Proceedings of Sixth ACM Symposium on Operating Systems Principles (November 1977) 117 -124.

EFFECT OF PROGRAM LOCALITIES ON MEMORY MANAGEMENT STRATEGIES

Takashi Masuda
Institute of Information Sciences and Electronics
University of Tsukuba
Japan

Programs tend to reference pages unequally and cluster references to certain pages in short time intervals. These properties depend on the tendency of program locality references and program phase transitions. The significant effects on system performances arise from the phase transition behavior. However, the phase transition behavior of programs has been rarely taken into account in the analysis of memory management strategies. This paper investigates the effect of the phase transition behavior on the total system performance. For this purpose, an elaborate simulation model of the multiprogrammed memory management has been developed for a time-sharing environment. The working set strategy and the local LRU strategy are modeled in the simulation system. A simple phase transition model and the simple LRU stack model are used as a program paging behavior model. Both cases are analyzed where (1) locality variations exist and phase transitions occur, and (2) only locality variations exist and phase transitions do not occur. The relations between the phase transition rate and the system performance are found in the above memory management strategies.

Key Words and Phrases: program locality, program behavior, multiprogramming, memory management, performance evaluation, computer system simulation

CR Categories: 4.3

1. INTRODUCTION

In virtual storage systems, memory management strategies have a critical effect on system performance. A number of memory management strategies have been proposed and analyzed [3]. These can be grouped with respect to 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. In variable partitioning strategies, the number of allocated page frames for each active task varies during program execution.

Programs tend to reference pages unequally and cluster references to certain pages in short time intervals [6, 7, 10]. Therefore, they can be run efficiently in memory spaces considerably smaller than the program size. These properties depend on the tendency of program phase transitions or program locality references. Excessive page faults occur during program execution when the locality set of a phase is not loaded into the main memory. Consequently, those variable memory management strategies are desirable which 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 [3]. Coffman and Ryan [1] analyzed the effect of locality variations. 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.

As stated in [4], the significant effects on system performances will arise from the phase transition behavior. Particularly, the effect on the variable partitioning strategies with explicit correlation to the locality properties of active programs, such as a working set strategy, will be significant. However, the phase transition behavior of programs has been rarely taken into account in the analysis of the memory management strategies.

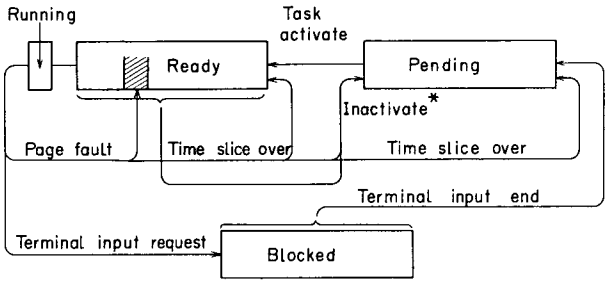
This paper investigates the effect of the phase transition behavior on the total system performance. For this purpose, an elaborate simulation model of the multiprogrammed memory management has been developed for a time-sharing environment. A simple phase transition model and the simple LRU stack model is used as a program paging behavior model [4, 14]. As a representative fixed partitioning strategy, the local LRU strategy is employed. As a representative variable partitioning strategy, the working set strategy is employed.

2. SIMULATION MODEL

2.1 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.



* only in case of working-set strategy

Fig. 1. Task states and transitions

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.

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 [8]. 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 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.

2.2 Model of user program behavior

A program's behavior can be characterized in terms of its residence in localities of various sizes and lifetimes, and the transitions between these localities [4, 9, 14]. Then a program's behavior model consists of two parts: the model of the phase transition behavior and the model of the reference pattern within each phase.

Madison and Batson [9] proposed the concept of a locality through a formal definition of what constitutes a phase of localized reference behavior, and gave a corresponding mechanism for the detection of localities in actual reference strings. They defined a phase as a maximal interval in which LRU stack distance does not exceed i and every one of the i top stack objects is referenced at least once. They concluded that programs do not gradually drift between localities, but rather that program execution can be modeled as a sequence of residencies in fairly long-lived stable phases and that the transitions between these phases can be fairly disruptive.

Denning [4, 14] tried to model these phase transition behaviors using as few parameters as possible. He assumed (1) that the holding time distribution in a locality was state independent, (2) that the type of observed locality distribution, together with its mean and standard deviation, was given, and (3) that at a phase transition, a locality set is entered with a fixed probability, irrespective of the previous phase. These choices require 2n+1 parameters to model the program behavior of the n locality sets. His conclusion is that the proposed simple phase transition model is capable of reproducing known properties of program lifetime functions.

As a phase transition model we use essentially the same model as proposed by Denning [4]. We simplify the model for our simulation as follows.

The holding time distribution in a locality is state independent and is assumed as a uniform distribution. The mean of the holding time distribution corresponds to the mean duration of a phase. The standard deviation of the distribution

is assumed small so as to make the simulation environments as stable as possible.

The phase transition sequence of a program is assumed to be predetermined. When a phase transition occurs, the next phase is determined from the predetermined sequence. In the preliminary experiment we used the same phase transition model as proposed above by Denning. However, if a phase is selected probabilistically independent of the previous phase, the effect of selected phases on the system performance is large in each case of simulation, because of the limited simulation time. Then we decided to use the predetermined phase transition sequences.

For the program model within a phase, the simple LRU stack model is used [12, 14]. Five programs were selected which operated under the HITAC8700/8800 virtual memory computer system [11], and the stack distance probabilities were obtained within the range where the page reference patterns are stable. The maximum stack distances of the programs are 24, 14, 34, 13 and 28. The probabilities q_0 that pages not included in LRU stacks are referenced are 0, 4.11×10^{-5}, 0, 0.19×10^{-5}, and 0.38×10^{-5} respectively.

Each of these five programs is considered to express a program phase. These phases are called phase 1, phase 2, ... , and phase 5 respectively hereafter. During program execution, when a phase change occurs, following page requests use the set of stack distance probabilities for a new program phase. The 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 are replaced by other pages in the local LRU strategy.

The execution steps between page faults are calculated from the SLRU stack probabilities when a model program is executed in a given memory size [12]. 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

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

where q_i is the i-th element of the SLRU stack, and p is the maximum stack depth. The number of memory references r between page faults can be sampled 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.

2.3 CPU time usage model

It is important to get the usage distribution of cpu time which each user program uses during an 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 used in the model was measured at the computer center of the University of Tokyo [8]. If the mean processing time per instruction is assumed to be 1.0 μsec, the average cpu time used for an interaction is 348.9 msec. The standard deviation is 693.6 msec, and the median is 74.9 msec.

2.4 Other conditions and environments of simulation

The think time distribution of the above system is used for the user's behavior at terminals. The average is 26.5 sec, the standard deviation is 38.6 sec, and the median is 10.0 sec.

Most analyses to date do not account for the time spent executing the operating system. However, especially in a time-sharing system, the time spent in the user program in an interaction is usually short and the paging rate tends to be high. The operating system execution time has a great effect on the total system performance. Some execution steps in the model are described.

It is assumed that 1K or 3K steps are needed for page fault handling according to whether available pages exist or not when a page fault occurs. Task inactivation is assumed to be 15K steps on the average. Terminal input requests are assumed to necessitate 20K steps, where input request handling and character editing are assumed to take 5K steps. Task activation is considered to take 200 steps for on-demand paging policy where only the task state change handling is necessary, and 15K steps for pre-loading policy where the swapping-in of the working set is necessary. The execution steps for task activation decision are also considered. The time necessary to calculate the working set size of the activation candidate task is assumed to be 200 steps, and the one for each active task is assumed to be 100 steps.

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

(1) cpu speed ~ 1 μ sec/instruction.
(2) Main memory size ~ 80 page frames for user area.
(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.

3. SIMULATION RESULTS

3.1 Effect of window size

First the basic properties of the memory management strategies are investigated. For this purpose, it is assumed that phase transitions during program execution do not occur. Each program requests pages according to the set of stack distance probabilities for one of the model program phases.

Window size is the most important control parameter 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 phenomena. 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 or phase transition. When a phase transition 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. 2 when the number of users is 80. As window size is decreased to around 10K instruction steps, response time increases rapidly, and the thrashing phenomena occur. When the window size increases to approximately 10^6 instruction steps, then the response time again increases in spite of no phase transitions 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 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.

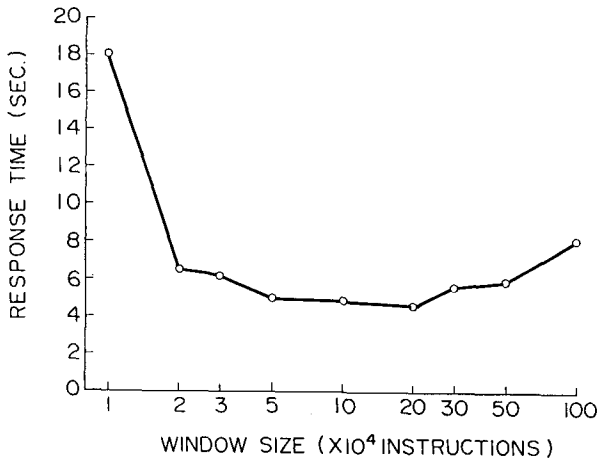


Fig. 2. Effect of window size on responsiveness when no phase transitions occur

{Main memory size: 80 page frames
{Number of users: 80 users

In this example, the responsiveness is satisfactory for window sizes between 30K and 300K 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.

3.2 Comparison of the local LRU strategy with the working set strategy

The average response times for the local LRU and working set strategies are found as a function of user number in fig. 3, when no phase transitions are assumed to occur during program execution. Simulation of the local LRU strategy was carried out for the partitioning numbers of the main memory equal to one to five. The average response time is optimal when the partitioning number is two. When the partitioning number equals one or two, the number of allocated page frames to a task is larger than the maximum stack depth of model programs. When the main memory is partitioned into three, two programs out of five have a maximum stack distance greater than the number of allocated page frames. The thrashing phenomena are observed when the partitioning number is four or five. The average response time is extremely long and cannot be expressed in fig. 3. As shown in this example, the partitioning number is essential to total system performances. In actual systems, it will be almost impossible to decide an ideal partitioning number since locality set size differs among users and varies during a program execution. The defects of the fixed partitioning strategy have been clarified in fig. 3.

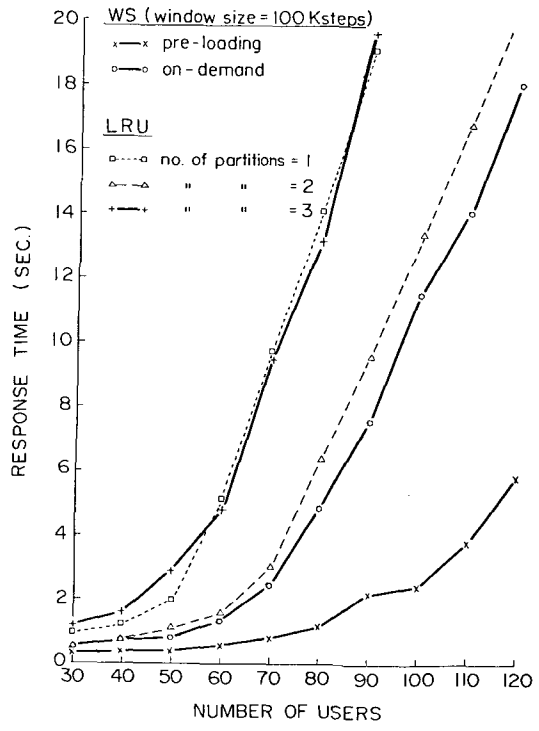


Fig. 3. Comparison of memory management strategies when no phase transitions occur

Main memory size: 80 page frames

The results of two variations in working set strategies are investigated; on-demand paging strategy and pre-loading strategy. The advantages of the pre-loading strategy 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 of page fault handling programs. 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 or the phase transition rate is low.

Figure 3 shows that the average response time of the pre-loading strategy is much better than that of the on-demand strategy, since no phase transitions are assumed for user programs. The average response time is about 5 sec for 80 users with the on-demand strategy, and for 110 ~ 120 users with the pre-loading strategy. The mean execution instruction steps between page faults are 7K steps and 70K steps respectively.

Comparing the working set strategy with the local LRU strategy, 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.

3.3 Effect of phase transitions

The effect of phase transitions in user programs is considered. If phase transitions 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 phase transitions occur.

The phase transition model is assumed to be the model described in 2.2 Each user program executes one of the five model program phases during a lifetime of the phase, and then selects another program phase for execution. The lifetime of a phase is designated as the phase duration. The effect of the phase duration on average response time is shown in fig. 4, when the number of users is 60 and the main memory size is 80 pages in the working set strategy of the on-demand paging policy. As the rate of phase transitions increases, responsiveness deteriorates rapidly, and in particular, becomes more sensitive to changes in window size. This is because when the rate of phase transitions is high, many unused pages are included in the working set as window size increases, and the multiprogramming degree decreases.

Some detailed measurements representing system behavior are shown in fig. 5 in the above cases when the phase transition occurs every 10^5 instruction steps and 10^8 instruotion steps. When a phase transition occurs every 10^5 instruction steps, the multiprogramming degree goes down rapidly as window size increases, and the paging and channel idle rate increases. Responsiveness deteriorates rapidly as window size increases over 10^5 instruction

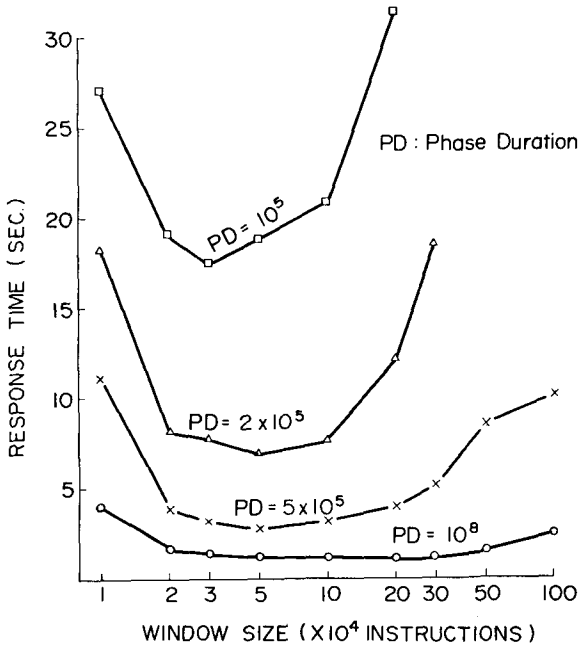


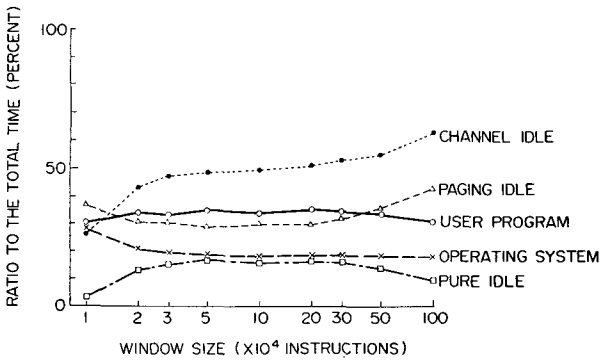
Fig. 4. Effect of the phase transition rate on responsiveness

{Main memory size: 80 page frames
{Number of users: 60 users

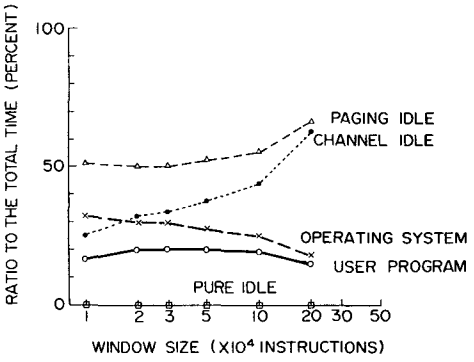
steps. In comparison, when a phase transition occurs every 10^8 steps, which means that actually no phase transition occurs during simulation, the multiprogramming degree decreases gradually as window size increases. As explained earlier, this is because some model program phases request pages not included 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 30K steps shows that about 30K 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 phase transitions 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. 5, the feasible window size range should be between 30K and 50K instruction steps.

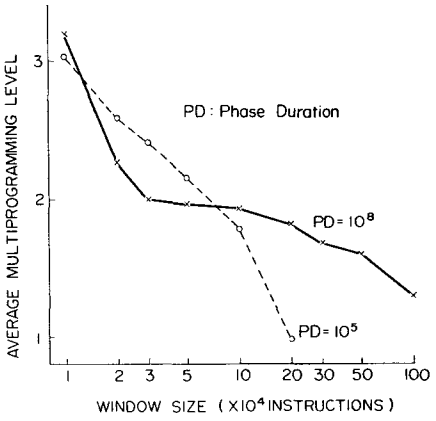
While phase transitions occur, working set size increases transiently and many pages of previous phases are included in the working set. To reduce this undesirable effect, it is useful to control the maximum number of pages allocated to a task. This mechanism is also useful for preventing the programs with extremely large working set size from reducing total system performance. The effectiveness of the maximum page allocation control on responsiveness is seen for the phase transition occuring every 200K steps and 500K steps in fig. 6, when the number of pages allocated to a task is limited to 35 pages. The solid lines indicate where maximum page allocation control is not specified, and the dotted lines where it is specified.



(a) Phase duration = 10^8 instruction steps



(b) Phase duration = 10^5 instruction steps



(c) Multiprogramming level for two cases

Fig. 5. System characteristic measurements of the working set strategy

{ Main memory size: 80 page frames
{ Number of users: 60 users

The effectiveness of pre-loading policy will decrease when the phase transition takes place, since the swapping-in probability for unused pages becomes high. Figure 7 shows how the phase transition effects on the effectiveness of pre-loading.

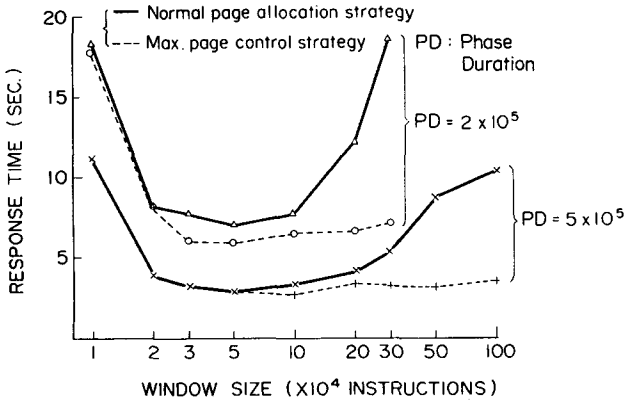


Fig. 6. Effect of maximum page control policy on responsiveness

{ Main memory size: 80 page frames
{ Number of users: 60 users

As a phase duration decreases, the responsiveness of pre-loading strategy degrades in comparison with the on-demand paging strategy. The dotted line which gives the optimal responsiveness is the result of adopting both a maximum page allocation control policy and pre-loading strategy.

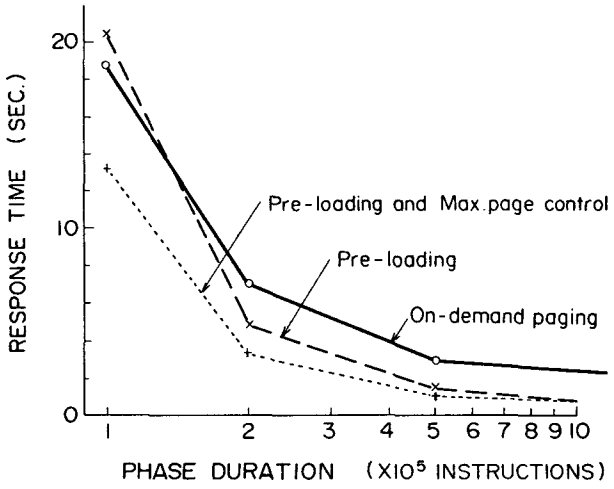


Fig. 7. Effect of the phase transition rate on the responsiveness of memory management strategies

{ Main memory size: 80 page frames
{ Number of users: 60 users
Window size: 50K instruction steps

We will mention the performance of the local LRU strategy, when the effect of phase transitions is taken into account. In the local LRU strategy system is controlled without any correlation to the locality properties of programs. Then there is no performance degradation due to the estimation failure of a locality set. Paging rate will increase by the new page requests due to the phase transitions. Figure 8 shows the effect of phase transitions on the responsiveness. The responsiveness is optimal when the number of memory partitions is two. In this case the response time is almost the same as that of the working set strategy shown in fig. 4. However, since, in the local LRU strategy, there is no way to decide the optimal number of

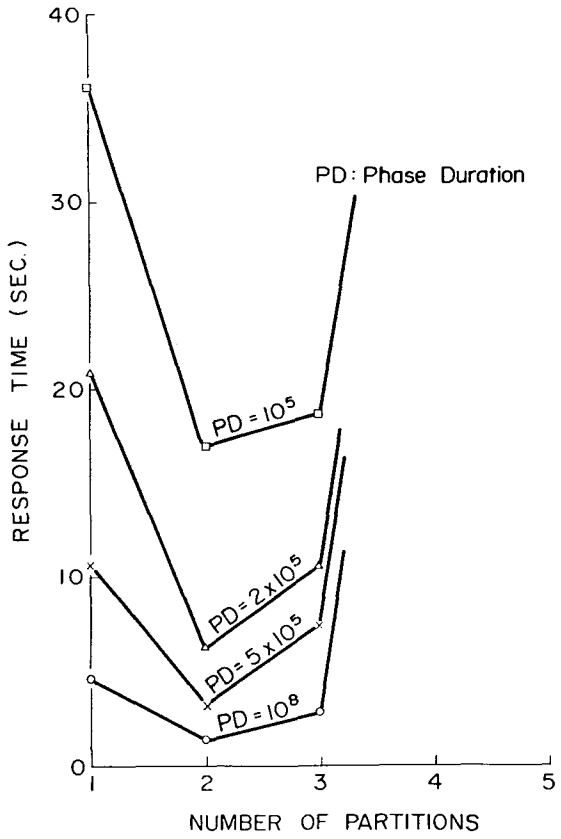


Fig. 8. Effect of the phase transition rate on responsiveness in the local LRU strategy

{ Main memory size: 80 page frames
{ Number of users: 60 users

memory partitions, and the system responsiveness is affected greatly by the number of memory partitions, it is difficult to use the strategy in actual systems.

4. CONCLUSION

This paper has analyzed the effect of the phase transition behavior of programs on actual systems. An elaborate simulation model has been developed for time-shared multiprogramming environments. As a representative fixed partitioning strategy, the local LRU strategy is employed, and as a representative variable partitioning strategy, the working set strategy is employed.

In the local LRU strategy, the effect of the number of main memory partitions is too large for the strategy to be used in actual systems, in both cases where phase transitions exist or do not exist.

When the effect of phase transitions is not accounted for, the working set strategy seems quite effective. The feasible range of window size is wide and the pre-loading policy improves responsiveness. However, as the phase transition rate becomes high, the responsiveness of the working set strategy degrades rapidly. The range of feasible window size becomes narrower. 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, as indicated in [3]. In our example, the feasible window size range should be between 30K and 50K instruction steps. The maximum number of pages allocated to a task should be controlled, so as to exclude unnecessary pages of past phases from the working set.

This paper does not measure the phase transition rate of actual programs. Few actual measurements [9] have been reported regarding the extent of the phase transition rates of locality set variations. More actual system measurements must be collected.

Finally some comments are given for the simulation programs used. The simulation programs are implemented by FORTRAN. Program size is about 4K statements for the working set strategy, and 3.3K statements for the local LRU strategy. The simulation speed is about one third of real time, using the HITAC 8700 [11], 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] E. G. Coffman and T. J. Ryan: A Study of Storage Partitioning using Mathematical Model of Locality, Comm. ACM, Vol. 15, No. 3, pp. 185-190 (1972).

[2] P. J. Denning: The Working Set Model for Program Behavior, Comm. ACM, Vol. 11, No. 5, pp. 323-333 (1968).

[3] P. J. Denning and G. S. Graham: Multiprogrammed Memory Management, Proc. of the IEEE, Vol. 63, No. 6, pp. 924-939 (1975).

[4] P. J. Denning and K. C. Kahn: A Study of Program Locality and Lifetime Functions, Proc. of SIGOPS Conf. SOSP-5, pp. 207-216 (1975).

[5] P. J. Denning, K. C. Kahn, J. Leroudier, D. Potier and R. Suri: Optimal Multiprogramming, Acta Informatica, Vol. 7, No. 2, pp. 197-216 (1976).

[6] D. Ferrari: Improving Locality by Critical Working Sets, Comm. ACM, Vol. 17, No. 11, pp. 614-620 (1974).

[7] D. J. Hatfield and J. Gerald: Program Restructuring for Virtual Memory, IBM Systems Journal, Vol. 10, No. 3, pp. 168-192 (1971).

[8] H. Ishida and M. Nomoto: Graphic Monitoring of a Large Scale Computer System, Information Processing Society of Japan, Document of System Performance Evaluation Meeting, March (1975).

[9] A. W. Madison and A. P. Batson: Characteristics of Program Localities, Comm. ACM, Vol. 19, No. 5, pp. 285-294 (1976).

[10] T. Masuda, H. Shiota, K. Noguchi and T. Ohki: Optimization of Program Organization by

Cluster Analysis, Proc. of IFIP Congress 74, pp. 261-265 (1974).
[11] K. Noguchi, I. Ohnishi and H. Morita: Design Considerations for a Heterogeneous Tightly-Coupled Multiprocessor System, Proc. of NCC, Vol. 44, pp. 561-565 (1975).
[12] H. Opderbeck and W. W. Chu: Performance of the Page Fault Frequency Algorithm in a Multiprogrammed Environment, Proc. of IFIP Congress 74, pp. 235-241 (1974).
[13] J. H. Saltzer: On the Modeling of Paging Algorithms, Comm. ACM, Vol. 19, No. 5, pp. 307-308 (1976).
[14] J. R. Spirn and P. J. Denning: Experiments with Program Locality, Proc. of FJCC, Vol. 41, pp. 611-621 (1972).