A Study of a Mechanism for Controlling Multiprogrammed Memory in an Interactive System

ALEXANDRE BRANDWAJN AND JEAN-ALAIN HERNANDEZ

Abstract—**This paper deals with the following mechanism for controlling the multiprogramming set in a demand paging system: processes are dynamically divided into several categories according to the number of page faults generated during their residence in main memory. A process is admitted into the multiprogramming set only if there is enough space free in the main memory to contain the number of pages corresponding to the current category of the process. Using a queueing network model of an interactive system with such a control mechanism we study the effectivenesss of the control considered, and, more particularly, its ability to partition the memory space according to the locality of processes.**

Index Terms—**Classes of processes, demand paging, hybrid analytical-simulation solution method, interactive system, multiprogrammed memory, process admission and memory allocation, queueing network models.**

I. Introduction

SINCE the experimental studies of the dynamic program behavior [1], and the introduction of the notion of locality and of the working set model [2], a number of system designers have attempted to include these results in their memory management algorithms (e.g., [3]). An interesting attempt to dynamically partition the memory space according to the observed behavior of processes was implemented in the interactive virtual memory system ESOPE [4].

The algorithm, apparently inspired from the EMAS system [5], is as follows (see Fig. 1): processes are divided into categories, and a process is admitted into the multiprogramming set (i.e., allowed to share real memory and to compete for other system resources) only if there is enough space free in main memory to contain the number of pages corresponding to the current category of the process. This number of pages are reserved for the process, but the actual fetch takes place on a page on-demand basis. The process category is adjusted, basically, in two instances.

1) Upon Command Completion: The category is reduced by one (if possible), if the number of pages fetched (i.e., of page faults generated) is smaller than the corresponding limit for the immediately preceding (i.e., lower) category.

2) Upon Category Transgression: If a process attempts to access more pages than reserved for it according to its current category, the process is ejected from the multiprogramming set. The memory pages that have been allocated to it are freed, and its category is increased by one (if possible). The process is then placed at the end of the admission queue.

A new process entering the system is assigned a category on an arbitrary basis.

The intuitive motivation behind this algorithm seems to be the following. The current category of a process is used as an estimate of its working set size. In order to prevent thrashing [6], processes are admitted into main memory only if the latter can contain their estimated "working sets." Note that this algorithm not only controls the partitioning of the memory space (and thus the multiprogramming level) but also automatically ensures replacement of pages (when a process leaves the multiprogramming set). Note also that it reacts to instantaneous changes in program behavior, and thus introduces a strong coupling between the system execution and control functions. This hinders the system's decomposability [7].

In summary, the algorithm controls the multiprogramming set via dynamic memory partitioning. The latter depends on an on-line process classification based on the virtual time paging behavior of a process. The number and limits of the categories are the parameters of the algorithm. Therefore, it is important to study the system throughput (or, equivalently, the mean response time) as a function of the category assignment, and its dependence on system and program behavior parameters. It is also important to study the effectiveness of the virtual time on-line classification of processes. (The latter point is suggested by the obvious result of recent modeling studies (see, e.g., [8]) which indicates that the number of pages needed by a process "to be executed efficiently" [9] is not an absolute process characteristic but depends strongly on, e.g., the average service time of the secondary memory device). These questions are addressed in this paper.

The problem of multiprogrammed memory management, and, more particularly, that of controlling the level of multiprogramming has received considerable attention in the literature, e.g., [10]-[19] are but a few recent references. The work by Schoute [17] seems the closest to this paper. Our work differs from [17] in that we explicitly take into account the effect of program loading [20], [21] on the performance of the admission mechanism, i.e., on the current category of processes. This allows us to study the influence

Manuscript received June 1, 1979.

A. Brandwajn is with the Amdahl Corporation, Sunnyvale, CA 94086.

J.-A. Hernandez is with the Ecole Nationale Supérieure des Télécommunications, Paris, France.

0098-5589/81/0500-0321$00.75

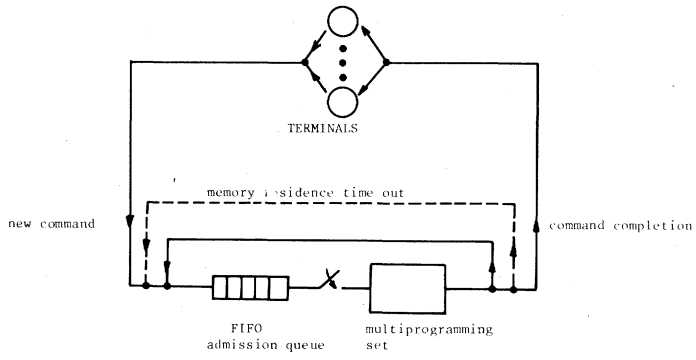


Fig. 1. Framework for algorithm considered.

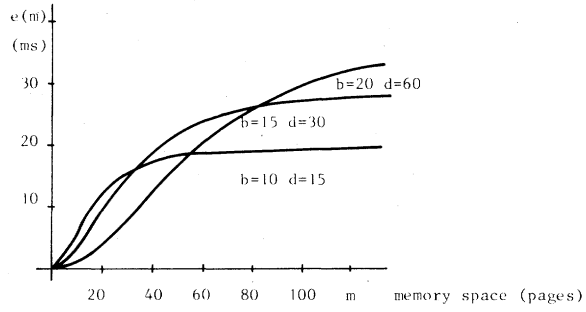


Fig. 2. Examples of lifetime curves.

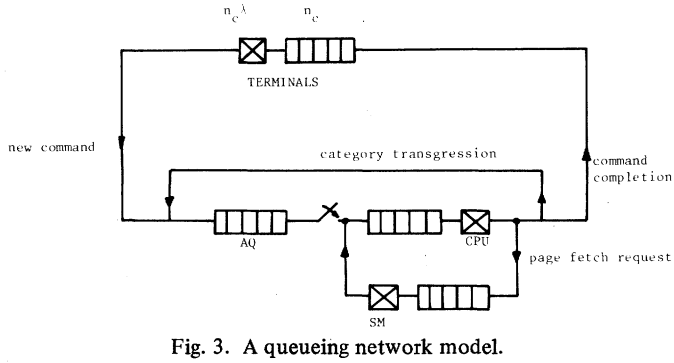


Fig. 3. A queueing network model.

of program behavior on system performance, and, in particular to consider the case where the processes belong to different classes as regards their locality and total compute time.

Let us now define the scope of our paper. We shall restrict our attention to a situation where only interactive processes are present in the system (no background batch jobs) and where the number of logged-in (active) terminals is constant over periods of time long enough for a stationary analysis to be valid. The arbitrary initial category assignment will thus be neglected and only the long-run behavior of the algorithm under consideration will be studied. It may be noted that in the algorithm as implemented in the ESOPE system, processes are ejected from the multiprogramming set if their residence time in real memory exceeds a given limit. We concentrate on the paging behavior of processes and, therefore, neglect this mechanism.

To start with, we shall assume that all the users are statistically identical and independent. Their paging behavior will be modeled by the lifetime function [1] which relates the average process execution time between two successive page faults $e(m)$, to the amount of memory m, allocated to the process. We shall use the two-parameters fit proposed in [9],

$$e(m) = 2b/(1 + (d/m)^2), \tag{1.1}$$

which accounts for the saturation effect at larger m. Fig. 2 shows examples of the life-time function for several sets of parameters b and d. I/O activity other than caused by paging, and most overheads, will be neglected.

The next section is devoted to the description of a queueing model of the system under study. In Section III the numerical results obtained with a single class of processes are discussed. Section IV is devoted to the numerical results with several classes of processes.

II. A Queueing Network Model

The queueing network model we shall use to study the properties of the algorithm under consideration is represented in Fig. 3. The behavior of a user is modeled by a sequence of think times followed by a generation of a command after which the user remains inactive until the system response [22]. (A somewhat more elaborate model of user interaction has been proposed recently in [23].) The user think time is assumed to be an exponentially distributed random variable with mean $1/\lambda$; the set of terminals is thus represented by an exponential server with service rate $n_c\lambda$, n_c being the current number of active terminals. We assume that there are q categories in the system, and we denote by m_j the page number limit for category j, $j = 1, \cdots, q$, where $m_1 < m_2 < \cdots m_{q-1} < m_q$. A command generated by a user (a process) enters a FIFO admission queue (AQ). According to the admission mechanism described, if the process at the front of AQ is of category j, it will be admitted only if there are at least m_j memory pages available.

The multiprogramming set is represented by a CPU and a secondary memory (SM) device with their queues of processes. We denote by $v(m)$ the page fault rate of a process which has m pages present in real memory, and we let

$$v_1(m) = \begin{cases} 1/e(m), & m = 1, \cdots, m_j \\ 1/\omega, & m = 0 \end{cases} \tag{2.1}$$

where $e(m)$ is given by (1.1), and ω is the average system overhead time per process admission into the multiprogramming set. A process having requested a page is either placed in the SM queue if $m < m_j$, or ejected if $m = m_j$. The mean service time of the SM is $1/u_1$, and it is assumed to account for a possible preliminary saving of a written-onto memory page before the actual fetch of the page requested. We assume that the SM service time is an exponentially distributed random variable, and that the SM queueing discipline is FCFS.

In the case of ejection, the memory pages belonging to the process are freed, and the process category is adjusted ($j := j + 1$, if $j < q$). The process is then placed in AQ, and, once admitted, will have to reacquire one by one its pages.

The service discipline at the CPU is assumed to be processor sharing. The CPU service time for a process is assumed to be an exponentially distributed random variable with rate

$$v(m) = \begin{cases} v_1(0), & \text{if } m = 0 \\ v_1(m) + v_0, & m = 1, \cdots, m_j \end{cases} \quad (2.2)$$

for a process of category j with m pages in memory. v_0 denotes the rate of command completions. We let

$$v_0 = 1/c \quad (2.3)$$

where c is the mean total CPU time per command.

If a process of category j has no more than m^*_{j-1} pages in memory at the moment of completion its category is adjusted ($j := j - 1$, if $j > 1$).

At this point, we shall use the mean system response time W as a measure of overall system performance and of the utilization of the CPU system resource (using Little's formula [24], one can easily show that $W = cN/B - 1/\lambda$, where N is the total number of terminals and B denotes user mode CPU utilization).

Note that despite a number of assumptions aimed at the tractability of the model its solution is far from obvious. A direct brute-force attack of the model balance equations would be difficult in practice because a very detailed state description is needed. The latter seems particularly inadequate with respect to the performance measures defined.

We summarize below the four main steps of the approximate hybrid solution method we have used. The interested reader may refer to the Appendix and [32] for details.

Step 1: We analyze the behavior of a user in its virtual (i.e., execution) time so as to obtain, for an active process of category $j, j = 1, \cdots, q$:

- f_j the rate of page faults;
- w_j the rate of category transgressions;
- z_j the rate of command completions;
- s_j the probability that process category will decrease upon command completion.

This is similar to the approach used in [20], [21], and yields a simple analytical solution (see also the Appendix).

Step 2: We use the average page fault rates f_j in a simple model of the multiprogramming set (see Fig. 4) to compute $A_j(\boldsymbol{n})$, the CPU utilizations for processes of each category under a constant load of $\boldsymbol{n} = (n_1, \cdots, n_q)$, where n_j is the number of category j processes in the multiprogramming set. Due to the assumptions on service time distribution and on queueing disciplines, the analytical solution of this model may be easily obtained [25].

Step 3: We use the $A_j(\boldsymbol{n})$, w_j, z_j, and s_j in a queueing model of the multiprogramming set control. The model (see Fig. 5) reflects only events pertaining to the operation of the admission control. Following the "short-circuit" approach [28], [33], it is studied under constant load of l processes in order to obtain $u(l)$, the rate of command completions with a total of l processes (waiting and admitted).

The multiprogramming set is represented by an exponential queue. Its current state is described by $\boldsymbol{n} = (n_1, \cdots, n_q)$, the vector of the numbers of processes of each category currently in the multiprogramming set. The rates of departures from the latter are set to be the products of $A_j(\boldsymbol{n})$ by the corresponding rate obtained in Step 1, e.g., the rate of category

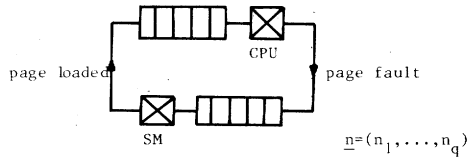


Fig. 4. Multiprogramming set submodel.

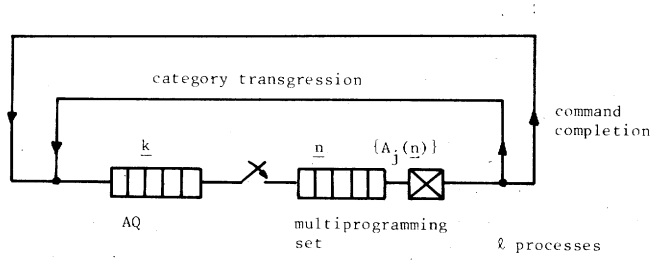


Fig. 5. Multiprogramming control model.

transgressions for processes of category j is $A_j(\boldsymbol{n})\, w_j$. The state of the admission queue is described by $\boldsymbol{k}$, the vector of process categories in FIFO order.

The model can be solved using discrete event simulation [26]. A confidence interval for the result may be derived applying the regenerative simulation method [27], so that we obtain two numbers $u'(l)$ and $u''(l)$ which are the bounds of the confidence interval for the rate of command completions with a total of l processes.

Step 4: We analyze the system behavior at a highly aggregate level via the queueing model of Fig. 6. The terminals and the system are represented by exponential servers with service rates $n_c\lambda$ and $u(l)$, respectively ($n_c + l = N$, the total number of terminals). The analytical solution of this model is well known and allows an easy computation of the average number of processes in the system, and, hence, via Little's formula [24], of the mean system response time. In our case, since $u(l)$ is given in the form of an interval, interval arithmetic would have to be used to finally obtain W' and W'', the two bounds of the confidence interval for the mean system response time. Note that we have used a decomposition technique whereby one computes approximate values for conditional probabilities by analyzing subnetworks under constant (full) load conditions [28], [29]. As a whole, our approach is similar to that discussed in [30], in that at each step in the solution of the decomposed problem, the solution method which seems the most appropriate (analytical, discrete-event simulation, . . .) is used. The next section is devoted to numerical results obtained from our model.

III. Numerical Results with a Single Class of Processes

Since the system under consideration is quite complex, and both the model elaborated and its solution involve a number of simplifications and approximations, it is important to gain some confidence in the accuracy of the results before using them to draw conclusions. Therefore, we have tested our model using results of measurements of the ESOPE system under simulated load [31]. A few modifications had to be introduced in our model to reflect the actual operation of the

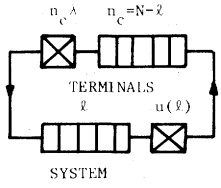


Fig. 6. Aggregate system model.

ESOPE system, and to be able to use measurement results. Mainly, in ESOPE not all page faults result in a page fetch. Some of the pages deallocated at the moment of process ejection, and requested again during subsequent residence periods, may still be present in main memory. Such pages are simply "recovered" by table update. The number of pages actually fetched has been measured. The ratio of the latter number of the total number of pages faults will be denoted by $1 - \beta$, so that β is the relative frequency of pages recoveries. Hence, the actual page fetch rate becomes $f_j = (1 - \beta) f_j$, and this value has to be used in Step 2 to compute the CPU utilization.

We also let

$$v(m) = \begin{cases} \text{"}\infty\text{"}, & m = 0, \\ v_1 + v_0, & m = 1, \cdots, m_j, \end{cases} \tag{4.1}$$

where v_1 is the average page fault rate (i.e., total number of page faults divided by total CPU time), v_0 is the rate of command completions (i.e., total number of interactions divided by total CPU time).

Table I shows the results of model tests. A 90 percent confidence interval will be used throughout this paper.

Clearly, the model may be used with reasonable confidence.

We now first use it to study the influence of system and program behavior parameters on the choice of category limits. To start with, we consider the case where the number of categories is equal to one ($q = 1$). The category mechanism then results in a fixed maximum degree of multiprogramming. For a given main memory size (available for paging), only a few discrete values of category limit m_1 have to be studied, viz., those for which m_1 is maximum with a given resulting degree of multiprogramming. All other values of m_1 may only cause the page fault rate f_1 to increase with no compensation by the multiprogramming effect, since the multiprogramming degree will not increase. Thus, for a memory of $M = 128$ pages, the values of m_1 to consider are 128, 64, 42, pages, etc. Note that with one category the solution Step 3 may be easily carried out analytically.

We have represented in Fig. 7 the average relative system response time i.e., the ratio W/c versus the number of terminals N for a set of model parameters with the mean service time of the secondary memory device, $1/u_1$, set to 20 ms. Fig. 8 shows the results obtained with $1/u_1$ set to 5 ms, the other parameters remaining unchanged. We observe the important effect of the mean service time of the secondary memory device on system performance, and, in particular on optimum (i.e., which minimizes mean system response time) category assignment. The latter changes from 64 pages for the set of parameters used in Fig. 7, to 42 pages in Fig. 8.

(In Figs. 7-14, and 16 and 17, the following values of model parameters are used: M = 128 pages, $1/\omega$ = 0.01 ms.) The effect of program behavior may be seen in Figs. 8 and 9 in which the values of life-time function parameters used are b = 15 ms, d = 30 pages, and b = 20 ms, d = 60 pages, respectively. Again, we observe that the optimum category assignment changes. A similar observation may be made as regards the influence of the average total CPU time per command c in Figs. 8 and 10, c being set to 500 ms and 250 ms, respectively. As a whole, the figures obtained clearly indicate that the optimum limit assignment for one category may be sensi-

TABLE I

values of model parameters:

category limits in number of pages (q=10):

m_1=4; m_2=8; m_3=16; m_4=24; m_5=32; m_6=48;

m_7=64; m_8=80; m_9=112; m_{10}=148;

category reduction:

$m_j^* = m_{j-1} - 1$, j=2, 3,..., 10;

total memory available for paging: M=184 pages;

I N=5 (number of active terminals)		
v_o =1/c=1/464 ms ; v_1 =0.04 ms ; β =0.76;		
$1/u_1$ =37 ms (mean service time per page fetch);		
$1/\lambda$ =10.4 s (mean user think time);		
response time W measured	:	0.70 s
obtained from model	:	(0.77 s , 0.85 s)
II N=5		
v_o =1/458 ms ; v_1 =0.058 ms ; β =0.69; $1/u_1$ =38 ms ; $1/\lambda$ =10.8 s;		
response time W measured	:	0.89 s
obtained from model	:	(0.93 s , 1.04 s)
III N=12		
v_o =1/490 ms ; v_1 =0.038 ms ; β =0.47; $1/u_1$ =41.1 ms; $1/\lambda$ =11.8 s;		
response time W measured	:	1.44 s
obtained from model	:	(1.37 s , 1.56 s)

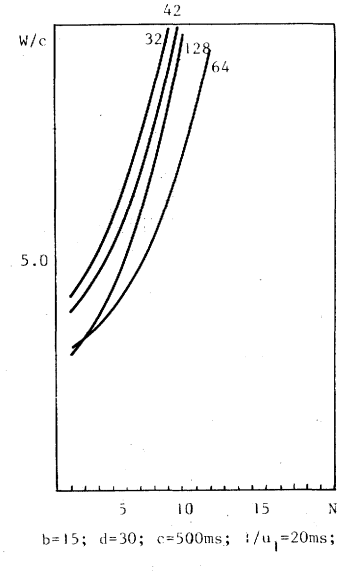


Fig. 7.

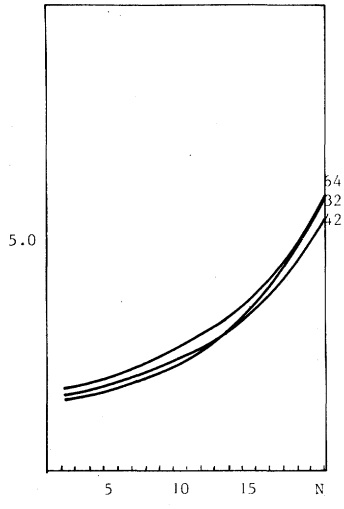


Fig. 8.

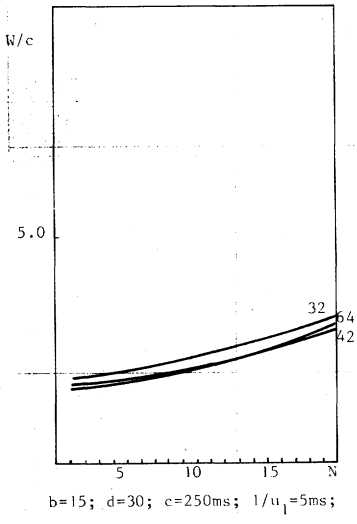


Fig. 10.

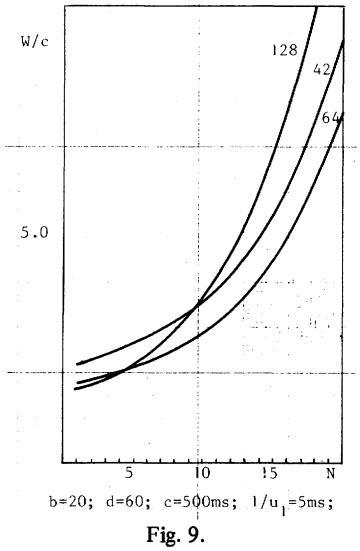


Fig. 9.

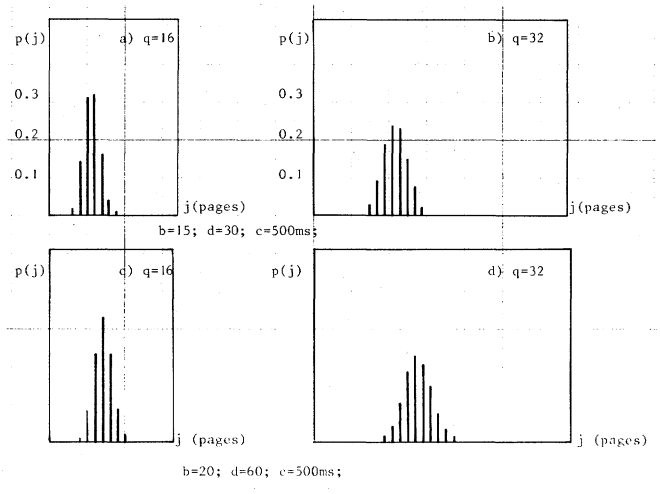


Fig. 11.

tive to both system and program behavior parameters. In particular, an important and undesirable effect to be noted is that the optimum category assignment may depend on the number of logged-on terminals. This may be seen in Fig. 8 where the category limit of 64 pages yields better results than the 42 pages limit up to $N = 12$ terminals. The curves of Fig. 10 exhibit a similar effect.

As a next step in the study of the category mechanism we consider where the number of categories is sufficiently large so that two successive category limits differ only by a small number of pages. For example, for a total memory space of 128 pages and $q = 32$ with equidistant category limits, two adjacent categories differ only by 4 pages. A high number of categories results in excessive computational difficulties in Steps 2 and 3 of our solution procedure. It may be noted, however, that the probability distribution of the current category of an active process [denoted $p(j)$] is highly nonuniform. As a consequence, if we use only the few most probable categories in Steps 2 and 3, the error introduced in the final result should not be important (this reduction of the number of categories has been actually used in the model validation for $N = 12$ terminals). Fig. 11 shows a few examples of the above probability distribution. We have represented there $p(j)$ versus j, the category number, in the case of equidistant categories with $q = 16$ and 32, for two different life-time functions. We observe that $p(j)$ is nonnegligible only for some five to six values of j out of 16 or 32.

In Fig. 12 we have plotted the results obtained (using the five most probable equidistant categories out of 32) for the values of model parameters used in Figs. 7 and 8.

A comparison of the curves indicates that a large number of categories, in the case studied, does not improve the system response time. The effect may even be adverse, as illustrated by the curves for $1/u_1 = 20$ ms. It is not difficult to understand why this is so. While with only one category the latter may be adjusted to match best the speed of the paging device, in the case of a large number of categories there is not much adjustment possible. The category mechanism "classifies" the processes according to their virtual time page fault rates. There is, in general, no reason for this classification to correspond to the optimum multiprogramming level, since the latter may be very sensitive to the speed of the paging device.

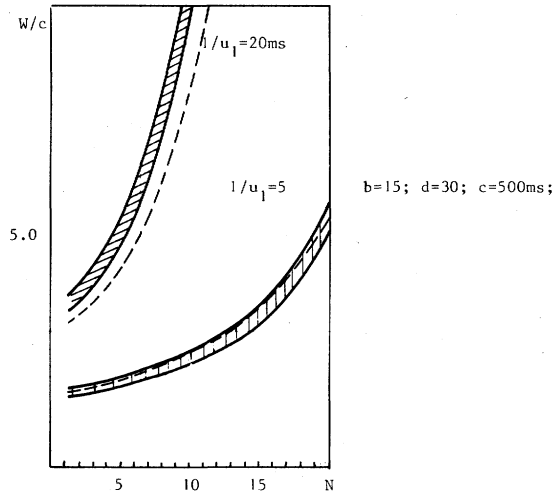


Fig. 12.

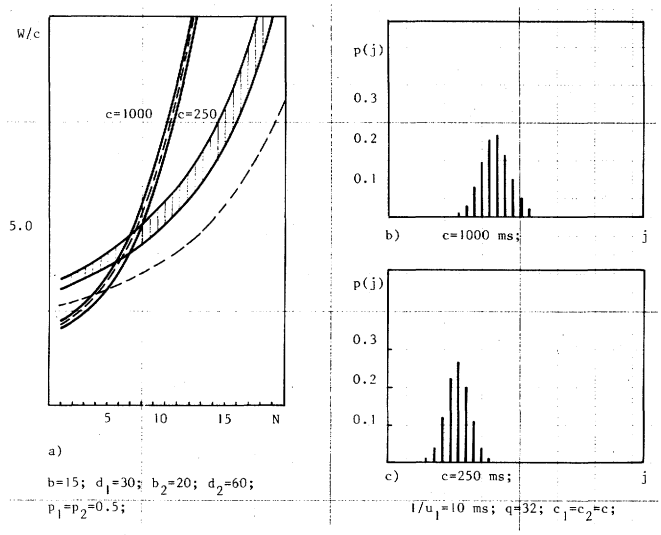


Fig. 13.

So far we have considered only a single class of processes. It is interesting and important to determine whether our observations carry over to the case where the processes form several classes as regards their paging characteristics and total CPU requirements. The next section is devoted to this subject.

IV. Several Classes of Processes

We now assume that there are ξ classes of processes present in the system. We denote by c_i, $(i = 1, \cdots, \xi)$ the average total CPU time for a class i command, and by $e_i(m)$ the mean CPU time between successive page faults for a class i process with m pages in main memory. Using obvious notational generalizations we have

$$e_i(m) = 2b_i/(1 + (d_i/m)^2) \tag{5.1}$$

$$v_{1i}(m) = \begin{cases} 1/e_i(m), & m = 1, \cdots, m_j \\ 1/\omega, & m = 0 \end{cases} \tag{5.2}$$

$$v_{0i} = 1/c_i \tag{5.3}$$

$$v_i(m) = \begin{cases} v_{1i}(0), & \text{if } m = 0, \\ v_{1i}(m) + v_{0i}, & \text{if } m = 1, \cdots, m_j \end{cases} \tag{5.4}$$

for $i = 1, \cdots, \xi$.

We denote by p_i, $(\Sigma_{i=1}^{\xi} p_i = 1)$ the probability that a newly generated command is of class i.

The analysis presented in Section II may be easily extended to include several classes of processes. In fact, it suffices to modify solution Step 1 (i.e., the analysis of process behavior in its virtual time), the other steps remaining unchanged. The detail of the modification is given in the Appendix and [32]. It is worth noting that we obtain a well-decomposed, easy to evaluate recurrent solution for Step 1. Hence, our model has no difficulty in coping with many classes of processes.

We now discuss the numerical results obtained from our model with classes of processes. We have studied the case when there are two classes of processes, equally probable $(p_1 = p_2 = 0.5)$, with different memory locality as represented by the lifetime function parameters b_i and d_i. In Fig. 13(a) we have plotted the relative average response time versus the number of terminals for a set of model parameters with 32 equidistant categories and two different values of the mean total compute time per command: $c_i = 1000$ ms, $i = 1, 2$, and $c_i = 250$ ms, $i = 1, 2$. The dotted curves correspond to the optimum category assignment with only one category.

We observe that not only a large number of categories does not result in an improvement over the optimum with one category, but may yield significantly worse results. This is in particular the case for $c_i = 250$ ms, $i = 1, 2$. Hence, we conclude that the "classification" performed by the category mechanism, based only on the virtual time behavior of processes, does not provide an effective control of the multiprogramming set. Parameters such as, e.g., the speed of the paging device clearly influence the performance of the control mechanism under consideration. We have represented in Fig. 13(b) and (c) the virtual time probability that a process belongs to category j, $p(j)$, for $c = 1000$ ms and $c = 250$ ms, respectively. We observe the important effect of the average total compute time on the process classification.

Since the category mechanism with many categories fails to control efficiently the multiprogramming set, and since its primary motivation is to provide estimates of the "working-set" sizes for different processes, i.e., to classify processes according to their locality, it is interesting to study its performance with respect to this objective. We have therefore computed the stationary (virtual time) conditional probability that a process is classed in category $j (j = 1, \cdots, q)$ given that it is of class $i (i = 1, \cdots, \xi)$, $p(j|i)$. The results obtained are represented in Fig. 14. They clearly indicate that this probability strongly depends on the total compute times c_i (which we have already observed), and on the relative frequencies of different classes of commands, p_i. In particular, increasing the total compute time c_i results in a classification in a higher category. A possible explanation for this effect is that with larger c_i more pages are referenced, and the category mechanism is such that when new pages are to be loaded the category must, quite often, be increased. Therefore, the most

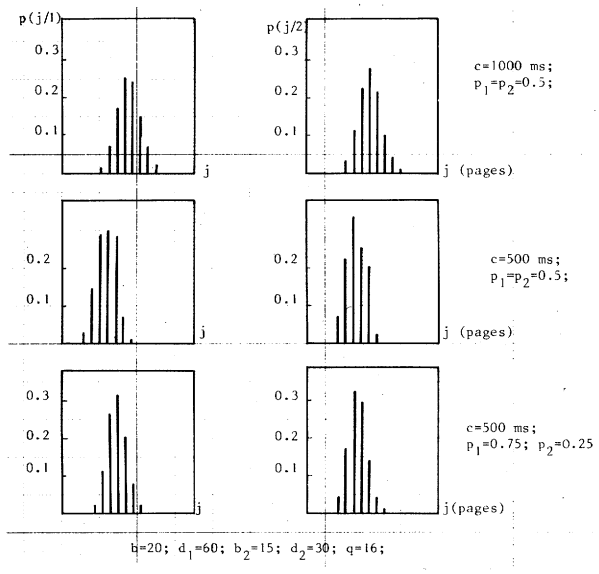


Fig. 14.

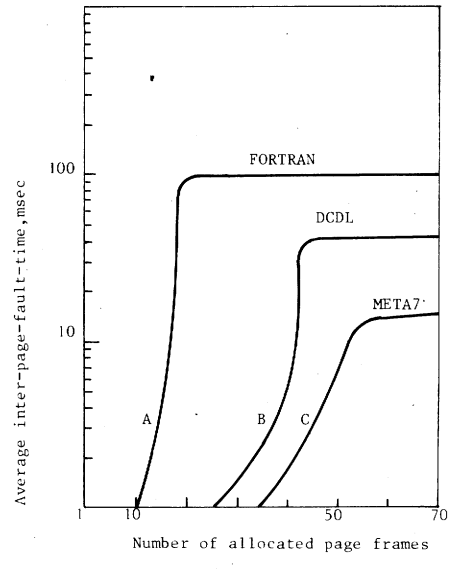


Fig. 15.

probable category for a given class of processes does not, in general, correspond to a "good" point on the lifetime curve of the process.

At this point it is important to determine to what extent our findings could be affected by the shape of the lifetime curve. In particular, it is interesting to study the multicategory versus the monocategory performance in the case where the lifetime curves of processes possess distinct sharp "knees." A sharp knee, as opposed to the relatively well-behaved shape of the lifetime curves considered so far, has been reported for a number of programs, e.g., [33].

The next section is devoted to the results obtained with measured lifetime curves of this type.

V. Results with a Sharp Knee in the Lifetime Curve

The lifetime curves used in this section are represented in Fig. 15. They are borrowed from [33]; curves A, B, and C correspond to a Fortran compiler, a DCDL compiler, and a META7 compiler, respectively.

The numerical results obtained from our model with a single class of processes indicate that the response time when several categories are used exhibits little improvement, if any, over that with optimum monocategory assignment. This is illustrated in Fig. 16(a) and (b) for lifetime curves A and B, respectively. The shaded areas represent the results with two categories, and the dotted curves correspond to the optimum (with respect to the mean response time) category assignment with only one category. As might be expected given the steep decrease in average interpage-fault-time below the knee of the lifetime curves used, the optimum monocategory assignment is strongly correlated with the knee. It corresponds, of course, to a number of pages beyond the knee, i.e., in the "flat" region of the lifetime curves.

It is somewhat more surprising to observe that, even when the processes in the system belong to two different classes with such distinct paging behaviors as represented by life-time curves A and C, the optimum monocategory assignment may

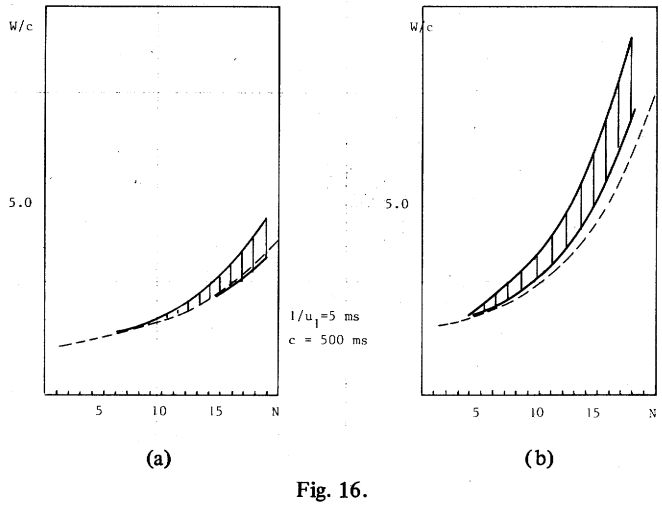


Fig. 16.

still yield better results than a large number of categories. This is illustrated in Fig. 17 for an equiprobable mixture of processes of type A and C.

VI. Conclusion

We have presented a study of the category mechanism for multiprogrammed memory management. This study has been performed in the context of a demand-paged interactive system. A queueing network model of such a system has been built, and a hybrid (analytical-discrete event simulation) decomposed solution has been obtained for classes of processes. The model neglects I/O activity other than paging, and overheads other than that incurred when a process is admitted into the multiprogramming set. (Numerical results, not reported in this paper, show that the latter overhead has little influence on the performance of the admission mechanism.) These features can be relatively easily incorporated in our model, and require that only Step 2 (i.e., the analysis of the CPU utilizations) of our solution procedure be modified.

The numerical results obtained indicate that the system

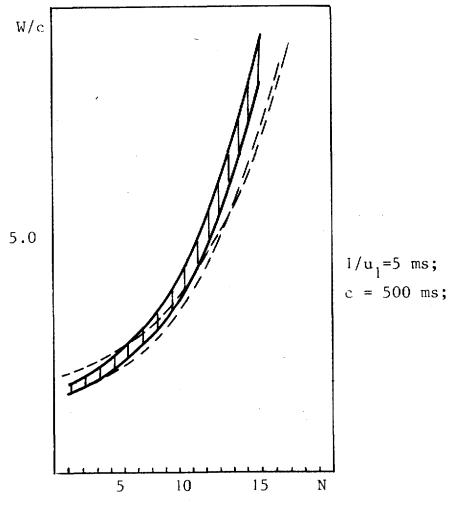


Fig. 17.

performance (as measured by the relative average response time) with several categories is, in general, worse than the optimum performance with a single category. This seems to be true both for cases when there is a single class and when there are several classes of processes with distinct paging behavior characteristics. Moreover, the category which is the most probable in the virtual time of a process of a given class does not seem to reflect in any clear way the paging characteristics of the process, as represented by its lifetime curve.

Hence, we must conclude that, for the types of life-time curves considered, the category mechanism fails both in optimizing the level of multiprogramming and in classifying the processes according to their paging characteristics, i.e., in dynamically partitioning real memory. We interpret the former failure as indicating that an effective control of multiprogramming, in general, cannot be based solely on virtual time behavior of processes. (The working set policy, which has proven effective, is not based only on virtual-time behavior since the choice of the window size depends on system characteristics.) Although, algorithms including the speed of the paging device have been proposed (e.g., [11]), the problem of optimal multiprogramming seems to require further study. Indeed, a (near) optimal policy should also incorporate features such as various I/O's and overheads (including that of the policy itself) since these may be shown to importantly affect the optimal multiprogramming level.

The observed fact that the expected response time when no process classes are distinguished (optimum monocategory assignment) may compare favorably with the system performance when classes are distinguished is of potential interest to system designers, and to people working on dynamically adaptative multiprogramming. It therefore deserves further investigation, under broader conditions.

Appendix

A. A Few Details of the Solution Steps

Step 1: Denote by $p(m,j)$ the stationary probability (*in process execution time*) that a process has m pages in real memory and that it is of category j; $m = 0, 1, \cdots, m_j; j = 1, \cdots, q$. The balance equations for $p(m,j)$ are easily obtained as

$$v_1(0)\,p(0,j) = v_0\left(\sum_{m=1}^{m_j^*} p(m,j+1) + \sum_{m=m_{j-1}^*+1}^{m_j} p(m,j)\right) + v_1(m_{j-1})\,p(m_{j-1}, j-1); \quad \text{for } j = 2, \cdots, q-1 \tag{A1}$$

$$v_1(0)\,p(0,1) = v_0\left[\sum_{m=1}^{m_1^*} p(m,2) + \sum_{m=1}^{m_1} p(m,1)\right]; \tag{A2}$$

$$v_1(0)\,p(0,q) = v_0 \sum_{m=m_{q-1}^*+1}^{m_q} p(m,q) + v_1(m_{q-1})\,p(m_{q-1}, q-1) + v_1(m_q)\,p(m_q, q); \tag{A3}$$

$$[v_1(m) + v_0]\,p(m,j) = v_1(m-1)\,p(m-1,j), \quad m = 1, 2, \cdots, m_j;\ j = 1, 2, \cdots, q. \tag{A4}$$

Let $p(m|j)$ be the stationary conditional probability (in process execution time) that the process has m pages in memory given that it is of category j. Using (A4) and the fact that

$$\sum_{m=0}^{m_j} p(m|j) = 1, \quad j = 1, \cdots, q$$

we readily get

$$p(m|j) = \frac{1}{G_j} \cdot \frac{(v_1(m_j) + v_0) \cdots (v_1(m+1) + v_0)}{v(m_j - 1) \cdots v_1(m)} \quad m = 0, \cdots, m_j; \tag{A5}$$

where G_j is a normalization constant.

Hence, we have

$$\bar{m}_j = \sum_{m=0}^{m_j} m p(m|j); \tag{A6}$$

$$f_j = \sum_{m=0}^{m_j} p(m|j)\,v_1(m); \tag{A7}$$

$$w_j = v_1(m_j)\,p(m_j|j); \tag{A8}$$

$$z_j = [1 - p(0|j)]\,v_0; \tag{A9}$$

$$s_j = \left[\sum_{m=1}^{m_{j-1}^*} p(m|j)\right] \Big/ [1 - p(0|j)]. \tag{A10}$$

In order to compute the stationary probability (in process execution time) that the process is of category j, denoted $p(j)$, we note that $p(j) = \Sigma_{m=1}^{m_j} p(m,j)$. Summing (A4) over $m = 1, \cdots, m_j$, we obtain

$$v_1(m_j)\,p(m_j,j) + v_0 \sum_{m=1}^{m_j} p(m,j) = v_1(0)\,p(0,j).$$

Adding to this equation (A1), (A2), and (A3), respectively, and using the fact that $p(m,j) = p(m|j)\,p(j)$ we get

$$\left[v_1(m_j)\,p(m_j|j) + v_0 \sum_{m=1}^{m_{j-1}^*} p(m|j)\right] p(j)$$

$$= p(j+1)\,v_0 \sum_{m=1}^{m_j^*} p(m|j+1)$$

$$+ p(j-1)\,v_1(m_{j-1})\,p(m_{j-1}|j-1),$$

$$j = 2, \cdots, q-1;$$

$$v_1(m_1)\,p(m_1|1)\,p(1) = p(2)\,v_0 \sum_{m=1}^{m_1^*} p(m|2);$$

$$v_0 \sum_{m=1}^{m_{q-1}^*} p(m|q)\,p(q) = v_1(m_{q-1})\,p(m_{q-1}|q-1)\,p(q-1).$$

Hence, $p(j)$ may be easily computed:

$$p(j) = \frac{1}{H} \prod_{i=2}^{j} v_1(m_{i-1})\,p(m_{i-1}|i-1) \Bigg/ \left[v_0 \sum_{m=1}^{m_{i-1}^*} p(m|i)\right],$$

$$j = 1, \cdots, q \quad \text{(A11)}$$

where H is a normalization constant. We have, of course,

$$p(m,j) = p(m|j)\,p(j). \quad \text{(A12)}$$

Step 4: Denote by $p(l)$ the stationary probability that there are l processes in the system. We have

$$p(l) = \frac{1}{G} \lambda^l \prod_{i=1}^{l} (N-i+1)/u(i), \quad l = 0, 1, \cdots, N \quad \text{(A13)}$$

where G is a normalization constant. Hence, we get

$$\bar{l} = \sum_{l=1}^{N} l p(l) \quad \text{(A14)}$$

and, from Little's formula [24]

$$W = \bar{l}/[(N-\bar{l})\,\lambda]. \quad \text{(A15)}$$

In order to estimate W' and W'', we compute

$$g'(l) = \lambda^l \prod_{i=1}^{l} (N-i+1)/u''(i);$$

$$g''(l) = \lambda^l \prod_{i=1}^{l} (N-i+1)/u'(i);$$

$$G' = 1 + \sum_{l=1}^{N} g'(l); \qquad G'' = 1 + \sum_{l=1}^{N} g''(l);$$

$$p'(l) = g'(l)/G'; \qquad p''(l) = g''(l)/G'';$$

$$\bar{l}' = \sum_{l=1}^{N} l p'(l); \qquad \bar{l}'' = \sum_{l=1}^{N} l p''(l);$$

$$W' = \bar{l}'/[(N-\bar{l}')\,\lambda]; \qquad W'' = \bar{l}''/[(N-\bar{l}'')\,\lambda].$$

B. Step 1 of the Solution Procedure with Several Classes of Processes

We denote by $p(m,j,i)$ the stationary probability (in the virtual time of a process) that a process has m pages in real memory, that it current category is j, and that it is of class i; $m = 0, 1, \cdots, m_j$; $j = 1, \cdots, q$; $i = 1, \cdots, \xi$. We also denote by $p(m,i|j)$ the stationary (virtual time) conditional probability of the number of pages and the class of a process given its current category, and by $p(m|j,i)$ the stationary (virtual time) conditional probability that a process has m pages in memory given its current category and class. Finally, we let $p(i|j)$ be the stationary (in process virtual time) probability that the process is of class i given that its current category is j.

With these notations we have

$$\bar{m}_j = \sum_{m=0}^{m_j} m \sum_{i=1}^{\xi} p(m,i|j); \quad \text{(A16)}$$

$$f_j = \sum_{i=1}^{\xi} \sum_{m=0}^{m_j} p(m,i|j)\,v_{1i}(m); \quad \text{(A17)}$$

$$w_j = \sum_{i=1}^{\xi} p(m_j,i|j)\,v_{1i}(m_j); \quad \text{(A18)}$$

$$z_j = \sum_{i=1}^{\xi} v_{0i}\,[p(i|j) - p(0,i|j)]; \quad \text{(A19)}$$

$$s_j = \left[\sum_{i=1}^{\xi} v_{0i} \sum_{m=1}^{m_{j-1}^*} p(m,i|j)\right] \Big/ z_j, \quad j = 1, 2, \cdots, q. \quad \text{(A20)}$$

We have

$$p(m,i|j) = p(m|j,i)\,p(i|j). \quad \text{(A21)}$$

A straightforward generalization of (A4) and (A5) yields

$$p(m|i,j) = \frac{1}{G_{ij}} \prod_{k=m+1}^{m_j} [v_{1i}(k) + v_{0i}]/v_{1i}(k-1)$$

$$m = 0, 1, \cdots, m_j \quad \text{(A22)}$$

where G_{ij} is a normalization constant.

Denote by $p(j)$ the stationary (in process virtual time) probability that a process is of category j. It is not difficult to show, summing the balance equations for $p(m,j,i)$ over $m = 0, \cdots, m_j$ and $i = 1, \cdots, \xi$, that

$$p(j) = \frac{1}{H} \prod_{l=2}^{j} w_{l-1}/(s_l z_l), \quad j = 1, \cdots, q \quad \text{(A23)}$$

where H is a normalization constant. It is not difficult to obtain balance equations for the joint probability $p(j,i)$ that a

process is of category j and of class i. Using the fact that

$$p(j,i) = p(i|j)\,p(j), \tag{A24}$$

together with (A23) one readily gets the balance equations for $p(i|j)$. Their solution may be obtained as follows. First we compute $p(i|j)$ for $j = 1$:

$$p(i|1) = \frac{1}{F_{i1}} \cdot p_i / \{p(m_1|1,i)\,v_{1i}(m_1) + v_{0i}[1 - p(0|1,i)]\} \qquad i = 1, \cdots, \xi \tag{A25}$$

where F_{i1} is a normalization constant. Then, $p(i|j)$ is obtained for consecutive values of j, $j = 2, \cdots, q$ from the following recurrence relation:

$$g_j(i) - h_j(i) = g_j(k) - h_j(k), \qquad i,k = 1,2,\cdots,\xi \tag{A26}$$

where

$$g_j(i) = \begin{cases} w_{j-1}p(i|j)\{p(m|j,i)\,v_{1i}(m_j) \\ \quad + v_{0i}[1 - p(0|j,i)]\}/p_i, \\ \qquad\qquad \text{for } j = 2,\cdots,q-1; \\ w_{j-1}p(i|j)\,v_{0i}[1 - p(0|j,i)]/p_i, \qquad \text{for } j = q. \end{cases} \tag{A27}$$

$$h_j(i) = s_j z_j p(i|j-1)\,p(m_{j-1}|j-1,i)\,v_{1i}(m_{j-1})/p_i, \qquad \text{for } j = 2,\cdots,q; \quad i = 1,\cdots,\xi. \tag{A28}$$

The set of equations (A26) is easy to solve using the normalization condition $\Sigma_{i=1}^{\xi} p(i|j) = 1$, for $j = 1,2,\cdots,q$, and the expressions for s_j and z_j, i.e., (A19) and (A20).

We now summarize the solution of Step 1 with classes of processes. First, we compute the $p(m|j,i)$ [(A22)]. Next, we recurrently compute the $p(i|j)$ for $j = 1,2,\cdots,q$ [(A25), (A26)]. Finally we use $p(m|j,i)$, $p(i|j)$ together with (A21) in (A16) through (A20) to obtain the inputs to following solution steps [w_j is in fact computed during the solution of (A25), (A26)].

As a final point, note that one may be readily obtain the average memory space actually used by an active user (in virtual time), which we denote by $\overline{m}$, as

$$\overline{m} = \sum_{j=1}^{q} \overline{m}_j p(j), \tag{A29}$$

where $p(j)$ is given by (A23).

References

[1] L. Belady and C. J. Kuehner, "Dynamic space sharing in computer systems," *Commun. Ass. Comput. Mach.*, vol. 12, pp. 282–288, 1969.

[2] P. J. Denning, "The working set model for program behavior," *Commun. Ass. Comput. Mach.*, vol. 11, pp. 323–333, 1968.

[3] J. B. Morris, "Demand paging through utilization of working sets on the MANIAC II," *Commun. Ass. Comput. Mach.*, vol. 15, pp. 867–872, 1972.

[4] C. Bétourné, C. Kaiser, S. Krakowiak, and J. Mossiere, "Process management and resource sharing in the multiaccess system ESOPE," *Commun. Ass. Comput. Mach.*, vol. 13, pp. 727–733, 1970.

[5] H. Whitfield and A. S. Wight, "EMAS–The Edinburgh multiaccess system," *Comput. J.*, vol. 16, pp. 331–346, 1973.

[6] P. J. Denning, "Trashing: Its causes and prevention," in *AFIPS Conf. Proc., Fall Joint Comput. Conf.*, 1968.

[7] P. Courtois, "Decomposability, instabilities and saturation in multiprogramming systems," *Commun. Ass. Comput. Mach.*, vol. 18, pp. 371–377, 1975.

[8] A. Brandwajn, J. Buzen, E. Gelenbe, and D. Potier, "A model of performance for virtual memory systems," in *Proc. ACM SIGMETRICS Symp.*, Oct. 1974.

[9] D. Chamberlin, S. Fuller, and L. Liu, "An analysis of page allocation strategies for multiprogramming systems with virtual memory," *IBM J. Res. Develop.*, vol. 17, pp. 404–412, 1973.

[10] A. Brandwajn, "A model of a time-sharing system with two classes of processes," in *Gesellschaft fuer Informatik*, vol. 5. Dortmund: Springer-Verlag, Oct. 1975, pp. 547–566.

[11] P. J. Denning, K. C. Kahn, J. Leroudier, and D. Potier, "Optimal multiprogramming," *Acta Informatica*, vol. 7, pp. 197–216, 1976.

[12] J. E. Neilson, "An analytic model of a multiprogrammed batch time-shared computer," in *Proc. Int. Symp. Comput. Performance, Measurement and Evaluation*, Harvard Univ., Cambridge, MA, Mar. 1976, pp. 59–70.

[13] C. E. Landwehr, "An endogenous priority model for load control in combined batch-interactive computer systems," in *Proc. Int. Symp. Comput. Performance Measurement and Evaluation*, Harvard Univ., Cambridge, MA, Mar. 1976, pp. 282–295.

[14] M. Reiser and A. G. Konheim, "Queuing model of a multiprogrammed computer system with a jobqueue and a fixed number of initiators," in *Modelling and Performance Evaluation of Computer Systems.* Ispra, Italy, Oct. 1976, pp. 319–334.

[15] J. M. Hine, I. Mitrani, and S. Tsur, "The use of memory allocation to control response times in paged computer systems with different job classes," in *Modelling and Performance Evaluation of Computer Systems.* Ispra, Italy, Oct. 1976, pp. 201–216.

[16] E. G. Coffman and T. A. Ryan, "A study of storage partitioning using a mathematical model of locality," *Commun. Ass. Comput. Mach.*, vol. 15, pp. 185–190, 1972.

[17] A. L. Schoute, "Comparison of global memory management strategies in virtual memory systems with two classes of processes," in *Modelling and Performance Evaluation of Computer Systems.* Ispra, Italy, Oct. 1976, pp. 389–414.

[18] Y. Bard, "The modeling of some scheduling strategies for an interactive system," in *Proc. Int. Symp. Comput. Performance Modelling, Measurement, and Evaluation*, Yorktown Heights, NY, Aug. 1977, pp. 113–137.

[19] G. S. Graham and P. J. Denning, "On the relative controllability of memory policies," in *Proc. Int. Symp. Comput. Performance Modelling, Measurement, and Evaluation*, Yorktown Heights, NY, Aug. 1977, pp. 411–428.

[20] M. Parent and D. Potier, "A note on the influence of program loading on the page fault rate," in *Modelling and Performance Evaluation of Comput. Systems.* Ispra, Italy, Oct. 1976.

[21] A. Brandwajn and B. Mouneix, "A study of a page-on-demand system," *Inform. Processing Lett.*, vol. 6, pp. 125–132, 1977.

[22] A. L. Scherr, *An Analysis of Time-Shared Computer Systems.* Cambridge, MA: M.I.T. Press, 1967.

[23] V. A. Abell and S. Rosen, "Performance of an ECS-based time-sharing subsystem," in *Proc. Int. Symp. Comput. Performance, Modeling, Measurement and Evaluation*, Yorktown Heights, NY, Aug. 1977, pp. 249–261.

[24] J. D. Little, "A proof of the queueing formula $L = \lambda W$," *Oper. Res.*, vol. 9, pp. 383–387, 1961.

[25] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed and mixed networks of queues with different classes of customers," *J. Ass. Comput. Mach.*, vol. 22, pp. 248–260, 1975.

[26] J. Leroudier and M. Parent, "Quelques aspects de la modélisation des systèmes informatiques par simulation à événements discrets," *RAIRO Informatique*, vol. 10, pp. 5–26, 1976.

[27] G. S. Fishman, "Statistical analysis for queueing simulations," *Management Sci.*, vol. 20, pp. 363–369, 1973.

[28] K. M. Chandy, U. Herzog, and L. Woo, "Approximate analysis of queueing network models," IBM Res. Rep. RC 4931, Yorktown Heights, NY, July 1974.

[29] P. J. Denning and J. Buzen, "Operational analysis of queueing networks," in *Proc. 3rd Int. Symp. Modelling and Performance Evaluation of Comput. Syst.*, Bonn, Germany, Oct. 1977.

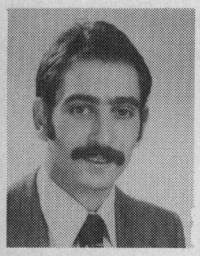
[30] H. D. Schwetman, "Hybrid simulation models: A speed-up

technique combining analytic and discrete-event modeling," in *Modelle fuer Rechensysteme, Workshop der GI*. Bonn, Germany: Springer-Verlag, Apr. 1977, pp. 226–237.

[31] A. Brandwajn, "Simulation de la charge d'un système conversationnel," *RAIRO Informatique*, vol. 10, pp. 25–40, 1976.

[32] A. Brandwajn and J. A. Hernandez, "A study of a mechanism for controlling multiprogrammed memory in an interactive system," Ecole Nationale Supérieure des Télécommunications, Paris, Res. Rep. ENST-D-78008, May 1978.

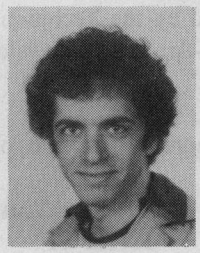
[33] W. W. Chu and H. Opderbeck, "The page fault frequency replacement algorithm," in *AFIPS Conf. Proc. Fall Joint Comput. Conf.*, vol. 41, 1972, pp. 597–608.



Alexandre Brandwajn received the Ingeuieur Civil des Télécommunications degree from the Ecole Nationale Supérieure des Télécommunications, Paris, France, in 1971, and the Docteur-Ingenieur and Docteur es-Sciences degrees from the University of Paris VI in 1972 and 1975, respectively.

While a Researcher at IRIA-Laboria, Rocquencourt, France, from 1971 to 1975, he worked on operating system evaluation and on solution methods for analytical models of computer performance. In 1975 he joined the Ecole Nationale Supérieure des Télécommunications where he was a Professor of computer science. A full Professor there from 1977 to 1979, he directed a research project in computer architecture aimed at the design of a dynamically adaptive system, while at the same time pursuing research in the areas of analytical and numerical methods for queueing models, and fault tolerance techniques. He is currently with Amdahl Corporation, Sunnyvale, CA, where he is participating in the activities of the Systems Performance Architecture Group.

Dr. Brandwajn is a member of the Association for Computing Machinery.



Jean-Alain Hernandez was born in Lyon, France, on March 16, 1951. He received the Ingenieur degree in 1974.

From 1974 to 1977 he worked in industry. In 1977 he joined the Department of Computer Science, Ecole Nationale Supérieure des Télécommunications, Paris, France, where he is currently an Associate Professor. His research interest are in the areas of computer structure, fault-tolerant computing, and high-level languages.