D. D. Chamberlin
S. H. Fuller
L. Y. Liu

An Analysis of Page Allocation Strategies for Multiprogramming Systems with Virtual Memory

Abstract: In a multiprogramming, virtual-memory computing system, many processes compete for the main storage page frames and CPU's of the real system. It is customary to define a subset of these processes called the "multiprogramming set" (MPS), and to allocate resources only to those processes currently in the MPS. Each process remains in the MPS for a limited time and is then demoted. The system paging manager controls the size of the MPS; it allocates the available page frames among the processes in the MPS and fetches appropriate pages into the page frames.

A model is described that assumes the most critical resources of the system to be page frames and the paging channel (i.e., there is no significant CPU contention). The model makes certain assumptions about the page fault rate of processes as a function of page frames allocated, and about the page fetch time as a function of mean load on the paging channel. The model also incorporates a definition of the value of a given page allocation in terms of system throughput.

The model is used to study various strategies for choosing an MPS and allocating page frames among processes. For simple cases, the model yields an exact optimal strategy. A heuristic strategy is proposed for dealing with more complex cases, and is shown by the model to be reasonably near optimal. The heuristic strategy monitors the page fault rate of each process and chooses an allocation such that each process can be executed at a reasonable rate, while ensuring that the paging channel is neither overloaded nor underloaded.

1. Introduction

In a multiprogramming, virtual-memory computing system, many processes compete simultaneously for the resources of the system. (In this paper, we define a *process* as a program with its own virtual memory, which requires an allocation of real memory space and a CPU in order to be executed.) The principal system resources are CPU's, main memory page frames, and the transmission capacity of the paging drum. Here we consider systems in which the scarce resources are page frames and the paging drum (i.e., the system is not CPU-bound). Subject to this assumption, we study ways of allocating resources to the processes in order to maximize system throughput.

It is customary to define a subset of the processes known to the system, called the *multiprogramming set* (MPS), and at each instant to allocate resources only to those processes currently in the MPS. While in the MPS a process is allocated a certain number of main storage page frames, and is allowed to be executed. Its execution is periodically interrupted by *page faults*, in which the process references a page that is not present in main storage and must wait for the page to be fetched. A process remains in the MPS until it finishes or exhausts its *time slice*, at which time it is demoted.

We assume the existence of two resource managers within the virtual-memory operating system: the paging manager and the scheduler. The paging manager controls the size of the MPS and allocates main storage page frames among those processes in the MPS. The scheduler assigns time-slice lengths to the various processes and defines a promotion order among those processes not currently in the MPS. The scheduler must ensure that system responsiveness is adequate, while the paging manager is primarily concerned with throughput. This paper studies possible strategies for the paging manager. A strategy for the scheduler is proposed in [1].

In order to evaluate various strategies for the resource managers, it was necessary to construct a model of a time-sharing system. Because of the large amount of data we wished to gather, we did not consider a pure simulation model to be feasible. Furthermore, since we wished to study the allocation of resources to a heterogeneous mix

of processes, we were unable to use conventional queuing models, which assume that all processes in the system are identical [2]. Therefore, we developed a model of our own, analytical in nature but not based on queuing theory.

The models and experiments to be described are representative of the following approach to the general problem of system resource allocation:

1. Assume a set of parameterized equations that characterize the environment in which resources are to be allocated.
2. Define an objective function.
3. For various values of the parameters, solve for the resource allocation that optimizes the objective function.
4. Evaluate various simple heuristic allocation schemes by comparing their performance with the optimum as defined by step 3.

2. Modeling the user load

In an earlier paper [3], Belady and Kuehner introduced the concept of a "life-time function," which relates e_i, the expected execution time between page faults for a given process, to p_i, the number of page frames allocated to the process. (In this paper, we assume that the page frame allocation to a given process is constant for the short term, and that a process can fetch a new page only by relinquishing a page it currently possesses in main storage. In the long term, by monitoring the behavior of a process, the paging manager may choose to change its page frame allocation.) Belady and Kuehner cited evidence that the lifetime function has two regions: a concave upward region, followed by a concave downward region, as shown in Fig. 1. This corresponds to the observation that if a process is allocated very many or very few page frames, it tends to use them inefficiently, but that each process has some intermediate number of allocated page frames, variously called its "locality" [4] or "working set" [5], which enables it to be executed efficiently. For the present paper, we fit the lifetime function curve with the simple equation

$$e_i = \frac{2\,B_i}{1 + \left(\frac{C_i}{p_i}\right)^2}. \qquad (1)$$

This equation enables us to describe a process by the following two parameters:

C_i(pages): A relative measure of the number of page frames needed to enable the process to be executed efficiently. More precisely, the number of page frames that provides the process with half of its largest possible lifetime.

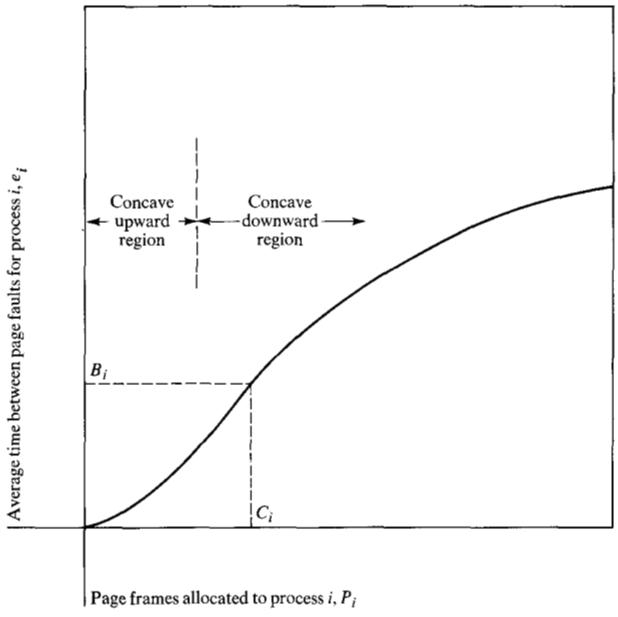


Figure 1 Lifetime function for process i.

B_i(ms): The expected execution time between page faults for process i when it is allocated C_i page frames.

Like Belady and Kuehner, we assume that the parameters B_i and C_i are invariant during the period of interest. Also, we assume that, during the period of interest, processes neither arrive at the system nor terminate. Therefore, we can completely describe the load on the system by specifying, for each process, the parameters B_i and C_i. Experience has shown that, on the IBM System/360 model 67 computer, appropriate values for C_i's are in the low tens of pages, and for B_i's are in the low tens of milliseconds [6,7].

3. Modeling the paging drum

Whenever a process sustains a page fault, it goes into a wait state until the required page can be fetched from the paging drum. The length of the waiting period depends on the capabilities of the drum and on the length of the queue of requests for pages to be fetched. In this paper, we assume that the circumference of the drum is divided into an integral number of sectors, and that pages are stored in such a way that they do not cross sector boundaries. When a request is made to read or write a page, the request is placed in the appropriate sector queue. As the drum rotates, its read-write heads reach each sector in turn and service the requests on each sector queue in first-in, first-out order. We assume that the drum has rotational period T and has N_S sectors.

We desire to find a relationship between the average time to service a page fetch request (W) and the total

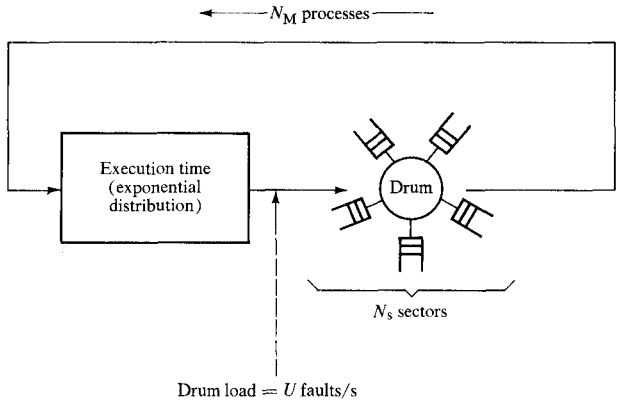


Figure 2 Simulation of paging drum.

load on the paging drum in requests per second (U). This problem has been solved analytically by Skinner [8] and Coffman [9] under the assumption that arrival times of drum requests form a Poisson process. In general, however, arrivals of drum requests in a computer system do not form a Poisson process. Therefore, we have chosen to model the computer system with the more realistic cyclic-queue structure shown in Fig. 2, which does not require this assumption. An experiment was performed in which the structure shown in Fig. 2 was simulated. A total of N_M processes were considered to be active in the system. After each page request is serviced, the process that issued the request is executed for a random interval of time before it makes another page request. The intervals are drawn from an exponential distribution. (Choosing execution intervals from the same distribution for all processes may seem to contradict our assumption that different processes have different characteristics. However, we will show that the paging manager can and indeed should, allocate pages to nonidentical processes in such a way that their mean execution intervals between page faults are the same. In any case, we do not expect the drum characteristics described here to be sensitive to differences among processes.) The page requests are placed on each of the N_S sector queues with equal probability. Consistent with our original assumption of a non-CPU-bound system, we assume that no process ever waits for a CPU after its page request has been satisfied. By adjusting the mean execution interval, we vary the total drum load (U) and observe the effect on the average wait time (W).

Clearly the results must be dependent on the degree of multiprogramming, N_M. If $N_M = 1$, the single process always sees an empty drum queue, and the average wait time to service a fault is the "no load" wait:

$$W_{NL} = \left(\frac{1}{2} + \frac{1}{N_S}\right)T. \tag{2}$$

The no load wait W_{NL} consists of one-half revolution average latency, plus one sector read time to transmit the page. If there are many processes in the system, we expect W to vary from W_{NL} to infinity, depending on the load U. Wait W should approach infinity as U approaches the maximum transmission capacity M of the drum, which is given by:

$$M = N_S/T. \tag{3}$$

The results of the simulation experiment confirmed the above expectations. A family of curves was plotted that gives W as a function of U for various values of N_M, as shown in Fig. 3. Several equations were fitted to the results of the simulation and the following basic hyperbolic form was found to be most appropriate:

$$W = [K_1/(M - U)] + K_2. \tag{4}$$

Least-squares analysis was used to evaluate the constants K_1 and K_2 for various degrees of multiprogramming N_M. The following specific equation was found to fit the results very closely:

$$W = \frac{2K_M}{M - U} + W_{NL} - \frac{2K_M}{M}, \tag{5}$$

where M and W_{NL} are defined by Eqs. (2) and (3) and K_M is a factor determined by the degree of multiprogramming, defined as follows:

$$K_M = \frac{N_M - 1}{N_M}. \tag{6}$$

The total drum load U consists of the sum of the real-time page fault rates of all the processes in the multiprogramming set:

$$U = \sum_{i=1}^{N_M} u_i, \tag{7}$$

where

$$u_i = 1/(e_i + W). \tag{8}$$

If we let $\rho = U/M$, Eq. (5) can be written as:

$$W = \left(\frac{1}{2} + \frac{1}{N_S}\right)T + \frac{2\rho K_M}{N_S(1 - \rho)}T.$$

It is interesting to note the marked degree of similarity between the above equation and the following expression for W derived by Skinner and Coffman for the special case of a Poisson arrival process:

$$W = \left(\frac{1}{2} + \frac{1}{N_S}\right)T + \frac{\rho}{2(1 - \rho)}T.$$

4. Definition of value

We now wish to make a reasonable definition of the *value* of a particular allocation of pages to a particular set of processes. First we note that the *rate of progress* r_i of

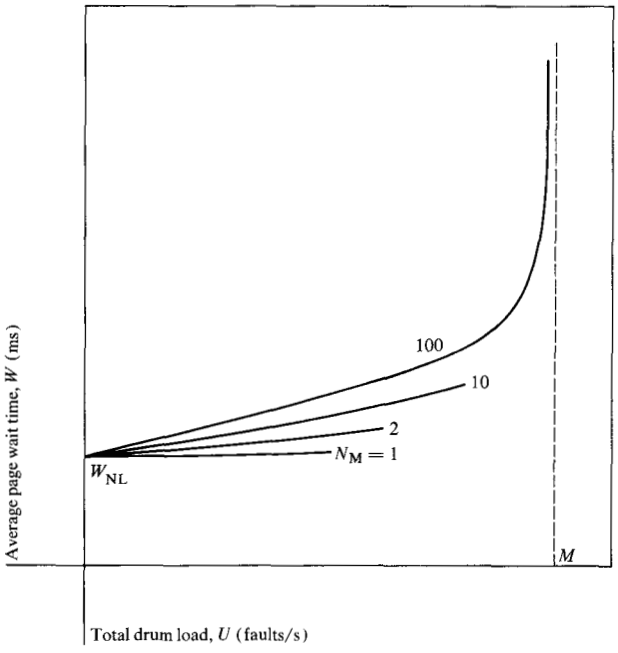


Figure 3 Drum wait as a function of load.

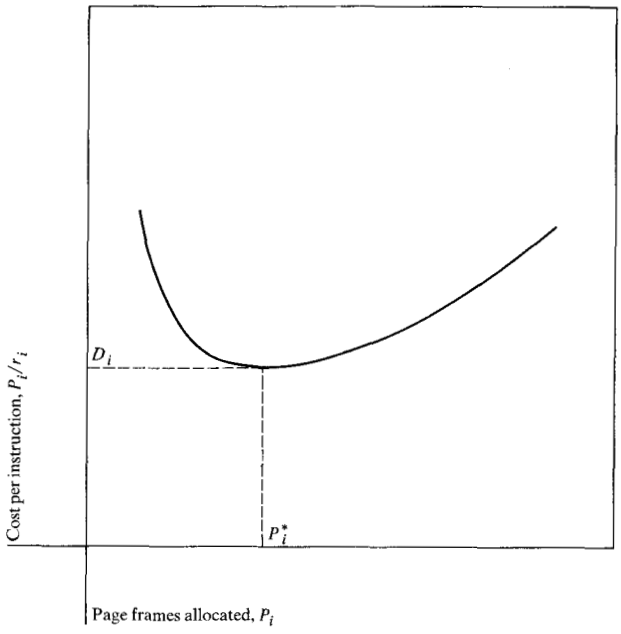


Figure 4 Derivation of demand of a process.

process i (expressed in instructions per second) is given by the equation:

$$r_i = \frac{e_i}{e_i + W} S, \tag{9}$$

where S is the speed of the CPU in instructions per second. We wish to take into account the fact that some processes are more *demanding* than others in the sense that they require more page frames in order to be executed efficiently, and that it is more difficult (and hence more valuable) to execute an instruction for a more demanding process than for a less demanding process. We will define the *rate of accrual of value* (v_i) of process i to be the product of its demand (D_i) and its rate of progress (r_i):

$$v_i = D_i r_i \tag{10}$$

We proceed to define the rate of accrual of value for the system as a whole (V) as the sum of the value rates of all the processes in the system:

$$V = \sum_{i=1}^{N_J} v_i. \tag{11}$$

We are now left with the task of defining the demand (D_i) of a process. One possible candidate is C_i, which has been defined as a measure of the number of page frames needed for process i to progress at half its maximum rate. But this measure would neglect B_i, which also has a strong influence on the behavior of the process (see Fig. 1). What we really want is a measure of the relative cost to the system (in page-seconds) of executing an instruction for process i. To gain such a measure, we consider a hypothetical experiment in which process i runs alone on a virtual-memory computer. We allocate to the process various numbers of page frames, p_i, allowing the other page frames to stand idle, and observe its behavior. As a function of p_i, we plot the average system page-seconds required to execute one instruction (represented by p_i/r_i). The results, derived from Eqs. (1) and (9), are shown in Fig. 4. There is some optimal number of page frames, p_i^*, that minimizes the system's cost per instruction when process i runs alone. The actual cost per instruction at this minimum point is defined to be the demand of process i:

$$D_i = \left(\frac{p_i}{r_i}\right) \qquad p_i = p_i^*. \tag{12}$$

From Eqs. (1) and (9), D_i may be expressed in terms of the characteristics B_i and C_i of the process:

$$D_i = \frac{C_i}{B_i} \sqrt{W_{NL}^2 + 2B_i W_{NL}}. \tag{13}$$

This definition of demand has the feature that it is a property only of the characteristics of the individual process, and does not depend on the environment in which the process is run.

5. Solution procedure

Our model of a multiprogrammed, virtual-memory computer system has been described in the preceding three

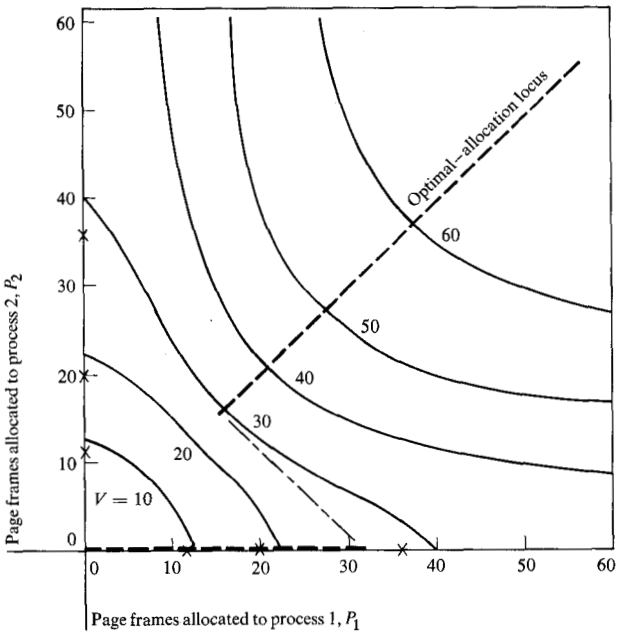
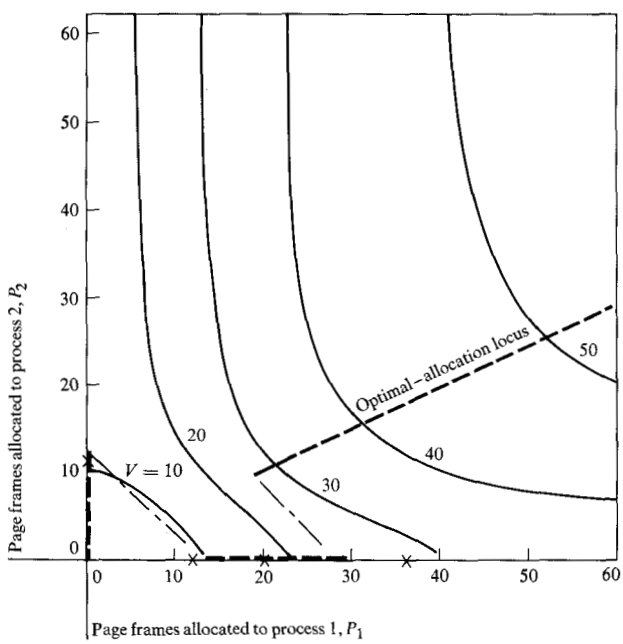


Figure 5 Experiment 1.

Table 1 Process characteristics for experiments 1, 2, 3.

	Expt. 1	*Expt.* 2	*Expt.* 3
B_1 (ms)	20	20	20
C_1 (pages)	50	50	50
B_2 (ms)	20	20	40
C_2 (pages)	50	25	50

Figure 6 Experiment 2.



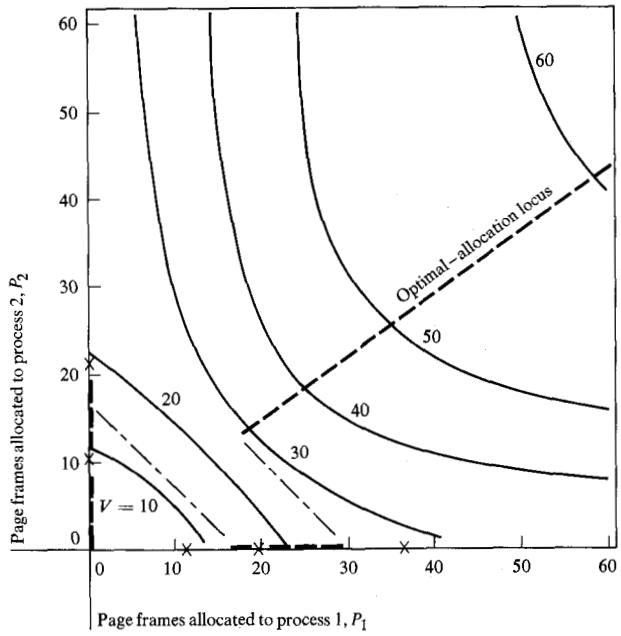


Figure 7 Experiment 3.

sections. The above definitions and assumptions must now be brought together into a procedure for finding the value of a particular page allocation.

Equations (5), (7), and (8) can be combined to give the following implicit relation for W:

$$W = \left\{2K_M \Big/ \left[M - \sum_{1 \le i \le N_M} (e_i + W)^{-1}\right]\right\} + W_{NL} - \frac{2K_M}{M}. \tag{14}$$

Equation (1) gives e_i as a function of B_i, C_i, and P_i for each process. Therefore, if we specify the lifetime functions of all processes in the multiprogramming set, the drum characteristics, and a particular page allocation, we can solve Eq. (14) for W. In our analysis we used the secant method and found it to converge consistently within a few steps to single-precision accuracy on an IBM System/360 computer.

Once W is determined, we can use Eq. (9) to find the rate of progress, r_i, for each of the processes, and then use Eqs. (10) and (11) to find V, the value of the page allocation. Note that V will have units of pages, and will be less than or equal to the number of page frames in the system. The ratio of V to the number of system page frames might be considered a measure of the efficiency of use of main memory.

An interactive PL/I program was written to aid in evaluating various page allocations and searching the space of possible allocations for the one that produces optimum value under various conditions. The following sections describe results obtained by use of this program.

6. Simple observations

The first experiments done with the modeling program were studies of an extremely simple system in which only two processes were active. The system drum has parameters $T = 10$ ms, $N_S = 5$, which corresponds to an IBM 2035 drum with page size = $2K$ bytes [10]. Three experiments were performed. The B_i's and C_i's chosen for the two processes in each experiment are shown in Table 1.

In each experiment, many possible allocations of pages to the two processes were tried, and their values were recorded. The results are shown in Figs. 5, 6, and 7, in which equal-value contours are drawn on the plane of possible page allocations. The following observations may be made from these figures:

1. The equal-value contour lines are sharply discontinuous on the axes, reflecting the fact that it is always much better to give n pages to one process and no pages to the other than to give $(n-1)$ pages to one process and one page to the other. (Axis intercepts of the equal-value contours are represented by X's in the figures.)
2. On the figures, any line of slope -1 represents all the possible allocations for a system with a fixed number of page frames.
3. The effect on the value contours of decreasing C_2 is similar to the effect of increasing B_2; both changes tend to decrease the demand of process 2.
4. The locus of optimal (best-value) allocations for various numbers of pages is represented on the figures by the heavy dashed line.
   a. For two identical processes, the optimal strategy is to run only one process if there are fewer than a certain number of pages available; if more than this number of pages are available, they should be split evenly between the two processes.
   b. For two nonidentical processes, the optimal strategy is as follows: If very few pages are available, run the less demanding process only; if somewhat more pages are available, run the more demanding process only; if relatively many pages are available, run both processes and allocate the pages to them in some fixed proportion depending on their B_i's and C_i's.

In addition to allocating pages, it is the task of the paging manager to control the size of the multiprogramming set. Once again, a simple experiment can give us insight into the nature of this task. For Experiment 4, we keep the 2305 drum characteristics as above, and we assume the user load consists of many identical processes, all with $B_i = 30$ ms, $C_i = 50$ pages. For various numbers of pages in the system, and for various degrees of multiprogramming, we find the optimal-value page allocation and record U (total drum load) and V (total system value) for this allocation. A family of curves showing drum load as a function of total pages for various degrees of multiprogramming is shown in Fig. 8. Superimposed on these curves is a locus that shows the degree of multiprogramming that yields optimum value for any given number of pages. Note that as the number of pages increases, the optimal degree of multiprogramming increases, tending to keep the drum load U within a certain band of values.

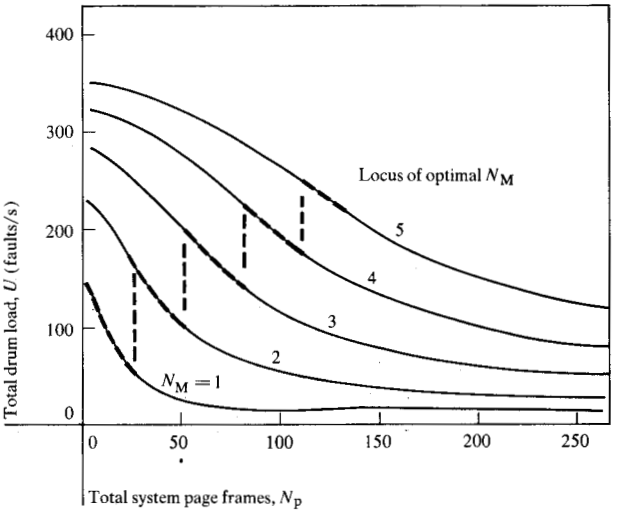


Figure 8 Experiment 4.

Table 2 Process characteristics for experiments 5, 6.

i	1	2	3	4
B_i (ms)	10	5	25	20
C_i (pages)	15	20	50	60

7. More complex cases

Having gained some insight into the problem by studying a simple case, we can now consider the more realistic case of several nonidentical processes competing for system resources. As an example of such a case, we will describe some experiments that assume the system is loaded with the four processes described in Table 2. As before, the paging manager must choose a multiprogramming set and allocate page frames to those processes chosen. The lifetime functions of these processes are shown superimposed on each other in Fig. 9. The experiments in this section employed the same 2305 drum parameters described above.

Experiment 5 studies the process of choosing a multiprogramming set (MPS) from the four available processes. There are 15 possible nonempty MPS's that

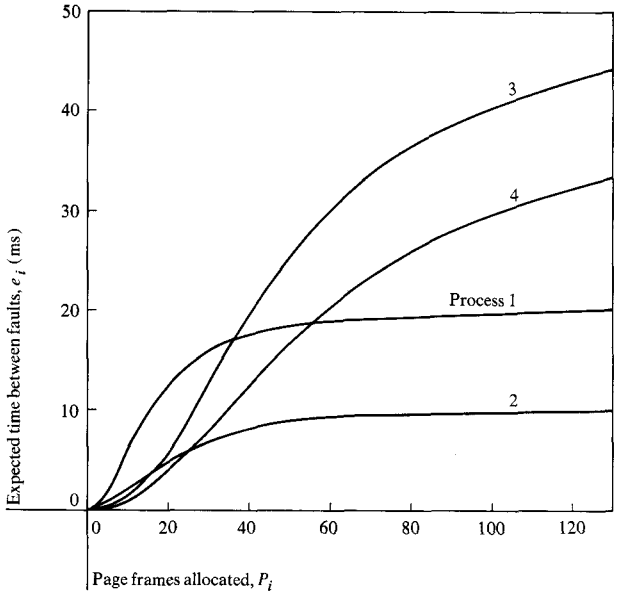


Figure 9 Lifetime functions of processes in experiments 5 and 6.

could be chosen. Each MPS has a curve of system value (V) vs total system pages (N_p), assuming that the pages are allocated in such a way as to maximize system value. Figure 10 shows those fragments of the V vs N_p curves that are optimal for various values of N_p. As expected, the best MPS when very few pages are available consists only of the smallest process; as more pages are added, the MPS grows until it contains all four processes. Some possible combinations of processes are never the optimal MPS for any value of N_p.

Experiment 6 studies the allocation of pages to processes in the MPS. We see from Fig. 10 that the optimal MPS for $N_p > 100$ consists of all four processes. So we find the optimum allocations of pages for $N_p = 100$, 150, and 200. The results are shown in Table 3.

Various attempts were made to find a pattern in these allocations; the only generalization that held for this and other similar experiments was that pages were allocated to the various processes approximately in the proportion of $C_iB_i^{-0.4}$. Figure 11 shows how this proportionality is preserved for each of the optimal allocations in Table 3.

Of course, the observation that optimal page allocations are proportional to $C_iB_i^{-0.4}$ is of little value in the design of a real paging manager, because B_i and C_i are very difficult to measure in real time for a real process. What is needed is a heuristic strategy that enables the paging manager to choose an MPS and allocate pages on the basis of quantities that are easily measured with very little overhead on a real system. Such a heuristic procedure is described in the next section.

8. Heuristic strategy

We propose that the paging manager monitor the real-time page-fault rate, u_i, of each process in the MPS, and the total system page-fault rate U. These quantities are very easy to measure in real systems. We further propose that the paging manager use these measurements to implement the following heuristic procedures:

Heuristic procedure 1: Control the size of the MPS in such a way that U is kept between limits U_{min} and U_{max}, which are defined as properties of a particular system.
Implementation: If $U < U_{min}$, promote another process into the MPS; the process to be promoted is chosen by the system scheduler [1]. If $U > U_{max}$, demote the process that has been longest in the MPS. Choose U_{min} and U_{max} far enough apart so that the probability of jumping from $U < U_{min}$ to $U > U_{max}$ is negligible.

Heuristic procedure 2: Allocate page frames to the processes in the MPS in such a way that all their page fault rates are equal ($u_i = \bar{u}$).
Implementation: If process i has $u_i < \bar{u}$ and process j has $u_j > \bar{u}$, take a page frame away from process i and give it to process j.

Evaluation: In order to evaluate the two heuristic procedures, experiments were performed to compare the system value V produced by them with the best possible value realizable under the same conditions. We report on an experiment in which the system was loaded by the eight processes described in Table 4. These processes were chosen to represent a broad spectrum of demand.

Experiment 7 was conducted assuming a system with $N_p = 50$ pages and a 2305 drum. The number of system pages was chosen small enough so that the optimum MPS would include only a subset of the available processes. The first part of the experiment was to find the best page allocation for each of the 256 possible MPS's, and to measure its value. The highest value realizable by any possible MPS, with optimal page allocation, is denoted by V^*; it was found to have the value $V^* = 46.02$ pages in Experiment 7. The next part of the experiment consisted in allocating page frames according to heuristic procedure 2 ($u_i = \bar{u}$) for each of the 256 possible MPS's. For each such allocation, the system paging rate U and the system value V were measured. These measurements are plotted on a U-V plane in Fig. 12, in which each point represents a possible MPS. (Note that this figure was produced by a computer printout. The appearance of a number such as 8 denotes the clustering of 8 points in the same print position. An asterisk denotes clustering of more than 9 points.) If we employ heuristic procedure 1 with $U_{min} = 150$ faults/s and $U_{max} = 250$ faults/s, we are confined to some MPS in

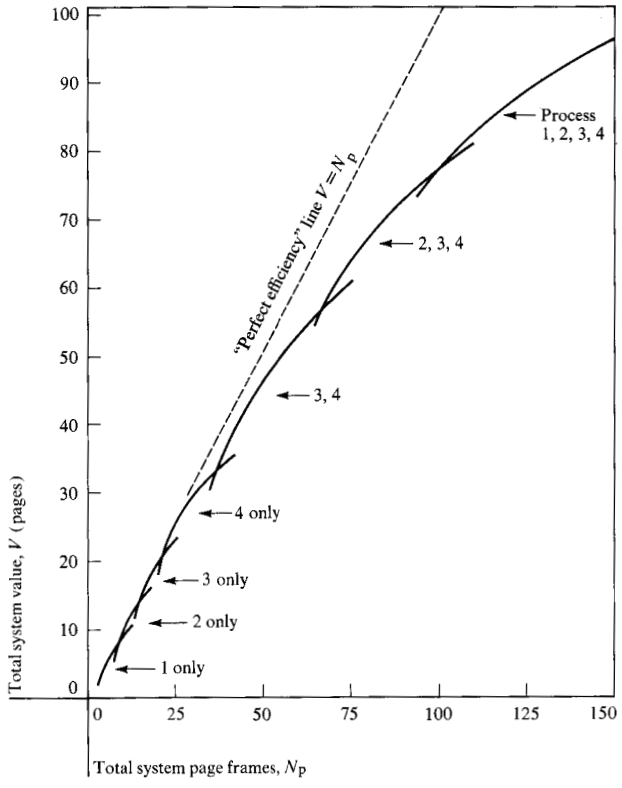


Figure 10 Experiment 5.

the center region of the graph. The mean value of all points in the center region is 40.1 pages, 87% of V^*. So we see that the two heuristic procedures used in conjunction will result in an average system value of 87% of the best possible value under the conditions of the experiment.

Other experiments were conducted with similar results. The choice of U_{min} and U_{max} depends on N_p and on the characteristics of the drum being used. For example, Experiment 7 was repeated with $N_p = 50$ and with the same load, but with two 2305 drums in the system rather than one. Under these conditions, with $U_{min} =$ 200 faults/s and $U_{max} = 400$ faults/s, the heuristic procedures produced a mean system value of 92.5% of V^*.

The system scheduler may require that the paging manager provide an estimate of the size of the working page set of each process as it is demoted, to be used in scheduling its next promotion [1,5]. Using the two heuristic procedures above, no precise measurement of working set can be made, since the number of page frames allocated to a process depends on the characteristics of the other processes that are in the MPS at the same time. However, a reasonable measure of the relative sizes of working sets can be made by informing the scheduler, when a process is demoted, of the average number of page frames allocated to it during its stay in

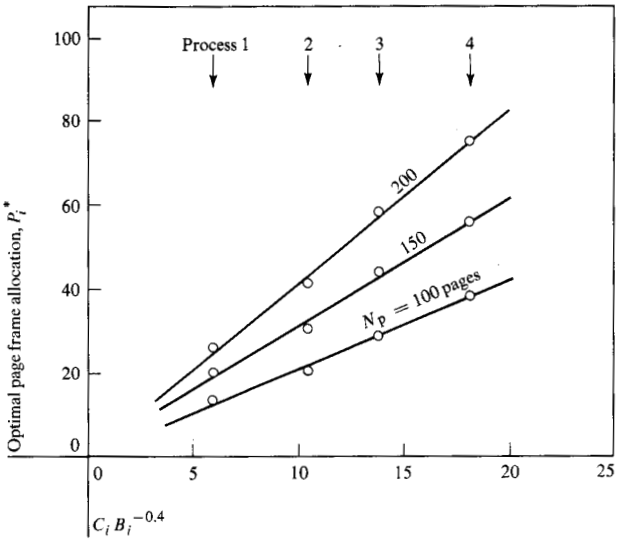


Figure 11 Experiment 6.

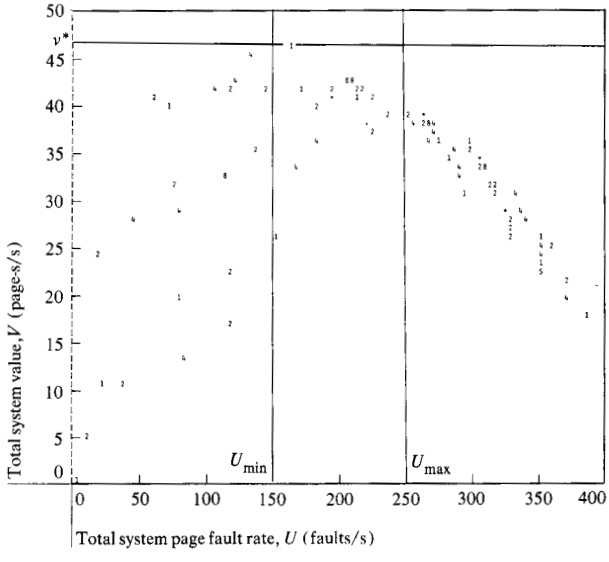


Figure 12 Experiment 7.

Table 3 Optimal page allocations.

N_p	p_1	p_2	p_3	p_4
100	13	20	29	38
150	20	30	44	56
200	26	41	58	75

Table 4 Process characteristics for experiment 7.

i	1	2	3	4	5	6	7	8
B_i (ms)	10	10	50	50	10	10	50	50
C_i (pages)	50	50	10	10	10	10	50	50

the MPS. Heuristic procedure 1, in maintaining the total fault rate within a certain range, tends to stabilize the environment in the MPS. If we assume that a given process encounters approximately the same MPS environment each time it is promoted, the number of page frames allocated to it is a reflection of its working set size.

9. Conclusions

We have proposed an analytic model for the behavior of processes in a non-CPU-bound virtual-memory system, and for the performance of the paging drum. Our model is unusual in that it takes into account the tradeoff between two scarce resources of the system: main memory page frames and paging channel capacity. Combining our definition for value with the model, we have developed methods of measuring the value of a given page-frame allocation, and of finding the optimal allocation of a given number of pages among a given set of processes. We have tested our methods on a simple system containing only two processes, and found the results to be intuitively reasonable. Using the same method with more complex systems, we have observed the following two results:

1. The optimal choice of a multiprogramming set is closely related to the load on the paging channel. If more page frames are added to the system, the optimal MPS grows, tending to stabilize the paging channel load.
2. Among those processes in the MPS, the optimal allocation of page frames is approximately proportional to $C_i B_i^{-0.4}$.

As a convenient, low-overhead means for the paging manager to control the allocation of page frames, we propose that it monitor the page fault rates of all active processes, and employ the following heuristic procedures:

1. Control the size of the MPS in such a way that total paging channel load U is kept between limits $U_{\min}$ and $U_{\max}$, which are defined as system parameters.
2. Allocate page frames among processes in the MPS in such a way that all active processes have the same page fault rate.

These heuristic procedures have been evaluated using the tools described above. Under the conditions of our experiment, they provide an inexpensive means for dynamically tuning a system for near-optimal throughput.

Our results illustrate the general approach of defining a model and solving it analytically for optimum resource allocation, and then evaluating heuristic allocation strategies by comparing their performance with the optimum.

Cited references

1. D. D. Chamberlin, *"A Scheduling Mechanism for Interactive Systems with Virtual Memory," IBM Research Report RC3911*, Yorktown Heights, New York.
2. J. M. McKinney, "A Survey of Analytical Time-Sharing Models," *Computing Surveys* **1,** 105 (1969)
3. L. A. Belady and C. J. Kuehner, "Dynamic Space-Sharing in Computer Systems," *Communications of the ACM* **12,** 282 (1969).
4. L. A. Belady, "A Study of Replacement Algorithms for a Virtual Storage Computer," *IBM Systems Journal* **5,** 78 (1966).
5. P. Denning, "The Working Set Model for Program Behavior," *Communications of the ACM* **11,** 323 (1968).
6. W. J. Doherty, "Scheduling TSS/360 for Responsiveness," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **37,** 97 (1970).
7. T. B. Pinkerton, *Program Behavior and Control in Virtual Storage Computer Systems*, Ph.D. Thesis, University of Michigan, 1968.
8. C. E. Skinner, "A Priority Queuing System with Server-Walking Time," *Operations Research* **15,** 278 (1967).
9. E. G. Coffman, "Analysis of a Drum Input/Output Queue under Scheduled Operation in a Paged Computer System," *Journal of the ACM* **16,** 73 (1969); Errata, *Journal of the ACM* **16,** 646 (1969).
10. *IBM Systems Component Summary: 2305 Fixed Head Storage*, Form GA26-1589-1, IBM Data Processing Division, White Plains, New York.

General references

1. Barbara S. Brawn and Frances G. Gustavson, *Program Behavior in a Paging Environment, IBM Research Report RC2194*, Yorktown Heights, New York.
2. G. H. Fine, C. W. Jackson, and P. V. McIsaac, "Dynamic Program Behavior under Paging," *Proc. ACM 21st Nat. Conf.*, pp. 223–228 (1966).
3. R. W. O'Neill, "Experience Using a Time-Sharing Multiprogramming System with Dynamic Address Relocation Hardware," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **30,** 611 (1967).
4. B. Randell and C. J. Kuehner, "Dynamic Storage Allocation Systems," *Communications of the ACM* **11,** 297 (1968).
5. W. W. Chu and H. Opderbeck, "The Page Fault Frequency Replacement Algorithm," *Proceedings 1972 Fall Joint Computer Conference* **41,** 597–609.

Received April 10, 1973

Dr. Fuller is located at Carnegie-Mellon University, Pittsburgh, Pa. Dr. Chamberlin and Dr. Liu are located at the IBM Research Division Laboratory, Monterey and Cottle Roads, San Jose, California 95114.