# A MODEL HIERARCHY FOR VIRTUAL MEMORY SYSTEM INVESTIGATION

Emily C. Placy and Jon C. Strauss

Washington University

## ABSTRACT

A hierarchy of simple models is presented for an idealized version of an IBM VS/370 computer system. The idealized system consists of a main memory, an auxiliary memory, and a processor. Demand paging is employed with an LRU replacement algorithm. A hierarchy of simulation and analytic models is developed which represents the system at several levels of detail. These models are compared and their suitability for representing the system for various purposes is discussed. Several interesting observations concerning the performance of the LRU page replacement algorithm are presented.

## INTRODUCTION

During the course of research into the range of possible effects of non-local process referencing behavior on the thruput of a batch processing virtual memory computer system such as the IBM VS/370, it became necessary to develop performance models of such systems. This paper presents two simulation models which represent the system at different levels of detail. The results of these models are compared and related to a simple analytic model of virtual memory system processor utilization.

The computer system modeled here is an idealized version of the IBM/370 with a virtual storage operating system (1,2). The modeled system has a single processor and a two level memory. The main memory is divided into fixed size page frames. Demand paging is used; i.e., a page is not brought from auxiliary memory until a process attempts to reference it. A reference to a page which is not in main memory is termed a page fault. When there is a page fault and all of the page frames in memory are being used, one of the pages in memory must be replaced. The page replacement algorithm used for this modeled system is global least recently used (LRU) which is a good approximation to the algorithm extant in OS/VS1 (2). A global LRU replacement algorithm removes the page from main memory which has not been referenced for the longest time regardless of which process owns the page.

This work was supported by NSF Grant GJ-33764X.

Several analytic models of virtual memory computer systems have been developed. Gaver (3) presents a probabilistic model for a multiprogramming system which relates I/O speed, core size, and processor speed to the processor and system performance. Coffman and Ryan (4) model the storage requirements of programs in a multiprogramming system for both fixed and dynamic storage partitioning. Oden and Shedler (5) develop a detailed mathematical model of the memory contention in a multiprogramming environment with the LRU replacement algorithm. The analytic model developed in this paper draws on the work of Williams (6).

This paper is organized as follows: a description and comparison of the two simulation models is given first. Next, an analytic model is developed and then compared to one of the simulation models. Finally, some results from the simulation model are presented and discussed.

## SIMULATION MODELS

Two simulation models are described. The first model simulates the system at the individual memory reference level, while the second simulates the system at the active process level.

### MEMORY REFERENCE LEVEL MODEL (MRLM)

For MRLM, a process LRU stack model, as described by Mattson et.al.(7), is employed to generate the time sequence of page references of each process (process reference string). The stack is an ordering of the pages of a process according to the time they were last referenced. The top position of the stack represents the page which was referenced most recently, and the bottom position represents the least recently used page. The process LRU stack model assumes that at each reference the probability of referencing the $j^{th}$ previously referenced page is $p_j$. This $p_j$ corresponds to the $j^{th}$ position in the LRU stack and $p_1+p_2+\ldots+p_n=1$, where n is the number of pages of a program. Thus, if a process has its $m(\leq n)$ most recently

referenced pages in main memory, then the probability P of a reference to a page in memory is

$$P = \sum_{j=1}^{m} p_j ,$$

and the probability of a page fault is (1-P). The use of a process LRU stack makes it unnecessary to keep track of the individual pages of a process.

Spirn and Denning (8) compare working set size and missing page probability for several models of programs with intrinsic locality; one of these models is the LRU stack model. They show that the LRU stack model produces good approximations to the behavior of actual programs. One problem of the LRU stack model, however, is that the stack probabilities are fixed, whereas in practice these probabilities tend to change. Also, processes seem to slowly change localities instead of jumping to a new locality. A second set of probabilities could be introduced to account for the periods when a process is changing localities.

MRLM uses a uniform random number generator to produce the page references of a process. At each reference, the random variate corresponds to the cumulative probability of reference in the process LRU stack. This determines the particular position in the stack of the process currently using the processor and given the number of pages of the process currently in memory, indicates a successful reference or a fault. The use of the stack model eliminates the need to keep track of the specific pages of a process. It is sufficient to know only how many pages a process has in memory.

The model has queues for each sector of the paging device. In a real system, read and write requests are placed into the sector queues according to the location of the respective pages in auxiliary memory. However, since the model does not simulate the individual pages of a process some other means must be used to place requests in the sector queues. The model uses the time of the page fault modulo the number of sectors to determine in which queue the read request is placed. Since the model employs LRU replacement it is necessary to know the time when the pages were last referenced. These times are stored in the process LRU stacks. These stored reference times are also used to determine in which queue to place a write request. This technique distributes read and write requests somewhat evenly among the sector queues.

In the current models all replaced pages are written to the auxiliary memory. In VS/370 (2), and most other virtual storage operating systems, care is taken to only write pages that have been altered during their residence in primary memory. The statistical effect of this feature could be easily added to the models to validate their predictions to actual measured behavior.

When a process has a page fault, a request to read a page from auxiliary memory is placed in the appropriate sector queue. When a page fault occurs and if no free page frames are in memory, a request to write the least recently used page of all processes in main memory is issued. Read requests within a queue are serviced first-come-first-served. A write request is serviced only if there are no read requests in the queue or if the number of free page frames in main memory is less than a threshold value. The threshold is used to help minimize waiting due to sparsely filled sector queues. The service discipline used for the sector queues is round robin.

ACTIVE PROCESS LEVEL MODEL (APLM)

APLM is also based on the process LRU stack and employs most of the page replacement structure of MRLM. However, instead of generating a process reference string, APLM computes the average number of references to pages in memory followed by one reference to a page not in memory, using the stack probabilities. This quantity is given in equation (1) for a process that runs to a page fault without interruption:

$$\bar{n}_i = \sum_{n=0}^{\infty} n\, P_i^{n-1}(1-P_i) = \frac{1}{1-P_i}, \qquad (1)$$

where $P_i$ is the sum of the stack probabilities for the pages process i has in memory.

For the case of the pre-emptive priority CPU scheduling characteristic of IBM OS Systems, equation (1) is only valid for the highest priority process since all other processes might not run until a fault.

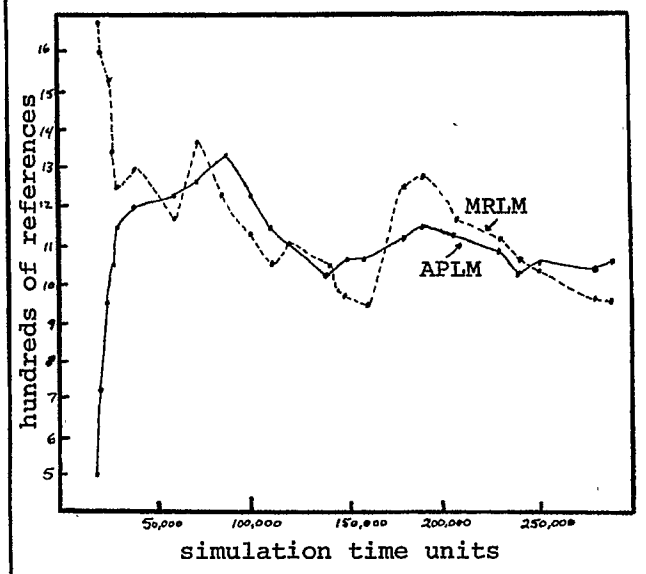
COMPARISON OF MRLM AND APLM

Since both MRLM and the APLM have the same theoretical basis, the two simulation models should on the average have the same number of memory references before a fault for the highest priority process. It is desirable that the two models compare favorably since the second simulator runs much faster than the first. Illustration 1 compares the average number of references before a fault for the highest priority process of the two models. Each time the process faults, the number of references it made before the fault is added to the total number of references of the process. This number is divided by the total number of times the process has used the processor and is the average plotted in Illustration 1. Due to different starting conditions, the initial number of references to pages in memory is much higher

for MRLM than for APLM.

ILLUSTRATION 1

Average Number of References




The probabilistic bases for the two models would suggest that their instantaneous number of successful references could be very much different. However, the average of the number of references should be close as is shown in Illustration 1.

## ANALYTIC MODEL

A simple analytic model of the expected processor utilization of the system is employed to verify the behavior of the simulation models for simple cases.

A system with two processes which have the same successful reference probability P is investigated. 1-P is the probabability of a fault and $\eta\beta$ is the time to transfer a page between the two memories, where $\eta$ is the number of times the main memory is accessed while a page is being transferred from auxiliary memory and $\beta$ is the cycle time of the computer. Williams (8) establishes that the expected processor utilization for this system is given by

$$E(U) = \frac{P}{P^{\eta}+\eta(1-P)} . \qquad (2)$$

It can similarly be shown that if the two processes have different successful reference probabilities, $P_1$ and $P_2$, then the expected processor utilization is

$$E(U) = \frac{P_1(1-P_2)+P_2(1-P_1)}{2\eta(1-P_1)(1-P_2)+P_1\eta(1-P_2)+P_2\eta(1-P_1)} .$$

For $P_1 = P_2$ this equation reduces to equation (2).

There are three differences between the analytic model of Williams and the APLM. First, Williams assumes that the read and write requests which result from a page fault may be serviced in parallel. The simulation model services all requests serially. For comparison of the two models $\eta$ is redefined as the time required to write a page from main memory and to read a page from auxiliary memory. Second, a process uses the CPU for P/(1-P) references in the analytic model and 1/(1-P) references in the simulation model. Upon adding this modification, E(U) becomes

$$E(U) = \frac{1}{P^{\eta}+\eta(1-P)} .$$

Third, the analytic model assumes that the service of a page request may begin at any unit of time while in the simulation model service must begin at a unit of time which is a multiple of $\eta/2$. Because of this difference E(U), for the analytic model is slightly higher than what it should be.

Illustration 2 shows E(U) for the analytic model and the APLM. One reason that the results for the APLM are higher than those for the analytic model is that the number of references for the APLM is constant rather than varying with an average of $\frac{1}{1-P}$.

ILLUSTRATION 2

E(U) for the Active Process Level Model and for the Analytic Model

| $\eta$ | P=.99 | P=.999 | P=.9999 | |
|---|---|---|---|---|
| 200 | .4949 | .9900 | .9999 | APLM |
| | .4639 | .9807 | .9997 | Analytic Model |
| 2000 | .0495 | .4995 | .9900 | APLM |
| | .0495 | .4679 | .9815 | Analytic Model |
| 20,000 | .0050 | .0500 | .4996 | APLM |
| | .0050 | .0500 | .4682 | Analytic Model |

## EXPERIMENTAL RESULTS

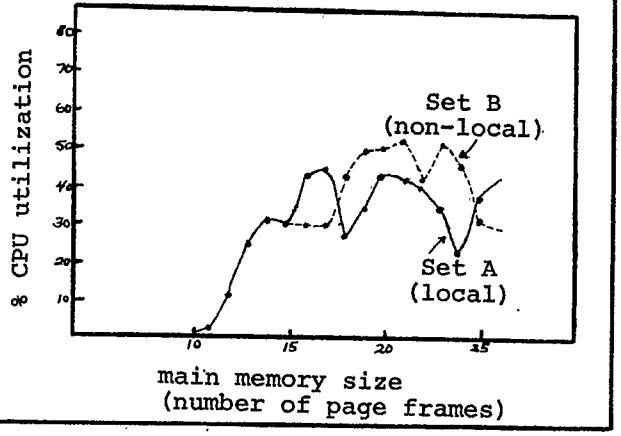
It is demonstrated that a global LRU replacement algorithm is a good replacement algorithm for programs of different locality.

The successful use of virtual memory memory systems implies that only a fraction of a program need be resident in main memory at any one time. This property of programs has been termed the principle of locality by Denning (9). Intuitively, a local program references a subset of its pages for a period of time moving slowly to new subsets of pages. All available experimental evidence corroborates that it is an exception for a program not to satisfy the principle of locality.

It has been established using APLM that when one process is replaced by a process which is relatively less local than the others, the performance of the other processes generally improves. When the process LRU stack model is used, the locality of a process can in some sense be measured by computing the second moment of the stack probabilities. Two sets of processes are used in the experimental study, and the results are presented in Illustration 3. Set A consists of five equally local processes ($\bar{n}$ for three pages of the process LRU stack = 1490). Set B consists of four processes from Set A plus one less local process ($\bar{n}$ for three pages of the process LRU stack = 59).

ILLUSTRATION 3

The Effect of a Less Local Process on the CPU Utilization of Local Processes




Pre-emptive priority scheduling is used and the less local process has the highest priority. The curves in Illustration 3 are the CPU utilizations of the four lowest priority processes of both process sets as a function of the number of page frames in main memory. The solid curve corresponds to Set A and the dashed curve to Set B. The points on Illustration 3 were obtained by running the simulations for an equal number of references. Due to the probabilistic basis of the model, the individual points are from a range of possible values and cannot be compared directly on the sketched curves for Set A and Set B. Rather, the individual points are characteristic of general behavior of CPU utilization as a function of memory size. The curves are however, indicative of general trends and Illustration 3 does indicate improved performance for the four local processes when run with a higher priority, less local process. One might suspect that the less local process would take an unfair share of the page frames in memory and degrade the performance of the other processes. However, Illustration 3 indicates that the less local process causes more memory to be available for the other processes and their performance improves as a result. This is because when using global LRU page replacement if a process does not reference all of its pages when it is active, the unreferenced pages become prime candidates for paging out while other processes are active or when it again becomes active. Hence, the less local process is taking pages from itself rather than from the other processes.

These models can also be employed to demonstrate thrashing. Thrashing is defined as system performance degradation or collapse due to too much paging and is discussed by Denning (10). Thrashing is related to program behavior, paging algorithms, and the mismatch of access times for main memory and auxiliary memory.

The principle of locality leads to the concept of a working set of a program (11). A working set is defined intuitively as the smallest subset of pages which a program must have in main memory in order to run efficiently; i.e., execute with few page faults. Hence, if the main memory of a multiprogrammed virtual memory computer is large enough to accommodate the working sets of the programs being run, the system performance, as measured possibly by processor utilization, will be high. Denning (10) has shown that when the size of the working sets of the programs being run equals the size of memory, the addition of just one more program will induce thrashing. The main cause of thrashing is the small ratio of main memory access time to the time required to copy a page from auxiliary memory to main memory.

For third generation computers this ratio is typically in the range $10^{-3}$ to $10^{-4}$.

Thrashing requires that the memory be over-committed. Thus, it seems likely that a program which becomes much less local and requires a larger working set could induce thrashing as well. To increase this program's working set, pages would have to be taken from the working sets of other programs. And if no program has a working set in main memory, very little useful computing would be accomplished.

ILLUSTRATION 4

CPU Utilization for Two Sets of Processes

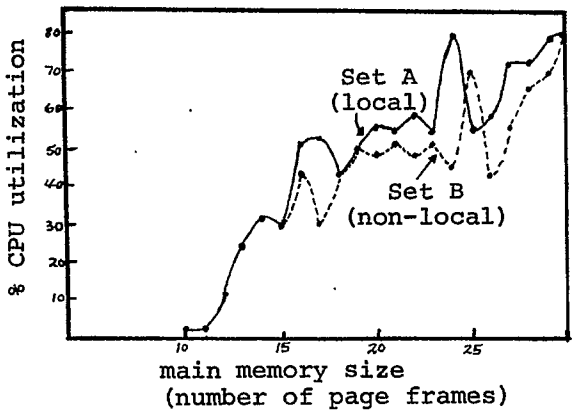



Illustration 4 shows CPU utilization for two different sets of processes for APLM. Set A consists of five processes with the same set of stack probabilities. The stack probabilities are defined such that a process in Set A requires three pages of memory to do a useful amount of computing before a fault. Set B has four processes identical to those of Set A along with another less local process which has the highest priority. Preemptive priority scheduling is used for both sets of processes. For three pages resident, the n for the local process is 1490 and the n for the less local process is 59. $\eta = 1000$. For Set A, 15 pages of memory are required for useful computing and thrashing occurs for a memory of less than 15 pages. Since thrashing occurs with 15 or less pages for Set B as well, the addition of non-local process did not cause thrashing to occur any sooner. However, the CPU utilization is generally lower for Set B than for Set A. For a memory of 10 to 15 pages, the processor utilization is the same for both sets of processes.

As in Illustration 3, the detailed location of points in Illustration 4 is not significant. The lengths of the simulation runs plotted in Illustration 4 were the same. Since the processor utilization is influenced by activity at the time it is measured, the curves should be considered as points in envelopes of possible values of the CPU utilization.

## CONCLUSIONS

This paper has presented and discussed several models of a virtual memory system. The two simulation models were shown through experiments to compare reasonably well. The analytic model and the active process level simulation model also give similar results. This simple active process level simulation model can be used to study the effect of processes and their stack probabilities on system performance and to study the effect of one process on others.

This paper also provides some justification for the use of the global LRU page replacement algorithm and shows experimentally how thrashing can occur.

## REFERENCES


1. Scherr, A.L., "The Design of IBM OS/VS2 Release 2", AFIPS Conference Proceedings, Vol. 42, 1973 NCC, AFIPS Press, Montvale, New Jersey, 387-394.
2. Wheeler, T.F., Jr., "IBM OS/VS1 - An Evolutionaly Growth System", AFIPS Conference Proceedings, Vol. 42, 1973 NCC, AFIPS Press, Montvale, New Jersey, 395-400.
3. Gaver, D.P., Jr., "Probability Models for Multiprogramming Computer Systems", JACM, Vol. 14, No. 3 (July 1967), 423-438.
4. Coffman, E.G., Jr., and Ryan, T.A., Jr., "A Study of Storage Partitioning Using a Mathematical Model of Locality", Communications of the ACM, Vol. 15, No. 3 (March 1972), 185-190.
5. Oden, P.H., and Shedler, G.S., "A Model of Memory Contention in a Paging Machine", Communications of the ACM, Vol. 15, No. 8 (August 1972), 761-771.
6. Williams, J.G., "Asymmetric Memory Hierarchies", Communications of the ACM, Vol. 16, No. 4 (April 1973), 213-222.
7. Mattson, R.L., Grecsei, J., Slutz, D.R., and Traiger, I.W., "Evaluation Techniques for Storage Hierarchies", IBM Systems Journal, Vol. 9, No. 2 (1970), 78-117.
8. Spirn, J.R., and Denning, P.J., "Experiments with Program Locality", AFIPS Conference Proceedings, Vol. 41, 1972 FJCC, AFIPS Press, Montvale, New Jersey, 611-621.
9. Denning, P.J., "Virtual Memory",

Computing Surveys, Vol. 2, No. 3 (September 1970), 153-190.
10. Denning, P.J., "Thrashing: Its Causes and Prevention", AFIPS Conference Proceedings, Vol. 33, 1968 FJCC, AFIPS Press, Montvale, New Jersdy, 915-922.
11. Denning, P.J., and Schwartz, S.C., "Properties of the Working-Set Model", Communications of the ACM, Vol. 15, No. 3 (March 1972), 191-198.