# A PAGE ALLOCATION STRATEGY *** FOR MULTIPROGRAMMING SYSTEMS

by

Donald D. Chamberlin*
Samuel H. Fuller**
Leonard Y. Liu*

## INTRODUCTION

In a multiprogramming, virtual-memory computing system, many processes compete simultaneously for system resources, which include CPU's, main memory page frames, and the transmission capacity of the paging drum. (We define a "process" here as a program with its own virtual memory, requiring an allocation of real memory and a CPU in order to execute). This paper studies ways of allocating resources to processes in order to maximize throughput in systems which are not CPU-bound.

As is customary, we define the multiprogramming set" (MPS) as the set of processes eligible for allocation of resources at any given time. Each process in the MPS is allocated a certain number of page frames and allowed to execute, interrupted periodically by page faults. 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 operating system: The Paging Manager and the Scheduler. The function of the Paging Manager is to control the size of the MPS, and to allocate main storage page frames among those processes in the MPS. The function of the Scheduler is to assign time-slice lengths to the various processes, and to define 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 (2).

In order to evaluate various strategies for the resource managers, it was necessary to construct a model of a time-sharing system. Our model is analytical in nature but is not based on queueing theory.

## MODELLING THE USER LOAD

In an earlier paper (1), 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 life-time function has two regions: a concave upward region, followed by a concave downward region, as shown in Figure 1.

*IBM Research Laboratory, San Jose, Calif.

**Departments of Computer Science and Electrical Engineering, Carnegie-Mellon Univ., Pittsburgh, Pa.

***This is an extended abstract of a paper which has been accepted for publication in IBM Journal of Research and Development. The work first appeared as IBM Research Report RC3848, Thomas J. Watson Research Center, Yorktown Heights, New York (May 1972).

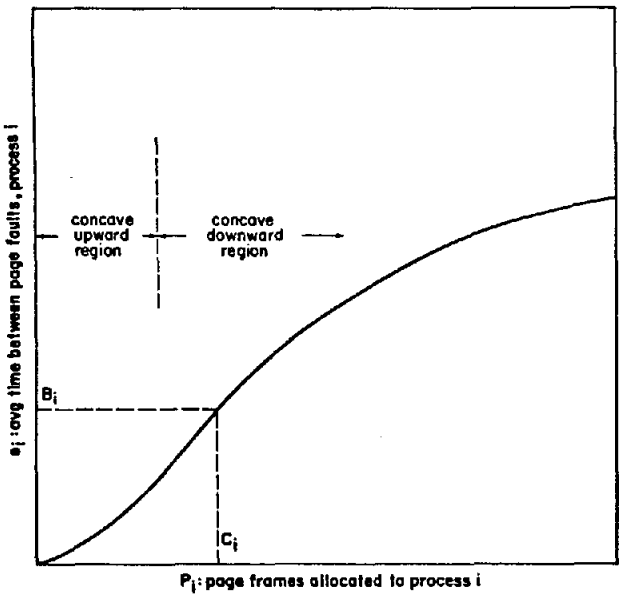


Figure 1

Lifetime Function for Process 1

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 execute efficiently. More precisely, the number of page frames which provides the process with half of its largest possible "lifetime."

$B_i$ (sec) — 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 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 360/67 system, 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. (4)(6)

## MODELLING 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 sectors. When a drum request is made, 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.

Simulation experiments were performed to investigate the relationship between the average time to service a page fetch request (W) and the total load on the paging drum in requests per second (U). A total of $N_M$ processes were assumed to be in the system. Each process was assumed to execute for a random (exponentially distributed) period of time, then make a page request which was placed on one of $N_S$ sector queues with equal probability. The drum was assumed to rotate with period T, servicing one request from each sector in turn. Consistent with our assumption of a non-CPU-bound system, we assumed that no process ever waits for a CPU after its page request has been satisfied. By adjusting the mean execution interval, we varied the total drum load (U) and observed the effect on the average wait time (W).

As expected, the results were dependent on the degree of multiprogramming, $N_M$. When $N_M = 1$, the single process always sees an empty drum queue,

and the average wait time to service a fault is the "no load" wait:

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

$W_{NL}$ consists of one-half revolution average latency, plus one sector-read-time to transmit the page. If there are many processes in the system, W varies from $W_{NL}$ to infinity, depending on the load U. W approaches infinity as U approaches the maximum transmission capacity M of the drum, which is given by:

$$M = \frac{N_S}{T} \qquad (3)$$

Simulation experiments produced a family of curves which give W as a function of U for various values of $N_M$, as shown in Figure 2

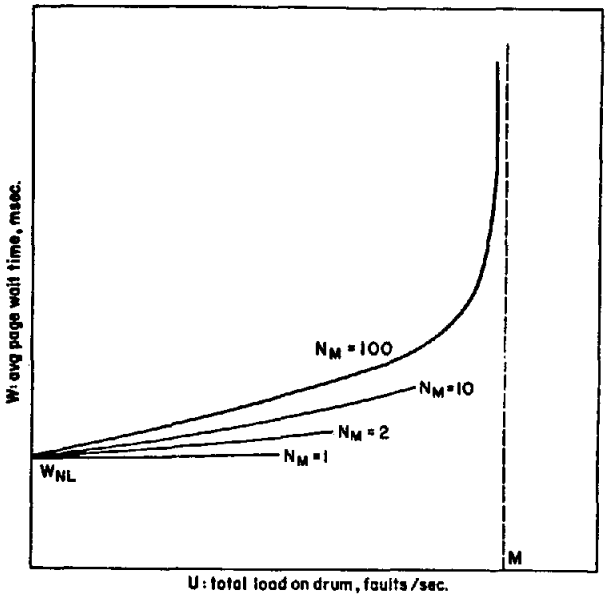


Figure 2

Drum Wait as a Function of Load

The following equation was found to fit the simulation data very closely:

$$W = \frac{2K_M}{M-U} + W_{NL} - \frac{2K_M}{M} \qquad (4)$$

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

$$K_M = \frac{N_M - 1}{N_M} \qquad (5)$$

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

$$U = \sum_{i=1}^{N_M} u_i \qquad (6)$$

where

$$u_i = \frac{1}{e_i + W} \qquad (7)$$

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 process i (in virtual seconds per real second) is given by the equation:

$$r_i = \frac{e_i}{e_i + W} \qquad (8)$$

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 execute 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 ($r_i$):

$$v_i = D_i r_i \qquad (9)$$

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:

$$V = \sum_{i=1}^{N_J} v_i \qquad (10)$$

We are now left with the task of defining the "demand" ($D_i$) of a process. 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.

From equations (1) and (8), we can find the average "cost" in system page-seconds required to give one virtual second to the process, as a function of $p_i$. There is some optimal page frame allocation $p_i^*$ which minimizes the system's cost to execute the process. The "demand" of process i is defined as the actual cost at this minimum point:

$$D_i = \left(\frac{p_i}{r_i}\right)\Bigg|_{p_i = p_i^*} \qquad (11)$$

From equations (1) and (8), $D_i$ may be expressed in terms of the characteristics $B_i$ and $C_i$ of the process:

$$D_i = \frac{C_i}{B_i}\sqrt{W_{NL}^2 + 2B_iW_{NL}} \qquad (12)$$

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 other processes which may be running concurrently.

Equations (4), (6), and (7) can be combined to give an implicit equation for W in terms of the $e_i$'s of the processes. If we know $B_i$, $C_i$, and $P_i$ for all processes, we can find the $e_i$'s from equ. (1) and then solve for W. Once W is known, equ. (8) gives us $r_i$ for each process, and equs. (9) and (10) give us V, the value of the given page allocation. An interactive PL/I program was written to execute the above procedure. Given $B_i$ and $C_i$ for all processes, and $N_S$ and T for the drum, the program can find the value of any given page allocation, or find the optimal allocation of any given number of pages.

EXPERIMENTS

The first experiments done with the program were studies of page allocation in a simple system containing only two processes. The system drum was given parameters T = 10 msec, $N_S$ = 5, corresponding to an IBM 2305 drum with page size = 2K bytes (5). Figures 5 and 6 show equal-value contours on the space of possible page allocations. In Experiment 1 (Figure 3), the two processes were identical: $B_i$ = 20 msec., $C_i$ = 50 pages. In experiment 2 (Figure 4), process 1 was the same as above but process 2 was less demanding ($B_i$ = 20 msec., $C_i$ = 25 pages). The locus of optimal allo-

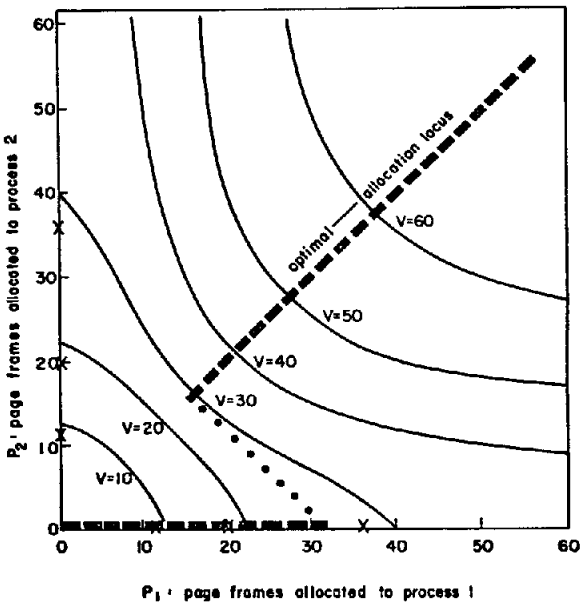


Figure 3: Experiment 1

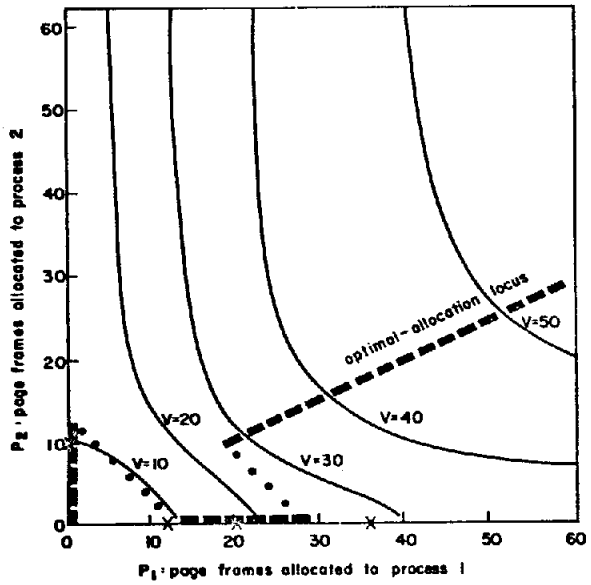


Figure 4: Experiment 2

cations of various numbers of pages is shown as a heavy line. The value contours are discontinuous on the axes (axis intercepts for the contours are represented by X's).

In addition to allocating pages, the Paging Manager must control the size of the multiprogramming set. Experiment 4 gives us insight into the nature of this task. We keep the 2305 drum characteristics as above, and assume the user load consists of many identical processes, all with $B_i$ = 30 msec, $C_i$ = 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 Figure 5.

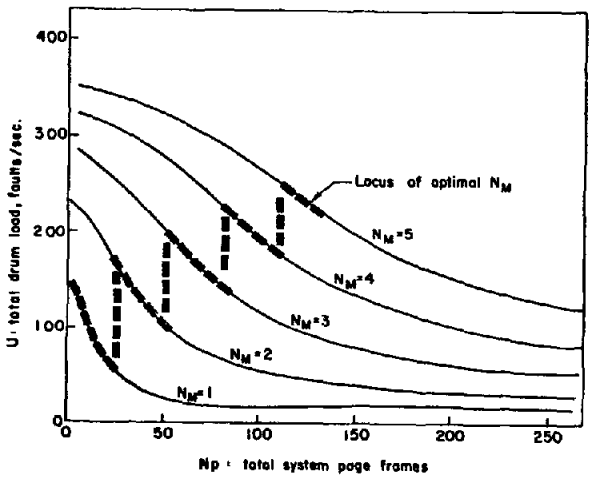


Figure 5: Experiment 4

Superimposed on these curves is a locus which shows the degree of multiprogramming which 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.

In addition to the above experiments, a number of other experiments (described in the full paper) were performed to investigate page allocations in more complex environments of many nonidentical processes. The objective was to find heuristics which would enable the Paging Manager to choose an MPS and allocate pages on the basis of measurements which can be easily made on real systems. The results of our experimentation are the following two heuristics, which depend only on the real-time page-fault rates of the various processes, $u_i$, and the total system page-fault rate U:

Heuristic 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 will be chosen by the system scheduler (2). If $U > U_{max}$, demote the process which has been longest in the MPS. $U_{min}$ and $U_{max}$ are chosen far enough apart that the probability of jumping from $U < U_{min}$ to $U > U_{max}$ is negligible.

Heuristic 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. (Note: Heuristic 2 is similar in concept to the Page Fault Frequency Replacement Algorithm proposed by Chu and Opderbeck (3)).

In order to evaluate the two heuristics, experiments were performed to compare the system value V produced by the heuristics with the best possible value realizable under the same conditions. In Experiment 7, for example, a system with 50 pages and a 2305 drum was loaded with the eight processes described below:

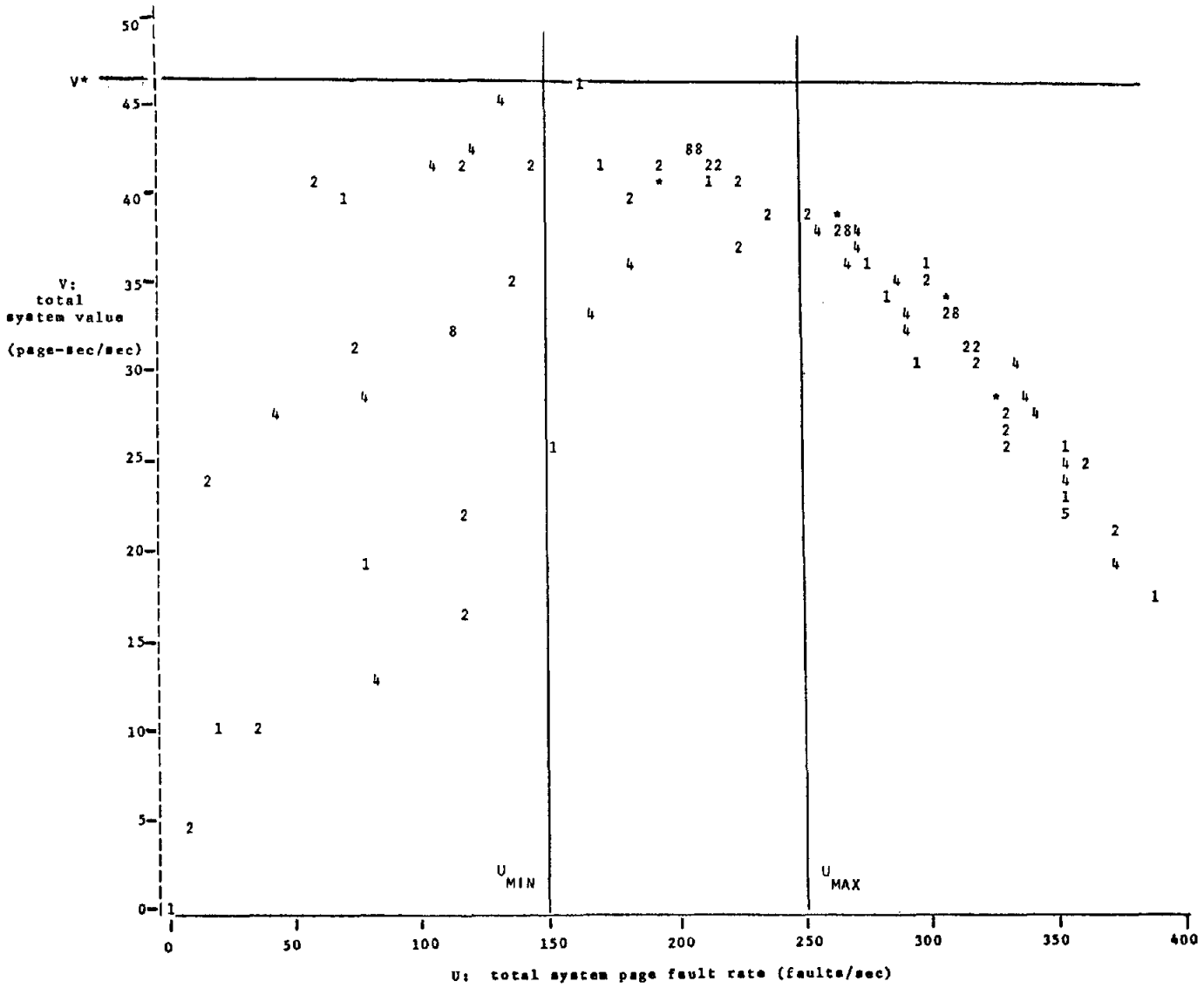


Figure 6: Experiment 7

| Process | $B_i$ (msec.) | $C_i$ (pages) |
|---|---|---|
| 1 | 10 | 50 |
| 2 | 10 | 50 |
| 3 | 50 | 10 |
| 4 | 50 | 10 |
| 5 | 10 | 10 |
| 6 | 10 | 10 |
| 7 | 50 | 50 |
| 8 | 50 | 50 |

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. The next part of the experiment consisted in allocating page frames according to Heuristic 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 found. These measurements are plotted on a U-V plane in Figure 6, in which each point represents a possible MPS. (Note: 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 1 with $U_{min}$ = 150 faults/sec and $U_{max}$ = 250 faults/sec, we are confined to some MPS in 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 heuristics 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 performed with similar results.

## 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 simple systems and found the results to be intuitively reasonable.

We have proposed a pair of simple, low overhead heuristics for page management, and evaluated them by means of the model. Under the conditions of our experiments, the heuristics provide an inexpensive means for dynamically tuning a system for near-optimal throughput.

## REFERENCES


(1) Belady, L. A. and Kuehner, C. J. Dynamic space-sharing in computer systems. Comm. ACM 12,5 (May 1969) pp. 282-288.

(2) Chamberlin, D. D. A scheduling mechanism for interactive systems with virtual memory. IBM Research Report RC 3911, Thomas J. Watson Research Center, Yorktown Heights, N.Y. (June 1972).

(3) Chu, W. W. and Opderbeck, H. The page fault frequency replacement algorithm. Proc. 1972 FJCC, Vol. 41, pp. 597-609.

(4) Doherty, W. J. Scheduling TSS/360 for responsiveness. Proc. 1970 FJCC, Vol. 37, pp. 97-111.

(5) IBM Systems Component Summary: 2305 Fixed Head Storage. IBM Publication No. GA26-1589-1 (June 1970).

(6) Pinkerton, T. B. Program Behavior and Control in Virtual Storage Computer Systems. PhD. Thesis, University of Michigan, 1968.