Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

A HYBRID SIMULATION-QUEUEING MODULE FOR MODELING UNIX I/O IN PERFORMANCE ANALYSIS

Barry L. Nelson

Department of Industrial Engineering
and Management Sciences
Northwestern University
Evanston, Illinois 60208, U.S.A.

William S. Keezer
Thomas F. Schuppe

LEXIS-NEXIS
Dayton, Ohio 45401, U.S.A.

ABSTRACT

LEXIS®-NEXIS® frequently develops simulation models to estimate the computer system capacity required to deliver on-line information services. The load imposed by Unix Input/Output (I/O) processes is a key factor in many of these models. However, the I/O processes themselves are seldom of interest, and explicitly modeling them results in simulations executing *much slower than real time*. This paper presents a hybrid simulation-queueing module that can be inserted into any simulation to accurately model I/O resource consumption and queueing delays without explicitly modeling each individual I/O process. This module is in use at LEXIS-NEXIS today (LEXIS and NEXIS are registered trademarks of Reed-Elsevier Properties, Inc., and are used under license).

1 INTRODUCTION

LEXIS-NEXIS uses a diverse collection of computer systems to deliver on-line information services worldwide. An important part of the development of new information services is estimating the system capacity required to support the product. Even for existing products, it is important to estimate capacity requirements for services experiencing growth.

At LEXIS-NEXIS, capacity estimates are typically based on a simulation of the processes involved in preparing and delivering a new service. Usually simulation models describe processes in great detail and include such tasks as computation of specific subprocesses, input/output (I/O) of data, remote procedure calls (RPC's), etc. The data used to describe time delays and resource consumption (CPU, memory, etc.) are based on historical data for familiar, well-understood processes. For an unfamiliar or completely new process, it is often necessary to estimate time delays and resource consumption based on expert opinion, since no data are available.

For I/O operations in Unix environments, no systematic collection of data had been done prior to 1994, so I/O times and resource consumption data were based on expert estimates. Most estimates were constants or simple functions of I/O size. These methods were used because they produced reasonable results and no better information was available. In 1994, however, extensive controlled testing was done on both HP and Sun servers to gather precise data on I/O times and resource consumption. At about the same time the modeling effort described in this paper was initiated to develop a better method for simulating resource consumption and time delays caused by I/O. The concept was to produce a portable module that could be inserted into any LEXIS-NEXIS performance simulation that required modeling local (as opposed to network) Unix I/O. The module needed to be customizable to the actual physical configuration of the hardware, and to simulate resource consumption and delays with high fidelity but minimal additional computational burden on the simulation. Ultimately, the HP and Sun laboratory tests were used to parameterize and validate this module. The remainder of the paper describes the processes used to develop, validate and apply this improved modeling process.

2 SYSTEMS MODELED

Both the HP and Sun systems of interest to us had similar overall architectures: Multiple CPUs are connected to memory modules via a high-speed backplane (about 240–960 megabytes per second). All I/O data passes through a converter that contains buffering to match the speed and the protocol of the peripheral bus to that of the backplane. SCSI controllers are connected to a peripheral bus with speeds on the order of 30–50 megabytes per second. The actual I/O devices are then attached to the SCSI de-

vices. The DASD (disk) devices for both systems had very similar architectures with on-board CPUs and buffers, as well as look-ahead capability. The details of each of the main components differs considerably between the two systems despite the similarity of the overall architectures.

The details of the architecture of the HP9000 Model T500 may be found in the article by Alexander, et al. (1994). The main concerns for our work were not the internals of the system, but rather the rates at which various modules process data and various buses and paths transmit data. The HP9000 backplane operates at a rate of 240–960 MB/s (megabytes per second), and our system had the maximum capability. Because of the high speed and the use of a packet-switching protocol on the bus to transfer data, we assumed that there would never be any queueing on this device, only transfer delays. Hewlett-Packard (HP) uses a device called a bus converter to provide an interface for the peripherals to the backplane. This device has two 64 MB/s Precision Buses, each of which can have 1–7 SCSI adapters attached to it. The actual data transfer rate after protocol overhead is about 42 MB/s. In our experience, this data bandwidth is sufficient to present no detectable queueing delays.

The DASD devices are connected to fast, wide SCSI adapters that have a nominal transfer rate of 20 MB/s. These adapters can accept up to 12 devices daisy-chained along a single data path. The data path from the device to the adapter has the capability of handling up to 20 MB/s for data buffered at the device. The details of the HP DASD devices may be found in Ruemmler and Wilkes (1993). Their paper gives the necessary details for calculating the device level delays and accounting for the effects of the on-board cache and the look-ahead algorithm of the device. More importantly, it provides an algorithm for calculating seek time that is much better than anything used previously.

At the operating system level, the HP9000 uses HP's version of Unix called HP-UX. This is a version of Berkeley Unix. The most important consideration here is that I/O is handled by a set of cache buffers that have their own management system. This became a major performance concern in trying to validate our module against the benchmark data. Based upon our results, the space devoted to I/O cache is divided into blocks of predetermined size, in our case 8172 bytes, and evenly distributed among buffer queues. The number of queues appears to be 128 plus the number of application files. To find out if a block of data is in cache, the operating system hashes to the correct queue and then sequentially searches the queue for the desired block. The implications are that the larger the cache is relative to the number of application files, the more time will be spent in sequentially searching the queues.

The Sun Microsystems SPARCcentera 2000 (SPARC 2000) is fully described in a white paper produced by Sun (1992). The main difference of importance in our model is that the backplane transfer rate is 640 MB/s. This still appears to be sufficiently fast that it creates no measurable queueing delay in the system. The internal architecture of the conversion module is considerably different from the HP9000, but only the timing is of importance. The Sun equivalent of the HP Precision Bus is the SBus. This bus has a transfer rate of 50 MB/s and can accept up to four SCSI adapters.

The SCSI adapters are labeled as differential, wide devices and have a transfer rate of 22 MB/s. Each adapter can accept up to 6 devices on a data path that transfers buffered data at 22 MB/s. From our viewpoint, there are no differences in the disk devices between the HP9000 and the SPARC 2000.

The operating system is Sun's proprietary version of Unix, SUNOS. It is different from HP-UX in its handling of I/O cache. In the SPARC 2000, cache is simply a part of memory and is handled by the memory manager. The overhead to access any block of cached data is a constant and the same as the overhead to access any other block of memory. This means that performance does not depend on the amount of cache versus the number of application files.

3 BENCHMARK DATA

The benchmark data used to validate the module and provide the values for some of the parameters was originally obtained during a stress test of the two different systems that were being considered for I/O-intensive applications. Nominally, the benchmark created a series of totally random READS of random length (1–64 kilobytes with an average of 32 kilobytes) at varying rates up to the maximum sustainable rate. A number of different configurations of SCSI adapters and DASD devices were tested. I/Os per second, CPU utilization, and response times were measured. In the case of the HP9000, several relatively complete throughput-response curves were obtained for the various configurations, and eventually a set of parameters was developed for our module that gave an error of 5% or less compared to the benchmark data. In the case of the SPARC 2000 tests, the system was greatly over-stressed, the goal being to see what the maximum throughput was. As a conse-

quence, not all parameters could be as carefully defined as for the HP9000. However, useful data were obtained.

4 THE UNIX I/O MODULE

This section describes the Unix I/O module. The module is a collection of Fortran subroutines that can be used in isolation, or can (more typically) be added to simulation models that need to account for local I/O resource consumption and queueing delays. A conceptual presentation of the module is shown in Figure 1; each box in the figure corresponds to a queue in the module. The philosophy of the module is to randomly sample processing times (service times, in queueing terminology) as a function of I/O size, but compute expected queueing delays based on system load as characterized by recent CPU utilization, read/write rates and the current probability of a memory cache hit. In this way the module achieves a high degree of fidelity without incurring the prohibitive computational cost of explicitly simulating each I/O.

Each subsection below describes one component of Figure 1, including the inputs that the component requires. Each component returns a (possibly randomly sampled) processing time and an expected queueing delay. Throughout the section, notation such as **iolam** indicates the name of a Fortran variable, and "ms" indicates milliseconds.

4.1 Cache

The CPU cache model has the following inputs:

$\lambda_{I/O} \equiv$ overall rate of reads and writes in MB/ms; this parameter may be dynamic (**iolam**).

$\lambda_{ru} \equiv$ rate at which data is reused in MB/ms; can also view $1/\lambda_{ru}$ as the expected time between requests for the same data; this parameter is not dynamic (**rulam**).

$c \equiv$ cache size in units of the average I/O size, $a_{I/O}$ (**avsize**); in Fortran cache size is **c**.

$p_{hit} \equiv$ Pr{memory cache hit}; this parameter can be specified or computed dynamically (**phit**).

Whether or not there is a local cache hit is modeled as a Bernoulli (0 or 1) random variable, where the probability of a hit, p_{hit}, on each trial is independent of the previous trial. This random variable is sampled.

The modeler can specify a value for p_{hit}, or it will be computed by a simple Markov-process model. The

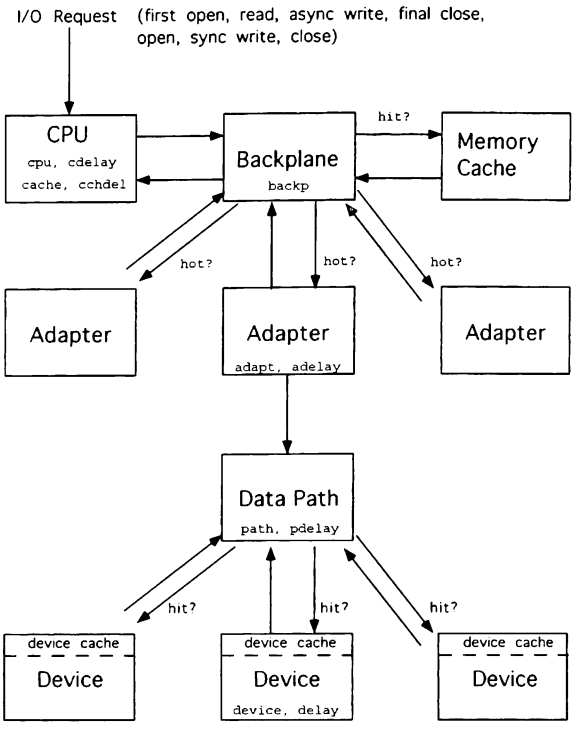

Figure 1: Conceptual View of the Unix I/O Module

Markov-process model assumes that data requests arrive to cache according to a Poisson process at a rate of $\lambda_{I/O}$, and data are reused at a rate λ_{ru}, with the time between reuse being exponentially distributed. A data quantum is pushed out of cache when it is the oldest quantum in cache and something new is pushed in. These assumptions allow a steady-state value of p_{hit} to be calculated analytically,

$$p_{hit} = 1 - \left(\frac{\lambda_{I/O}}{\lambda_{ru} + \lambda_{I/O}}\right)^c$$

4.2 CPU

The CPU model has the following inputs:

$\rho_{cpu} \equiv$ utilization per CPU for tasks with which I/O has to contend (i.e., other system calls); this parameter is dynamic (**crho**).

$k \equiv$ number of CPUs (**k**).

$\mu_{cpu} \equiv$ service rate of CPU in MIPS (**cmu**).

$\lambda_{cpu} = k\mu_{cpu}\rho_{cpu} \equiv$ the effective arrival rate to the CPU in the same units as μ_{cpu} (**clam**).

For each task performed, the constants γ_0 and γ_1 such that the total CPU processing time consumed by that task is given by $t(\text{I/O size}) = \gamma_0 +$

(I/O size)/γ_1, where there are different values of γ_0 and γ_1 depending on task and whether or not there was a cache hit, if appropriate (in Fortran **gamma0(task,hit)** and **gamma1(task,hit)**).

$1/\mu_{\text{cache}} \equiv$ the expected time to read the cache buffer (**cacherd**), which is a function of the number of pages in a string (**cachstrg**) and the time to read one page (**pagerd**).

$\lambda_{\text{cache}} = \lambda_{I/O}/a_{I/O} \equiv$ arrival rate to the cache-buffer queues in units of I/O's per time (**cchlam**).

The model returns a total CPU processing time and an expected total delay waiting for the CPU; the same quantities are returned for contention and use of the cache-buffer queues.

The collection of CPUs is modeled as an M/G/k processor-sharing queue. Processing times at the CPU are calculated as a deterministic function of overhead plus a linear function of I/O size, depending on I/O task and whether or not there is a cache hit. Expected delays waiting for the CPU are proportional to the total processing time for a processor-sharing model, and depend only on the service rate and not the distribution (Cooper, 1981).

Contention for the cache-buffer queue is modeled as an M/G/1 queue. The actual processing time depends on whether or not there is a cache hit or miss; when there is a hit then the time to search out the appropriate page is sampled.

4.3 SCSI Host Adapter

The adapter model has the following inputs:

$n \equiv$ number of adapters (**n**).

$n^* \equiv$ number of "hot" adapters (**nhot**).

$f_{\text{hot}} \equiv$ is as measure of imbalance between "hot" and not "hot" adapters, given as a fraction of all I/O requests routed to the "hot" adapters (**fhot**).

$\lambda_{\text{adpt}} \equiv$ arrival rate to the adapter that is used in the current calculation in MB/ms (**alam**). This value is computed as an appropriate fraction of $\lambda_{\text{I/O}}$.

$\mu_{adpt} \equiv$ service rate of adapter in MB/ms (**amu**).

For each task performed, the constants α_0 and α_1 such that the total adapter processing time consumed is given by t(I/O size) $= \alpha_0 +$ (I/O size)/α_1 (in Fortran **alpha0(task)** and **alpha1(task)**).

Whether or not the current I/O requires a hot adapter is a Bernoulli random variable with f_{hot} as the probability of requiring a hot adapter. The concept of a "hot" adapter is included to account for nonuniform load across devices. If a hot adapter is required, then

$$\lambda_{\text{adpt}} = f_{\text{hot}}\lambda_{\text{I/O}}/n^*$$

otherwise

$$\lambda_{\text{adpt}} = (1 - f_{\text{hot}})\lambda_{\text{I/O}}/(n - n^*)$$

The model returns a total adapter processing time and an expected total delay waiting for the adapter. Each adapter is modeled as an M/D/1 queue. Processing times at the adapter are calculated as a deterministic function of overhead plus a linear function of I/O size, depending on I/O task. Since two delays are experienced at the adapter (outbound to the data path and inbound from the data path), the total delay is

$$2 \times \frac{\lambda_{\text{adpt}}/\mu_{\text{adpt}}}{2(\mu_{\text{adpt}} - \lambda_{\text{adpt}})} = \frac{\lambda_{\text{adpt}}/\mu_{\text{adpt}}}{\mu_{\text{adpt}} - \lambda_{\text{adpt}}}$$

4.4 Data Path

The data-path model has the following inputs:

$\lambda_{\text{adpt}} \equiv$ arrival rate to the adapter that is used in the current calculation in MB/ms (**alam**). This value is computed as an appropriate fraction of $\lambda_{\text{I/O}}$.

For each task performed, the constants ω_0 and ω_1 such that the total data-path processing time consumed is given by t(I/O size) $= \omega_0 +$ (I/O size)/ω_1 (in Fortran **omega0(task)** and **omega1(task)**).

$\mu_{\text{path}} \equiv$ service rate of the data path in MB/ms (**pmu**).

The model returns a total data-path processing time and an expected total delay waiting for the data path. The data path is modeled as an M/D/1 queue. Processing times are calculated as a deterministic function of overhead plus a linear function of I/O size, depending on I/O task. Since two delays are experienced (outbound to the device and inbound from the device), the total delay is

$$2 \times \frac{\lambda_{\text{adpt}}/\mu_{\text{path}}}{2(\mu_{\text{path}} - \lambda_{\text{adpt}})} = \frac{\lambda_{\text{adpt}}/\mu_{\text{path}}}{\mu_{\text{path}} - \lambda_{\text{adpt}}}$$

4.5 Backplane

For each task performed, the backplane model requires constants β_0 and β_1 such that the total backplane processing time consumed is given by $t(\text{I/O size}) = \beta_0 + (\text{I/O size})/\beta_1$ (in Fortran `beta0(task)` and `beta1(task)`).

The backplane is modeled as pure processing time (overhead plus a linear function of I/O size) with no contention (queueing). Backplane time accounts for all transmission time down to the adapter level and back up from the adapter level.

4.6 Device

The disk storage device model has the following inputs:

$d \equiv$ number of devices per adapter (`d`).

$p_{\text{dhit}} \equiv$ probability of a device cache hit (`pdhit`).

$r \equiv$ total rotation time, in ms (`rotate`).

$p_{\text{short}} \equiv$ probability of a short head movement (`pshort`).

$t_{\text{short}} \equiv$ time to make a short head movement and settle, in ms (`short`).

$t_{\text{long}} \equiv$ time to make a non-short head movement and settle, in ms (`long`).

$a_{\text{I/O}} \equiv$ size of the average I/O (`avsize`).

$b \equiv$ size of a block in the same units as $a_{\text{I/O}}$.

$\delta_0 \equiv$ seek time + rotational latency; this value is sampled.

$\delta_1 \equiv$ time to transmit data in MB/ms on a physical read/write (`delta1`).

$\eta_0 \equiv$ set-up time for I/O when a device cache hit occurs, in ms (`eta0`).

$\eta_1 \equiv$ marginal transmission time when a device cache hit occurs, in MB/ms (`eta1`).

The device is modeled as an M/G/1 queue for the purpose of calculating delay. The service time at the device is composed of seek time, rotational latency and transmission time when there is a cache miss; it is composed of a set-up time and marginal transmission time when there is a cache hit.

The seek time is modeled as a two-point distribution on t_{short} and t_{long}. The expected value and variance of this random variable are used in the queueing model, but a sampled value is returned.

The rotational latency is modeled as a random variable that is uniformly distributed on the time to complete one rotation of the disk. The expected value and variance of this random variable are used in the queueing model, but a sampled value is returned.

The transmission time for a physical I/O is modeled as a linear function of the I/O size. The size of a block is used in the queueing model, but the actual transmission time is returned. Specifically, $t(\text{I/O size}) = \delta_0 + (\text{I/O size})/\delta_1$, where δ_0 = seek + rotational latency.

When a cache hit occurs, the device processing time is $t(\text{I/O size}) = \eta_0 + (\text{I/O size})/\eta_1$.

4.7 Task Models

The Unix I/O module can simulate the seven I/O tasks described below.

1. Open an unopened file without accounting for prefetch:
   (a) CPU processing and delay; always a memory cache miss
   (b) adapter processing and delay
   (c) data path processing and delay
   (d) device processing and delay; always a device cache miss
2. Read
   (a) determine if memory cache hit or miss
   (b) if memory cache hit, CPU processing and delay
   (c) if cache miss...
      i. adapter processing and delay
      ii. data path processing and delay
      iii. device cache hit or miss
      iv. device processing and delay
3. Write; write-back version with no housecleaning. That is, only a write to cache. Includes CPU processing and delay.
4. Close; concluding close.
   (a) CPU processing and delay; always a memory cache miss
   (b) adapter processing and delay
   (c) data path processing and delay
   (d) device processing and delay; always a device cache miss

5. Open an already open file. Treated as always a CPU cache hit. Includes only CPU processing and delay.

6. Write; synchronous version, so always a CPU cache miss.

   (a) CPU processing and delay

   (b) adapter processing and delay

   (c) data path processing and delay

   (d) device delay and processing time to write to cache

7. Close, but not concluding close. Treated as always a cache hit. Includes only CPU processing and delay.

Values are obtained by making calls to `subroutine io` as follows:

```
 subroutine io(cp,task,size,crho,iolam,p,
$  cache,cchdel,cpu,cdelay,backp,adapt,
$  adelay,device,ddelay,path,pdelay,hit,hot)
```

The inputs are the configuration platform `cp`, the `task` type (as listed above), the `size` of the I/O in MB, the current input/output rate `iolam` in MB/ms, and the current CPU utilization $0 <$ `crho` < 1. The dynamic parameters `crho` and `iolam` should not be based on instantaneous snapshots, however, since the delay values returned by `subroutine io` are based on steady-state queueing models. Rather, average values for a recent time period should be passed. In this way, the I/O module can reflect current load on the system. Some experimentation is required to determine an appropriate time window for averaging.

5 PARAMETRIZATION

In its final form, the Unix I/O module requires two different types of parameters: (1) those that change with each simulation scenario or group of scenarios and deal with the configuration of the system and the data layout on the devices, and (2) those that deal with the internal operating parameters of the platform. In a multi-system test, a different configuration parameter file is required for each computer system in the model, and a different internals file for each type of system. For example, if a system of two HP9000s and three SPARC 2000s was being simulated, five different configuration files would be required and two different internals files (one for HP and one for Sun).

The system configuration values that might change from scenario to scenario of the model are the amount of cache defined (in the case of the SPARC 2000 it is 0); the reuse rate of data in cache (used to calculate cache-hit probabilities if that value is not supplied); the effective number of CPUs (since these are multi-CPU systems); the total number of application files (required for calculating buffer queues); the number of adapters and the fraction defined as "hot" (this allows the model to account for heavily-used devices); the probability of a device cache hit (very important in response-time modeling); the transfer block size; and the total number of devices. All of these parameters may be defined by the Systems Administrator in a real system or are part of the what-if testing when trying to design a system. The only one that requires calculation is the probability of a device cache hit. In our applications, the size of the I/O and the totally random access patterns created a probability of a device cache hit of near zero, and we often used zero for this value. In cases where there are a high proportion of records shorter than a block and a high degree of sequential access, the probability of a device cache hit becomes quite high. Ruemmler and Wilkes (1992) discuss this issue in detail.

System internals are values that change only with a change of platform. They are implemented as permanent reference files that are called as needed when the overall computing environment changes its configuration. Specifically, they are matrices of values, based on various divisions of the model: CPU, backplane, cache manager, SCSI adapter, data path from device to adapter, and the device. For each component of the Unix I/O module there is a set of seven parameters, one for each of the I/O tasks. Each set of values includes a base service time and a data-size dependent rate for cache miss situations and the same pair for cache hit situations. For those elements that do not have dependence on the task or for those elemental parameters that are not task dependent, a single set of parameters was created.

The internal parameters for consumption of CPU, transfer on the backplane, adapter processing and transfer on the data path from device to the SCSI adapter each require a parameter for the fixed delay (in some cases zero) and another for the transfer rate for calculating data-size dependent delays. In the case of CPU processing, an additional set of parameters is created to account for the differences between memory cache hit and cache miss. There are a pair of these values for each of the seven I/O operations modeled. The CPU values are for setting up the I/O and not for doing the cache search, which was calculated separately then added to the calculated CPU time. These parameters were obtained primarily from the manufacturer's literature on the system. In the

case of the CPU, estimates were made of the fixed delays for operations other than READ based on the amount of work required to accomplish the operation. The amount of CPU for the READ operation was determined from the benchmark data. The module is not critically sensitive to any of these values.

Caching of I/O data is an important constraint on the performance of the systems in reality, and therefore requires considerable care to be represented properly in the module. To maintain integrity of the data, only one processor may access the cache data at a time. This means that when the CPU consumption of cache access and management is equivalent to one fully utilized processor, no higher throughput is possible.

In the case of the HP9000, the cache processing was particularly important for obtaining a good fit of the module to the benchmark data. Early plots of throughput response curves from the benchmark data indicated different curves at the device level for different configurations. We found that the curves were essentially the same for the same number of files. Coupling this information with the cache buffer management scheme, we were able to understand that this effect was due to the decrease in the sequential search time on a buffer queue. After considerable study of the benchmark data it was found that a good approximation for the amount of CPU time due to cache buffer processing could be obtained via a linear function of the number of application files. The y-intercept of this equation is the time to process a buffer when only one application file is present, and the slope is negative indicating a decrease in processing per buffer with an increase in the number of files. The slope is multiplied by the number of files and subtracted from the intercept to determine the delay per buffer. This value is then multiplied by the number of buffers in a cache buffer queue to determine the overall delay to search a single queue. This value is corrected for the probability of a cache hit and the fact that on average half of the queue is required to be searched to find the correct buffer.

In the case of the SPARC 2000, data were found in the benchmarks that indicated that the limit had been reached on cache processing. This information was used to determine the cost per I/O for cache handling. This is fixed for each I/O, regardless of configuration, because cache in Sun systems is handled as part of general memory. Therefore, the parameters for calculating cache buffer management were set to zero, and only a fixed time was used to determine the service time and queueing for this part of the module.

The individual disk devices presented the most important and sensitive portions of the module. Disk access is usually described by three parameters, seek time, rotational delay, and transfer time. However, as Ruemmler and Wilkes (1993) point out, the seek time is not linear, and is itself composed of several parts, acceleration, linear travel, deceleration, and settling. They provide an algorithm based on the distance the arm has to travel for calculating the seek time. This algorithm depends on the distance traveled, using one equation for less than one third of the full width of the disk, and a second for traveling the full width of the disk. We chose to approximate this as a two-point probability distribution using the values for a short seek, a long seek, and the probability of a short seek. The short seek time was set equal to that required to move one track. The long seek time was set equal to the time to move the distance from the middle of the file nearest to the spindle to the middle of the file farthest from the spindle in the benchmark test. The probability of a short seek (which implies the probability for a long seek) was calculated using the average response time for a very lightly loaded disk (essentially no queueing), the average rotation and transfer times, and the short and long seek times. The module is extremely sensitive to this value, because it directly affects the variance of the service time and therefore the queueing at heavy loads. Despite the sensitivity, once this parameter is established and when the I/O is distributed relatively evenly across the disk surface, the module will work for simulations other than the one used to create it. The values we calculated work quite well if the short and long seeks are correctly estimated. Thus, if the device changes or the data layout changes greatly, this parameter should be recalculated. Rotational delay and transfer rates are obtained from the manufacturer's literature. Other than the differences in rotational delay and transfer rates, both HP9000 and SPARC 2000 devices used the same calculations for disk device performance.

6 VALIDATION

Validation of the parameters was coupled with validation of the module in an iterative process. The data for validation came from the benchmarks, and the comparisons were made to throughput-response curves plotted from that data. The goal was to be able to reproduce all the curves from a benchmark run with one set of parameters. The process was essentially trial-and-error. Multiple runs would be made for a set parameters with varied throughput and an empirical best fit obtained. When the behavior of the module did not correspond to the benchmark system, further research into the details of Unix and the hardware was done, and the module modified

to account for the new findings. The process of fitting parameters was then repeated. The module required two modifications from the first edition, one to add queueing behavior to the connection between the device and the SCSI adapter, and the second and more important one to add the cache manager behavior. Our final validation gave results that could be superimposed on the HP9000 benchmark data with no more than a 5% error at any data point.

Because of the difference in the quality of the SPARC 2000 benchmark data when used for our purposes, a fit such as we obtained for the HP data was not possible. However, as mentioned above, the cache processing delay could be obtained as well as the CPU cost per READ I/O. The manufacturer's literature and experience were used to complete the set of parameters for this platform.

7 AN APPLICATION

After the basic modeling concept was developed and validated against the HP and Sun benchmark data, the module was implemented in an existing simulation model. At LEXIS-NEXIS the primary simulation modeling tool used is SLAMSYSTEM. The basic simulation package has been supplemented with about two dozen FORTRAN subroutines known as the Data Driven Modeling System (DDMS). DDMS was created at LEXIS-NEXIS to allow very large models to be represented by data files and to allow easy modification of these models. A detailed description of the process used at LEXIS-NEXIS is provided by Robinson (1994).

DDMS operates by using a basic network diagram that allows numerous options (await a resource, delay, free a resource, return to a network diagram, etc.) for each step in a process. The option selected is controlled by attribute values read in from a data file. This data file also provides all parameters required to accomplish a step or call additional subroutines to compute other required values.

The I/O module is a collection of three Fortran subroutines, and calling `subroutine io` one time provides estimates of the queueing delays and resource consumption for one I/O process. All information passes between subroutines via calling arguments and common blocks. To make the I/O module work in the SLAMSYSTEM/DDMS structure, it was necessary to incorporate these three subroutines into this structure by specifying five new attributes to carry additional information to the I/O subroutines.

The simulation model chosen to test the Unix I/O module was one that had been developed to provide capacity estimates for a new product. This new product requires that a large number of documents be processed and stored and that sets of relevant documents be prepared and shipped to customers each day. When the number of I/O's per document is scaled up to the number of documents and customers processed per day, we obtain a very large number of I/O's, on the order of 1,000,000 per day. These I/Os are accomplished on two servers, an HP hosting a Sybase database and a Sun acting as a file server.

Four different scenarios representing different levels of business volume were run for both the old I/O method and the new I/O module. While each run of the simulation using the new I/O module took longer than the corresponding run using the old I/O method, the time delays and resources consumed as generated by the new Unix I/O module were much more realistic. For example, the time required to process a batch of documents was now sensitive to the overall loading (utilization of the CPUs) of the system. Formerly, I/O processing times were assumed to be independent of current CPU utilization rates.

8 DISCUSSION

The potential for the Unix I/O module is substantial, since the simulation of every I/O to the detail accounted for in the module requires prohibitive amounts of simulation execution time. Because the module is parameter-driven, considerable attention must be paid to establishing the parameters. Many of the parameters may be obtained from manufacturers' literature. However, for critical parameters, thorough testing and understanding of the workings of the operating system and the hardware is required. The most critical areas are cache management and the disk device. In our experience, a combination of manufacturers' literature, a detailed description of the operating system (Leffler, et al. 1990), and careful analysis of benchmark data to understand cache management was required. Ruemmler's work (1992, 1993) provided the needed information on modeling the physical devices.

To fully and adequately parametrize the module, good benchmark data is essential. We were fortunate to obtain the HP9000 benchmarks, since they were designed for a different purpose. In the case of the Sun data, we obtained usable data but not as complete as the data for HP. We strongly recommend that specially designed benchmarks be commissioned for proper parameter determination. The characteristics of a good benchmark include systematically varied loads and configurations, with careful measurements of CPU utilization, response times, I/O rates, service versus user time, interrupt counts and memory

utilization.

When the module was tested by including it in the DDMS model, the run times for a single simulation of each scenario increased by 33–70%. However, the I/O module replaced two lines of SLAM code with a large subroutine. Furthermore, the results obtained with the Unix I/O module could not have been obtained in any reasonable computation time with a fully simulated I/O model. There are also a number of improvements that could be made to increase the efficiency of the module in the DDMS environment, most of them focusing on the interface between DDMS and the module. We estimate that about 50% of the current increase in run time could be eliminated, resulting in a 15–35% increase in run time over deterministic methods.

9 SUMMARY

A hybrid queueing-simulation module was developed for modeling Unix I/O. This module is parameter-driven and allows great flexibility in modeling both varying system hardware configurations and varying workloads. We found that cache management and the physical disk devices were the most important factors influencing the accuracy of the module. We also found that manufacturers' literature could provide many, but not all, of the necessary parameters. The remaining parameters had to be obtained by iterative validation and modification of the module against benchmarks. As might be expected, benchmarks are a vital link, and should be custom designed and run for this purpose. The hybrid module runs longer than a deterministic model of I/O, but provides results that could only be obtained from prohibitively long simulation runs.

REFERENCES

Alexander, T. B., K. G. Robertson, D. T. Lindsay, D. L. Rogers, J. R. Obermeyer, J. R. Keller, K. Y. Oka, and M. M. Jones, II. 1994. Corporate business servers: An alternative to mainframes for business computing. *Hewlett-Packard Journal* 45: 8–30.

Cooper, R. B. 1981. *Introduction to queueing theory*, 2d ed. New York: North Holland.

Leffler, S. J., M. K. McKusick, M. J. Karels, and J. S. Quarterman. 1990. *The design and implementation of the 4.3SD Unix operating system.* New York: Addison-Wesley.

Robinson, J. N. 1994. Capacity and performance analysis of computer systems. In *Proceedings of the 1994 Winter Simulation Conference*, ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Selia, 34–41. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Ruemmler, C., and J. Wilkes. 1992. Unix disk access patterns. In *USENIX Winter 1993 Technical Conference Proceedings*, San Diego, California.

Ruemmler, C., and J. Wilkes. 1993. Modeling disks. HP Laboratories Technical Report HPL-93-68, revision 1.

SPARCcentera 2000. 1992. Technical White Paper, Sun Microsystems, Inc.

AUTHOR BIOGRAPHIES

WILLIAM S. KEEZER has been with LEXIS-NEXIS for ten years and is currently a Staff Systems Engineer focusing on mainframe system storage management and performance issues. He has also done considerable work at LEXIS-NEXIS on mainframe and Unix platform application and communication performance. Before coming to LEXIS-NEXIS, he was an in-house consultant on OLTP system performance problems for the Data Pathing Division of NCR. He holds B.S. and Ph.D. degrees from the University of Oklahoma, and is a member of ACM and CMG.

BARRY L. NELSON is an associate professor in the Department of Industrial Engineering and Management Sciences at Northwestern University. He is interested in the design and analysis of computer simulation experiments, particularly statistical efficiency, multivariate output analysis and input modeling. He is the simulation area editor for *Operations Research* and will be Program Chair for the 1997 Winter Simulation Conference.

THOMAS F. SCHUPPE is the Manager of System Integration at LEXIS-NEXIS. He received a B.S. in Mechanical Engineering from the University of Wisconsin, an M.S. in Systems Engineering from the Air Force Institute of Technology, and a Ph.D. in Operations Research from The Ohio State University. His primary research interest is in simulation modeling of complex man-machine systems. He is currently a member of INFORMS and SCS.