Explored by simulation is the performance of a probabilistic model of a multiprogrammed single-processor computing system operating under demand paging.

Results of experiments on statistical methods for improving the efficiency of the simulation are presented.

Estimates of the response variables in the simulation are reported for a variety of conditions of system overhead, queuing delays, and transient response. Sensitivity of these factors to the assumptions of the model are discussed.

Simulation of a model of paging system performance

by G. S. Shedler and S. C. Yang

Many problems that arise in operating systems lead rather naturally to the study of queue networks. The models formulated, however, are frequently of greater complexity than can be handled by existing mathematical techniques. Queue network simulation by straightforward sampling is typically adopted, and often requires long simulation studies to adequately represent the behavior of queuing systems. It is advantageous, therefore, to investigate alternatives to straightforward sampling. Gaver[1] has proposed several such alternatives. Their value is suggested by data derived from a simple (though mathematically difficult) queuing situation.

In this paper, we apply Gaver's methods to simulating the more complex queuing model of a paging machine previously described by Lewis and Shedler.[2] Our approach compares three methods of simulation with straightforward sampling and then estimates the sensitivity of the system response variables to the assumptions of the model. In our discussion, we attempt to go beyond a description of the experiment by illustrating a methodology for the simulation of probabilistic models.

The computer system in our model is a multiprogrammed, single processor operating under demand paging. Overhead functions of the Central Processing Unit (CPU) such as construction and execution of channel control programs, execution of replacement algorithms, and queue management are represented explicitly. Similarly,

other system notions such as a channel-complete interruption and a postponable interruption are also represented explicitly. Although expressions for the long-run channel idle time, CPU idle time, and CPU overhead time have been obtained by Lewis and Shedler,[2] the transient or short-term response of the system is not easily characterized mathematically. Further, it is difficult to assess mathematically the sensitivity of the analytic results to distributional assumptions. Accordingly, a simulation of the model addressed to these matters is desirable. Such a simulation study as discussed in this paper, also provides a quantitative assessment, not obtained in Reference 2, of the queuing delays in the system.

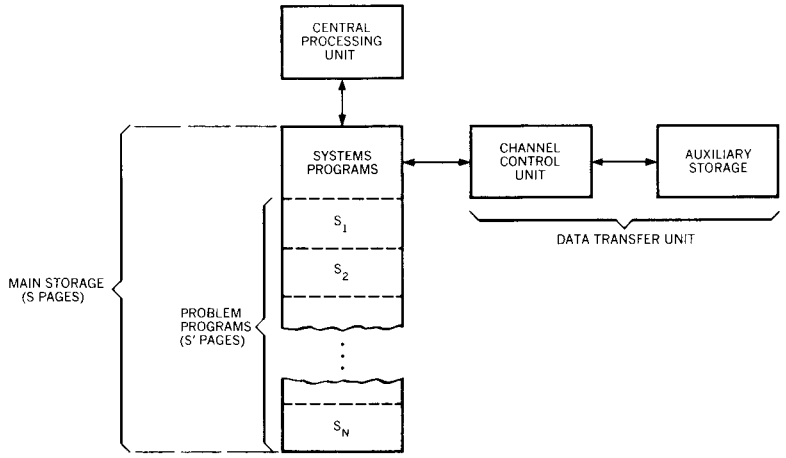
Structure of the model

The computer system with which this paper is concerned is a single processor system with two-level storage, multiprogrammed at a fixed level (i.e., with a constant number of problem programs), and operated in a demand paging environment. Such systems are described in References 3 and 4. The following brief discussion gives the background necessary for an understanding of the model evaluated in our discussion.

In a paging system, all information that is explicitly addressable by the Central Processing Unit (CPU) is divided into units of equal size called *pages*. Main storage (or the "execution store") is similarly divided into page-size sections called *page frames*. In such machines, it is possible to execute a program by using only a few page frames of main storage. When the page containing the first executable instruction has been loaded into a page frame, execution begins and continues until information is required that cannot be found in main storage. The system fetches the page containing the missing information, and may overwrite a page frame in main storage. Thus, under this procedure, called *demand paging*, information is brought into main storage only as a result of an attempt to use information not currently stored there. An instance of demand for a page that is not in main storage is termed a *page exception*. When executing large programs, or operating in a multiprogramming mode in which main storage is shared among several programs, main storage is frequently filled when another page has to be fetched from auxiliary storage. Consequently, a choice must be made as to which page frame in main storage is to be overwritten. The rule governing this choice is called the *replacement algorithm*. The control program must be capable of saving the content of the page frame chosen to be overwritten before the overwriting is performed.

The essential components of the hardware configuration that we discuss in this paper and show in Figure 1 are the following: main storage containing S page frames, a Channel Control Unit (CCU),

Figure 1 Model system hardware configuration



and an auxiliary storage device. We assume that N (where $N \geq 2$) problem programs $P_1, P_2, \cdots, P_N$ are being run in the system. Thus N is the level of multiprogramming. Part of main storage is used as the residence of system (control) programs. Of the remaining S' page frames of main storage, s_i page frames are allocated to problem program P_i. After counting the control programs, we assume that all the remaining page frames are allotted among the problem page frames, that is,

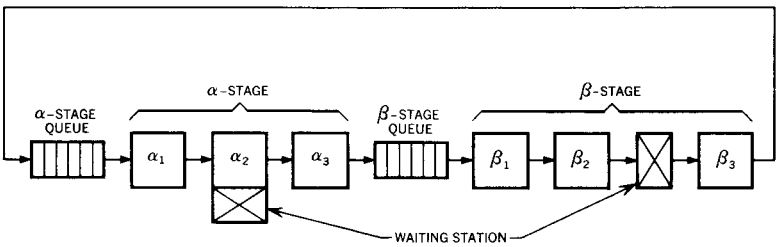
$$\sum_{i=1}^{N} s_i = S'$$

If l_i is the number of pages in problem program P_i, the case of interest to us is the one wherein $1 \leq s_i < l_i$ for $1 \leq i \leq N$.

Under the multiprogramming assumption, there is more than one program resident in main storage, thereby giving rise to contention for processing resources. Hence a conceptual queue is formed for processing services to be provided by the CPU. Whenever a program that is receiving processing service from the CPU references a page that is not in main storage, a request for data transfer service is made by the CPU to a Channel Control Unit (CCU) to move the referenced page from auxiliary storage to main storage so as to be available for processing. Having initiated this request, the CPU is free to service the next available program. A data transfer service consists of activity of a CCU and an input/output device (say a drum or a disk).

Since we have assumed multiprogramming, there can be more than one request waiting for data transfer service. Thus a second conceptual queue is formed for data transfer services to be provided by a Data Transfer Unit (DTU). As soon as a referenced page is moved from auxiliary storage to main storage, the requesting

Figure 2 Schematic of system overhead model



program is again logically available for processing. It is assumed that the CPU can be operated concurrently with the DTU. Thus, in the multiprogramming mode, the CPU can process one program while the DTU is processing a page request for another program.

The foregoing discussion has made no mention of system overhead: CPU processing to switch from one problem program to another to construct and execute appropriate channel programs, and such other activities as queue management and execution of the replacement algorithm. In the model with which this paper is concerned, system overhead functions are represented explicitly. The results of the analysis presented in Reference 2 provide a quantitative assessment of the effects of system overhead in terms of parameters that describe CPU overhead functions and the processing requirements and paging characteristics of the program load.

A schematic representation of the system overhead model is shown in Figure 2. Basically, the model consists of two sequential stages, the α-stage and the β-stage, joined in a loop. The system serves a constant number of N problem programs $P_1, P_2, \cdots, P_N$, where $N \geq 2$. Each program goes through both stages sequentially and returns to the first stage in a continuously repeated process. Within the α-stage, a program receives in order each of three services α_1, α_2, and α_3. Similarly, within the β-stage, a program receives the three ordered services β_1, β_2, and β_3. The assumption that N is constant is an approximation justified by the usual practice of operating such a system in a saturated mode.

Our interpretation of the six services, α_1, α_2, α_3, β_1, β_2, and β_3, in a demand paging system is as follows. Problem program processing corresponds to α_2 service, and data transfer service (paging) corresponds to β_2 service. The remaining services α_1, α_3, β_1, and β_3 are interpreted as system overhead functions. We think of α_1 service as the picking up of the next program for processing from the α-stage queue and restoring the machine state for the program. Associated with α_3 are the following services: saving the machine state of the program that is relinquishing the CPU, executing the replacement algorithm, constructing the channel control program for the required page, and placing an entry onto the paging (β-stage)

queue. The interpretation of β_1 is the picking up of the next page request and starting execution of the channel control program. The function of placing a new entry on the CPU (α-stage) queue is associated with β_3 service and terminating the I/O operation.

Under this interpretation, the major overhead activity is represented in the α_3 service. Note that all six services are provided by two servers, a single CPU and a single DTU; α_1, α_2, α_3, β_1, and β_3 services are provided only by the CPU. A β_2 service can be provided only by the DTU. It is assumed that the CPU and the DTU can provide service simultaneously, subject to the restriction that no β_1 or β_3 service can be rendered by the CPU while the DTU is rendering a β_2 service. Assume further that after receiving α_3 service, a program moves instantaneously from the α_3 service station to the tail of the β-stage queue, and after receiving the β_3 service, the program moves instantaneously from the β_3 service station to the tail of the α-stage queue.

The single CPU renders α_1, α_2, α_3, β_1, and β_3 services to the N programs in the system. Having begun to render an α_1, α_3, β_1, or β_3 service to one program, the CPU completes that service without interruption. Interruption is possible for an α_2 service. An α_2 service is interrupted at the epoch at which a β_2 service is completed by the DTU, and the α_2 service continues at the point of service where the interruption occurred. The β_2-complete interruption of an α_2 service is thus called *pre-emptive resume* interruption.

CPU priority rule

At an epoch of completion of an α_1, α_2, α_3, β_1, or β_3 service and at an epoch of interruption of an α_2 service (i.e., completion of a β_2 service), the CPU chooses the next service according to the following priority rule:

1. If a program is waiting for β_3 service, begin that service.
2. Otherwise, if a program is waiting for β_1 service, begin that service—provided no DTU β_2 is in progress.
3. Otherwise, if the last α service was a completed α_2 service, begin an α_3.
4. If the last α service was an interrupted α_2, resume the α_2.
5. If the last α service was an α_1, begin an α_2.
6. If the last α service was an α_3, and the α-stage queue is not empty, begin an α_1.

If no claim is made on the CPU according to the priority rule, the CPU is assumed to remain idle until the completion of the next β_2 service, at which time the priority rule is invoked. We assume that both the queue in the α-stage and the queue in the β-stage are served under a first-in, first-out (FIFO) queuing discipline.

Since no interruption of an α_1 or α_3 service can occur, a program completing a β_2 service during an α_1 or α_3 service to another program

must wait in the β-stage queue until the completion of that service before receiving its β_3 service. Similarly, a program whose α_2 service is interrupted by a β_2 completion must wait in the α-stage queue until a β service has been rendered before its α_2 service is resumed. In fact, if any program is in the β-stage queue, the α_2 service is not resumed until β_2 service by the DTU begins again. These priority rules are based on the tacit assumption that the bottleneck in a multiprogrammed system is in fetching pages from auxiliary storage, which is generally true in today's systems.

distributional assumptions

The system overhead model has been previously analyzed under the following distributional assumptions, which we use in our simulation studies:

- Service times at the six service stations are independent of one another.
- Successive α_1 and α_3 service times are independent and identically distributed random variables A_1 and A_3 with arbitrary distributions $F_{A_1}(t)$ and $F_{A_3}(t)$, e.g.,

  $$F_{A_1}(t) = \text{prob}\{A_1 \leq t\}$$

- For $i = 1, 2$, and 3, the successive β_i service times are independent and identically distributed random variables B_i with arbitrary distributions $F_{B_i}(t)$.
- The successive α_2 service times are independent and identically distributed random variables A_2 with exponential distribution having rate parameter λ_2, i.e.,

  $$F_{A_2}(t) = \text{prob}\{A_2 \leq t\} = 1 - e^{-\lambda_2 t} \text{ for } t \geq 0$$

The model is analyzed by concentrating on particular epochs at which changes occur in the state of the system as defined in Reference 2. These changes in state are generated by a one-step Markov chain, and the times between changes (given the initial and final states) are independent of the previous history of the process. Such a stochastic process is termed a semi-Markov process.

The exponential distributional assumption for the α_2 service time is essential for the analysis. An assessment of the sensitivity of certain response variables to this assumption is a result of the present simulation study, and is discussed later in this paper.

Some further remarks about the assumptions concerning α_2 service times are in order. We have assumed that each of the programs $P_1, P_2, \cdots, P_N$ is constrained to run in storage that is smaller than its length (i.e., for all i, where $s_i < l_i$). Under the demand paging assumption, a page is moved from the drum to main storage only when needed and when not in main storage. Whenever P_i references a missing page while the portion of main storage allocated to P_i is filled to its capacity, a page in s_i is replaced by the newly referenced

page, in accordance with the replacement algorithm. If the page to be replaced can only be one of the s_i pages in main storage belonging to program P_i, the replacement algorithm is said to operate *locally*. If the replacement algorithm is applied to the entire S' area of the main storage, it is said to operate *globally*. We consider an *execution interval* of a program to be a time interval during which the CPU can continue to process the program without referencing a page that is not in main storage. Thus program P_i, under a replacement algorithm that operates locally in a region of size s_i, gives rise to a sequence of execution intervals independently of the other programs. The length of an execution interval of program P_i is independent of the length of an execution interval of program P_j for $j \neq i$. Under a replacement algorithm that operates locally, however, successive execution intervals of a single program might well not be statistically independent. We assume, nevertheless, that the combined sequence of execution intervals of the set of N programs that comprise the program load is such that successive execution intervals are independent.

Further, for the given program load, under a given memory partition $\pi(s_1, \cdots, s_n)$ and a specified replacement algorithm that operates locally, we assume that successive execution intervals are all exponentially distributed. Although this exponential distribution is essential for the analysis given in Reference 2, limited experimental evidence suggests that it is not an unreasonable assumption. It should be emphasized that the parameter of the exponential distribution is a function of the sizes s_i, which comprise the storage partition π.

The result of the analysis given in Reference 2 is the determination of the long-run expected fraction of time that each of the six services is rendered, and hence the long-run expected fraction of time that each of the two servers is busy. From this, the effectiveness of the multiprogramming can be assessed.

Simulation methods

In multiprogrammed computer systems of the type considered in this paper, certain response variables are of particular interest. Four such response variables are: (1) the fraction of time spent by the CPU doing problem program processing (as opposed to processing associated with overhead functions or being idle), (2) fraction of time the DTU is busy, (3) lengths of queues, and (4) waiting times in queues. These response variables are, in turn, influenced by such other variables as level of multiprogramming (i.e., constant number of problem programs), nature of the program load, characteristics of the physical devices, control strategy, and the like. In general, a response variable W is influenced by a set of system input variables, say $X_1, X_2, X_3, \cdots$, that we denote collectively by the vector $\mathbf{X}$.

The probabilistic model provides a way of relating W to the collection of random variables $\mathbf{X}$, and gives rise to a known but very complicated function.

$$W = f(\mathbf{X}) \tag{1}$$

We seek information about such characteristics of W as the expected value $E[W]$ and the probability distribution of W. Simulation methods have been used for studying the distribution of W in which one observes sample values of that variable. Following Gaver, we now briefly outline the four simulation methods with which the paper is concerned.

straightforward sampling

Straightforward sampling is the basic technique that is used as a standard of comparison for the other methods discussed in this paper. An observation of W is computed from Equation 1 by using a sample value of $\mathbf{X}$. To find a sample value of $\mathbf{X} = (X_1, X_2, \cdots)$, choose a vector of pseudo-random numbers that are uniformly distributed over the interval (0, 1). Then convert these numbers to realizations or samples of $X_1, X_2, \cdots$, perhaps by means of the probability integral transformation

$$\mathbf{X} = F_{\mathbf{X}}^{-1}(R)$$

where $F_{\mathbf{X}}$ is the probability distribution function of $\mathbf{X}$ and R is a random number uniformly distributed on (0, 1). A description and evaluation is given in Reference 5 of the pseudo-random number generator used in the experiments reported in this paper. In *straightforward sampling*, n independent realizations of W, denoted by $W_1, W_2, \cdots, W_n$, are obtained and averaged to give $\bar{W}$, an unbiased estimator of $E[W]$, as follows:

$$\bar{W} = \frac{1}{n} \sum_{j=1}^{n} W_j$$

The variance of the estimator $\bar{W} = 1/n \text{ Var } [W]$. Thus it is clear that the estimate can be brought closer to $E[W]$ by increasing the number of independent realizations n. The investigation of alternatives to straightforward sampling is of interest because, in general, rather long simulation studies are required to adequately represent the behavior of queuing systems.

antithetic variables

One alternative is that of *antithetic variables*, proposed for queuing problems by Page[6] and further discussed in Reference 7. The antithetic idea is to create companion realizations $W_j^{(1)}$ and $W_j^{(2)}$ resulting from antithetic samples $\mathbf{X}_j^{(1)}$ and $\mathbf{X}_j^{(2)}$, which in turn are the result of $\mathbf{R}$ and $1 - \mathbf{R}$. The two antithetic realizations $W_j^{(1)}$ and $W_j^{(2)}$ are averaged to obtain the estimate W_j. The average $\bar{W}_a$ of $W_1, W_2, \cdots, W_n$ is unbiased and is taken as the antithetic estimator of $E[W]$ in the expression

$$\bar{W}_a = \frac{1}{n} \sum_{j=1}^{n} W_j = \frac{1}{n} \sum_{j=1}^{n} \frac{W_j^{(1)} + W_j^{(2)}}{2}$$

stratification

In the method known as *stratification* discussed in Reference 1, m (for $m \geq 2$) companion realizations $W_i^{(1)}, W_i^{(2)}, \cdots, W_i^{(m)}$ are averaged to form a single estimate W_i. To obtain the $W_i^{(k)}$, the unit interval is divided into the following m equal parts:

$$r_1 = \left(0, \frac{1}{m}\right), \cdots, r_k = \left(\frac{k-1}{m}, \frac{k}{m}\right), \cdots, r_m = \left(\frac{m-1}{m}, 1\right)$$

Using one random number that is uniformly distributed over (0, 1), an interval r_i is chosen. Within r_i, a value $R^{(1)}$ is chosen by means of a second random number that is uniformly distributed over the interval $(0, 1/m)$. Obtain the sample $X^{(1)}$ associated with the first of the m companion realizations from $R^{(1)}$ by using the appropriate probability integral transformation. For the second of the companion realizations, i.e., the sample $X^{(2)}$, use $R^{(2)}$, which is obtained by adding $1/m$ to $R^{(1)}$, and possibly subtracting 1 to place $R^{(2)}$ in the unit interval. Subsequent additions of $1/m$ and possible subtractions of 1 yield $R^{(3)}, \cdots, R^{(m)}$, from which $X^{(3)}, \cdots, X^{(m)}$ are obtained. This stratification procedure is carried out for each variable in **X**. Note that two independent random numbers are used to generate m samples of an input variable, one for each of the m companion realizations. The unbiased stratification estimator $\bar{W}_s$ of $E[W]$ is given by the equation

$$\bar{W}_s = \frac{1}{n}\sum_{i=1}^{n} W_i = \frac{1}{n}\sum_{i=1}^{n} \frac{(W_i^{(1)} + \cdots + W_i^{(m)})}{m}$$

We wish to point out that stratification is an extension of the antithetic idea in that it has the tendency to provide an equal distribution of X across companion realizations. If the values $W_i^{(1)}, W_i^{(2)}, \cdots, W_i^{(m)}$ are sufficiently negatively correlated, the final estimate obtained from Equation 2 tends to have a variance smaller than that obtained from mn independent realizations. The experiments reported in this paper use the value $m = 3$.

antithetic variables and stratification

The fourth method studied is a combination of antithetic variables and stratification. By treating the second random number as uniformly distributed over $(0, 1/m)$ antithetically inside the interval $(0, 1/m)$, $2m$ companion realizations can be generated, and the final estimate $\bar{W}_{s,a}$ is obtained by averaging.

Results of our experiments on the four estimating procedures are given in Tables 1 and 2. We consider two instances of a response variable W: CPU utilization, and DTU utilization. In both cases, the system input variable X is the exponentially distributed α_2 service time. For positive integral c, CPU utilization $T_{\text{CPU}}(c)$ is defined as

$$T_{\text{CPU}}(c) = \frac{A(t_c)}{t_c} \times 100$$

Here $A(t_c)$ is the total amount of time that the CPU renders service during the time interval $(0, t_c)$, and t_c is the epoch of simulated

Table 1 Straightforward and antithetic-variable estimates of CPU and DTU utilization

			Straightforward		*Antithetic*				
	c	λ_2	*Mean*	*Variance*	*Mean*	*Variance*	*Range of correlation coefficient*		*Average correlation coefficient*
CPU	100	2.00	71.41	0.1935	71.56	0.0572	−0.8528	−0.4966	−0.7042
		1.50	81.87	0.2514	81.87	0.0266	−0.9363	−0.7365	−0.8263
		1.00	92.79	0.0810	92.78	0.0233	−0.8439	−9.5603	−0.7091
	200	2.00	71.00	0.0886	71.06	0.0353	−0.8672	−0.3930	−0.7119
		1.50	81.56	0.1226	81.61	0.0211	−0.9254	−0.6697	−0.8199
		1.00	92.64	0.0285	92.71	0.0232	−0.8199	0.5930	−0.7276
	300	2.00	70.87	0.1573	70.91	0.0195	−0.8722	−0.4958	−0.7300
		1.50	81.43	0.0899	81.46	0.0237	−0.9018	−0.7225	−0.8240
		1.00	92.70	0.0318	92.66	0.0139	−0.8629	−0.5993	−0.7561
DTU	100	2.00	91.26	0.0255	91.12	0.0301	−0.5469	−0.0075	−0.2517
		1.50	86.12	0.0783	85.99	0.1618	−0.4943	+0.1410	−0.2063
		1.00	72.44	0.3776	72.26	0.1503	−0.6859	−0.2400	−0.4139
	200	2.00	91.72	0.0132	91.65	0.0175	−0.6913	+0.0175	−0.2093
		1.50	86.64	0.0686	86.60	0.0701	−0.6002	+0.1344	−0.2228
		1.00	72.82	0.0713	72.80	0.1076	−0.6824	−0.0700	−0.3556
	300	2.00	91.96	0.0114	91.86	0.0104	−0.4222	+0.1960	−0.2012
		1.50	86.85	0.0525	86.81	0.0394	−0.5470	−0.0435	−0.2480
		1.00	72.89	0.0745	73.01	0.0764	−0.5648	−0.2016	−0.3991

time at which the c + 1st customer begins his α_1 service. Similarly, the DTU utilization $T_{\text{DTU}}(c)$ is defined as

$$T_{\text{DTU}}(c) = \frac{B(t_c)}{t_c} \times 100$$

where $B(t_c)$ is the total amount of time that the DTU renders service during the time interval $(0, t_c)$.

The results shown in Tables 1 and 2 are for the case in which the level of multiprogramming is 3. The α_1, α_2, and α_3 service times are exponentially distributed, and the β-stage service times are constant. Unit time is taken to be the duration of a β_2 service. The average duration of the dominant overhead service α_3 is 0.2, and the duration of an average α_1, β_1, or β_3 service is 0.02.

In each realization, all customers are in the CPU queue at time $t = 0$. We observed that for a given customer c, the result of terminating each realization when the next customer $(c + 1)$ is about to start his α_1 service varies only slightly from defining a time interval $(0, t_c)$ by the first realization, and terminating all subsequent realizations at simulated time t_c.

Table 2 Four methods of estimating CPU and DTU utilization

λ_2	Method	CPU *utilization*		DTU *utilization*	
		Mean	*Variance*	*Mean*	*Variance*
2.00	Straightforward	71.41	0.1935	91.26	0.0255
	Antithetics	71.56	0.0572	91.12	0.0301
	Stratification	71.57	0.0830	91.00	0.0267
	Stratification & antithetics	71.55	0.0135	91.09	0.0193
1.50	Straightforward	81.87	0.2514	86.12	0.0783
	Antithetics	81.87	0.0266	85.99	0.1618
	Stratification	81.91	0.0802	85.91	0.0911
	Stratification & antithetics	81.93	0.0097	86.07	0.0206
1.00	Straightforward	92.79	0.0810	72.44	0.3776
	Antithetics	92.78	0.0233	72.26	0.1503
	Stratification	92.79	0.0280	72.38	0.1511
	Stratification & antithetics	92.83	0.0239	72.14	0.0509

comparative results

In Table 1, we display results of a comparative assessment of straightforward sampling and the antithetic estimating procedure. Here the variances are estimates of Var $\{\overline{W}_a\}$, each obtained from a set of $k = 20$ independent observations of $\overline{W}_a$, and the means are obtained from k observations. Each of the m observations of $\overline{W}_a$ is based on $n = 30$ pairs of companion realizations. Also displayed is the mean correlation coefficient and its range for each of the m observations of $\overline{W}_a$. For straightforward sampling, the means and variances displayed were obtained from $k = 20$ independent observations of $\overline{W}$, each based on $n = 60$ realizations. Thus the total number of realizations used to obtain the estimates is the same for both the straightforward and antithetic-variable methods.

We conclude from the results in Table 1 that, for CPU utilization, the antithetic estimating procedure is useful in variance reduction. Note, however, the lack of evidence that a similar gain is obtainable for DTU utilization. This observation may well have to be considered in future investigations of policies for determining whether the antithetic procedure should be employed in the simulation of different queuing problems. Table 2 shows some results for the straightforward method compared with the antithetic and stratification procedures, and the combination of stratification with antithetics. For stratification (with $m = 3$), the given means and variances are obtained from $k = 20$ independent observations of $\overline{W}_s$, each observation being based on $n = 20$ sets of companion realizations. For the combination of stratification and antithetics, the means and variances given are obtained from $k = 20$ independent

observations of $\bar{W}_{s,a}$, each observation being based on $n = 10$ sets of companion realizations.

Our results in Tables 1 and 2 provide no evidence that stratification is preferable to antithetics. There is evidence, however, that for CPU utilization an additional gain over antithetics is obtainable by the use of the combination of stratification and antithetics.

Simulation results

In this section, we define several quantities of interest in the model, and present results of further simulation studies. In accordance with our findings in the previous section, all simulation results are obtained by the method of antithetic variates.

In the model, the single CPU processes problem programs (α_2 service) and performs overhead functions (α_1, α_3, β_1, β_3 services). The DTU provides paging service (β_2 service). Quantities of primary interest in the model are the percentages of time that the CPU spends rendering each of the several services that it provides (as opposed to being idle), and the percentage of time that the DTU renders paging service (as opposed to being idle).

Thus, for example, we define the random variable $T_{\alpha_1}(c)$ as the percentage of the total time that the CPU performs α_1 service, so that

$$T_{\alpha_1}(c) = \frac{A_{\alpha_1}(t_c)}{t_c} \times 100$$

Here c, referring to the cth customer, is a positive integer. The percentage is obtained by multiplying by 100 the quotient of the total amount of time spent by the CPU performing α_1 service $[A_{\alpha_1}(t_c)]$ during the time interval and the total time t_c.

Definitions of the other random variables representing the times that the CPU performs α_2, α_3, β_1, and β_3 services follow correspondingly:

$$T_{\alpha_2}(c) = \frac{A_{\alpha_2}(t_c)}{t_c} \times 100$$

$$T_{\alpha_3}(c) = \frac{A_{\alpha_3}(t_c)}{t_c} \times 100$$

$$T_{\beta_1}(c) = \frac{A_{\beta_1}(t_c)}{t_c} \times 100$$

$$T_{\beta_3}(c) = \frac{A_{\beta_3}(t_c)}{t_c} \times 100$$

Table 3 shows the results of the simulation by giving estimates of expected values of the quantities just defined as a function of the

Table 3 Estimates of expected values

λ_2	c	*Overhead* $T_{\alpha_1}(c)$	$T_{\alpha_3}(c)$	$T_{\beta_1}(c)$	$T_{\beta_3}(c)$	*Total*	$T_{\alpha_2}(c)$	CPU *idle* $100-T_{CPU}(c)$	DTU *busy* $T_{DTU}(c)$	DTU *idle* $100-T_{DTU}(c)$
0.50	50	0.952	4.758	0.941	0.927	7.573	92.186	0.241	46.296	53.704
	250	0.936	4.681	0.934	0.930	7.482	92.314	0.204	46.563	53.447
	500	0.934	4.669	0.933	0.931	7.466	92.286	0.248	46.570	53.430
	1000	0.929	4.647	0.929	0.928	7.434	92.312	0.254	46.417	53.583
1.00	50	1.649	9.244	1.609	1.576	13.077	80.386	6.537	79.135	20.865
	250	1.631	8.153	1.623	1.616	13.023	80.481	6.494	80.893	19.107
	500	1.626	8.130	1.622	1.618	12.996	80.367	6.637	80.979	19.021
	1000	1.618	8.091	1.616	1.614	12.940	80.358	6.702	80.760	19.240
2.00	50	1.997	9.985	1.919	1.879	15.779	49.119	35.102	94.185	5.815
	250	1.931	9.656	1.916	1.909	15.412	47.733	36.855	95.475	4.525
	500	1.923	9.617	1.916	1.912	15.369	47.570	37.061	95.621	4.379
	1000	1.918	9.592	1.915	1.913	15.338	47.635	37.027	95.658	4.342

Table 4 Average queue lengths and waiting times

λ_2	c	CPU *queue* $L_{CPU}(c)$	$W_{CPU}(c)$	DTU *queue* $L_{DTU}(c)$	$W_{DTU}(c)$
0.50	50	2.357	4.879	0.168	0.330
	250	2.367	5.037	0.156	0.328
	500	2.366	5.057	0.158	0.336
	1000	2.365	5.085	0.159	0.342
1.00	50	1.383	1.670	0.872	1.044
	250	1.329	1.626	0.904	1.107
	500	1.321	1.623	0.912	1.122
	1000	1.328	1.640	0.909	1.122
2.00	50	0.265	0.265	2.130	2.180
	250	0.191	9.197	2.212	2.301
	500	0.183	0.190	2.221	2.315
	1000	0.185	0.193	2.218	2.315

customer ordinal number c and the rate parameter λ_2 of the exponentially distributed α_2 service times. The level of multiprogramming is 4, and the average duration of an α_3 service is 0.10. The other parameter values are the same as those used in the simulation results reported in the previous section. These parameter values apply to results shown in Tables 3 through 6.

Estimates of expected values obtained by simulating several quantities of interest in connection with queuing delays in the model are given in Table 4. These quantities—average waiting time in the

CPU queue (W_{CPU}) and in the DTU queue (W_{DTU}), and average length of CPU queue (L_{CPU}) and the DTU queue (L_{DTU})—are defined as follows: Let c be a positive integer representing the customer ordinal number. Then

$$W_{CPU}(c) = \frac{1}{c} \sum_{j=1}^{c} [t_f(j) - t_i(j)]$$

and

$$W_{DTU}(c) = \frac{1}{c} \sum_{j=1}^{c} [t'_f(j) - t'_i(j)]$$

where $t_i(j)$ is the epoch at which the jth customer joins the CPU queue, and $t'_i(j)$ is the epoch at which the jth customer joins the DTU queue. Similarly for the CPU queue, $t_f(j)$ is the epoch at which the jth customer begins his α_1, service and $t'_f(j)$ is the epoch at which the jth customer begins his β_1 service.

For $m \geq 1$, let $\{t_m\}$ be the sequence of epochs (such that $t_i < t_i + 1$) at which changes in the length of the CPU queue occur. Similarly, $\{t'_m\}$ is the sequence of epochs (wherein $t'_i < t'_{i+1}$) at which changes in the DTU queue length occur. Finally, let l_k be the length of the CPU queue during the interval (t_i, t_{i+1}), and let l'_k represent the DTU queue length during the interval (t'_i, t'_{i+1}). Using this notation, we define average CPU and DTU queue lengths as the following random variables:

$$L_{CPU}(c) = \frac{\sum_{i=1}^{k} l_i(t_{i+1} - t_i)}{t_{k+1}}$$

and

$$L_{DTU}(c) = \frac{\sum_{i=1}^{k'} l'_i(t'_{i+1} - t'_i)}{t_{k+1}}$$

Here, t_{k+1} is the epoch at which the $c + 1$st customer begins his α_1 service, and t'_{k+1} is the epoch at which the $c + 1$st customer begins his β_1 service.

Table 4 gives estimates of the expected values of the four quantities just defined as a function of c and the rate parameter λ_2 of the exponentially distributed α_2 service time. Again, the other parameter values have been defined earlier in this paper.

An essential assumption of the analysis given in Reference 2 requires that the probability distribution of α_2 service be exponential. An indication of the sensitivity of the response variables to this assumption is provided by the simulation results given in Tables 5 and 6. Here estimates of CPU utilization and DTU utilization are given as functions of c for six different α_2 service distributions. For $k = 2$ and $k = 3$, we consider the Erlang-k distributions, which represent

Table 5 CPU utilization for several α_2 service time distributions

λ_2	c	*Exponential*	*Erlang-2*	*Erlang-3*	*Constant*	*Uniform*	$F_{\alpha_2}(x) = 1 - \frac{1}{(1+\lambda_2 x)^2}$
0.50	50	99.759	99.999	100.000	100.000	99.983	97.747
	250	99.796	99.993	100.000	100.000	99.986	98.018
	500	99.752	99.996	100.000	100.000	99.974	98.008
	1000	99.746	99.995	100.000	100.000	99.977	97.993
1.00	50	93.463	97.440	98.778	100.000	98.208	84.327
	250	93.504	97.416	98.775	100.000	98.300	84.049
	500	93.363	97.342	98.714	100.000	98.202	83.941
	1000	93.298	97.257	98.690	100.000	98.158	84.163
2.00	50	64.898	65.231	65.464	66.303	66.096	58.622
	250	63.145	63.364	63.497	64.007	63.730	57.800
	500	62.939	63.301	63.411	63.728	63.433	57.648
	1000	63.121	63.286	63.285	63.569	63.263	58.018

Table 6 DTU utilization for several α_2 service time distributions

λ_2	c	*Exponential*	*Erlang-2*	*Erlang-3*	*Constant*	*Uniform*	$F_{\alpha_2}(x) = 1 - \frac{1}{(1+\lambda_2 x)^2}$
0.50	50	46.296	46.311	46.124	45.380	45.443	49.804
	250	46.563	46.597	46.490	46.122	46.123	48.003
	500	46.570	46.487	46.410	46.218	46.217	47.906
	1000	46.417	46.405	46.419	46.266	46.265	47.017
1.00	50	79.135	82.685	83.613	84.514	82.259	76.915
	250	80.893	84.202	85.222	85.878	84.242	75.917
	500	80.979	84.111	85.179	86.073	84.451	75.898
	1000	80.760	83.977	85.251	86.170	84.557	74.895
2.00	50	94.185	94.599	94.670	94.721	94.449	90.485
	250	95.475	95.673	95.707	95.880	95.468	90.655
	500	95.621	95.878	95.859	96.038	95.600	90.767
	1000	95.658	95.908	95.941	96.117	95.667	90.259

the intervals between k events in a completely random series. In addition, the constant distribution and uniform distribution, as well as a long-tailed distribution specified by

$$F_{A_2}(x) = 1 - \frac{1}{(1+\lambda_2 x)^2}$$

are considered. Both CPU utilization and DTU utilization are relatively insensitive (i.e., less than ten percent) to all the observed distributions, with the exception of the long-tailed distribution where somewhat greater differences are observed.

Concluding remarks

This paper compares three simulation methods with straightforward sampling to study the efficiency of these methods in estimating the performance of a demand-paging system model. Measures of system overhead, queuing delays, and transient response of the modeled system are related to parameters describing the processing demands of the programs load. Our results for this specific system model suggest that variance reduction from straightforward sampling techniques is obtainable by the method of antithetic variables. However, one should not necessarily expect a similar gain for all response variables in a model. With regard to variance reduction, additional gains may be obtainable by a combination of stratification and antithetics.

CITED REFERENCES

1. D. P. Gaver, "Statistical methods for improving simulation efficiency," *Third Conference of Applications of Simulation* December 8–10, 1969, Los Angeles, California, 38–46 (December 1969).
2. P. A. W. Lewis and G. S. Shedler, "A cyclic queue model of system overhead in multiprogrammed computing systems," *Journal ACM* **18,** No. 2 (April 1971).
3. B. Randell and C. J. Kuehner, "Dynamic storage allocation systems," *Communications of the Association for Computing Machinery* **11,** No. 5, 297–305 (1968).
4. C. J. Kuehner and B. Randell, "Demand paging in perspective," *AFIPS Conference Proceedings Fall Joint Computer Conference,* **33,** 1011–1018 (1968).
5. P. A. W. Lewis, A. S. Goodman, and J. M. Miller, "A pseudo-random number generator for the System/360," *IBM Systems Journal* **8,** No. 2, 136–146 (1969).
6. E. S. Page, "On Monte Carlo methods in congestion problems: II," *Operations Research* **13,** No. 2, 300–305 (March-April).
7. J. Hammersley and D. Handscomb, *Monte Carlo Methods,* Methuen and Co., Ltd., London (1964).