AN ANALYTICAL MODEL FOR *UNIX*® SYSTEMS

Gopalakrishnan Ramamurthy

This paper presents an analytical model for a single-processor interactive computer system running under the UNIX operating system. The model is a closed multichain, multiclass priority-queueing network model. The model solution is an approximation and is based on mean value analysis. The validity of the model has been established by using it with a variety of workloads and workload mixes and with a number of different computer systems. Mean response times predicted by the model are within 10 percent of measured values. The model can be extended to tightly coupled multiprocessor systems and multiple UNIX systems running with network file systems over local-area networks.

Gopalakrishnan Ramamurthy *is a member of technical staff in the Performance Analysis Department at AT&T Bell Laboratories in Holmdel, New Jersey. He works on modeling and analysis of systems based on the UNIX operating system, factory information systems, packet voice multiplexers, and congestion control in broadband networks. He received a bachelor of engineering degree in electronics engineering from Bangalore University, a master of engineering in electrical communication engineering from the Indian Institute of Science, and a Ph.D. in electrical engineering from the University of Aston, Birmingham, United Kingdom. He joined AT&T in 1984.*

Introduction

The UNIX system is a general-purpose time-shared operating system that runs on many of today's mainframes, super minicomputers, and microcomputers. It is used not only to support software development, text processing, office automation, and business applications, but also in many dedicated applications such as operations support systems and process control systems. Despite the growing popularity of UNIX systems, however, a suitable model for analyzing and quantifying the performance of UNIX system-based computers has been lacking.

Traditionally, computer systems have been modeled as product-form-queueing networks[1,2,4] in which jobs are served at the central processing unit (CPU) either on a first-come, first-served basis or according to a processor-sharing discipline. An example is the central server model.[1-4] These models give accurate predictions of mean throughput and mean response time for certain workloads and workload mixes, where *workload* characterizes the resources required for a job. However, they do not take into account completely the subtle effects of resource management in the operating system. Therefore, they often fail when several different types of workload run concurrently on the system.

The UNIX operating system, for example, not only does book-

keeping, handles interrupts, schedules processes, and manages memory, but also services user requests via system calls which cannot be preempted by other processes. Although the UNIX system kernel is nonreentrant, the system services interrupts immediately so that interactive users see a fast response. Workloads may be CPU-intensive or they may be input-output-intensive—or both. CPU-intensive workloads may spend significant amounts of their time either in the kernel mode or in the user mode. I/O-intensive workloads may cause frequent interrupts. The operating system treats these different types of workload very differently, resulting in distinctly different performance for each type of workload. A simple product-form-queueing network model such as the central server model does not capture these effects.

This paper presents a model of a single-processor interactive computer system that takes into account some of the key characteristics of the UNIX system kernel. The model is a closed-multichain, multiclass queueing network model. The CPU serves jobs according to either a first-come, first-served or a processor-sharing discipline, depending on the class to which they belong. The solution of the model is by an approximate method that is based on mean value analysis.[5]

To use the model, we simply characterize commands in each application by their resource requirements—for example, CPU time, number and types of system calls, number of logical blocks read and written, and relative frequencies with which the command types are used. We execute the model with the chosen configuration of the system, the set of applications running concurrently on the system, the workload characterization and number of users associated with each application, and the hardware device parameters. The model predicts hardware component utilization as well as the mean response time of each type of command and the mean throughput of each application.

The model can be used for performance evaluation, performance prediction, and capacity planning of systems running under the UNIX operating system. Configuration changes (for example, adding disks or upgrading hardware with a faster CPU) and changes in workload mix can also be evaluated conveniently.

The validity of the model has been checked with a variety of workloads and workload mixes and a range of hardware from microcomputers to superminicomputers running under the UNIX operating system. The model predicted mean response times within 10 percent of experimental results.

The paper gives a brief summary of process scheduling in UNIX systems and a high-level view of an interactive computer system running under the UNIX operating system. It describes the model, analyzes it, and presents some validation results. Readers are assumed to have basic knowledge of the UNIX operating system (such as that provided by References 6 to 8), although the abstract mathematical model is fairly general.

Process Scheduling in UNIX System

Processes may run in the user mode or the kernel mode. In the user mode, a process executes user programs and accesses user data segments. In the kernel mode, a process executes kernel codes and accesses kernel data segments. When a process executing in the user mode needs system service, it makes a system call, which is executed in the kernel mode.

Processes are scheduled to run according to their priorities. Higher priorities are assigned to kernel processes for events such as interrupts from the disk subsystem and the terminals. User processes are assigned priorities according to their recent CPU usage. If a user process switches to the kernel mode, it still has a user-level priority unless it has to wait for kernel resources to become available or for an event to occur. When a waiting process is ready to run again, it is assigned a kernel-level priority associated with the resource or event that caused it to wait, or "sleep." For example, a process sleeping on a disk I/O subsystem has a higher priority than a process sleeping on a terminal input. All processes with kernel-level priorities are nonpreemptible and, in turn, can preempt processes executing with user-level priorities.

Once the kernel allocates the CPU to a user process, the user process can hold the CPU until the next 1-second clock interrupt, if no other interrupt occurs. If a hardware interrupt occurs while the CPU is executing a process in the kernel mode, execution of the interrupted

process resumes after all the interrupts have been handled. If an interrupt (hardware or software) occurs while the CPU is executing a process in the user mode, execution of that process is preempted. After the CPU has served the interrupt, the scheduler assigns the CPU to the process with the highest priority that is ready to run. Periodically (once every 10 milliseconds in UNIX System V), a clock handler checks on the recent CPU usage of the currently executing process. Once every second, the clock handler adjusts the recent CPU usage of all the active processes and recalculates the priority of every process that is ready to run. Hence, no user process can hold the CPU for more than a second if there are other processes ready to run. In UNIX System V, the adjustment reduces the recent CPU usage of every active process by a factor of two. The user-level priority of a process is given by the sum of its recent CPU usage and the base-level user priority, where the base-level user priority is the threshold priority difference between the kernel mode and the user mode.[6] A low numerical value for the user-level priority means a high scheduling priority.

The recalculation upgrades the priorities of processes that have been waiting for the CPU and penalizes processes that have used the CPU recently. This approximates a round-robin scheduling policy for processes with user-level priorities.

System Description

This section provides a high-level view of an interactive computer system running under the UNIX operating system. Users at terminals submit commands (for example, UNIX system shell commands) to be executed by the system. Each active user has a process in either a sleep state or in a run state. A user process, after completing the execution of the current command, executes a read system call to read the next command from the user's terminal. If the user has not submitted a new command, the corresponding user process enters a sleep state. The arrival of a new command at the CPU causes an interrupt, which, on being served by the CPU, awakens the user process that has been sleeping on the read system call. The awakened user process assumes kernel-level priority and completes the read system call. The user process then returns to the user mode and begins to execute the command. If another process is ready to run in the kernel, the process under consideration relinquishes the CPU and is scheduled to run at a later time. During execution of the command, the user process may make several system calls, some of which may result in block and character I/O operations.

Each read system call for block I/O produces a sequence of block reads. The block requested by the read system call may be found in the disk cache buffer. If the required block is not in cache, the kernel initiates a physical read to the disk subsystem. Because all block reads to the disk subsystem are synchronous, the user process that initiates the block read enters a sleep state until the requested block has been completely transferred to the disk cache buffer. Then an interrupt is generated to wake the sleeping process. The user process once again can run in the kernel with kernel-level priority, and the block is copied into the user space. Once the read system call is complete, the user process switches back to the user mode.

Each block I/O write system call from a user produces a sequence of block writes. Block writes are generally asynchronous. After a block is copied into the disk cache buffer, the user process does not wait for the block to be written to the disk. At a later time, when the kernel requires a free disk cache buffer and if none is available, the kernel selects a buffer on which a delayed write is pending. The kernel then writes the contents of the buffer to the disk.

Model Description

This section describes a priority queueing network model of an interactive computer system under the UNIX operating system. The hardware resources in the model consist of a set of interactive terminals (or users), a single CPU, and a set of disks. We assume that users at each terminal submit commands to the system and wait for a response. When the system responds, a user enters a "think" state. The user can then submit the next command and exit the think state. Users who submit the next command before they receive a response for the command currently being processed can be accounted for by choos-

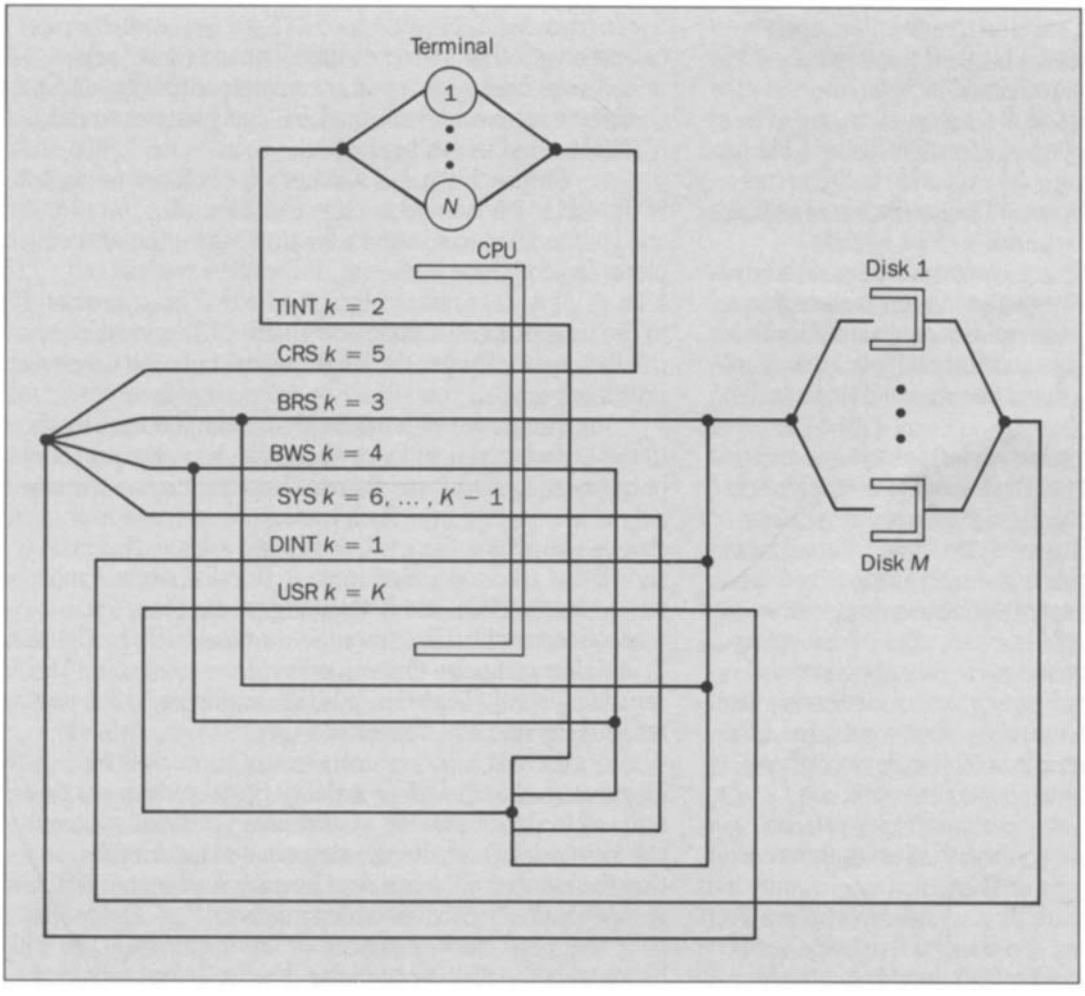

Figure 1. Queueing network model of an interactive computer system running under the UNIX operating system.

ing an appropriate value for the think time.

Figure 1 shows the priority queueing network model. N simultaneously active users are represented by N circulating jobs. R concurrent but distinct applications run on the system, and N_r users are associated with application r ($r = 1, 2, \ldots, R$). Thus

$$\sum_{r=1}^{R} N_r = N$$

The R concurrent applications and the number of users associated with each application are represented by R closed chains with a population vector $\mathbf{N} = [N_1, N_2, \ldots, N_R]$. Commands in application r can be grouped further into distinct command types. However, to keep the notation simple and without loss of generality, we will treat only the case where, in each application, all commands belong to a single type. Once a job leaves the terminal, or think node, it may visit the CPU several times before returning to the think node. The visits to the CPU

may also be made between visits to the disk I/O subsystem. Jobs can also revisit the CPU when they change from the user mode to the kernel mode and vice versa.

During a CPU visit, a job is assigned to one of K classes, depending on its mode of execution at the CPU. Class 1 and class 2 represent jobs executing in the kernel mode with kernel-level priorities. They are nonpreemptible and can preempt jobs in other classes. Jobs in class k ($k = 3, 4, \ldots, K - 1$) are those executing in the kernel mode with user-level priorities. A job in class K is a job executing in the user mode and has only a user-level priority. For simplicity, we assume that multiple classes are possible only at the CPU node; at the disk and think nodes, all jobs belong to a single class.

To complete the model, we further assume that the CPU is a fixed-rate server (that is, all jobs receive service at the same rate and the service rate is independent of the queue population). It serves class 1 and class 2 jobs according to a nonpreemptive first-come, first-served discipline, class 1 jobs having nonpreemptive priority over class 2 jobs. Jobs in classes 3, 4, . . . , $K - 1$ executing in the kernel mode and jobs in class K executing in the user mode are served according to a processor sharing discipline, all with the same priority. In other words, the CPU first serves class 1 and class 2 jobs as they arrive, giving preference to class 1 jobs. It will not interrupt a class 2 job for a newly arrived class 1 job, however. If there are no class 1 and class 2 jobs waiting, the CPU serves the remaining jobs.

The model represents the disk subsystem by a set of M first-come, first-served queues with fixed-rate servers. It represents the interactive terminals or the think node by an infinite server node.[1,2,4]

To determine the routing probabilities of jobs of various classes, let us follow the execution of a job in the model with the aid of Figure 1. A job is at the think node when the corresponding user is in a think state. When the user submits the next command, the corresponding job joins the CPU queue as a class 2 job. In UNIX system terminology, the arrival of a class 2 job represents an interrupt from a terminal. The terminal awakens the user process that has been sleeping on a read system call. Class 2 jobs (denoted in Figure 1 as TINT for *terminal interrupt*) can preempt other jobs executing with user-level priorities. As the read system call is completed, the class 2 job completes its execution at the CPU and switches to class K (denoted in Figure 1 as USR).

Clearly, if there is another job of class 1 or class 2 at the CPU, the job now in class K relinquishes the CPU, and is scheduled to run at a later time. Once the job completes its execution in class K, it switches to class k ($k = 3, 4, \ldots, K - 1$) at the CPU. This corresponds to the user process making one of ($K - 3$) system calls, a call that is executed in the kernel mode, but with user-level priorities.

The model views transitions from the user mode to the kernel mode, and vice versa, as class changes. For purposes of analysis, we assume that execution in the kernel mode resulting from such transitions is in one of k classes ($k = 3, 4, \ldots, K - 1$). We assume that class 3 and class 4 are read and write system calls, respectively, for block I/Os (BRS and BWS in Figure 1). Class 5 is a read system call to read from the terminal (CRS in Figure 1) and thus complete the execution of the command. The remaining non-I/O system calls (SYS in Figure 1) are represented by one of k classes ($k = 6, 7, \ldots, K - 1$).

A chain r job currently executing in class K switches to class 3 with probability $P_{r,3}$ to make a read system call for block I/O. We assume that each read system call for block I/O results in a sequence of block reads, and that the number of blocks read by each read system call is geometrically distributed with parameter $P_{R,r}$. The probability that each block requested by the read system call will be found in the disk cache buffer (that is, a read cache hit will result) is $P_{Rc,r}$. Hence, with probability $(1 - P_{Rc,r})$, a physical read is initiated to the disk subsystem. The probability that the requested block is located on disk m is $P_{Dm,r}$ ($m = 1, 2, \ldots, M$). All block reads to the disk are assumed to be synchronous. Therefore, the user process that initiates the block read waits, or enters a sleep state, until the requested block becomes available in the disk cache buffer. Hence, in our queueing model, a chain r job executing in class 3 visits disk queue m with probability $(1 - P_{Rc,r})P_{Dm,r}$.

After it has been served at the disk subsystem, the chain r job visits the CPU as a class 1 job. This corresponds to an interrupt from the disk subsystem signaling that the requested block has been transferred to the disk cache buffer. The class 1 job (denoted in Figure 1 as DINT for *disk interrupt*) has preemptive priority over jobs executing in the CPU with user-level priorities. After the chain r job has completed its execution in class 1 at the CPU, the job revisits the CPU in class 3 with probability $P_{R,r}$ to read another block. With probability $(1 - P_{R,r})$, the chain r job returns to the CPU as a class K job. The change to class K corresponds to the user process completing the read system call and switching back to the user mode. Because interrupts from the disk I/O subsystem have a higher priority than interrupts from character I/O devices, we assume that all class 1 jobs have nonpreemptive priority over class 2 jobs.

Each block I/O write system call results in a sequence of block writes. In our queueing model, a chain r job switches from class K to class 4 with probability $P_{r,4}$ to make a write system call. We assume that the number of blocks written by a job in chain r while executing a write system call is geometrically distributed with parameter $P_{W,r}$. We also assume that all block writes are delayed writes, and hence are asynchronous; that is, the user process copies the block into the disk cache buffer and does not wait for the block to be written to the disk.

After executing in class 4, a chain r job revisits the CPU as a class 4 job with probability $P_{W,r}$ and writes another block into the disk cache buffer. With probability $(1 - P_{W,r})$ the chain r job returns to the CPU as a job in class K, thus completing the write system call. Because the writes are asynchronous, a block written into the disk cache buffer by a chain r job is transferred to the disk subsystem with probability $(1 - P_{Wc,r})$ at a later time.

We can characterize straightforwardly an application and map the load it places on the system (that is, its resource requirement) into parameters for the model. For the sake of portability, we characterize the workload at the system call level—that is, by the number and types of system calls, the number of logical block reads and writes, the cache hit ratios for block reads and writes, and the mean time spent in the user mode and in the kernel mode. Many of these parameters can be either estimated or derived with the help of UNIX system accounting and performance monitoring packages such as the System Activity Report and the Process Status Report.[9]

Solution of the Analytical Model

The detailed queueing network model of the UNIX system described above does not have the simple, well known product-form solution[1,2,4] because of its priority classes. Although such networks can be treated, in principle, by solving global balance equations, the solution is computationally expensive and applies only to very small networks with a small number of classes and small population. An example is the preemptive and nonpreemptive M/G/1/K priority queue discussed by Jaiswal,[10] which can be considered a closed queueing network with two service centers and K jobs, where one center is a priority center and the other is an infinite server. Avi-Itzhak and Heyman[11] solved a homogeneous central server model in which all priority classes have identical service times and routing probabilities. Morris[12] has developed exact solutions for homogeneous open and closed networks and a number of nonhomogeneous, two-node network models with two priority classes, in which both nodes are priority centers.

An alternative, computationally efficient approach is to solve approximately. Approximations use a product-form solution for the priority queueing network that does not satisfy local balance. The accuracy of the approximation can be established through validation—that is, by comparing its predictions with actual performance or with results obtained from simulation.

A variety of approximate models are available. The reduced service rate approximation[13] is based on the fact that, in a preemptive priority queueing system, lower-priority jobs see a server whose mean service rate is reduced because it serves higher-priority jobs. Sevcik's shadow server approximation[14] applies the reduced-service-rate approximation to a more general class of networks. Kaufman[15] developed an improved reduced-service-rate approximation. Schmitt[16] developed a state-dependent reduced-service-rate approximation. Bryant, Krzesinski,

Teunissen, Lakshmi, and Chandy[17-19] and Van Doremalen and Wijbrands developed methods to solve closed priority queueing networks,[20] based on the reduced-service-rate approximation and heuristic extensions to the mean value analysis algorithm.[5] The approximation combines the results of the M/M/1 priority queue[10,21] and the reduced-service-rate approximation. The MVA algorithm is used to solve the resulting model.

The models in the literature[10-21] assume that the priority of a job is associated with the chain to which the job belongs and that jobs cannot move from one chain to another. They also assume that the priority center has either a preemptive or a nonpreemptive discipline, but not both. In this section we present an approximate (analytical) solution of our model based on results for the M/M/1 priority queue and on mean value analysis. In this model, priorities are associated with the class to which a job is assigned when it visits a priority service center (that is, the CPU), rather than the chain to which it belongs. Hence a job can have several different priorities during different visits to the same priority service center. Furthermore, some job classes have preemptive priority, while others have nonpreemptive priority at the same priority service center.

Our queueing network model of the system described in the section "Model Description" is a closed queueing network with $(M + 2)$ service centers (one think node, one CPU node, and M disk nodes), R chains with K classes at the CPU node, and N_r users in chain r $(r = 1, 2, \ldots, R)$. The think node is an infinite server node. Each of the M disks is a first-come, first-served queue with an exponential service-time distribution that is common to all jobs. The CPU node is a priority service center with three levels of priority. All service times at the CPU node are exponentially distributed.

Jobs visit the CPU node in one of K classes. Class 1 and class 2 jobs do not preempt each other, but have preemptive priority over jobs in class k $(k = 3, 4, \ldots, K)$. Jobs can arrive at the CPU node in one of three ways:

- Jobs from the disk I/O subsystem are in class 1 and have the highest priority in the model.
- Jobs from the terminals or any other character I/O device are in class 2 and have the next higher priority.
- Jobs in class K can change to any one of k classes $(k = 3, 4, \ldots, K - 1)$.
- A class K job can arrive at the CPU node only through a class change from class k $(k = 1, 2, \ldots, K - 1)$.

Approximate Analysis

We now develop an approximate analysis based on mean value analysis for the model discussed in the previous sections. Consider an M/M/1 priority queue fed by K Poisson arrival streams with mean arrival rates $\lambda_1, \lambda_2, \ldots, \lambda_K$. Let jobs belonging to class j have priority over jobs belonging to class k, with $k > j$ (that is, $k = j + 1, j + 2, \ldots, K$). Then the class j mean sojourn time W_j when priorities are preemptive is[10,21]

$$W_j = \frac{S_j + \sum_{p=1}^{j} L_p S_p}{(1 - \sum_{p=1}^{j-1} \rho_p)} \qquad (1)$$

where S_j = mean service time for a job with priority j
L_p = mean number of priority p jobs in the queue
ρ_p = utilization of priority p jobs

When priorities are nonpreemptive, the mean sojourn time is[10,21]

$$W_j = S_j + \frac{\sum_{p=1}^{j} L_p S_p + \sum_{p=j+1}^{K} \rho_p S_p}{(1 - \sum_{p=1}^{j-1} \rho_p)} \qquad (2)$$

Although the arrival process in our model does not have a Poisson distribution, we nevertheless use equations (1) and (2) as the basis of our approximation.

In our model, jobs in class k $(k = 3, 4, \ldots, K)$ receive service (according to the processor sharing discipline) only when there are no class 1 or class 2 jobs

requiring service at the CPU. When a class 1 or class 2 job arrives (from the disk or the terminal) all the jobs executing in class k ($k = 3, 4, \ldots, K$) are preempted and forced to wait. The CPU resumes serving these jobs only after the high-priority busy period is completed. However, if a high-priority class 1 job finds a class 2 job in service at the arrival instant, the class 2 job is allowed to finish before the class 1 job goes into service.

Consider a chain r class 1 job arriving at the CPU node from the disk I/O subsystem. Its mean sojourn time $W_{C,r1}(\mathbf{n})$ for this visit to the CPU node is

$$W_{C,r1}(\mathbf{n}) = S_{C,r1} + \sum_{p=1}^{R} \left(L^{r}_{C,p1}(\mathbf{n}) - \rho^{r}_{C,p1}(\mathbf{n}) \right) S_{C,p1} + \sum_{p=1}^{R} \sum_{j=1}^{2} \rho^{r}_{C,pj}(\mathbf{n}) S_{C,pj} \quad (3)$$

where $\mathbf{n}$ = the population vector
$S_{C,pj}$ = mean service time at CPU for chain p class j job
$L^{r}_{C,pj}(\mathbf{n})$ = mean number of chain p class j jobs seen by arriving chain r job at CPU when population vector is $\mathbf{n}$
$\rho^{r}_{C,pj}(\mathbf{n})$ = P (arriving chain r job sees chain p class j job in service at CPU when population vector is $\mathbf{n}$)

The second term in equation (3) is the time to serve all high-priority class 1 jobs already waiting in the queue. The third term is the time to complete a class 1 or class 2 job already in service.

Next consider a chain r class 2 job arriving at the CPU node from the terminal. The mean sojourn time $W_{C,r2}(\mathbf{n})$ for this visit to the CPU node is

$$W_{C,r2}(\mathbf{n}) = S_{C,r2} + \sum_{p=1}^{R} \sum_{j=1}^{2} L^{r}_{C,pj}(\mathbf{n})\, S_{C,pj} + [W_{C,r2}(\mathbf{n}) - S_{C,r2}] \sum_{p=1}^{R} \lambda^{r}_{C,p1}(\mathbf{n}) S_{C,p1} \quad (4)$$

where $\lambda^{r}_{C,p1}(\mathbf{n})$ is the mean arrival rate of chain p class 1 jobs at the CPU node when our tagged chain r class 2 job is waiting at the CPU. This equation is analogous to equation (3) with a third term representing the additional delay caused by the higher priority class 1 arrivals when the tagged class 2 job is waiting for its turn. If we solve equation (4) for $W_{C,r2}(\mathbf{n})$, we have

$$W_{C,r2}(\mathbf{n}) = S_{C,r2} + \left(\frac{\sum_{p=1}^{R} \sum_{j=1}^{2} L^{r}_{C,pj}(\mathbf{n}) S_{C,pj}}{1 - \sum_{p=1}^{R} \lambda^{r}_{C,p1}(\mathbf{n}) S_{C,p1}} \right) \quad (5)$$

Next, consider the arrival of a chain r class k ($k = 3, 4, \ldots, K-1$) job at the CPU (executing in the kernel mode with user-level priority). Such arrivals result only from transitions from class K, and are served immediately (in a processor sharing discipline), because there can be no class 1 or class 2 jobs at the CPU at the instant they arrive. However, if class 1 and class 2 jobs arrive at the CPU subsequently, all the jobs executing in class k ($k = 3, 4, \ldots, K$) are preempted and forced to wait. The CPU resumes serving these jobs only when the high-priority busy period ends. Let $L^{r}_{C,pk}(\mathbf{n})$ be the mean number of chain p class k ($k = 3, 4, \ldots, K$) jobs at the CPU node, at chain r job arrival instants when the population vector is $\mathbf{n}$. The mean sojourn time $W_{C,rk}(\mathbf{n})$ at the CPU for class k jobs ($k = 3, 4, \ldots, K-1$) holding user-level priorities but executing in the kernel is

$$W_{C,rk}(\mathbf{n}) = S_{C,rk} \left(1 + \sum_{p=1}^{R} \sum_{j=3}^{K-1} L^{r}_{C,pj}(\mathbf{n}) + \sum_{p=1}^{R} L^{r}_{C,pK}(\mathbf{n}) \right) + W_{C,rk}(\mathbf{n}) \sum_{p=1}^{R} \sum_{j=1}^{2} \lambda^{r}_{C,pj}(\mathbf{n}) S_{C,pj} \quad (6)$$

The first term on the right is the mean sojourn time of a chain r class k ($k = 3, 4, \ldots, K-1$) job when there are no interruptions due to class 1 and class 2

arrivals. The second term is the mean wait while newly arrived class 1 and class 2 jobs claim their preemptive priority and are served, and $\lambda^r_{C,pj}(\mathbf{n})$ is the mean arrival rate of chain p class j jobs at the CPU when our tagged chain r job is resident at the CPU node. Solving equation (6) for $W_{C,rk}(\mathbf{n})$, we have

$$W_{C,rk}(\mathbf{n}) = \frac{S_{C,rk}\left(1 + \sum_{p=1}^{R}\sum_{j=3}^{K-1} L^r_{C,pj}(\mathbf{n}) + \sum_{p=1}^{R} L^r_{C,pK}(\mathbf{n})\right)}{\left(1 - \sum_{p=1}^{R}\sum_{j=1}^{2} \lambda^r_{C,pj}(\mathbf{n}S_{C,pj}\right)} \quad (7)$$

Finally, we consider the arrival of a chain r class K job at the CPU executing in the user mode with a user-level priority. Such arrivals result only from transitions from class k ($k = 1, 2, \ldots, K-1$). Class K jobs resulting from transitions from class 1 and class 2 may also see a backlog of class 1 and class 2 jobs at arrival instants. Class K jobs are also served (in a processor-sharing discipline) only when no high-priority class 1 or class 2 jobs require service. The mean sojourn time $W_{C,rK}(\mathbf{n})$ is

$$W_{C,rK}(\mathbf{n}) = \sum_{p=1}^{R}\sum_{j=1}^{2} L^r_{C,pj}(\mathbf{n})\, S_{C,pj} + S_{C,rK}\left(1 + \sum_{p=1}^{R}\sum_{j=3}^{K-1} L^r_{C,pj}(\mathbf{n}) + \sum_{p=1}^{R} L^r_{C,pK}(\mathbf{n})\right) + W_{C,rK}(\mathbf{n}) \sum_{p=1}^{R}\sum_{j=1}^{2} \lambda^r_{C,pj}(\mathbf{n})\, S_{C,pj} \quad (8)$$

The first term on the right represents the mean time a class K job has to wait before the CPU serves the backlog of class 1 and class 2 jobs. The second term represents the mean sojourn time when there are no interruptions due to class 1 and class 2 arrivals. The last term is the mean wait while the newly arrived class 1 and class 2 jobs claim their preemptive priority and are served. Solving equation (8) for the mean sojourn time $W_{C,rK}(\mathbf{n})$, we have

$$W_{C,rK}(\mathbf{n}) = \frac{\sum_{p=1}^{R}\sum_{j=1}^{2} L^r_{C,pj}(\mathbf{n})\, S_{C,pj}}{\left(1 - \sum_{p=1}^{R}\sum_{j=1}^{2}\lambda^r_{C,pj}(\mathbf{n})S_{C,pj}\right)} + \frac{S_{C,rK}\left(1 + \sum_{p=1}^{R}\sum_{j=3}^{K-1} L^r_{C,pj}(\mathbf{n}) + \sum_{p=1}^{R} L^r_{C,pK}(\mathbf{n})\right)}{\left(1 - \sum_{p=1}^{R}\sum_{j=1}^{2} \lambda^r_{C,pj}(\mathbf{n})\, S_{C,pj}\right)} \quad (9)$$

We also make the assumption that the arrival theorem[22] holds so that a chain r job arriving at the CPU node sees the equilibrium queue for states given by $\mathbf{n} - \mathbf{e}_r = n_1, n_2, \ldots, N_r - 1, \ldots, n_R$. We then have

$$\begin{aligned} L^r_{C,rk}(\mathbf{n}) &= L_{C,rk}(\mathbf{n} - \mathbf{e}_r) \\ \rho^r_{C,rk}(\mathbf{n}) &= \rho_{C,rk}(\mathbf{n} - \mathbf{e}_r) \\ &= \lambda_{C,rk}(\mathbf{n} - \mathbf{e}_r)S_{C,rk} \end{aligned} \quad (10)$$

where $L_{C,rk}(\mathbf{n})$ = mean number of chain p class k jobs at the CPU when the population vector is $\mathbf{n}$

$\lambda_{C,pk}(\mathbf{n})$ = mean throughput of chain p class k jobs at the CPU when the population vector is $\mathbf{n}$

This completes the discussion of the CPU node.

The mean waiting time for chain r jobs at the infinite server think node is just its mean service time:

$$W_{T,r}(\mathbf{n}) = S_{T,r} \quad (11)$$

Finally, we consider the disk nodes. At each disk node m ($m = 1, 2, \ldots, M$), there are two types of jobs, namely read and write requests. Since reads from the disk are synchronous, the job that initiates the read must visit the disk node for each read to the disk. However, because our model assumes that disk writes are

asynchronous, a job that initiates a write does not wait for the block to be written to the disk. Reads to disk m by chain r jobs have a mean throughput given by

$$\lambda_{R_m,r}(\mathbf{n}) = \lambda_r(\mathbf{n}) V_{R,r} \frac{1}{(1 - P_{R,r})} (1 - P_{R_c,r}) P_{D_m,r}$$

where $\lambda_r(\mathbf{n})$ = mean throughput of chain r when the population vector is $\mathbf{n}$
$V_{R,r}$ = mean number of times a chain r job issues a block read system call between successive visits to the think node
$1/(1 - P_{R,r})$ = mean number of block reads per chain r read system call
$1 - P_{R_c,r}$ = probability of a chain r block being read from the disk subsystem
$P_{D_m,r}$ = probability of a chain r block being located on disk m

Similarly, $\lambda_{W_m,r}$, the throughput of writes to disk m by chain r jobs, can be estimated. The mean sojourn time for a chain r job visiting disk node m ($m = 1, 2, \ldots, M$) to read a block is

$$W_{D_m,r}(\mathbf{n}) = S_{D_m}\left(1 + \sum_{p=1}^{R} \left[L_{R_m,p}(\mathbf{n} - \mathbf{e}_r) + L_{W_m,p}(\mathbf{n} - \mathbf{e}_r\right]\right) \quad (12)$$

where $L_{R_m,p}(\mathbf{n})$ = mean number of chain p read requests at disk m when the population vector is $\mathbf{n}$
$L_{W_m,p}(\mathbf{n})$ = mean number of chain p write requests at disk m when the population vector is $\mathbf{n}$

The above set of equations [(3) to (12)] can be solved if $\lambda^r_{C,pk}(\mathbf{n})$ is known. The literature suggests several possibilities. Bryant, Krzesinski, and Teunissen[17] assumed $\lambda^r_{C,pk}(\mathbf{n}) = \lambda_{C,pk}(\mathbf{n})$. Later, Chandy and Laksmi[18,19] proposed using $\lambda^r_{C,pk}(\mathbf{n}) = \lambda_{C,pk}(\mathbf{n} - L_{C,pk}(\mathbf{n}))$, where $L_{C,pk}(\mathbf{n})$ is the mean number of chain p class k jobs at the CPU.

In this analysis we adopt a different approach. In a product-form network, the mean arrival rate of chain l jobs to center i, while a tagged chain r job resides at center i, is[23]

$$\lambda^r_{i,l}(\mathbf{n}) = \frac{\lambda_{i,l}(\mathbf{n} - \mathbf{e}_r)\,[L_i(\mathbf{n} - \mathbf{e}_r - \mathbf{e}_l) + 1]}{L_i(\mathbf{n} - \mathbf{e}_r) + 1}$$

where $\lambda_{i,l}(\mathbf{n})$ = mean throughput of chain l at center i when the population vector is $\mathbf{n}$
$L_i(\mathbf{n})$ = mean queue length vector at center i when the population vector is $\mathbf{n}$

We assume that the same holds true for the model developed here. Thus the quantity $\lambda^r_{i,lq}(\mathbf{n})$ in our model can be approximated by

$$\lambda^r_{i,lq}(\mathbf{n}) = \frac{\lambda_{i,lq}(\mathbf{n} - \mathbf{e}_r)\left[1 + \sum_{p=1}^{R}\sum_{k=1}^{K} L_{i,pk}(\mathbf{n} - \mathbf{e}_r - \mathbf{e}_l)\right]}{\left[1 + \sum_{p=1}^{R}\sum_{k=1}^{K} L_{i,pk}(\mathbf{n} - \mathbf{e}_{r3})\right]} \quad (13)$$

Applying Little's law,[1,2,4] we have

$$\lambda_r(\mathbf{n}) = \frac{n_r}{\sum_{i=1}^{M+2}\sum_{k=1}^{K} V_{i,rk} W_{i,rk}(\mathbf{n})} \quad (14)$$

and

$$L_{i,rk}(\mathbf{n}) = V_{i,rk}\lambda_r(\mathbf{n}) W_{i,rk}(\mathbf{n}) \quad (15)$$

where $\lambda_r(\mathbf{n})$ = mean throughput of chain r when the population vector is $\mathbf{n}$
$V_{i,rk}$ = mean number of times a chain r class k job visits node i between successive visits to think node

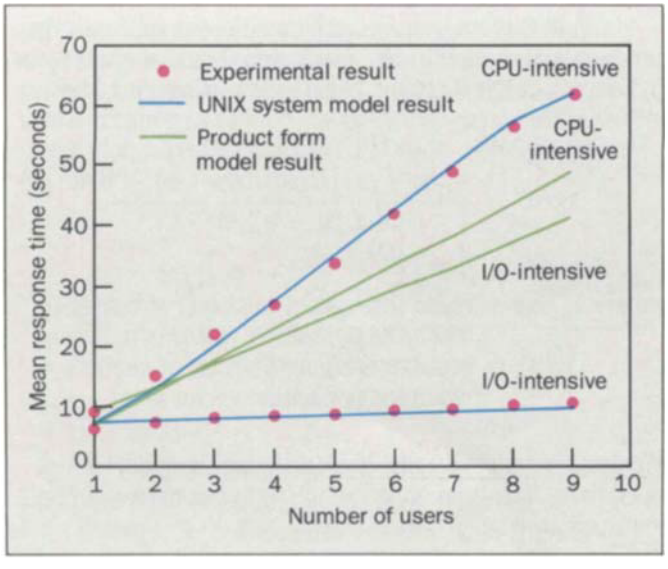

Figure 2. Mean response time for CPU-intensive and disk I/O-intensive jobs.

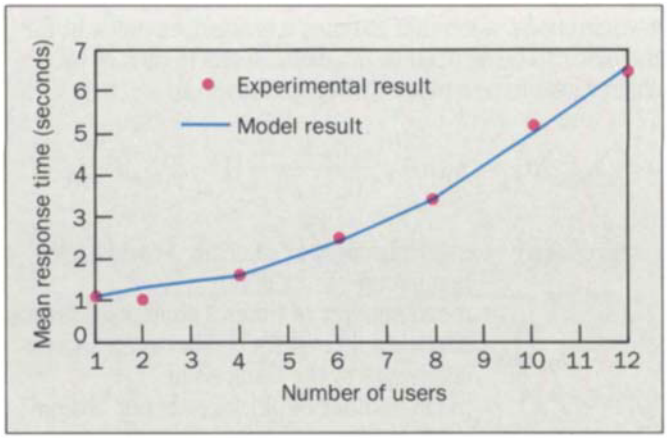

Figure 3. Mean response time as a function of number of users for a software development load on a VAX 11/750 computer.

We can determine these visit ratios by solving a set of linear equations. We can now solve the model recursively with equations (3), (5), (6), and (9) to (15). For large numbers of chains (applications) and classes, and large population, we can use efficient iterative methods such as the Linearizer.[24] The numerical results in the next section were obtained by the Linearizer method.

Results

The model predictions were validated by using several benchmarks with a remote terminal emulator on Digital Equipment Corporation VAX™ 11/750 and VAX 11/785 computers and an AT&T 3B2/300 computer, all running under UNIX System V.2. This section presents some of the validation results.

The first workload considered here consists of two types of jobs: one disk I/O-intensive and the other computation-intensive. The I/O-intensive job spends 3.5 seconds in the kernel mode and 0.03 seconds in the user mode and reads 1258 logical blocks with a cache hit ratio of 55 percent. The computation-intensive job spends 5.2 seconds in the user mode and 0.144 seconds in the kernel mode.

Figure 2 compares the mean response times predicted by the model with those obtained by running the corresponding benchmarks on a 3B2/300 computer. The results are also compared with the mean response time predicted by a simple central server model having a product-form solution. The number of I/O-intensive jobs is set at one and the number of computation-intensive jobs is varied from 1 to 10. The think time is 3 seconds for both types of jobs. The results clearly show that the predictions based on a product-form network have poor accuracy, while the predictions made by our model are well under 10 percent of the experimental results.

Figure 3 compares the mean response time predicted by the model with experimental results for a benchmark resembling a UNIX software development workload running on a VAX 11/750 computer. The number of users is varied from 1 to 12, with a think time of 3 seconds. The results show that the mean response time predicted by our model is within 7 percent of the measured results.

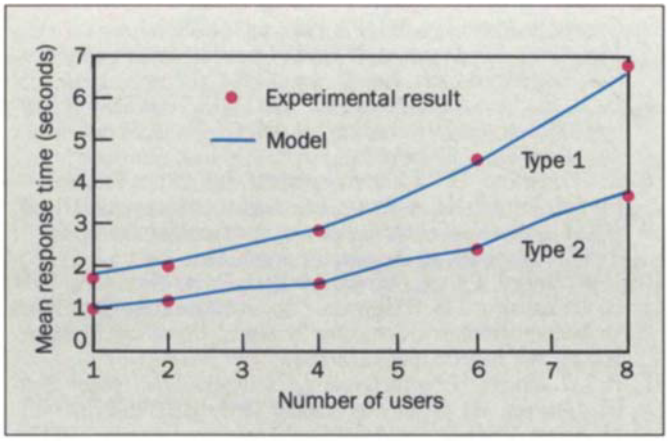

Figure 4. Mean response time as a function of number of users for an application with two types of commands.

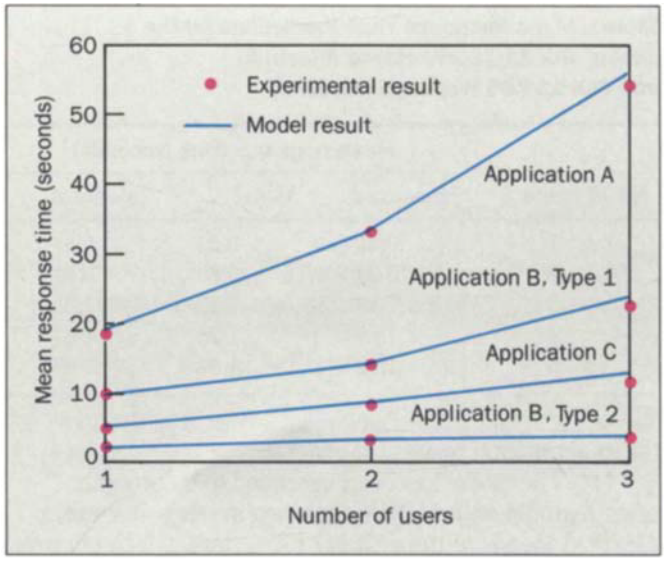

Figure 5. Mean response time as a function of number of users in each application when three different applications are running concurrently.

To illustrate how one may use the model to predict performance when the workload is moved from one type of hardware system to another, we conducted the following experiment. Starting with the workload measured on a VAX 11/750 computer, we reduced the CPU service time by a factor of 2.4 in the user mode and by a factor of 2 in the kernel mode to predict the response time of the same workload on a VAX 11/785 computer. We obtained these scaling factors from typical benchmarks for the VAX 11/750 and VAX 11/785 machines.

The model was then executed with these scaled workload parameters. The results are shown in Table I under Model A. The model was also executed with the workload parameters from the measurement on the VAX 11/785. The results are shown in Table I under Model B. Clearly, the scaling approach yields reasonably high accuracy; the maximum error is about 12 percent and decreases as the number of users is increased.

Figure 4 shows the mean response time for a benchmark with two types of commands. Type 1 commands were mostly CPU-intensive, while type 2 commands perform a moderate amount of disk I/O operations. The figure compares the model results with those obtained by experiment. The predicted mean response times are within 10 percent of the measured values.

Finally, Figure 5 shows the mean response time as a function of number of users when three different applications are running concurrently. Commands in applications A and C are assigned to a single type, and commands in application B are grouped into two types. This example is a case of multiple workloads with multiple types of commands running concurrently. The analytic results are again in excellent agreement with the experimental results.

Summary

In this paper, an analytical model for a single-processor interactive computer system running under the UNIX operating system has been developed. The model has been validated against several benchmarks on three different computer systems running under UNIX System V.2. Validation results show that the mean response time predicted by our analytical model is within 10 percent of

Table I. Mean Response Time Predictions for the Scaled VAX 11/750 Workload (Model A) and VAX 11/785 Workload (Model B)

	Mean response time (seconds)		
No. of users	Measured	Model A	Model B
1	0.61	0.54	0.64
2	0.90	0.80	0.94
3	1.28	1.22	1.23
4	1.69	1.58	1.67
5	1.98	1.96	2.05

the experimental results, in general.

The model has been extended to incorporate other features of the UNIX operating system—for example, read aheads to the disk I/O subsystem, which improve concurrency between the CPU and the disks and, consequently, improve system performance as well.[25] The approach can also be extended with little effort to tightly coupled multiprocessor systems and multiple UNIX systems running with network file systems over local-area networks.

Acknowledgment

The author is specially indebted to Y. T. Wang for his many suggestions, assistance, and discussions.

References

1. S. S. Lavenberg, *Computer Performance Modeling Handbook*, Academic Press, New York, 1983.
2. E. D. Lazowska et al., *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Englewood Cliffs, New Jersey, 1984.
3. A. Perez-Davila and L. W. Dowdy, "Parameter Interdependencies of File Placement Models in a UNIX System," *Proceedings of the 1984 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Cambridge, Massachusetts, August 1984, pp. 15-26.
4. C. H. Sauer and K. M. Chandy, *Computer Systems Performance Modeling*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
5. M. Reiser and S. S. Lavenberg, "Mean Value Analysis of Closed Multichain Queueing Networks," *Journal of the Association for Computing Machinery*, Vol. 27, 1980, pp. 313-322.
6. M. J. Bach, *The Design of the UNIX Operating System*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.
7. D. M. Ritche and K. Thompson, "The UNIX Time Sharing System," *Bell System Technical Journal*, Vol. 57, No. 6, Part 2, July-August 1978, pp. 1905-1929.
8. K. Thompson, "UNIX Implementation," *Bell System Technical Journal*, Vol. 57, No. 6, Part 2, July-August 1978, pp. 1931-1946.
9. *UNIX System V—Release 2.0 Tuning and Configuration Guide*, AT&T, Basking Ridge, New Jersey, April 1984.
10. N. K. Jaiswal, *Priority Queues*, Academic Press, New York, 1968.
11. B. Avi-Itzhak and D. P. Heyman, "Approximate Queueing Models for Multiprogramming Computer Systems," *Operations Research*, Vol. 21, No. 6, 1973, pp. 1212-1230.
12. R. J. T. Morris, "Priority Queueing Networks," *Bell System Technical Journal*, Vol. 60, No. 8, October 1981, pp. 1745-1769.
13. M. Reiser, "Interactive Modeling of Computer Systems," *IBM Systems Journal*, Vol. 15, No. 4, 1976, pp. 309-327.
14. K. C. Sevcik, "Priority Scheduling Discipline in Queueing Network Models for Computer Systems," *Information Processing Congress 77*, B. Gilchrist, editor, IFIP, North Holland Publishing, New York, 1977, pp. 565-570.
15. J. S. Kaufman, "Approximate Methods for Networks of Queues with Priorities," *Performance Evaluation*, Vol. 4, 1984, pp. 183-198.
16. W. Schmitt, "On Decomposition of Markovian Priority Queues and Their Application to the Analysis of Closed Priority Queueing Networks," *Performance 1984*, E. Gelenbe, editor, North Holland Publishing, New York, 1984.
17. R. M. Bryant, A. E. Krzesinski, and P. Teunissen, "The MVA Preempt Resume Priority Approximation," *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Minneapolis, August 1983, pp. 12-27.
18. K. Chandy and M. S. Lakshmi, "An Approximation Technique for Queueing Networks with Premptive Priority Queues," technical report, Department of Computer Sciences, University of Texas at Austin, February 1983.
19. R. M. Bryant et al., "The MVA Priority Approximation," *ACM Transactions on Computer Systems*, Vol. 2, No. 4, November 1984, pp. 335-359.
20. J. van Doremalen and R. Wijbrands, "Approximate Analysis of Priority Queueing Networks," *Teletraffic Analysis and Computer Performance Evaluation*, O. J. Boxma and J. E. Cohen, editors, North Holland, New York, 1986, pp. 117-131.
21. A. Cobham, "A Priority Assignment in Waiting Line Problems," *Operations Research*, Vol. 2, February 1954, pp. 70-76.
22. K. C. Sevcik and I. Mitrani, "The Distribution of Queueing Network States at Input and Output Instants," *Journal of the Association for Computing Machinery*, Vol. 28, No. 2, April 1981, pp. 358-371.

23. J. Zahorjan, "The Distribution of Network States During Residence Times in Product Form Queueing Networks," *Performance Evaluation*, Vol. 4, 1984, pp. 99-104.
24. K. M. Chandy and D. N. Neuse, "Linearizer: A Heuristic Algorithm for Queueing Network Models of Computer Systems," *Communications of the Association for Computing Machinery*, Vol. 25, No. 2, February 1982, pp. 126-134.
25. G. Ramamurthy, "An Analytical Model for Read Aheads to the Disk I/O Subsystem," *Proceedings of Computer Measurement Group 1986, International Conference on Management and Performance Evaluation of Computer Systems*, Las Vegas, December 1986, pp. 196-205.

(Manuscript received August 18, 1988)