Analysis of Memory Management Strategies for Multiprogrammed Virtual Storage Systems

TAKASHI MASUDA*

Performance of multiprogrammed memory management strategies for virtual storage systems is extensively investigated. Detailed simulation models are developed for this purpose. In particular, the working set strategy, which is a representative variable partitioning strategy, is analyzed in the multiprogramming environment approximating the operating conditions of actual systems. Practical usage problems inherent to the strategy are discussed. Investigated are the effect of window size, the performance of the on-demand paging policy compared with that of the pre-loading policy, the effectiveness of controlling the maximum number of pages allocated to a task, etc. In this analysis, the effect of locality set transfers in user programs is evaluated. The local LRU strategy, which is a representative fixed partitioning strategy, is also analyzed for comparison with the working set strategy.

1. Introduction

In virtual storage systems, memory management strategies have a critical effect on system performance. Programs tend to reference pages unequally and cluster references to certain pages in short time intervals. Therefore, they can be run efficiently in memory spaces considerably smaller than the program size. These properties depend on the tendency of program locality references. It is known that excessive page faults occur during program execution time when the locality set is not loaded into the main memory. Consequently, those memory management strategies are desirable that estimate a program's locality set at any time of execution and assign main memory pages to the program so as to load the locality set. And the multiprogramming degree should be decided on the basis of the locality set sizes of active programs.

A number of memory management models have been proposed and analyzed [3]. However, only a few models and analyses have proved feasible for practical use. This is partly because, as stated in [7], the mathematical models do not usually formulate actual systems and program paging behaviors in enough detail to be of practical use. In addition, a few reports ([3], [6]) have analyzed memory management strategies in a multiprogrammed environment; although many reports have analyzed such strategies and program paging behavior in a single program environment.

Memory management strategies can be grouped with respect to the two basic strategies of partitioning storage: fixed partitioning and variable partitioning. In fixed partitioning strategies, a fixed number of page frames is allocated to each active task; while in variable partitioning strategies, the number of allocated page frames for each active task varies during program execution. Coffman and Ryan [1] used the simple statistical model of locality variations to compare these two storage partitioning methods. They concluded that the total memory size required for the variable partitioning strategies is around 30 percent less than that required by the fixed partitioning strategies for a given performance level when the variation in working set sizes is relatively large.

In this paper, we compare performances of these memory management strategies by using a simulation technique and examine practical usage problems inherent to the strategies. A detailed simulation model of the multiprogrammed memory management has been developed for a time-sharing environment. The effects of locality set transfer on total system performance are also discussed.

2. Memory Management Strategies to be Analyzed

In order for a program to be executed efficiently, that is, without excessive page faults, the locality set of the program must be loaded. It is not useful to assign more memory spaces to a program than the locality set size, and it causes excessive page faults to assign less memory spaces to a program than the locality set size. Even if the page replacement algorithm is adequate, the system operates inefficiently when the memory assignment policy is not adequate.

The locality reference properties of programs show that variable partitioning strategies are generally superior to fixed partitioning strategies because of the locality set size variations. However, the locality properties of each program are not known to the operating system before program execution, and so the memory management program must estimate the locality set of each program from the past behavior of the program.

*Institute of Information Sciences and Electronics, University of Tsukuba.

The memory management strategies can be classified as follows:

(1) Fixed partitioning strategy~The main memory is partitioned into the fixed number of areas, and each active program is assigned a partitioned area.

(2) Variable partitioning strategy without explicit correlation to the locality properties of active programs~ Though the number of pages assigned to each active program varies during program execution, they are not explicitly correlated to the locality references of active programs.

(3) Variable partitioning strategy with explicit correlation to the locality properties of active programs~ The memory management strategy estimates the locality set size of active programs, and main memory pages are assigned dynamically during program execution according to the locality set size of each active program.

There are a number of memory management strategies which are classified into the above three kinds of strategies. Some typical strategies which deserve consideration are as follows:

(1) Local LRU strategy~This is a typical fixed partitioning strategy. When a page fault occurs and a page replacement is needed, the least recently used selection is made from pages belonging to the task which generated the page fault.

(2) Global LRU strategy~This is a typical variable partitioning strategy with no explicit correlation to the locality properties of active programs. When a page replacement is needed, the replaced page is the one that has not been referenced for the longest period of real time, regardless of the task to which it belongs.

(3) Working set strategy~This is a variable partitioning strategy which has explicit correlation to the locality properties of active programs. The working set $W(t, T)$ at a given time t is the set of distinct pages referenced in the virtual time interval $(t-T+1, t)$; that is, the set of pages referenced during the last T instructions where T is called the window size parameter. The working set size $w(t, T)$ is the number of pages in $W(t, T)$. The working set strategy uses the working set as an estimator of a locality set and keeps in the main memory those pages of active tasks which are included in their working sets. The tasks are activated to the maximum degree under the condition that their working sets can be loaded into the main memory.

(4) PFF (Page Fault Frequency) strategy [6]~This is a modification of the working set strategy. Window size varies for each active program according to the page fault rate of the program in the PFF strategy, while it is the same for all programs in the working set strategy. In the PFF strategy, pages are allocated to keep the paging rate constant for all active programs.

A representative fixed partitioning strategy and variable partitioning strategy are selected for our simulation analysis. As a fixed partitioning strategy, the local LRU strategy is employed. The effect of memory partitioning, which determines the multiprogramming degree, is investigated.

The global LRU strategy and its modification have been employed in many real systems. However, there are two weak points in this strategy. The first is that it is difficult to control the multiprogramming degree adequately. Thrashing tends to occur when the sum of locality set sizes of active programs exceeds the main memory size. The second point is that the LRU property of program behavior is assured only in a single program environment. In the multiprogrammed environment, the global LRU property of programs is affected by the task scheduling policy. There is no guarantee that the least recently used page will be reclaimed with the least probability in the multiprogrammed environment.

In the working set strategy, the working set is used as an estimate of a locality set, and those pages of active tasks which have been referenced during the window size T are resident in the main memory. Therefore, it provides the capability to adapt to changes in the working set and its size. Another distinguishing feature of the working set strategy is that multiprogramming degree control is inherent in the definition of the working set strategy. The multiprogramming degree varies according to the working set sizes of active programs. Program activation and inactivation decisions must satisfy the condition that locality set estimates of all active programs are resident. Therefore, the probability of thrashing can be kept low by selecting a suitable window size.

The PFF strategy is a variant to the working set strategy. The page fault rate of each active task, instead of window size, is used to control program activation and inactivation decisions. Pages are allocated to active tasks so as to keep the page fault rate of each active task at a predetermined value. In the PFF strategy, some active tasks with poor locality of references tend to occupy more page frames than in the working set strategy. For instance, when a task requests new data pages at a certain rate greater than the predetermined control value of the PFF strategy, the number of pages allocated to the task increases monotonously. In the working set strategy, the maximum number of page frames allocated to an active task is limited by the window size. The working set strategy will be preferable to the PFF strategy, especially when quick response is requested for tasks with good locality references, as in the time-sharing system.

For these reasons, the working set strategy is employed as a representative variable partitioning strategy in our simulation analysis. The variable partitioning strategy is particularly useful in a batch system or in a time-sharing system, where the page reference characteristics vary greatly among jobs. In a time-sharing system, in particular, the cpu time needed per interaction is less than a few tens of milliseconds in most interactions; and the dynamic program behavior changes greatly in a short time. Thus, the capability of memory allocation in units of one page and the dynamic control of multiprogramming degree, which are the advantages

of virtual storage systems, can be used quite effectively. Therefore, a time-sharing system is employed as an environment of our simulation analysis. Most of the results obtained, of course, are useful for the memory managements of other kinds of systems.

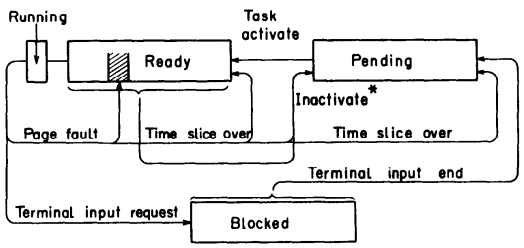
3. Simulation Model

3.1 Task States and Transitions

Task states and transitions are shown in Fig. 1. Since memory management strategies are to be analyzed, only the terminals and paging devices are modeled as peripheral devices. There are four states: running, ready, pending and blocked. In the working set strategy, another state, "pre-loading", exists when the pre-loading policy is adopted. In the pre-loading policy, a task which is to be activated stays in the pre-loading state during loading of the working set. The ready queue includes those tasks which are waiting for cpu service and are in the paging state. The tasks in the pending queue are waiting for promotion to ready states due to the congestion of the multiprogramming degree. The running, ready and pre-loading tasks are called active tasks. Both the pending and blocked tasks are called inactive tasks.

For the local LRU strategy, tasks are activated to keep the number of active tasks at the maximum degree within the number of the main memory partitions. The activation candidate task is taken from the top of the pending queue. Tasks are inactivated when the time slice is over and when a terminal input request occurs.

For the working set strategy, tasks are activated to keep the number of active tasks at the maximum degree under the condition that the sum of working set sizes of active tasks does not exceed the main memory size. The activation policy of the on-demand paging strategy, in which every page is loaded individually into the main memory on demand, differs from that of the pre-loading strategy, in which the working set is loaded when the task is activated. In the pre-loading strategy, a task is activated when the number of free pages is greater than the working set size of the activation candidate task. The working set size is evaluated as the number of referenced pages in the window size T of the immediate past virtual time. In the on-demand paging strategy, the activation policy is more complicated. In this case, even if the number of free pages is greater than the working set size of the activation candidate task, some active tasks may not yet have been assigned working sets in the main memory. The task can be activated when the number of main memory page frames minus the sum of the working set sizes of the active tasks is greater than the working set size of the activation candidate task.



* only in case of working-set strategy

Fig. 1 Task states and transitions.

As for the task inactivation, the transition of a ready task to the top of the pending queue occurs only in the working set strategy. When all main memory page frames are occupied by the working sets of active tasks and the running task requests an allocation of a free page frame, the ready task most recently activated is inactivated. It is then placed at the top of the pending queue to supply pages to be paged out.

Those pages which have never been changed during their lifetime in the main memory do not need to be actually transferred to the secondary memory in case of page out. The probability that a page is changed in the main memory is reported to be about 33% in the steady state measurements made at the computer center of the University of Tokyo [4]. That is, 67% of paged-out candidate pages are not actually transferred. Since it is expected that the value does not vary so greatly in each environment, this value is used in our simulation model. In the case of the pre-loading strategy, however, all paged-out candidate pages are actually transferred in order to load the whole working set from the contiguous area of the secondary memory at the next loading time.

3.2 Model of User Program Behavior

3.2.1 Paging Behavior Model

The SLRUM (Simple LRU Stack Model) [8] is employed as the paging behavior model for user programs. This model can be used, as seen from the definition, only in the range in which page references of a program are stable. The locality transfer of a program needs another model, which is described later. Five programs, which operated under the HITAC 8700/8800 virtual operating system [5], were selected as model programs, and the execution processes of these programs were traced interpretively. Then the stack distance probabilities were calculated within the range where the page reference patterns are stable. Fig. 2 shows the stack distance probabilities for these programs.

The execution steps between page faults are calculated from the SLRU stack probabilities when a model program is executed in a given memory size [6]. When the most recently referenced j pages of a program exist in the main memory, the probability Q_j that the next memory reference will cause a page fault is given by

$$Q_j = \sum_{i=j+1}^{p} q_i + q_0$$

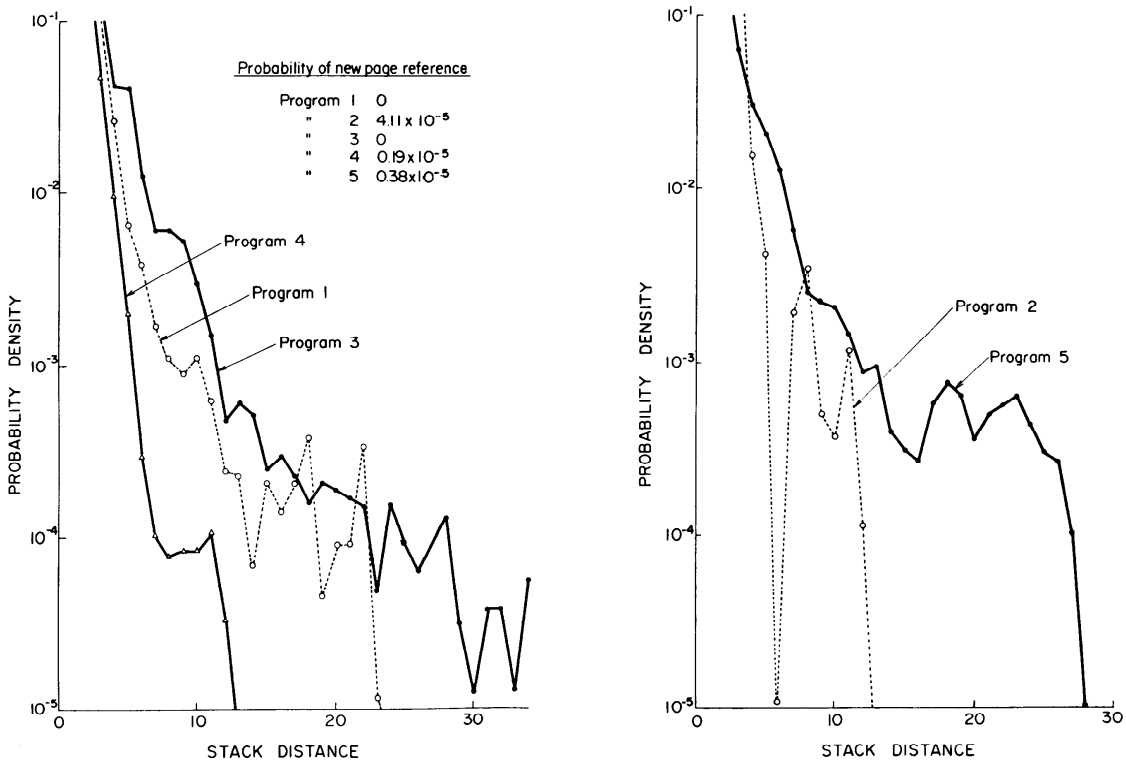


Fig. 2 Stack distance probabilities of the model programs.

where q_i is the i-th element of the SLRU stack, p is the maximum stack depth, and q_0 is the probability that pages not included in the LRU stack will be referenced. Then the number of memory references r between page faults is calculated as a sample from a geometric distribution with the mean $1/Q_j$. Then r can be found by

$$\frac{\log \alpha}{\log (1-Q_j)} - 1 \leq r < \frac{\log \alpha}{\log (1-Q_j)}$$

where α is randomly sampled from a uniform distribution [0, 1]. The execution steps between page faults can be obtained by multiplying r by a constant, which is the ratio of the number of instruction memory references to the total number of memory references.

3.2.2 Locality Transfer Model

As stated above, the SLRUM cannot express the locality set transfer of a program. In a time-sharing system, programs tend to use only short cpu times in each interaction, and the transfer rate of the locality set and the variation in the locality set sizes will be large. The locality set transfer is modeled as described hereafter.

Each SLRUM consists of a set of stack distance probabilities and represents page references in one locality set during program execution. This is called program phase. A program is modeled by combining a series of program phases and the respective execution times. During program execution, when a phase change occurs, the related page requests take on the stack distance probabilities of a new program phase. Those pages referenced by previous program phases are kept in the main memory as long as they are in the working set of window size T in the working set strategy, and until replaced by other pages in the local LRU strategy.

3.2.3 CPU Time Usage Model

It is important to obtain the cpu time usage distribution which each user program uses during one interaction. An interaction is defined as an interval between the time when a user finishes an input line and the time when the user's program requires input again. The cpu time usage model which was measured at the computer center of the University of Tokyo is used in the simulation model. The original measurements were slightly modified to fit the purpose of the simulation. For instance, those interactions which spend more cpu time than 5 seconds are excluded, because the effect will be small for the evaluation of memory management strategies and also the simulation time is limited. The cpu time usage distribution per interaction is shown in Fig. 3, when the mean processing time per instruction

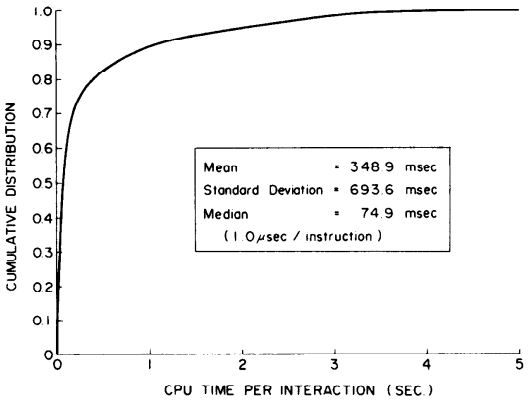


Fig. 3 CPU usage time distribution per interaction.

is adjusted to be 1.0 μsec.

3.3 User Behavior Model

The think time distribution of the above system is used as a user behavior model at terminals. The average is 26.5 sec. the standard deviation is 38.6 sec. and the median is 10.0 sec.

3.4 Dynamic Execution Steps in Operating Systems

Most studies to date do not account for the time spent executing the operating system. However, especially in a time-sharing system, the time spent per interaction in the user program is usually short and the paging rate tends to be high. Therefore, the operating system execution time has a great effect on total system performance. The execution steps used in the simulation model are assumed as shown in Table 1.

Table 1 Dynamic execution steps in operating system.

Processing module		Dynamic steps (instructions)
Page fault handling	when free pages exist.	1000
	when no free pages exist.	3000
Task scheduler		200
Interrupt handling from paging drum		1000
State change	running to ready	100
	running to pending	15000
	running to blocked	20000
	ready to running	100
	ready to pending	15000
	pending to ready	200
	pending to pre-loading	15000
	blocked to pending	200
	pre-loading to ready	200
In working set strategy	to decide task activation	200
	to calculate working set size per active task	100

3.5 Simulation Environment

Since the purpose of this work is to analyze the characteristics of memory management strategies, the system resource which has an essential effect on total system performance should be the main memory. After some trial-and-error experiments were performed satisfying this condition, the following environment was employed:

(1) cpu speed ~ 1 μsec/instruction.

(2) Main memory size ~ 80 page frames for user programs.

(3) Paging drums are used as paging devices, which have the following characteristics:

- 10 sectors/band, 4096 bytes/sector.
- mean access time ~ 10.3 msec.
- transfer rate ~ 2 msec/4096 bytes.

(4) Two paging channels are assumed.

(5) The real time intervals of 660 seconds are simulated, and measurements are collected beginning at the point where 60 seconds have passed in the simulation system.

4. Simulation Results

4.1 Local LRU Strategy

In **4.1** and **4.2**, the basic properties of the memory management strategies are found. For this purpose, it is assumed that the locality transfer of model programs does not occur. Each program requests pages according to the set of stack distance probabilities of one of the model programs. The effect of the locality transfer is discussed in **4.3**.

The disadvantage of the local LRU strategy is that the number of page frames allocated to each active task cannot be changed dynamically during execution, and the multiprogramming degree is not controlled. The maximum multiprogramming degree is equal to the number of the main memory partitions, and tasks are activated up to the maximum degree without any consideration of the paging traffic.

Simulation was carried out for the number of partitions one to five. The results are shown in Fig. 4 when the number of terminal users is 60. This number of the allocated page frames for each partition is 80, 40, 26, 20 and 16, respectively; since the main memory size for user programs is assumed to be 80 pages. Fig. 4 shows the average response time; the average multiprogramming degree; the average number of pending tasks; the average inter-page fault intervals (MSBPF); the total number of interactions during the simulated interval (600 seconds); the average channel idle rate; the average ratios of user program running time, operating system running time, paging idle time and pure idle time to the total elapsed time; etc. The numbers in brackets in Fig. 4 show the scale of the vertical axis. The average response time is optimal when the number of partitions is two. When the number of partitions exceeds three, the average response time gets worse quite rapidly.

The number of allocated page frames to each partition is larger than the maximum stack depth of any of the model programs when the number of partitions is equal to one or two. When the main memory is divided

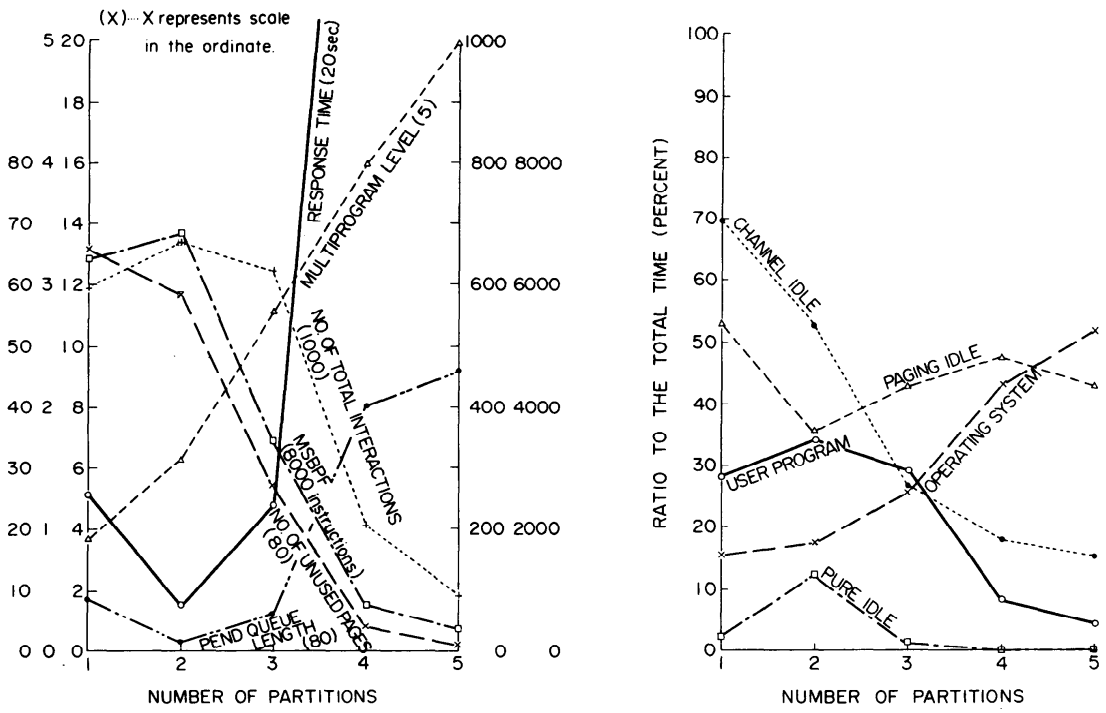


Fig. 4 Effect of the number of partitions on the system performance in the local LRU strategy
~{Main memory size: 80 page frames
{Number of users: 60 users

into three partitions, the maximum stack depths of programs 3 and 5 are greater than the number of allocated page frames, which is 26, and the replaced pages will be reclaimed again. When the number of partitions is equal to four and five, the maximum stack depth of program 1 also exceeds the number of allocated page frames. In this case, the inter-page fault intervals decrease rapidly, the time needed for page fault handling increases rapidly and the responsiveness gets worse rapidly. This is the "thrashing" phenomenon. When the number of partitions is one, the paging idle rate increases.

From this analysis, it can be concluded that a small difference of pages allocated to each program has a great effect on total system performance. In the actual system, it will be almost impossible to decide an ideal number of partitions since locality set size differs among users and varies during program execution. The defects of the fixed partitioning strategy have been clarified in Fig. 4.

4.2 Working set Strategy

4.2.1 Effect of Number of Users

The basic properties of the working set strategy are discussed where no locality transfer of user programs occurs. First, the effect of the number of users on system performance is found.

The simulation was executed for 30 to 120 users. As for the user program model, each user program executes one of the five model programs. The window size is assumed to be 100 K instruction steps which can be found sufficiently large to include the locality sets of the model programs. The average working set sizes for this window size are 24, 17, 34, 13 and 28 pages, respectively. Simulation results are shown in Fig. 5. All the pages are loaded on demand. Some advantages of the working set strategy will be found.

As the number of users exceeds 70, the number of free pages converges to a constant value and the main memory becomes the resource which effects system performance critically. This can be seen from the fact that the average pending queue length increases rapidly as the number of users exceeds 70. The reason why more tasks are not activated in this case, although about 45 free pages exist, is that some active tasks have not loaded their whole working sets into the main memory.

As the number of users increases, the ratio of user program running time to the total time increases. When the number of users exceeds 80, the ratios of user program running time, operating system running time, etc. to the total elapsed time converge. When the number of users is large, these ratios do not change and thrashing phenomenon does not occur. In this example, the maximum permissible number of users is between 70 and 80.

4.2.2 Effect of Window Size

Window size is the most important control parameter

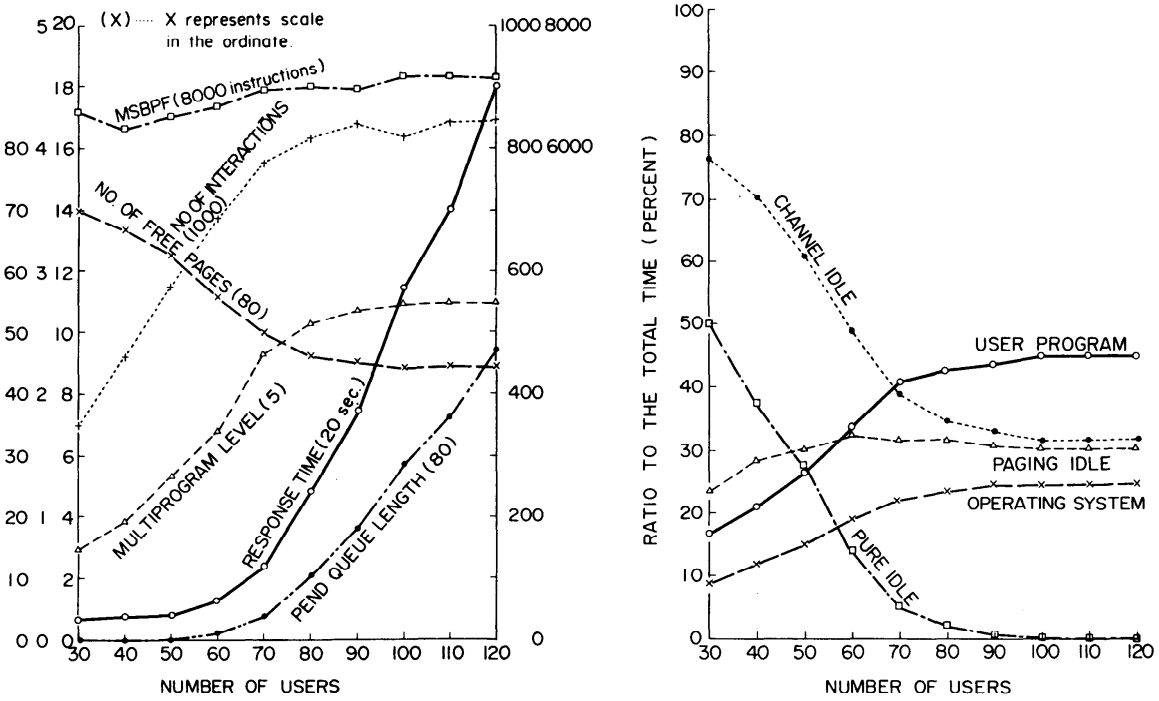


Fig. 5 Performance of the working set strategy with the on-demand paging policy
~{Main memory size: 80 page frames
~{Window size: 100 K instructions

in the working set strategy. When window size is too small, the number of page frames allocated to each active task becomes insufficient, causing the thrashing phenomenon. When the window size is too large, many pages not referenced in the near future reside in the working set and the multiprogramming degree decreases.

When the window size is too large, two kinds of pages with the possibility of not being referenced in the near future will be included in the working set. One type is caused by the properties of locality reference and transfer. When the locality transfer occurs, many pages of the past locality sets will reside in the working set for a long time if the window size is large. The other type is caused when a program requests new data pages with high probability, which become unnecessary in a short time interval.

The effect of window size on average response time is shown in Fig. 6 when the number of users is 80. As window size is decreased to around 10 K instruction steps, response time increases rapidly, and the thrashing phenomenon occurs. When the window size increases to approximately 10^6 instruction steps, then the response time again increases in spite of no locality transfer in user programs. This is because three model programs out of five request pages not contained in LRU stack at fixed probabilities q_0, even after the number of page frames allocated becomes greater than the maximum

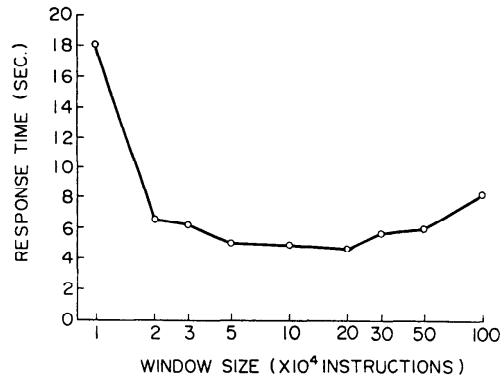


Fig. 6 Effect of window size on responsiveness.
~{Main memory size: 80 page frames
~{Number of users: 80 users

stack depth. In the model program 2, for instance, the average working set size is calculated as 71 pages, when the window size is 10^6 instruction steps. Consequently, when a user task of the model program 2 is active, no other user tasks are activated.

In this example, the responsiveness is satisfactory for window sizes between 30 K and 300 K instruction steps. Any window size will be allowed in this range. It is a great advantage that the range of available window sizes is wide.

4.2.3 Task Activation Criterion

In the on-demand paging policy, as shown in Fig. 5, about 50% of the main memory pages are available even when the system is fully loaded. These free pages are reserved for those active tasks for which the working sets have not wholly been loaded. In a time-sharing system, a considerable amount of interactions goes into the blocked states before loading the whole working set into the main memory, because many interactions use the cpu time less than a few tens of milliseconds as shown in Fig. 3. Consequently, the task activation decision policy described in **3.1** will be too strict in the on-demand paging policy. Therefore, the effect of relaxing the activation decision policy is analyzed. Some notations are defined below:

a set of current active tasks: $A=\{i\}$
the current working set size of a task i: $w_i(t, T)$
the number of page frames currently allocated to a task i: m_i
the activation candidate task: j
the working set size of the task j: $w_j(t, T)$
the main memory size: M pages

The task j is activated if

$$\sum_{i\in A} m_i+\alpha_1\cdot\sum_{i\in A}(w_i(t, T)-m_i)+\alpha_2\cdot w_j(t, T)\leq M$$

Here $0\leq\alpha_1\leq 1$ and $0\leq\alpha_2\leq 1$ are activation control parameters. When $\alpha_1=\alpha_2=1$, this condition is consistent with the one described in **3.1**. $\sum_{i\in A}(w_i(t, T)-m_i)$ expresses the expected number of additional pages needed to load entirely the working sets of the current active tasks into the main memory. The $w_i(t, T)$ is equal to m_i for those active tasks which have already spent more cpu time than the window size T since activated. As α_1 and α_2 are set small, the activation condition is relaxed. The response time is shown in Fig. 7 for various values of α_1 and α_2, when the window size is 100 K instruction steps and the user number is 80. In this case the best response time is found for $\alpha_1=0.6$ and $\alpha_2=0.4$ When $\alpha_1=\alpha_2=0$, every pending task is activated independent of the memory load. Consequently, inactivation of the task to a pending state occurs frequently and the responsiveness is quite poor. The optimal values of α_1 and α_2 should be decided in each system.

4.2.4 Effect of Pre-loading Policy

The effect of pre-loading, where the working set is loaded into the main memory at the task activation time, is analyzed. Page faults which occur after loading the working set are, of course, handled by the on-demand policy.

The advantages of the pre-loading policy are twofold. One is the possibility of improving the utilization rate of paging channels. This is because pages belonging to the working set can be transferred from consecutive sectors by one input request. The other is the possibility of decreasing the paging rate to reduce execution time

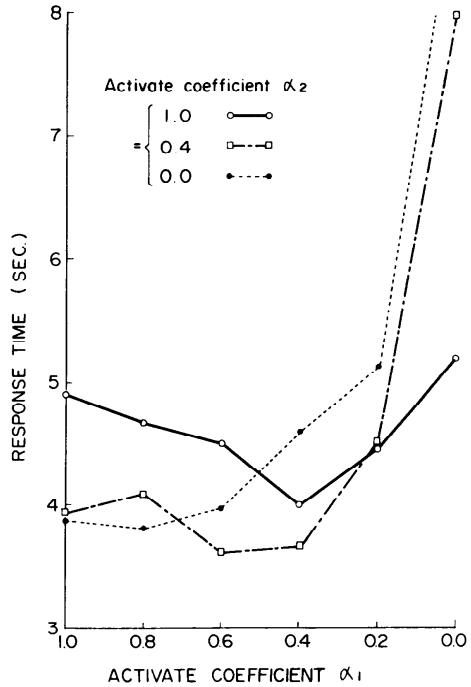


Fig. 7 Effect of the task activation control on responsiveness.
~ {Main memory size: 80 page frames; Number of users: 80 users; Window size: 100 K instructions}

of page fault handling. The disadvantage of pre-loading is that some pages may not be referenced at all after the working set is loaded. Therefore, the pre-loading strategy is advantageous when the page reference patterns of user programs are stable. No locality transfer in user programs is assumed in this section.

Simulation results show that the pre-loading policy is much superior to the on-demand policy in case of no locality transfer. The average response time is 5 sec. for 120 users in the pre-loading policy, while it is 5 sec. for 80 users in the on-demand policy. The mean instruction steps executed between page faults are 70 K steps and 7 K steps, respectively. In the pre-loading policy, the number of free pages is very few when the system is fully loaded. Since the page fault rate is low, both the operating system running time and the paging idle time decrease, and the user program running time increases. From this analysis, it can be concluded that the pre-loading policy is much superior to the on-demand strategy when the locality transfers of user programs are not considered.

4.2.5 Comparison of the Working Set Strategy with the Local LRU Strategy

The basic properties of the working set strategy and the local LRU strategy have been brought out by the foregoing analysis. Now we will compare these two strategies with regard to responsiveness. The result is

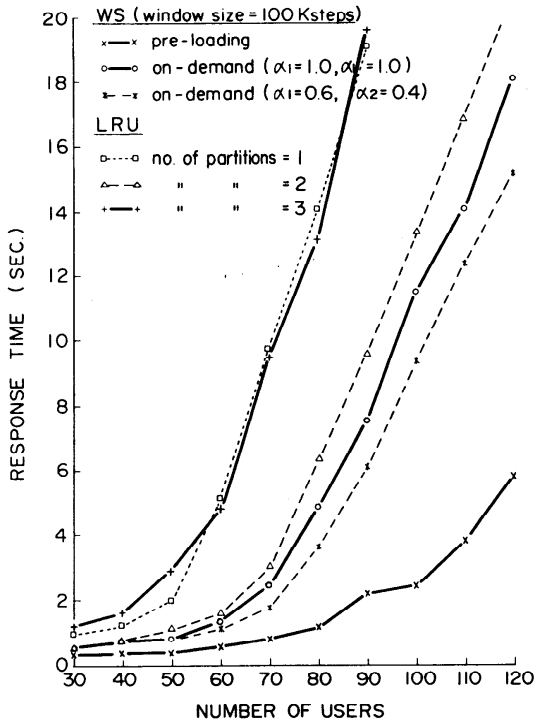


Fig. 8 Responsiveness of the working set strategy and the local LRU strategy.
~Main memory size: 80 page frames

shown in Fig. 8. In the local LRU strategy, the effect of the number of memory partitions on the responsiveness is very large, and the system performance tends to be degraded as variations of working set sizes among user programs increase.

The response time of the working set strategy with the on-demand paging policy is almost always 20% better than that of the local LRU strategy with the partitioning number equal to two. Furthermore, the average response time of the pre-loading policy is much better than that of the on-demand policy, since no locality transfer is assumed for user programs.

4.3 Program Model with Locality Transfer

The effect of locality transfers in user programs is considered. If locality transfers occur very frequently, any memory management strategies, which estimate the locality set, will be useless. In the actual systems, however, memory management strategies estimating the locality set are known to have positive effects on system performance, even if locality transfers occur.

The locality transfer model is assumed to be the model described in **3.2.2**. Each user program executes one of five model program phases during a fixed period and then selects another program phase for execution during the next period. This period is designated as the locality length. The effect of locality length on average response time is shown in Fig. 9, when the number of users is

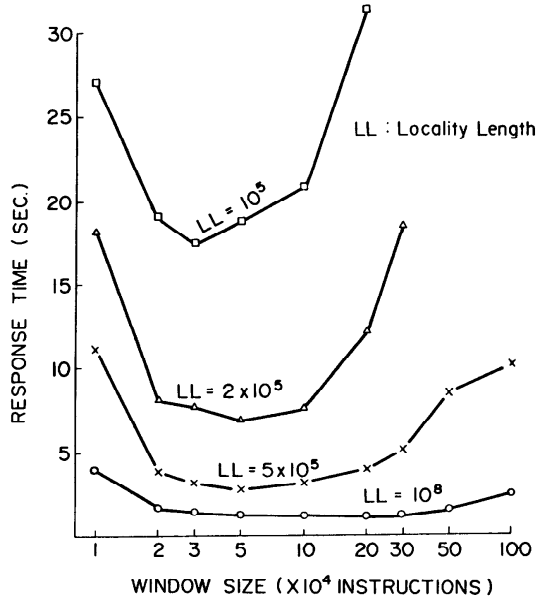


Fig. 9 Effect of the locality transfer rate on responsiveness (on-demand paging strategy).
~{Main memory size: 80 page frames
{Number of users: 60 users

60 and the main memory size is 80 pages. As the locality transfer rate increases, responsiveness deteriorates rapidly and, in particular, becomes more sensitive to changes in window size. This is because when the locality transfer rate is high, many unused pages are included in the working set as window size increases and the multiprogramming degree decreases.

The multiprogramming degree is shown in Fig. 10, when the locality transfer occurs every 10^5 instruction steps and 10^8 instruction steps. When a locality transfer occurs every 10^5 instruction steps, the multiprogramming

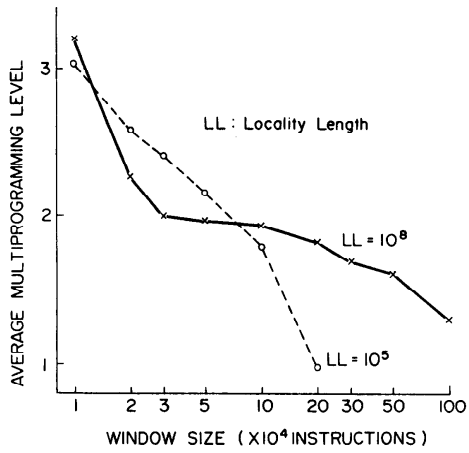


Fig. 10 Effect of the locality transfer rate on the multiprogramming degree (on-demand paging strategy).
~{Main memory size: 80 page frames
{Number of users: 60 users

degree goes down rapidly as window size increases and the paging and channel idle rates increase. Responsiveness deteriorates rapidly as window size increases over 10^5 instruction steps. In comparison, when a locality transfer occurs every 10^8 steps, which means that actually no locality transfer occurs during simulation, the multiprogramming degree decreases gradually as window size increases. As explained earlier, this is because some model program phases request pages not included in their LRU stacks at fixed probabilities q_0, and the working set size increases slowly. The rapid decrease in the multiprogramming degree for window sizes up to 30 K steps shows that about 30 K steps are necessary to reference the whole working set of a program phase.

As shown above, the range of feasible window sizes becomes narrower when locality transfers exist. Window size should be as small as possible, so as not to include unnecessary pages in the working set, but large enough to include the locality set of a program phase. In Fig. 9, the feasible window size range should be between 30 K and 50 K instruction steps.

When locality transfers occur, working set size increases intermittently and many pages belonging to previous phases are included in the working set. To reduce this undesirable effect, it is useful to control the maximum number of pages allocated to a task. This mechanism is also useful for preventing the programs with extremely large working set size from degrading the total system performance. Fig. 11 shows the effect of the maximum page allocation control on responsiveness for the locality transfer occuring every 200 K steps and 500 K steps, when the number of pages allocated to a task is limited to 35 pages. The solid lines indicate where the maximum page control policy is not specified, and the dotted lines where it is specified. Responsiveness improves greatly where the window size is large.

The effectiveness of pre-loading policy will decrease when the locality transfer takes place, since the swapping-in probability for unused pages becomes high. Fig. 12 shows how the locality transfer affects the effectiveness of pre-loading. As locality length decreases, the responsiveness of pre-loading policy degrades as is the case with the on-demand paging strategy. The dotted line which gives the optimal responsiveness is the result of adopting both the maximum page control policy and the pre-loading policy.

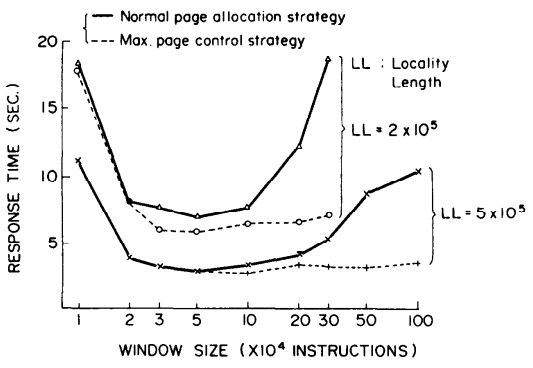


Fig. 11 Effect of the maximum page control on responsiveness (on-demand paging strategy).
Main memory pages: 80 page frames
Number of users: 60 users

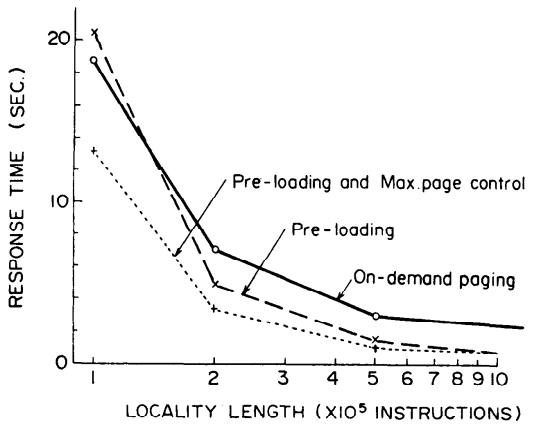


Fig. 12 Effect of the locality transfer rate on the responsiveness of memory management strategies.
Main memory size: 80 page frames
Number of users: 60 users
Window size: 50 K instructions

5. Conclusion

Multiprogrammed memory management strategies have been extensively investigated using simulation techniques. In particular, the working set strategy, which is a representative variable partitioning strategy, has been analyzed in an environment approximating actual systems operating conditions. The local LRU strategy, which is a representative fixed partitioning strategy, has been also investigated for comparison with the working set strategy. In these analyses, the effect of locality transfers in users programs has also been considered. Simulation results are summarized as follows:

(1) In the fixed partitioning strategies, such as the local LRU strategy, the effect of the number of main memory partitions is too great for the strategies to be used in actual systems.

(2) In the working set strategy, the feasible range of window size is wide. For the five model programs selected in this paper, the window size for referencing the whole locality set is approximately 30 K ~ 50 K instruction steps.

(3) The pre-loading policy of the working set strategy improves responsiveness when the locality transfer rate is not excessively high.

(4) It is useful to control the maximum number of

pages allocated to a task, so as to exclude unnecessary pages from the working set and prevent programs with extremely large working set size from degrading total system performance.

(5) The locality transfer rate of user programs has a great effect on the total system performance in the working set strategy.

The program behavior models proposed to date have not produced working models in which locality transfer rates or locality set size variations are accounted for precisely. In addition, few actual measurements have been reported regarding the extent of locality transfer rates or locality set size variations. Further work is required for modeling these aspects more precisely, and more actual system measurements of these aspects must be collected.

Finally, some comments are given for the simulation programs used. The simulation programs are implemented by FORTRAN. Program size is about 4 K statements for the working set strategy and 3.3 K statements for the local LRU strategy. The simulation speed is about one third of real time, using the HITAC 8700; *i.e.*, the simulation of 10 minutes requires about 200 sec. cpu time in the HITAC 8700.

Acknowledgements

The author gratefully acknowledges many stimulating discussions with Professor M. Hosaka and Professor S. Osuga of the University of Tokyo during the course of this research. The author is also indebted to Mr. I. Ohnishi and Dr. K. Noguchi of the Software Works of Hitachi for their cooperation in designing simulation environments.

References

1. COFFMAN, E. G. AND RYAN, T. J. A Study of Storage Partitioning using Mathematical Model of Locality. *Comm. ACM*, **15**, 3 (March 1972), 185–190.
2. DENNING, P. J. The Working Set Model for Program Behavior. *Comm. ACM*, **11**, 5 (1968), 323–333.
3. DENNING, P. J. AND GRAHAM, G. S. Multiprogrammed Memory Management. Proc. of the IEEE, **63**, 6, (1975), 924–939.
4. ISHIDA, H. AND NOMOTO, M. Graphic Monitoring of a Large Scale Computer System. IPSJ, Document of System Performance Evaluation Meeting (March 1975) (in Japanese).
5. NOGUCHI, K., OHNISHI, I. AND MORITA, H. Design Considerations for a Heterogeneous Tightly-Coupled Multiprocessor System. Proc. of NCC, **44**, (1975), 561–565.
6. OPDERBECK, H. AND CHU, W. W. Performance of the Page Fault Frequency Algorithm in a Multiprogrammed Environment. Proc. of IFIP congress 74, (1974), 235–241.
7. SALTZER; J. H. On the Modeling of Paging Algorithms. *Comm. ACM*, **19**, 5 (May 1976) 307–308.
8. SPIRN, J. R. AND DENNING, P. J. Experiments with Program Locality. Proc. of FJCC **41**, (1972), 611–621.