be half of the full rotation time, and (c) *data transfer time*, which is dependent on the amount of data being transferred. The seek and latency times can be obtained from the disk drive technical specifications and are constant for each disk model. The transfer time can be approximated on the basis of the average size of the I/O request; therefore, the total disk service time can be encapsulated in a single parameter for a given application. The disk service time dictates the acceptable frequency of the disk requests. Total disk requests are the sum of I/O accesses and paging interrupts. Normally, the primary concern in selecting a disk drive is its capacity. However, its seek time (the largest part of the disk service time) is the most important factor in assembling a balanced hardware configuration.

3.2 Software Parameters

Parameters for each software process are *Mode of Operation, Time Parameter, CPU Service, I/O Blocks, Paging,* and *Variability.*

3.2.1 Mode of Operation

Each process is classified according to its mode of operation as *periodic, open, closed, or dependent.*

a. periodic—processes that are invoked cyclically at predefined times. Typical processes in this class are diagnostic or monitoring procedures called at constant intervals.

b. open—processes that execute cyclically at irregular intervals. These processes are called irregularly, independent of the system load. For example manually invoked maintenance processes or service processes (executed in response to customer requests) belong to this class.

c. closed—processes that include terminal interactions, where the user spends some time thinking about the next request after receiving a response from the computer.

d. dependent—processes that are dispatched with certain probability at the end of execution of another process (normally a periodic process). One periodic process can have many dependent processes. This class was introduced so that we could reliably simulate a situation in which a periodic process detects a problem and dispatches one or more diagnostic processes in response.

The difference between the closed class and the others is important. Processes in the closed class cannot overload the system, since the start of a new execution waits until the previous run has terminated. On the other hand, a process from the periodic, open, or dependent classes may request a new start before the current execution has terminated. This situation indicates serious overloading of the system. The simulation package delays the start of a new periodic or open process and cancels the request for start of a new dependent process.

The advantage of simulation over queuing network modeling is its ability to model overloading even though the average load of the system may not be extraordinarily high. Since the simulation does not rely on the average values of the system parameters, it is able to simulate maximum system requirements brought on by bursts of activity.

3.2.2 Time Parameter

The significance of the Time Parameter is different for each mode of operation. For a *periodic* process, this parameter represents the time between two consecutive process invocations. For an *open* process, the time parameter is the average interval between two consecutive executions of the process. The distribution of intervals is assumed to be exponential. The *closed* process time parameter represents the user thinking time that elapses between the termination of one execution and the start of the next one. The thinking time distribution is assumed to be exponential. For a *dependent* process, the probability of being dispatched is represented by the time parameter of the dispatched process and is obtained by dividing the time parameter of the dispatching process by the time parameter of the dispatched process. Each time the dispatched process terminates, it selects a random number between 0 and 1 and compares it to the probability of dispatching the dependent process. If the selected number is smaller than the probability, the dependent process is started.

3.2.3 CPU Service

The CPU Service parameter is the amount of CPU time required per invocation of the process. It is assumed that the CPU service time has an exponential-tail distribution, discussed in Section 3.4, "Simulating Process Variability."

3.2.4 I/O Blocks

The I/O Blocks parameter describes the I/O activities of the process. It can be evaluated and compared to the I/O block counts produced by system accounting programs. The I/O block counts are averaged over all the measured runs of the process in the report produced by the ACCTCOM program for analyzing UNIX accounting files. (ACCTCOM was developed at the GE Research and Development Center by William Davidsen. It provides I/O activity information not available with vendor-supplied tools.) In the simulation it is assumed that the blocks read/written in each run are distributed according to the exponential-tail distribution, discussed in Section 3.4, "Simulating Process Variability."

3.2.5 Paging

The Paging parameter is taken from the report produced by the XENIX utility VMSTAT. Since paging activity is kept equal for all active processes running under the XENIX system, a single system-wide paging factor is used for the application. The paging factor defines the frequency of paging interrupts of the running process, so the value reported by VMSTAT has to be divided by the CPU utilization in the reported period before it is entered to the simulation package.

3.2.6 Variability

The Variability parameter reflects the amount of variability in the processing time and I/O manifested by the process. See Section 3.4, "Simulating Process Variability."

3.3 Output Parameters

Two parameters define the output results of the simulation software for each process to be monitored, i.e., the process for which the maximum response time and average response time will be displayed in graphs.

3.3.1 Frequency of Output

The Frequency of Output parameter defines the period of simulated time that must elapse before new data is produced.

3.3.2 Number of Output Points

The Number of Output Points parameter specifies the number of simulated events desired for a given run. Specifying this parameter in terms of event count is a small inconvenience caused by the time variability of the simulation steps. Extend treats this output parameter as a fixed, total elapsed simulation time. For the reasons given in Section 2.3, "Simulation Tool–The Extend Simulation Environment," we use the ticks of Extend's standard clock to represent event counts.

3.4 Simulating Process Variability

Distribution of the random values that characterize software and hardware is important in building a reliable and trustworthy model. It is often assumed that *I/O requests, page faults,* and *processing times* are random in character and are well represented by a Poisson arrival process and that these events have an exponential distribution. Our experience indicates that these events behave differently (with smaller variance) than predicted by the exponential distribution. We attribute the smaller variance to the operating system activity and the design characteristics of the application software.

The UNIX operating system buffers *I/O requests* and empties unfilled buffers only when the periodic "flush" signal is sent (typically every minute). In addition, software modules often have regular patterns of requesting I/O, which vary little from run to run.

The UNIX operating system attempts to equalize the *page fault* rate of all the running processes. Any increased paging activity causes an extension of the process working set that results in a decrease in the paging rate of that process. Thus, the operating system tends to smooth interpage time distribution.

Processes can be classified as data-independent or data-dependent. Data-independent processes, like monitoring and reporting, require roughly the same *processing times* for each run. Processes in this category tend to generate similar amounts of I/O activity.

For expert system components, database software, and other data-dependent processes, both *processing time* and I/O activity are strongly dependent on the problem or query that is being addressed. The traditional approach to modeling this type of modified behavior is to use a hyper-exponential or Erlang distribution [Cohen 1982; Lee 1966] because they can be treated analytically.

It is unrealistic to assume that the processing time of a program has a non-zero probability of falling below some small value, as required by the hyper-exponential and Erlang distributions. Therefore, in our model, the inter-event time is represented by an exponential-tail distribution comprising a constant component and a variable component. The constant component represents the minimum inter-event time. The variable component has an exponential distribution with a mean value equal to the average event time minus the constant component. The average inter- event time and its variance can be determined from the monitoring data. The constant component should then be selected to match the measured data.

The exponential-tail distribution has the following density function:

$$f(x) = \begin{cases} 0, \text{ if } x < mt \\ \alpha * exp(-\alpha * (x - mt)), \text{ otherwise} \end{cases} \quad (4)$$

where $\alpha = 1/(at-mt)$, and at and mt are the average and minimum inter-event times.

By simple integration it can be established that the average value in this distribution is at and the variance is $(at-mt)*(at-mt)$, which is always less than the variance of the exponential distribution, $(at*at)$. The measured average inter-event time, at, and its variance, vt, can be computed from the monitored results; then the constant part of the exponential-tail distribution is given by the formula:

$$mt = at - \sqrt{vt} \quad (5)$$

3.5 Estimating Parameters for Modeling Applications

Resource requirements for actual system processes are needed for creating a realistic simulation model input, as well as for validating the accuracy of simulation results. The UNIX environment provides two sets of tools that are used for this purpose: ADM, a system administration package, and VMSTAT, a UNIX utility. ADM provides an accounting tool that tracks resource utilization for each process. This package provides a number of useful accounting statistics. For our model we used actual CPU requirements for a process *(cpu-sec)*; time elapsed while a process is active *(realtime)*; and number of blocks transferred while a process is active *(ioblks)*.

The I/O block count parameter could not be used directly because UNIX transfers blocks only when the I/O buffer is full or at the end of a fixed period. The *ioblks* parameter therefore records the transfers that take place while the process is active, not the transfers for which the process was specifically responsible. The *ioblks* parameter for a process was averaged over several runs so that we could use an average transfer parameter in our model.

The UNIX utility VMSTAT provides process count, paging, and CPU utilization information for the system from the time it was last booted or over a specified period. A 30-second period was used with three parameters: Number of processes *(procs)*; Memory paging *(paging)*, and CPU utilization *(cpu)* expressed as a percentage of the elapsed time.

4. EXAMPLE APPLICATION

The initial application of the simulation model was a field-based system for diagnosing electromechanical devices. This type of system is of particular interest because processes invoke other processes as a result of changes in the devices. The ability to understand the capacity of the computer system hardware to deal with the dynamic software demands of a diagnostic system is crucial.

Because field-based diagnostic systems are typically dedicated applications, the unused capacity of a system cannot be absorbed by the addition of other, unrelated processes. Since several of these systems are to be installed, simple overcapacity design is not an economically feasible solution.

The recent growth in diagnostic system capability indicates that upgraded functionality for systems already in the field must

be planned for. System simulation provides a reliable tool to estimate the effect of proposed enhancements on overall system performance prior to embarking on the expensive software development process.

5. MODEL VALIDATION AND RESULTS

A 25 MHz CPU with 4 MB of memory was the target machine for the first of two study cases. To validate the model, components of the actual system were run and performance and timing statistics were gathered using the accounting package ACCTCOM and the VMSTAT utility. The timing statistics were then used to estimate parameters of the model, and simulations were run to obtain simulated performance data, including overall system throughput and page faults. As can be seen from Figures 3 and 4, the average values and the general shape of performance data for actual and simulated runs are very close. Likewise, Figures 5 and 6 show close agreement on the actual and simulated page faults. The agreement shown in these two pairs of figures confirms the validity of the simulation model for the application being considered.

The model parameters were extrapolated to an 8 MHz CPU by changing the CPU-speed parameter to reflect an environment with one-third the CPU horsepower. A factor of 0.42 (rather than 0.34, which would be obtained from a CPU clock speed adjustment) was used, based on equation 2.

The results shown in Figures 3 and 4 show generally constant processing load with intermittent bursts of peak activity. The peaks are characterized by three factors: *height,* which defines the amount of a resource required (in this case the CPU); *duration,* which indicates how long the processes required the resource; and *frequency,* which indicates how often the resource was required.

Our goal is to be able to model the height, duration, and frequency of peaks accurately in order to provide the capability required to profile resource utilization and therefore estimate the unused capacity in the system. To achieve this goal a close match of the model results with reality is needed. Some portions of the actual system are implemented as a sequence of calls to UNIX utilities. As of this writing they are modeled as a single process, which may account for the distorted shape of the peak in Figure 3. The authors plan to extend the current model to enable them to accurately address this issue by providing a refined view of the software components of the modeled system, including interaction with a database system.

Peak height can be traded off for peak duration by adjusting CPU and disk parameters. For example, a slower disk will decrease the height because more processes will be waiting in the disk queue, causing an increase in total running time. The result is longer, flatter peaks. The frequency is dictated by the complexity of the monitored electromechanical device. Simplifying the computer view of the device will decrease the frequency of the peaks brought on by running processes designed to analyze external events.

6. DISCUSSION

Our experience shows that a dedicated system can be accurately simulated for predicting performance and estimating usage of resources. The model presented is a tool of universal application that can easily be validated by comparing the results it generates with reports produced by the UNIX system accounting tool VMSTAT.

Our approach makes use of the UNIX accounting tools to validate the model as well as determine simulation parameters. First an initial set of parameters is obtained for each process in the system under investigation; then these parameters are used in a simulation run. The results of the simulation are compared with the accounting output from the actual system. Once the model is validated, it can be used for determining the effects of changes in the hardware or software of the system.

During the course of this project, we learned that modeling tools such as Extend can make the construction of the model fairly straightforward. However, a thorough understanding of what the system being simulated is doing, as well as how it interacts with the operating system is required. For example, the

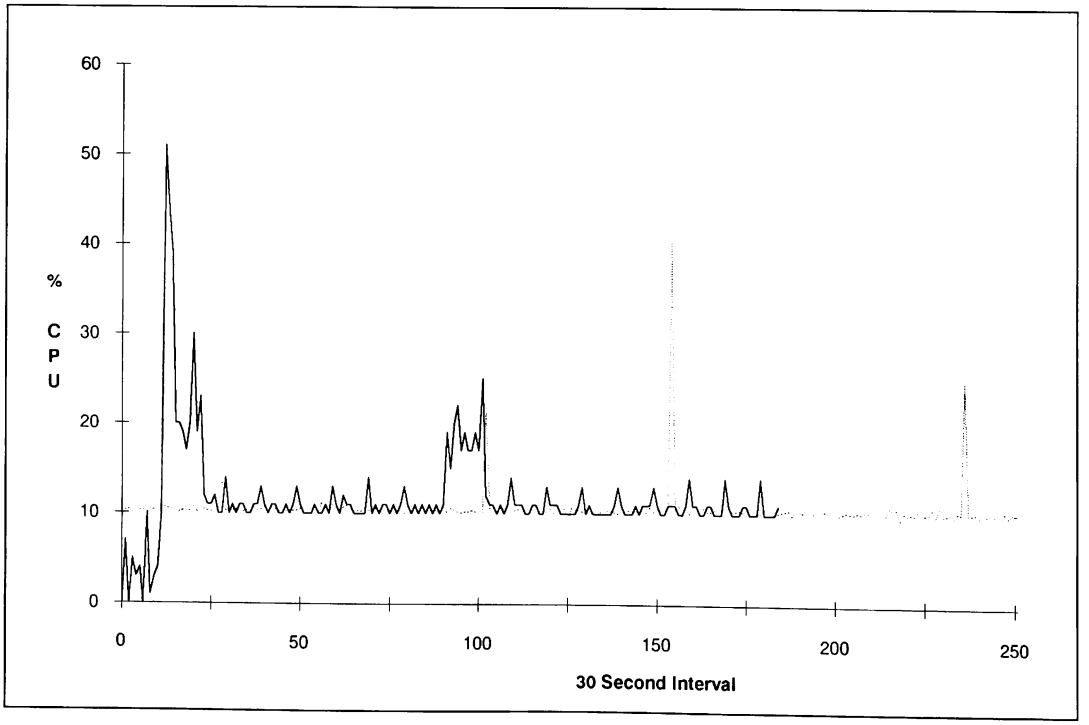

Figure 3. Comparison of Actual and Modeled CPU Utilization for a 25MHz CPU: Results Obtained Using VMSTAT from a Run of the System Shown as Solid Lines, Modeled Results Shown As Dotted Lines

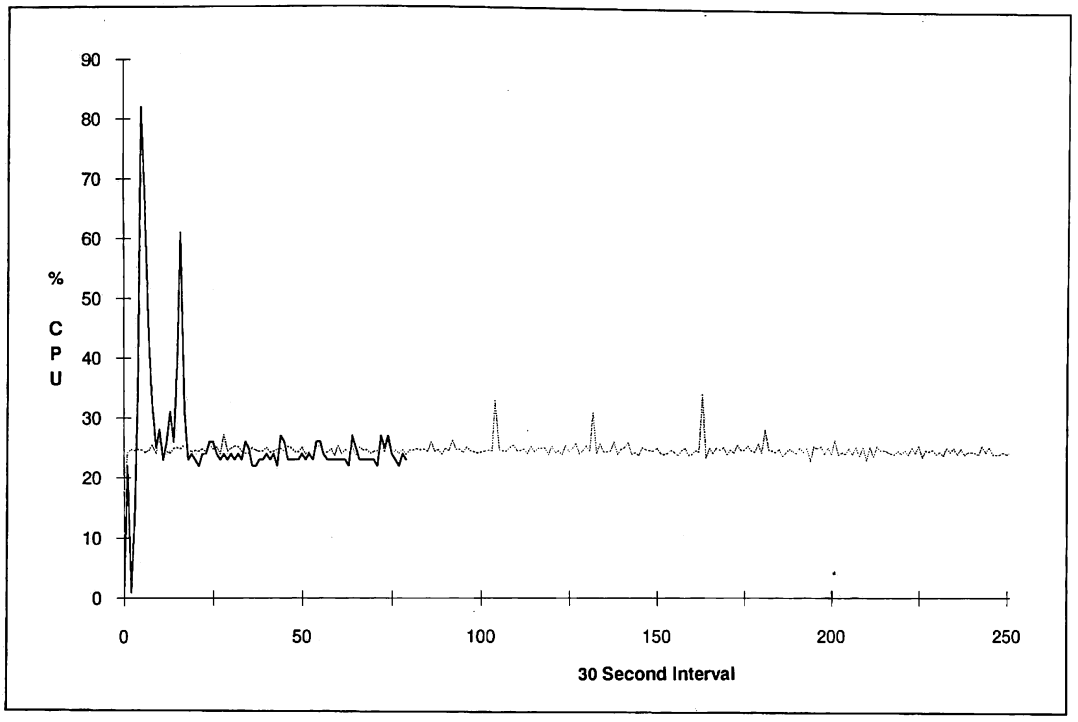

Figure 4. Comparison of Actual and Modeled CPU Utilization for an 8MHz CPU: Results Obtained Using VMSTAT from a Run of the System Shown as Solid Lines, Modeled Results Shown As Dotted Lines

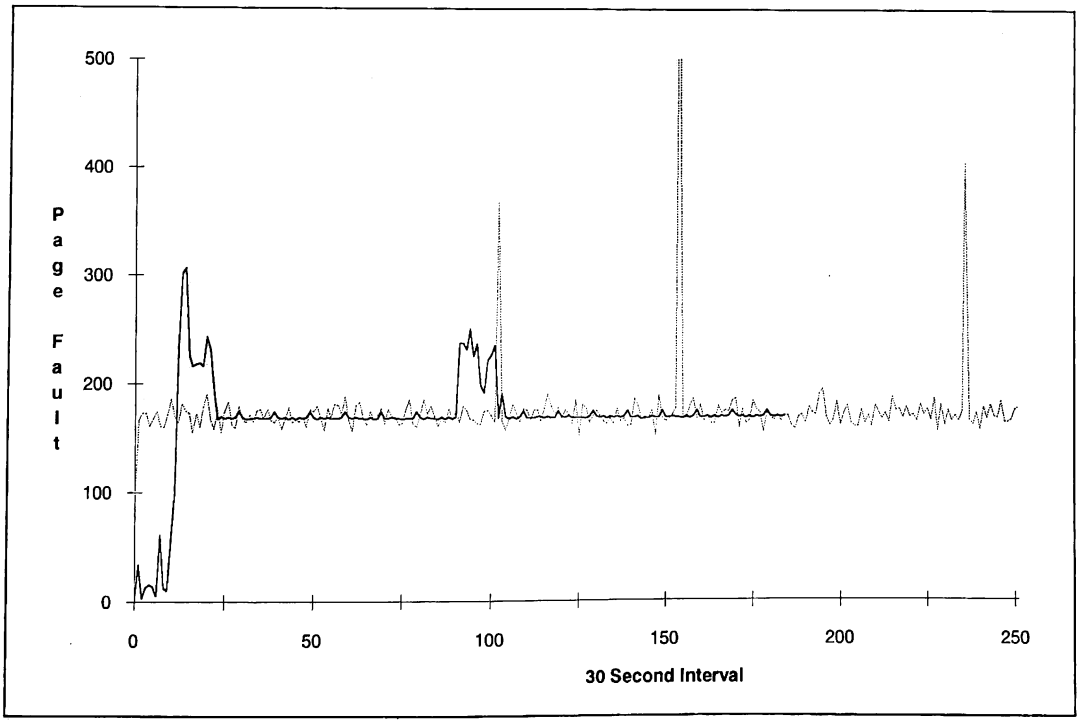

Figure 5. Comparison of Actual and Modeled Page Faults for a 25MHz CPU: Results Obtained Using VMSTAT from a Run of the System Shown as Solid Lines, Modeled Results Shown As Dotted Lines

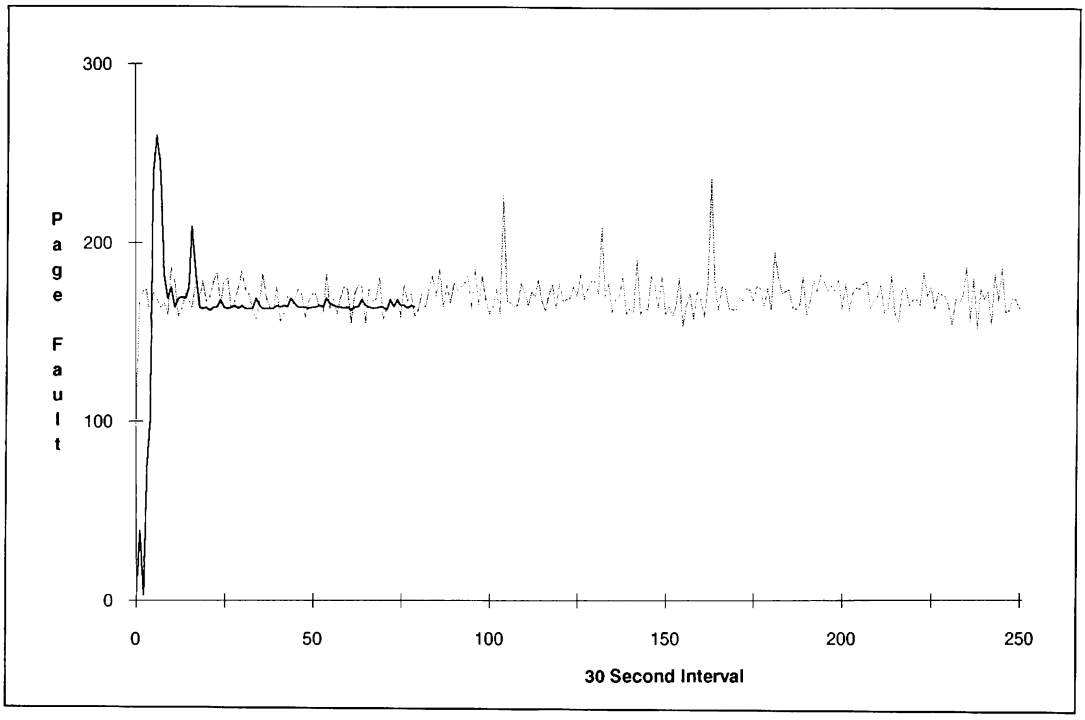

Figure 6. Comparison of Actual and Modeled Page Faults for an 8MHz CPU: Results Obtained Using VMSTAT from a Run of the System Shown as Solid Lines, Modeled Results Shown As Dotted Lines

issues of page faults and CPU normalization for wait states, discussed in detail above, had a major effect on the accuracy of the results.

Capturing I/O parameters in the UNIX environment was particularly challenging because the system posts information to the disk only when the I/O buffers are full or when a fixed time period has elapsed. The accounting tools attribute all I/O activity to the process that was running at the time it occurred. To overcome this limitation, I/O activity was averaged over several runs of the process.

Insights developed during this activity include understanding how variation in the software process affects accurate simulation of peak activity.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the help of William Davidsen, Richard Pfeiffer, and Gary Livingston of the GE Corporate Research and Development Center, whose UNIX expertise and experience were invaluable on this project.

REFERENCES

Anderson, G.E. (1984), "The Coordinated Uses of Five Performance Evaluation Methodologies," *Communications of the ACM 27*, 2,119-125.

Bard, Y. (1975), "Performance Analysis of a Virtual Memory, Time Sharing System," *IBM System Journal 14*, 4, 366-383.

Cohen, J.W. (1982), *Single Server Queue*, North Holland, New York.

Extend [Manual] (1988), Imagine That, Inc., 7109 Via Carmela, San Jose, CA.

Lazowska, E.D., J. Zahoran, G.S. Graham, and K.C. Sevick (1984), *Quantitative System Performance*, Prentice-Hall, Englewood Cliffs, NJ.

Lee, A.M. (1966), *Applied Queuing Theory*, Macmillan, New York.

Lucas, Jr., H.C. (1975), "Performance Evaluation and Monitoring." In *Software Systems Principles: A Survey*, P. Freeman, Ed., SRA, 509-524.