Analysis of block-paging strategies

by W. H. Tetzlaff
M. G. Kienzle
J. A. Garay

The performance of interactive paging systems in general and Virtual Machine/System Product (VM/SP) systems with the High Performance Option (HPO) in particular depends upon locality of reference. This storage-management dependency, often considered in the context of individual programs, extends in fact to a significant degree across most virtual-machine transactions. This paper investigates strategies to exploit locality of reference at the system level by analyzing page-reference strings gathered from live systems. Alternative strategies are evaluated using trace-driven simulations.

Introduction

The concept of *working sets* has usually been applied to individual programs and their data. Experience with time-sharing systems, such as IBM's Virtual Machine/System Product (VM/SP), suggests that the concepts of locality of reference and working sets apply similarly to series of interactions with the system, where each interaction invokes a program. In interactive sessions, users issue many different commands, invoking programs and referencing a large variety of system and user data. The VM/SP system with the High Performance Option (HPO) uses a real-storage management policy that is a close approximation to a detuned working-set policy [1]. The primary purpose of this real-storage management is to identify groups of pages that can be "block paged" together. To implement the policy, the real-storage management system must keep track of the reference behavior of the virtual machines. The VM/Monitor makes some of these reference data available for analysis.

The second section discusses some concepts that are necessary as background for the paper. The next section describes the tools used in the data-collection and data-reduction processes. The fourth section shows the interaction between locality of reference and the algorithms used in HPO. We then discuss the degree to which the data show possibilities for improving the real-storage management algorithms. The final section summarizes the most important points of the paper.

Overview of concepts

In the areas of modeling program behavior and measuring the memory demands of programs, much work has been done [1–3] that makes use of the concept of the *working set*—the collection of segments (or pages) recently referenced by the program.

The data for this study have been collected from the IBM VM/SP system with the High Performance Option. The window size is assumed to equal the time during which a virtual machine remains in the multiprogramming set. Because members of the multiprogramming set are kept in a queue, the time in the multiprogramming set is sometimes called a *queue stay*. Normally a queue stay corresponds to a single user interaction. In some cases the scheduler divides a transaction into multiple queue stays, using a standard CPU time slice to determine the time in queue.

Introduction to HPO paging and scheduling

The HPO paging algorithms tie real-storage management to scheduling. The following material discusses HPO scheduling with only the degree of detail needed for this paper. For additional background information, Koeppel [4] provides an overview of resource management in VM systems, Miles [5] gives a more complete review of HPO scheduling, and Tetzlaff and Beretvas [6] offer an overview of paging in all VM operating systems.

The VM scheduler moves virtual machines among the categories Q1, Q2, Q3, E1, and E2. The broadest distinction among these categories consists of three sets: a) those virtual machines that are *in queue* (Q1, Q2, Q3), b) those *eligible* for a queue (E1, E2), and c) those that are *idle* (neither Q_n nor E_n). If a machine is in queue, either it is ready to run or it has recently been run and is expected to be ready to run soon. The virtual machines that are in a queue constitute the multiprogramming set. All machines in queue are sorted into a single list according to an internally assigned priority. The list is searched by the dispatcher, which selects the highest-priority machine that is able to run. If a virtual machine has work to process but there is insufficient main storage, it is not placed in queue; rather, it is given eligible status. If a virtual machine is idle, it will not run until some event occurs, such as an input from a terminal or a message from another virtual machine.

The virtual machines in queue are classified into three categories. The system attempts to place short transactions in Q1, longer-running transactions in Q2, and very long-running transactions in Q3. This is accomplished by starting transactions in Q1 and then moving them to Q2 and Q3 as they consume resources. Q1 is intended as a means by which interactive users doing short transactions can be identified and given good response time. A virtual machine is placed in Q1 after a long idle period, which normally corresponds to the beginning of a transaction.

Virtual-machine storage concepts

The virtual address space of a virtual machine is a set of addresses whose total memory space is that of the virtual machine. This address space is divided into *segments* of 64K bytes. A segment is divided into *pages* of 4096 bytes each. Segment tables and page tables describe the mapping of the virtual addresses into real storage; this mapping is not necessarily contiguous. A page table shows whether a page is in real storage and correlates virtual addresses with real-storage addresses. A segment table describes which page tables, each describing one segment, a virtual machine uses for its storage. These tables are updated by the VM control program (CP) and reflect the allocation of virtual-storage pages to blocks of real storage.

The initial and maximum storage sizes of a virtual machine are defined as part of the virtual-machine configuration in the system directory. The user can redefine the virtual-storage size to any multiple of 4K not greater than the maximum value defined in the directory.

Storage in the real machine is logically and physically divided into 4096-byte *page frames.* If there is a shortage of available real storage, CP will try to keep only referenced virtual-storage pages in real storage, making use of the paging (and swapping) mechanism described in the remainder of this section.

VM/HPO paging and block paging

Paging and block paging are done by components of the HPO Control Program (CP). The basic objective of the HPO block-paging algorithms [7] is to improve interactive response time by reducing page waits and reducing the CPU overhead, especially for large systems. In addition, blocking techniques are used to reduce the number of times paging paths are used.

I/O operations that move only a single page are not very efficient for the (disk) paging devices or for the CPU. Therefore, the algorithms try to move several pages in one I/O operation whenever possible. This *block paging* exploits the high data-transfer-rate capability of disks. The cost to the system is the same as if one large page were transferred with a single I/O operation. Grouping several pages into one large page yields a granularity advantage over using larger pages. The components of this "big page" are changed with time as the contents of the working set change; this would not be true for a single large page. The block-paging concept reduces both seek and rotational latency, as well as the CPU overhead of paging.

For a physical swap, the working-set pages of a virtual machine are organized into *swap sets.* The working set of a machine may require several swap sets. When a physical swap is necessary, one or several swap sets, but not necessarily all swap sets belonging to a machine, will be written to the swap data sets. Thus, a machine can be *partially swapped.* If this is the case, the next physical swap-out operation forces another swap-out for this virtual machine, since all the logically swapped pages of a machine are swapped out before another machine is selected from the swap list as a candidate for physical swapping.

Swap-set affinities

At queue-drop, HPO identifies all pages that have been referenced during the previous queue stay. These can be considered the working-set pages, since the queue slice defines the reference interval. Regardless of whether these referenced pages are changed or unchanged, they are logically swapped. The unreferenced pages are trimmed from the working set and moved to a "flush list."

When physical swapping takes place, swap sets are formed by grouping the logically swapped pages of a given virtual machine in order of their *virtual addresses* into swap sets. Thus, swap sets are related by virtual address and by *time of*

reference, since all the pages in a swap set were referenced during the same queue stay. The collection of pages forming a swap set is constructed for a swap-out, and the swap set exists on DASD until it is swapped in, after which the affinity of pages is lost. After each new queue stay, swap sets are formed afresh for swap-out, and the pages in them may be different than before. Note that not all swap sets of a virtual machine necessarily come from the same queue stay or working set.

HPO systems analyze the references to pages within each interaction or time slice to predict which pages will be used together in later interactions or time slices. When sufficient real memory is available, the entire working set of the machine will be left loaded for the next transaction (this amounts to full working-set preloading). In the event that real storage must be made available, one or more swap sets will be paged out together. Because the pages in a swap set will all be brought in together when one page is referenced, it is important that the grouping algorithm predict which pages will be used together in a future interaction. This operation is called *swap faulting* or *demand swapping.*

Pages that are either logically or physically swapped in, but are not subsequently referenced, are paged out at a later time. Because the paging rate of system-owned pages and shared pages is quite low, the page-out rate of the system is a measure of the failures to predict the working set. In turn, the page-reads represent later reference to these same pages.

The goal of block paging is to do the best possible job of predicting future working sets. Perfect prediction would mean that all pages that have been logically or physically swapped in will be referenced; this would have the effect of reducing the paging rate to zero. The thrust of this paper is to analyze the predictive capability of the HPO algorithms.

Available data

All data presented in the following sections were recorded at three different installations from *interactive* (CMS) [8] user sessions or from service machines running CMS. One installation is at an educational institution, another supports a manufacturing plant, and the third is in a service industry. All three installations were running VM/HPO 3.4 [7] at the time of recording, and from each installation we analyzed the data of five virtual machines running CMS. In the data collection and reduction process, we used the tools described below.

- *Data collection: VM/Monitor*

All the data used in our analysis of paging behavior are available through the VM/Monitor [9], which is a standard part of the VM system control program.

This paper used trace data recorded at *logical swap-out* and at *voluntary queue-drop* events. Since trimming of unreferenced pages also occurs at involuntary queue drops, for which reference data are not available, the reference strings used for this analysis are not totally complete. For transactions that complete in Q1, however, the reference information is complete.

- *Data reduction*

GRIN

For the reduction of VM/Monitor data, we used a special data-reduction package developed at the IBM T. J. Watson Research Center, known as the *Generalized Reduction of Information* (GRIN) program [10]. This program is a *program generator* that has been designed to increase the productivity and effectiveness of performance analysis by allowing analysts to spend more time on analysis and less time on either defining or writing data-reduction programs.

GRAFSTAT

After we extracted the relevant data from the VM/Monitor files using the GRIN package, we used the GRAFSTAT system [11] to analyze the data and present them graphically. GRAFSTAT is an interactive APL system that combines the versatility of APL with a high-level, full-screen user interface, many built-in statistical routines, and a high-resolution storage display. Much of the preliminary data analysis and all of the plots in the final paper were done using GRAFSTAT.

Data presentation

- *Data selection*

We analyzed three tapes with VM/Monitor data, one from each of the three installations. From each tape we selected five virtual machines for a preliminary analysis. This included viewing the page-reference pattern and some statistical analysis. From these reviews of fifteen machines, we selected six for detailed analysis in this paper.

- *Page-reference pattern*

Figure 1 shows the page-reference pattern of a virtual machine running CMS. The x-axis dimension represents virtual page numbers. The reference data are collected using the reference bit provided by the hardware for each 4K page of storage. Therefore, the resolution of the virtual address referenced is in units of page size. Each point along the x-axis represents one page of storage of the virtual machine, sorted in the order of the virtual addresses of the pages. The user whose reference pattern is shown by the graph has a virtual machine with 256 pages, or one megabyte, of storage.

The points on the y-axis represent queue stays. The VM/Monitor records the reference data only at the end of queue stays that cause logical swaps, that is, stays in either Q1 or Q2 that terminate with *voluntary* queue drops. The sequence of reference within a queue stay is not recorded. This limits the resolution of the sequence of reference to units of queue

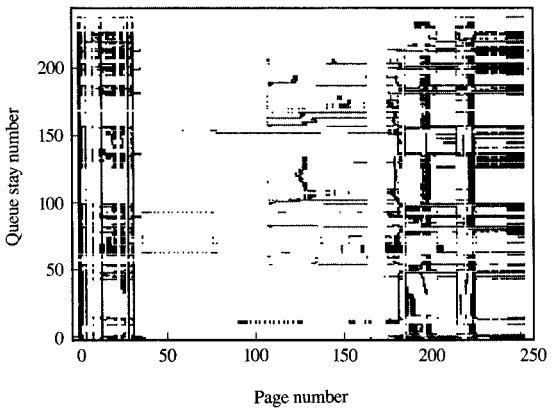

Figure 1

Page-reference pattern of VM 1.

stays. Since each user interaction ends with a voluntary queue drop, each point on the y-axis corresponds to one user interaction. The references shown are the references in the last queue stay of the interaction. The user represented in the graph experienced 267 queue stays during the measurement period. The number of queue stays is not proportional to real time, but it is approximately proportional to the virtual time of the user's virtual machine. The user's think time is not included in this approximation.

Each dot in the graph represents a page reference. A vertical line shows all of the queue stays during which a particular page has been referenced. A horizontal line depicts the pages that were referenced during the corresponding queue stay. The HPO algorithm that sorts the pages into swap sets by virtual addresses is an excellent choice, as the reference patterns show.

• *Relation of reference pattern to CMS virtual-storage use*

In another paper [12] the authors have studied the reference patterns of these same six users and have come to a number of conclusions.

The reference behavior of highly interactive systems under VM is dominated by the system structure of the virtual operating system, and by the users' behavior, rather than by the behavior of the individual application programs. There are several specific areas that each have their own characteristic reference patterns. As individual transactions get more computation-intensive, the related application will probably start to dominate the reference pattern. The fact that data are recorded for only one queue stay of a transaction masks this effect.

In the environment we studied, the data references have a far larger impact on the overall reference pattern than the code references. Most of the code references are to shared code segments. The data structures for such shared code must be defined very carefully, since their impact on system performance increases with the degree of sharing.

Analysis

In the remainder of this paper, the selected reference traces are analyzed to help us to better understand the HPO working-set predictions. We first consider those pages that are not predicted by HPO to be in a particular future working set. Second, we analyze those pages that are incorrectly predicted to be in a particular working set. Third, we consider the affinity of pages that are placed in swap sets. Finally, we evaluate the possibility of creating swap sets out of unreferenced (trimmed) pages.

Table 1 gives an overview of some working-set statistics of the virtual machines analyzed. The *Comment* column helps us to relate the data more easily to the discussion in the preceding section. The *VM size* column gives the virtual machine size in megabytes. The column headed *Queue drops* shows the total number of queue drops the virtual machines experienced during the recording interval. The column *WS size mean pages* gives the mean working-set size in pages as estimated by the Scheduler. In order to illustrate how the working-set size varies, the next column, *WS size stdev/mean WS* shows the standard deviation of the working-set size divided by the mean. The columns referring to working-set change show the difference from one queue drop to another in the pages that are in the working set. The first of the two columns, *WS change mean chg/mean WS*, shows the mean

Table 1 Overview of working-set statistics.

	Comment	*VM size* (Mbytes)	*Queue drops*	*WS size*		*WS change*	
				mean pages	*stdev/ mean WS*	*mean chg/ mean WS*	*stdev/ mean WS*
VM 1	Initial example	1	296	37.2	0.68	0.67	1.12
VM 2	Interactive application	2	425	29.2	0.65	0.43	1.63
VM 4	Service machine	3	553	54.4	0.08	0.09	1.00
VM 7	Typical example	4	298	95.4	0.47	0.35	1.60
VM 8	Example of phases	7	357	83.7	0.35	0.17	2.18
VM 9	Service machine	1	400	38.3	0.20	0.35	0.71

number of pages exchanged as a fraction of the mean working-set size. The second column, *WS change stdev/mean WS*, shows the standard deviation of the working-set change as a fraction of the mean of the working-set change.

• *Trimming considerations*

At the end of voluntary and involuntary queue drops, the HPO real-storage management algorithms trim the working sets. They remove from the working set the pages that have not been referenced in the preceding queue stay, and move them onto the flush list. **Table 2** gives the mean number of pages trimmed at a queue drop.

Generally, the number of pages trimmed is small compared to the number of pages referenced. For a particular page to be trimmed frequently over the course of a session, that page must be referenced and dropped from reference frequently. To this extent, the trimming frequency follows the reference frequency. But if a page is referenced very frequently, it cannot be trimmed very frequently, so there is no direct correlation between the trim and the reference frequencies.

The trimming algorithm works optimally if it trims precisely the pages that will not be used in the next queue stay, but this is not possible without perfect foresight. The current trimming algorithm is an approximation. It assumes

1. That pages that have not been referenced in the current queue stay will not be referenced in the next queue stay.
2. That if a page has been referenced in the current queue stay, it will likely be referenced again.

How realistic are these assumptions?

If a page has been trimmed in queue stay $i - 1$ and is then referenced in queue stay i, assumption 1 is violated. This effect of trimming too many pages is called the *over-trim error*. Table 2 shows the mean over-trim error per queue stay, which is about 5 percent of the working set. This in itself is not very much. **Figure 2** shows the distribution of the behavior of the over-trim error over time for VM 1, VM 2, and VM 7.

The distribution of the over-trim error is highly skewed. Most of the over-trim errors are very small, but for a small number of queue stays, the error is very large. It would be interesting to be able to detect the large over-trim cases and prevent them. However, we could not find a way to predict them, and the potential gain from reducing the over-trim error is probably not very large.

• *Block-paging considerations*

In a block-paging system, certain pages are grouped together when they are written to external storage. At a later time they are brought back into real storage. Some of those pages that are read back into memory in advance of their reuse will not be reused at all. The movement of these pages into real memory constitutes an error.

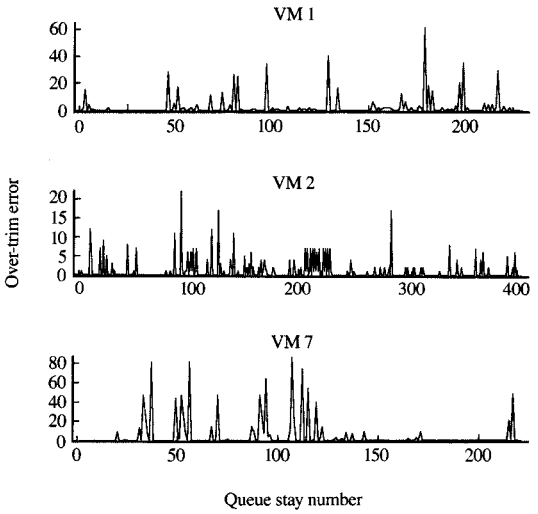

Figure 2

Over-trim error history for VM 1, VM 2, and VM 7.

Table 2 Mean number of pages trimmed and mean over-trim error per queue stay.

	VM 1	*VM 2*	*VM 4*	*VM 7*	*VM 8*	*VM 9*
Referenced pages	32.9	27.8	52.3	97.6	83.1	38.4
Trimmed pages	12.5	6.3	2.5	16.6	7.4	6.6
Over-trim error	2.3	1.0	0.8	4.8	1.0	1.9

In a system that preloads whole working sets, the preloaded but unreferenced pages would be termed *under-trim errors*. This terminology is consistent with the over-trim concept. However, in the early stages of experimenting with the HPO 3.4 algorithms, preloading of full working sets was discarded [13].

In a system that loads swap sets into memory on demand, the pages that are preloaded but not referenced are called *pre-page errors*. The algorithm predicts that all pages within the swap set will be used together, but inevitably some of the pages are not. This error has much more to do with the swap-set creation process than with the trimming process.

While it might seem natural to swap entire working sets, the experience of the prototype developers indicated that it should not be done. The cost of full working-set preloading is higher in terms of real-storage requirements and more I/O traffic. Swapping swap sets reduces the storage waste, even though it does not completely eliminate it. Demand paging would give the best storage economy. But demand swapping has some significant advantages over pure demand paging. First, the total elapsed time to bring pages into memory is

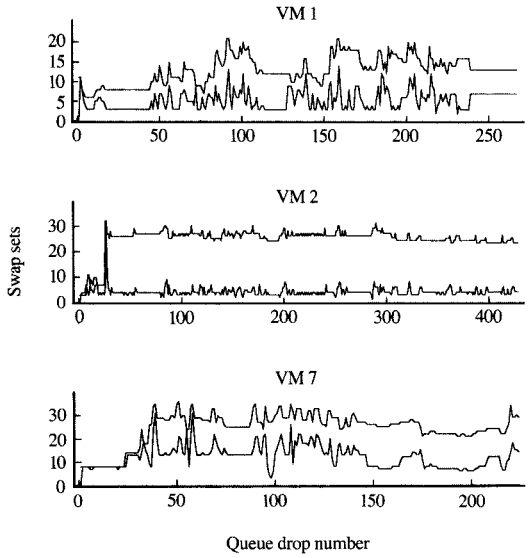

Figure 3

Total number of swap sets vs. swap sets swapped in.

Table 3 Page-reference percentages under swap-set swapping and working-set swapping.

	VM 1	*VM 2*	*VM 4*	*VM 7*	*VM 8*	*VM 9*
Demand paging only	100	100	100	100	100	100
Swapping swap sets	78	83	95	88	95	84
Swapping working sets	65	75	94	81	91	82

Table 4 Swap-set reuse distance.

Reuse distance (Queue stays)	*Number of swap sets*					
	VM 1	*VM 2*	*VM 4*	*VM 7*	*VM 8*	*VM 9*
$x = 1$	971	1502	3200	2466	3228	1118
$x = 2$	54	58	34	71	26	14
$2 < x \leq 5$	83	65	15	83	61	13
$5 < x \leq 10$	34	13	0	23	19	9
$10 < x$	36	46	0	42	51	9
% $x > 1$	17.5	10.8	1.5	8.1	4.6	3.8

much shorter. Second, the CPU overhead per page for both swap-out and swap-in is much less. These advantages are paid for with slightly higher real-storage and I/O-bandwidth cost. Swapping swap sets instead of the entire working set maintains most of the advantages of blocking, while significantly reducing the disadvantages.

We simulated the swap algorithms using recorded reference data to determine the percentage of swapped-in pages that are referenced. We also simulated the swapping of entire working sets to obtain their reference percentage. **Table 3** shows the page-reference percentages for the virtual machines analyzed in this study. Pure demand paging would result in 100 percent page reference.

The reference percentage for swapping swap sets with eight pages is somewhat closer to the percentage for swapping working sets than to the ideal percentage of 100. If the swap-set sizes were decreased, the percentages would move toward 100 percent. If the swap-set sizes were increased, the percentages would move toward full working-set swapping. The optimal swap-set size for a particular workload would be a function of real memory size, and of swapping and paging throughput capacity.

The swap-set approach is a compromise between swapping working sets and performing pure demand paging. The swap-set size determines the point between the two extremes at which a particular implementation lies. In practice, a swap-set size of eight pages proved very effective.

Some of the swapped-in swap sets originate from queue stays much earlier than the immediately preceding queue stay. **Table 4** provides some insight into the time distance (measured in queue stays) from the point at which the swap sets are created to the point at which they are brought back to memory.

Most of the swap sets are reused at the queue stay following their creation, but the portion of the swap sets that are used later is still significant. The importance of later reuse is greater for those virtual machines with more irregular reference behavior. Keeping swap sets for later use is another advantage of swapping swap sets over swapping entire working sets.

Figure 3 shows the development during the measurement interval of the total number of swap sets in the system and the number of swap sets swapped in for VM 1, VM 2, and VM 7. The data are results of the simulations mentioned above. Like any simulation data, they show start-up effects for the time the simulation takes to reach equilibrium. All three graphs appear to be in equilibrium after about 50 queue drops. Despite the fact that some swap sets are being reused in later transactions, the total number of swap sets in existence is relatively stable. The distance between the two curves remains relatively constant for all three virtual machines. This distance corresponds to the number of swap sets not referenced during a queue stay.

- *Page-affinity criteria*

The affinity criterion determines how the pages in a working set get ordered into swap sets. Since swapping swap sets gives a significantly higher reference percentage than swapping working sets, there must be some locality of reference within the swap sets. The goal of the affinity criterion is to achieve a

higher locality within a swap set than within the entire working set. When reference affinity does not exist, a page is swapped out and back in again without a reference. The system subsequently must devote real memory to a page that is not referenced, and that page is ultimately trimmed and paged out. In this way, the success of the affinity criterion clearly affects system performance.

There are three obvious affinity criteria that one might try: *most recently referenced* (or sequence of last reference), *sequence of first reference in a queue stay*, and *virtual address*.

Most recently referenced would be a true LRU (Least-Recently-Used) algorithm. The current management algorithms do not provide the time of last reference at a granularity smaller than a queue stay, but since recording that time more accurately would be extremely expensive, this is clearly not a practical alternative.

Affinity according to sequence of first reference in a queue stay would require action only once for each of the pages referenced in a queue stay. Pages would be set valid in the page tables only after their first reference, so that each page would incur the page fault needed to record its reference sequence number. Unlike the current algorithm, this approach would incur overhead for recording the first references of pages that were brought in "for free" by swap faults, but this overhead would not be prohibitively expensive. There are no indications, however, as to the swap-set locality this affinity would produce.

HPO chose virtual-address affinity, the approach that appeared to be the most obvious and also the least costly. This affinity exploits the piecewise sequential nature of many references by programs using and running under CMS. This is an extension of the successful strategy of CPU caching (which depends upon affinity within a cache line), and paging (which depends upon affinity within a 4K page). It seems reasonable to assume that virtual-address affinity would continue to exist at granularity above the page. At the same time, it requires no extra information for its implementation. All of the discussion in this paper so far is based on the virtual address criterion; i.e., pages are grouped in their virtual-address ascending order. The address affinity proved to work very well, so alternatives were not tried in developing the HPO 3.4 algorithms.

It is nonetheless legitimate to determine how important the choice of affinity criterion actually is. There are no time-of-reference data available that would allow us to simulate any of the other two criteria mentioned earlier, so we used *random affinity* and considered it a lower bound. We simulated swap-set algorithms using the recorded reference strings and grouping them into swap sets at random.

The data in **Table 5** and **Figure 4** show that the affinity criterion clearly affects the reference percentage of the swapped-in pages. The random affinity has a reference percentage very close to the reference percentage of swapping

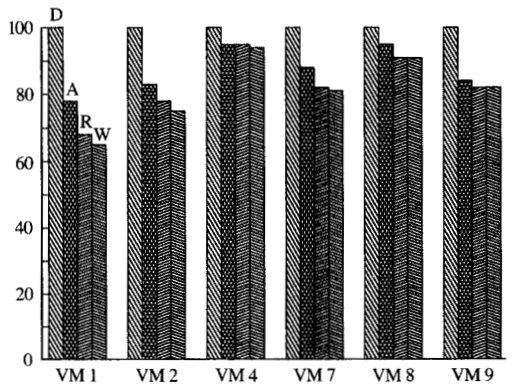

Figure 4

Page-reference percentages under random-affinity criterion: D, demand paging; A, swap sets with address affinity; R, swap sets with random affinity; W, working sets.

Table 5 Page-reference percentages under random-affinity criterion.

Swap granularity	*VM 1*	*VM 2*	*VM 4*	*VM 7*	*VM 8*	*VM 9*
Single pages	100	100	100	100	100	100
Swap sets, address affinity	78	83	95	88	95	84
Swap sets, random affinity	68	78	95	82	91	82
Working sets	65	75	94	81	91	82

entire working sets. The address affinity clearly creates improved locality in the swap sets. This proves that the affinity criterion is important for the performance of the algorithms. It also supports the conjecture, based on the shapes of the reference patterns, that many of the page references are sequential. Considering the options available, address affinity is clearly the superior choice.

• *Creating trim sets*

If swap sets are a good idea that exploits locality of reference, would trim sets also be a good idea? To create trim sets, the trimming algorithm would collect the trimmed pages into trim sets and swap them out as sets. On reference, they would be swapped in as sets. To investigate this question, we used the recorded reference patterns to simulate this modified trim algorithm.

The trim-set reference percentage (**Table 6**) is far lower than the working-set reference percentage. This clearly shows that there is far less locality of reference in the trim sets, and that it is correct to treat them very differently from the swap sets that contain the working-set pages.

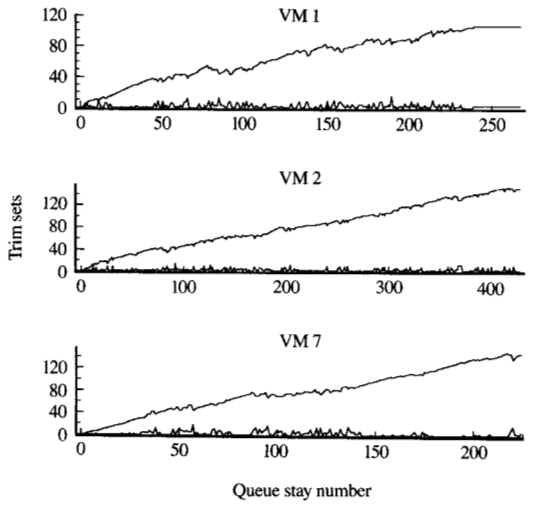

Figure 5

Total number of trim sets and number of trim sets swapped in.

Table 6 Trim-set reusage.

	VM 1	*VM 2*	*VM 4*	*VM 7*	*VM 8*	*VM 9*
Trim-set reference percentage	43	40	98	44	51	33
Mean trim-set size	6.14	5.34	1.94	6.02	4.01	6.30
Mean pages referenced	2.64	2.14	1.90	2.65	2.05	2.08

Another indication of the random nature of the references to pages in the trim sets is the total number of trim sets in the system. These numbers were also produced by the simulations. **Figure 5** shows that during the simulation, the number of trim sets in existence increases monotonically, never reaching equilibrium.

During the simulation, the size of the trim sets and their reference percentage did not change noticeably. This, and the fact that the number of trim sets increases, shows that most trim sets are never swapped back in. The few trim sets that are swapped in are related to the over-trim error, and are swapped in very soon after their creation.

The authors thus conclude that it is inappropriate to attempt to block trim pages. The dichotomy between paging and swapping seems to be a necessary characteristic of swapping systems.

Conclusions

- *Trimming*

There is some over-trim error and some under-trim error. It appears that neither type of error can be reduced without extensive additional reference history.

- *Use of swap sets*

Use of swap sets and swap faulting is an effective compromise between demand paging and swapping entire working sets. The HPO swap sets show a good locality of reference. This is a requirement for successful blocking of pages into swap sets. The locality of reference in the swap sets is a function of both the reference behavior of the virtual machines and the affinity criterion used to create swap sets.

- *Swap-set affinity*

The swap-set affinity is important in creating a higher locality of reference within a swap set than within the entire working set. Virtual-address affinity is much better than random affinity. The data do not allow any conclusions on other affinity criteria.

- *Swap-set reuse*

Typically about 90 percent of the swap sets are reused in the transaction following their creation. The total number of swap sets in the system remains fairly constant.

- *Trim sets*

Sets created from trimmed pages tend to be small and have poor affinity. Their rereference behavior is clearly different from that of swap sets. Demand-paging them appears to be the correct choice.

Acknowledgments

We would like to express our appreciation to Tom Beretvas, Larry Brenner, Paul Van Leer, and Jerry Spivak. Through their discussions of the algorithms, the analysis, and the results, they contributed greatly to this paper. We would also like to thank David Potter, who helped us untiringly in dealing with the VM/Monitor data and with his support of the GRIN system.

References and note

1. P. J. Denning, "Working Sets Past and Present," *IEEE Trans. Software Eng.* **SE-6,** No. 1, 64–84 (January 1980).
2. M. C. Easton and B. T. Bennett, "Transient-Free Working-Set Statistics," *Commun. ACM* **20,** 93–99 (February 1977).
3. A. P. Batson and W. Madison, "Measurements of Major Locality Phases in Symbolic Reference Strings," *Proceedings, ACM Sigmetrics and IFIPS W.G. 7.3 Conference*, March 1976, pp. 75–84.
4. K. W. Peter Koeppel, "Measuring VM Systems—An Introduction to a Systems Perspective," *Computer Measurement Group Trans.* **14,** No. 53, 27–40 (1986).
5. Richard J. Miles, "VM/HPO Scheduler Overview," *Computer Measurement Group Trans.* **14,** No. 53, 41–44 (1986).
6. William Tetzlaff and Thomas Beretvas, "Paging in VM/370 Operating Systems," *Computer Measurement Group Trans.* **14,** No. 53, 65–76 (1986).
7. T. Beretvas and W. Tetzlaff, "Paging Enhancements in VM/SP HPO 3.4," *Technical Bulletin GG22–9367*, IBM Washington Systems Center, Gaithersburg, MD, May 1984.
8. The Conversational Monitor System (CMS) is a conversational operating system designed to run under the VM control program (CP).

9. *VM/SP HPO System Programmer's Guide*, Order No. SC19-6224, available through IBM branch offices.
10. D. Potter and W. Tetzlaff, "Generalized Reduction of Information," *Research Report RC-6676*, IBM T. J. Watson Research Center, Yorktown Heights, NY, June 1977.
11. G. Burkland, P. Heidelberger, P. Welch, L. Wu, and M. Schatzoff, "An APL System for Interactive Scientific–Engineering Graphics and Data Analysis," *Proceedings of APL 84*, Finland, June 1984.
12. M. G. Kienzle, J. A. Garay, and W. H. Tetzlaff, "Analysis of Page-Reference Strings of an Interactive System," *IBM J. Res. Develop.* **32,** No. 4, 523–535 (July 1988).
13. W. Tetzlaff. T. Beretvas, W. Buco, J. Greenberg, D. R. Patterson, and G. A. Spivak, "A Page-Swapping Prototype for VM/HPO," *IBM Syst. J.* **26,** No. 2, 215–230 (1987).

Received May 13, 1987; revised manuscript received August 29, 1988; accepted for publication November 9, 1988

William H. Tetzlaff *IBM Research Division, T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Mr. Tetzlaff joined the Service Bureau Corporation in 1966. In 1969, he joined the IBM Research Division. He has done research in the areas of information retrieval, system performance, capacity planning, file systems, and paging subsystems. Mr. Tetzlaff has published many papers on that research, and has received two IBM Outstanding Contribution Awards for his work, the first for the Statistics Generating Package and the second for a prototype page-swapping subsystem for VM. He is a frequent speaker on system performance and paging at SHARE, GUIDE, and CMG meetings. Mr. Tetzlaff studied engineering sciences at Northwestern University, and he is a graduate of the IBM Systems Research Institute. He is currently Manager of Operating System Structure and File Systems in the Computer Sciences Department.

Martin G. Kienzle *IBM Research Division, T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Mr. Kienzle received a Diplom in Informatik from the University of Karlsruhe in 1976, and an M.Sc. in computer science from the University of Toronto in 1977. He joined IBM in 1978 at the T. J. Watson Research Center in Yorktown Heights, where he has been working on performance measurements for operating systems, and on operating-system structures. Mr. Kienzle is currently the manager of the Supervisor Kernel Studies group. His main interests are in operating-system structures and primitives for multiprocessors. Mr. Kienzle is a member of the Association for Computing Machinery and the Institute of Electrical and Electronics Engineers.

Juan A. Garay *Computer Science Department, The Pennsylvania State University, University Park, Pennsylvania 16802.* Mr. Garay received the degree of Electrical Engineer from the Universidad Nacional de Rosario in Rosario, Argentina, in 1975, and the Master of Electronic Engineering degree from the Netherlands Universities Foundation in Eindhoven, the Netherlands, in 1981. He also worked for IBM in Argentina as a systems engineer. At present, he is a doctoral candidate in computer science at The Pennsylvania State University. In 1985, he spent six months at the IBM T. J. Watson Research Center, Yorktown Heights, New York, working on the page-reference-pattern problem.