A page-swapping prototype for VM/HPO

by W. H. Tetzlaff
T. Beretvas
W. M. Buco
J. Greenberg
D. R. Patterson
G. A. Spivak

This paper discusses a series of changes that were made to a system running the Virtual Machine/System Product with the High Performance Option to enhance paging. The motivation and the background for these enhancements are discussed, and the design of a series of experimental paging subsystems is described and contrasted with the old design: specifically, the new algorithms for main memory management, block paging, working set identification, trimming, prepaging, page replacement, page-out device selection, and page-out slot selection. The performance impact of these changes is illustrated by results of benchmark measurements, which are then contrasted to measurements without the enhancements. Some things learned in running the prototype are discussed and conclusions drawn.

The productivity of Virtual Machine/System Product (VM/SP) users is improved through lowered system response times.[1,2] Analysis of VM/SP systems frequently shows that the greatest leverage in improving response times comes from improving the paging subsystem.[3] This paper explains a series of experiments conducted with the paging algorithms using a VM/SP, or VM, system with the High Performance Option (HPO). These experiments and prototypes led directly to the HPO Release 3.4 system paging enhancements.

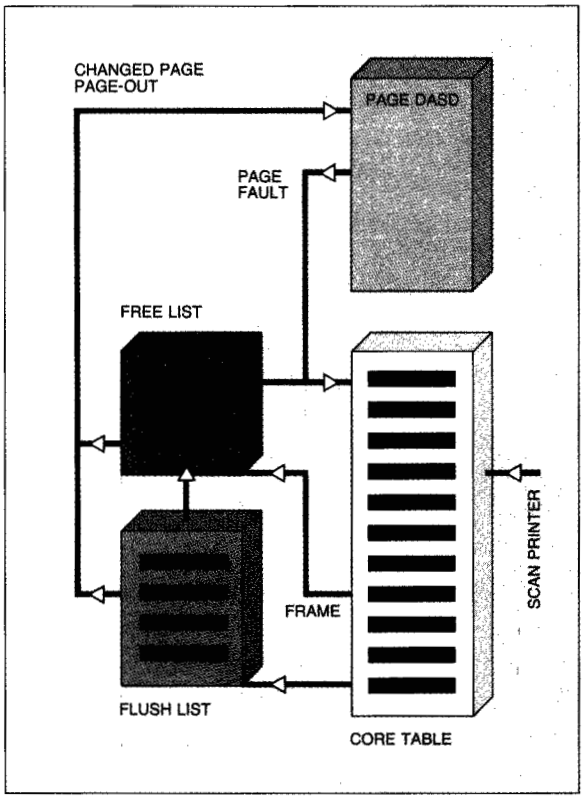
We have studied VM paging extensively[4] on a number of real systems. These studies indicated some problem areas, but more importantly, they suggested that interactive users had working sets that were largely repeatable across transactions. These studies then led to some small experiments on real systems. Next, an extensive prototype was coded and benchmarked. Finally, the prototype was used in production on one of the VM systems at the IBM Research Center in Yorktown Heights, New York.

Work on the paging prototypes was started in 1980, and they became operational in 1981. The performance characteristics were explored during 1981 and 1982. Data were obtained on IBM 4341-2 and 3081-D processors. A prototype production system was operational at the Research Center in 1983 on a 3081K (16MB) processor.

Background

An increasing desire for low subsecond interactive response times led us to analyze existing interactive response times. We found that in many cases the largest component of the response time was paging delay. This finding in turn led us to consider how paging response time could be improved. Our analysis of paging in Conversational Monitor System (CMS)-intensive systems suggested that interactive users had working sets that were repeatable across transactions. We theorized that page reference his-

Figure 1 Paging schema



tory and scheduler information could be used to improve the page-replacement algorithm and consequently to reduce the paging delays contained in the response times.

Another consideration was the tremendous growth in CPU power, "MIPS," or millions of instructions per second, which was not matched by a similar reduction in Direct Access Storage Device (DASD) paging access time. Simply put, paging DASD access time was too long. In order to reduce paging delays, it appeared necessary to move pages into or out of main storage faster than the existing VM demand paging was capable of doing. Since access time reduction was not readily provided by hardware evolution, an appropriate software solution had to be found.

VM[5] classifies users in the multiprogramming set as either queue one (Q1) or queue two (Q2). The terminology comes from the fact that they are conceptually on one of two lists or queues. Users are normally placed in Q1 at the beginning of a transaction and later moved to Q2 if they consume a specific amount of CPU time (called a Q1 time slice). When a user completes his transaction, the virtual machine goes idle, and the machine is said to be dropped from queue. If the scheduler chooses to remove a virtual machine from the multiprogramming set, the machine is said to be involuntarily dropped from queue.

During our analysis of the system, we found several cases where users were unnecessarily dropped from queue during the life of an essentially Q1 transaction. In the prototype (and subsequently in the product) these "false" queue-drops were eliminated. Thus, the Q1 drop counts more closely correspond to actual transaction counts. Experience has shown that while the real response time is not affected significantly, the Q1 drop rate may be reduced to as little as one third of its previous value, and a corresponding threefold increase in the VM Monitor Analysis Program (VMMAP)[6] response time measure (Q1SEC) value may occur.

Figure 1 shows a conceptual paging schema for VM/SP and the HPO (prior to HPO Release 3.4), referred to as VM for short in the rest of this paper.

The algorithm that is used to select a page for paging out in a paging system is called the *page-replacement algorithm.* The selected page frees up a four-kilobyte-size place in main memory, called a *frame.* VM uses a global Least Recently Used (LRU) demand paging page-replacement algorithm. The LRU algorithm[7] calls for removing the page that has gone unreferenced for the longest time. The frame freed up can be used for a newly referenced page that was not in main memory. The LRU algorithm is used because of its ability to predict the page that is most likely to go unreferenced for the longest time in the future. The algorithm is categorized as "global" LRU because all users compete for frames on an LRU elapsed-time basis equally, as one collective working set irrespective of individual working sets.

The free list in VM is a list of immediately available page frames. They are immediately available because the contents are written to backing store as part of the process of placing them in the free list. This list is kept at a size that is equal to the multiprogramming[8] set plus one. The size of the list ensures that if each task requires one free frame, one spare frame is left over. The free list is replenished from the pool of assigned real memory locations whenever it falls below the threshold. Requests for

frames come one at a time because of page faults, so the free list tends to fall one below the threshold. The page-replacement algorithm then has to find one free frame for the free list. Because the threshold is satisfied when only one frame is placed on the free list, it is refilled one frame at a time. If the selected page has been changed since it was last written, the new copy must be written to a paging device. This process in turn leads to writing pages out one at a time. The only chance for multiple-page transfers is when the paging device is overloaded, therefore accumulating a queue of paging requests. In this special case VM combines the paging requests into a single Start Input/Output Operation (SIO) if the queued requests are directed to the same paging cylinder.

The first choice for frames to refill the free list is among those on the flush list. The flush list is a first-in-first-out (FIFO) list of frames that belong to users who had been dropped from queue and who are thus out of the multiprogramming set. In normal operations, pages belonging to dropped users are not placed on the flush list; instead they are left in storage so that they reduce the need for paging in the next transaction. Flushing is only invoked as an overload control mechanism. In well-tuned systems the flush list is empty, so that the second choice, the LRU algorithm known as "core table scan," is used.

VM uses an approximation to the global LRU algorithm in selecting a page to be removed from main memory to make a page frame available. The VM algorithm is sometimes called a one-bit approximation to LRU, because the page reference bit is treated much like a two-state unreferenced interval timer.

The LRU algorithm usually provides a frame belonging to an out-of-queue user. If a frame belonging to an in-queue user is taken, this action is called a "page steal."

The core table scan process has a pointer steadily advancing through a map of main storage, the "core table," resetting the reference bit associated with each frame. If the reference bit of a page is not set, the scan process moves the frame onto the free list, thereby making the frame available for other use. Having completed its task, the scan is then suspended. The next scan will pick up at the next frame. The time taken through the entire core table is called the scan period.

The effect of the core table scan is that in-queue users, who reference their pages, tend to maintain their frames, and out-of-queue users tend to lose their frames. Because of the random distribution of pages in the core table, a user tends to lose only one page at one time. Since the free list requires replenishing whenever it falls below its threshold, pages are usually moved onto the free list one at a time.

In VM, the process of removing a virtual machine from the multiprogramming set is called a queue-drop. At queue-drop the individual page reference bits of pages belonging to a user are reset, thereby making the frames available the very next time the core scan analysis examines them. Thus, all of the user's pages will be taken in one full core scan period, and the average page life is one half of the core scan period for queue-dropped users. For users who are in the multiprogramming set (in-queue users), it takes one full core table scan period to reset all the reference bits associated with their (usually recently) referenced pages. During a subsequent scan, the now-unreferenced pages are taken away, resulting in an average page life of 1.5 scans for members of the multiprogramming set. Therefore, the LRU algorithm is biased strongly toward in-queue users. But note that out-of-queue users are allowed to hold pages for a while, depending on the speed of the core scan. The totality of frames belonging to users dropped from queue is called the paging buffer, and specifically, the set of pages belonging to former Q1 users is called the *Q1* or *interactive buffer.*

Flushing is a mechanism that VM may invoke at queue-drop time.

In VM the device for a page-out allocation is selected on a basically *round-robin* basis; i.e., each device is selected in turn. The paging algorithm remembers the last cylinder used on a paging device, and it attempts to find an unused slot for a page-out on that cylinder.

If it cannot do so, it locates the available slot closest to the center of the volume. If the center of the volume is within the paging area, the pages will be

clustered around that point. If the paging area does not include the center cylinder, the pages will be packed at the edge nearest to the center. The intention of this algorithm is to reduce seek distances on the device by clustering the pages. The algorithm is sometimes called *zigzag*, because of the tendency to move back and forth across a volume when the paging area spans the center of the volume. In practice, the very middle area of the device is fully allocated with pages. New allocations and page reads tend to come at the edges of the area.

Simple tuning experiments. Trying to decide how to improve VM paging, we evaluated the results of several experiments. These experiments are interesting, even though they did not yield positive results.

Flushing experiments. "Flushing" is a mechanism that VM may invoke at queue-drop time if the system is experiencing paging problems. The exact feedback algorithm that triggers flushing is unimportant to this discussion. When flushing is enabled, the pages of users are placed on the flush list when the user is dropped from queue. The flush list is used as the first source of frames. If it is empty, core table scan provides the frames. Flushing has the effect of protecting the in-queue users from core table scan by making all of the frames of out-of-queue users immediately available. Since the flush list is a FIFO list, it is essentially an LRU algorithm applied to dropped users.

Some installations have found that their systems are more stable if the flushing mechanism is disabled. It is possible that the flushing mechanism actually made the system worse by overprotecting the pages of in-queue noninteractive users at the expense of interactive users.

In order to better understand flushing, we modified a real system (with real users) so that it flushed all the time. The rationale was that if flushing was a good idea with heavy loads, it should be reasonable with lighter loads. In fact, the change caused an unsustainably high paging load. The users demanded an immediate removal of the change.

Changing the flushing algorithm to "last in first out" (LIFO) produced the same adverse result.

We concluded that flushing is inappropriate to use as an overload control mechanism. If it has a place, it is to protect long-running batch, service, or guest machines by sacrificing the interactive users.

Reference bit resetting experiments. At queue-drop time the reference bits of the pages of dropped users are reset. Resetting has the effect of biasing the LRU algorithm so that it will take these pages earlier, yielding a form of protection for the in-queue users, though one that is much less severe than the flushing just described.

In the next experiment we disabled the resetting of the reference bit at queue-drop time. The idea was that the interactive users could hold their pages potentially longer between transactions; they could get absolute protection for one core scan time.

Performance became worse. The core scan rate became much greater, with the core scan time appropriately reduced. The reduction in paging for the interactive users was more than offset by the increase in paging by the in-queue users. The system was characterized by a higher rate of stealing of pages from in-queue users. Although it was possible to run the system for a whole day without removing the change, it was clear that the change was not beneficial.

The core scan rate became faster because of the increase in page residency of the interactive users. With the modification, the working set of a queue-dropped user can stay resident for 1.5 times the core scan time instead of 0.5 times the scan time. With an increase in the residency of these pages, other pages must be paged out sooner. The core table scan algorithm reacts automatically by scanning more pages, thus reducing the core scan period. The result was higher CPU overhead for the page-replacement algorithm and a higher overall paging rate.

Observations on VM paging. VM achieves a very delicate balance between the protection of the pages belonging to the in-queue users and the need to buffer interactive user pages, from transaction to transaction. Attempts to shift the balance either toward the in-queue (continuous flushing) or toward interactive users (no reference bit reset) resulted in higher paging rates and poorer performance.

Although it is attractive to do paging experiments on a real system, they must be preceded by controlled experiments with a known workload and no real users to be disrupted.

Prototype experiments

Our basic objective was to improve interactive (CMS) response time by reducing page waits and making

the paging subsystem more efficient, especially for large systems.

A series of prototype experiments was started when we discovered indirect evidence in VM/Monitor data[9] that CMS working sets were highly repeatable. The prototypes aimed to exploit this observation.

Fortuitously, high-data-rate movable-head direct access devices (i.e., the IBM 3380 DASD) became available in this time frame. The prototypes tried to make use of the fast data transfer capability of these DASD devices by using page-blocking techniques.

Working set considerations. One observes that VM systems with more storage page less. One may assume that this occurs because fewer pages are stolen from active users. In fact, the pages belonging to queue-dropped users are retained in main storage and are reused during the next transaction. Traces of user activity demonstrated that the more frames a CMS user still owns in main storage at queue-add time (when the user becomes active), the fewer page faults the user encounters. Figure 2 illustrates this phenomenon. We concluded that CMS users have a repeatable working set, i.e., one that goes across "transactions," which are queue-stays (not CMS transactions). From this conclusion we inferred that explicit prepaging, i.e., fetching the pages belonging to a user's working set all at once, instead of one at a time, might work.

DASD considerations. The use of single-page SIOs is not very efficient for DASD (or for the CPU), so we decided to make an attempt to use "block paging" when possible.

Figure 3 illustrates the conceptual advantages of block paging. Each SIO operation incurs an "overhead" A, consisting of control unit protocol time, seek to the cylinder sought, rotational latency to reach the desired page, and potentially additional lost rotations if the path is busy when the attempt is made to reconnect to do the data transfer (RPS miss). Thus, in the case of a single page being transferred per SIO, there is one overhead A associated with each page transfer B. In contrast, when multiple consecutive pages are transferred with a single SIO operation, there is a single overhead A associated with multiple Bs. Clearly, the importance of the overhead diminishes.

Table 1 shows the results of *modeling* the comparison for demand and block paging use of 3380s. The left column indicates that when 3380s are used in demand paging mode, a paging response time of 29 milliseconds (ms) is obtained with 15 pages per

Figure 2 The importance of resident pages

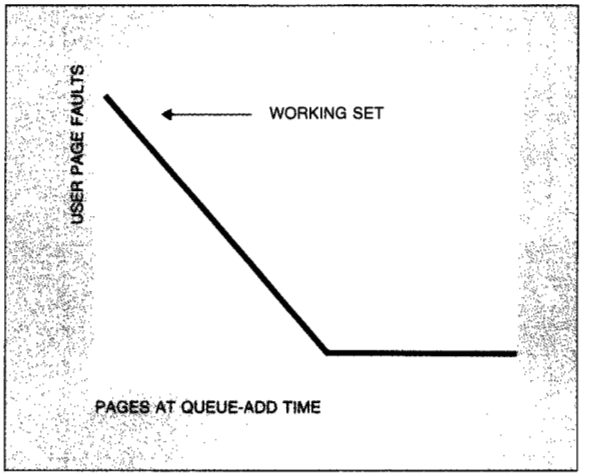


Figure 3 Advantages of blocked I/O operations

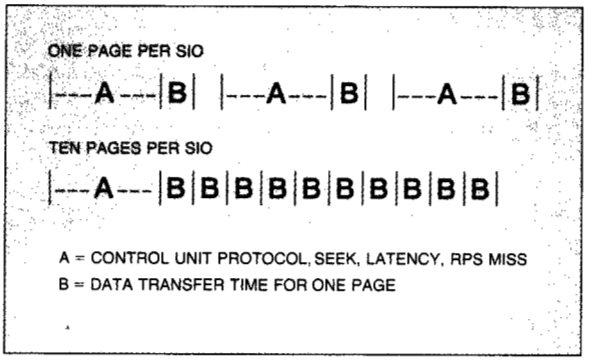
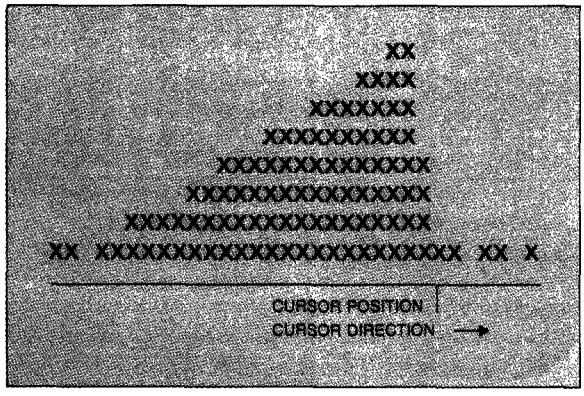


Table 1 3380 Paging performance in demand and block paging modes

	Demand Paging	Block Paging
Pages per SIO	1	10
Actuators per path	4	2
Paging rate per path	60/s	200/s
Paging rate per actuator	15/s	100/s
SIO response time	29 ms	48 ms
Average time for one page	29 ms	4.8 ms

Figure 4 Allocation density by cylinder



second (s) per actuator, and four actuators per path. The right column indicates that with block paging (10 pages per SIO) 100 pages per second per actuator per path can be supported with only two actuators per path, and the SIO completion takes 48 ms, i.e., 4.8 ms per page. Clearly, blocking is advantageous.

Its effect is to essentially establish "big pages," since the result is as if a single big page were transferred when multiple related pages are transferred with a single SIO.

Note that a granularity advantage is obtained in comparison with a scheme that used genuine big pages, since the individual small-page components of this big page can be (are) changed individually with time, which would not be true for a single big page.

Real storage use considerations. In VM, pages belonging to inactive users are retained in main storage without any consideration as to whether such a user is likely to be dispatched or the page is likely to be used. We change this situation by more closely relating real storage use and prepaging to scheduling, which is done by letting the scheduler provide frequent hints to the paging subsystem.

In VM, the size of a user working set was indirectly estimated in a way that was dependent on the workload. We reduce this workload dependency by determining the real working set of users, which is done by trying to identify the actual working set pages. Furthermore, if main storage is overcommitted, we keep only the "needed" working set pages for scheduled users. Specifically, the number of in-queue users is controlled on the basis of storage use, thereby avoiding a strong overcommitment of main storage.

Explicitly maintaining an interactive or Q1 buffer by holding Q1 users' working sets in memory as long as possible after queue-drop is an attempt to improve interactive response time.

Round-turkey prototype. The page-allocation algorithm that places pages into paging areas on DASD is changed in this prototype in order to encourage the frequency of block page-outs. Our goal was to write multiple pages on each device as it was selected in turn; we aimed to reduce

- The number of I/O operations initiated
- The number of times that disk arms need be moved
- The distance when they are moved

Three algorithms were changed: the ones for device and slot selection and the one for handling of unchanged pages.

Device selection. VM selects a device to be used for page-out by placing one page on one device and placing the second page on the next device on a round-robin basis.

The basic concept of the new device-selection algorithm in the prototype is to select the same page-out device (up to) eight times, unless a cylinder boundary is reached. The repeated selection of the same device allows the grouping of multiple page-out requests within the same SIO operation, resulting in blocked page-out requests. Thus, in view of the previous terminology, the new algorithm is seen as a bigger but slower-moving bird—a "round turkey." The prototype itself took its name from the device-selection algorithm. Note that page reads remain unblocked.

New slot-selection algorithm. The zigzag slot-selection algorithm of VM is replaced with a "moving-cursor" algorithm. This algorithm maintains a pointer that moves across the paging area. The cursor points at a slot that is to be used for allocation if the slot is free. The cursor is advanced steadily across the paging area, as required by the allocations taking place. The expectation is to find empty, unused areas ahead of the cursor. This construction was devised to provide a high probability for the existence of contiguous empty slots, which in turn provides effective blocked page-outs. It also reduces nonzero seek frequency and seek distances. Experience indicates

that as the allocation cursor is advanced, a wave of page references moves across the DASD volume. The allocation occurs ahead of the cursor, and page reads occur behind the cursor, but the band of pages referenced frequently is rather narrow; i.e., there is a well-defined temporal locality of reference. The cursor is reflected at the end of the paging area to avoid long seeks which would be necessary if allocation were then to return to the beginning of the area. Figure 4 shows the density of the pages allocated to cylinders in the paging area at a point of time.

We designed the prototype so that unchanged user pages are paged out.

Writing unchanged pages. VM (as do other demand paging operating systems) retains the backing slot location of page frames that are brought into main memory. If the frame is needed later, the contents need not be written out if the page has not been changed. This algorithm reduces the number of I/O operations required.

We designed the prototype so that unchanged user pages are paged out in order to aid the moving-cursor-allocation algorithm. It is intended to move referenced but unchanged pages along with the moving wave of pages across the paging area. This reduces the need to move the disk arm to a random point away from the current arm position when such a page is later referenced. Thus, arm movement is reduced at the expense of a higher page-out rate. Since page-outs are blocked, this design seemed to be a good trade-off.

Importance of contiguous slot allocation. An experiment was done in which the moving-cursor algorithm was disabled. The system was changed to allocate at the right edge of the paging area. This allocation is very similar to the one that VM does within an off-center paging area. The scheduling of multiple page-outs to a device was still attempted, but the lack of contiguous allocation caused scattered allocation. As a result, the use of a single channel program for the set of pages to be written was prevented. Performance of the system with this change was worse than that of the base system.

Experimental results. The combination of round-turkey and moving-cursor algorithms tends to result in an effective page-out blocking method. Care must be used that the paging area is large enough to allow contiguous page allocation, yet small enough that seeking does not become excessive. If the area is overly large, the occasional seek to a page at a random point in the area may be a long distance from the cursor. It was shown that the contiguous slot allocation was critical to the other algorithms in the prototype. The prototype illustrated reduced response times, reduced frequency of paging SIO operations, and improved throughput.

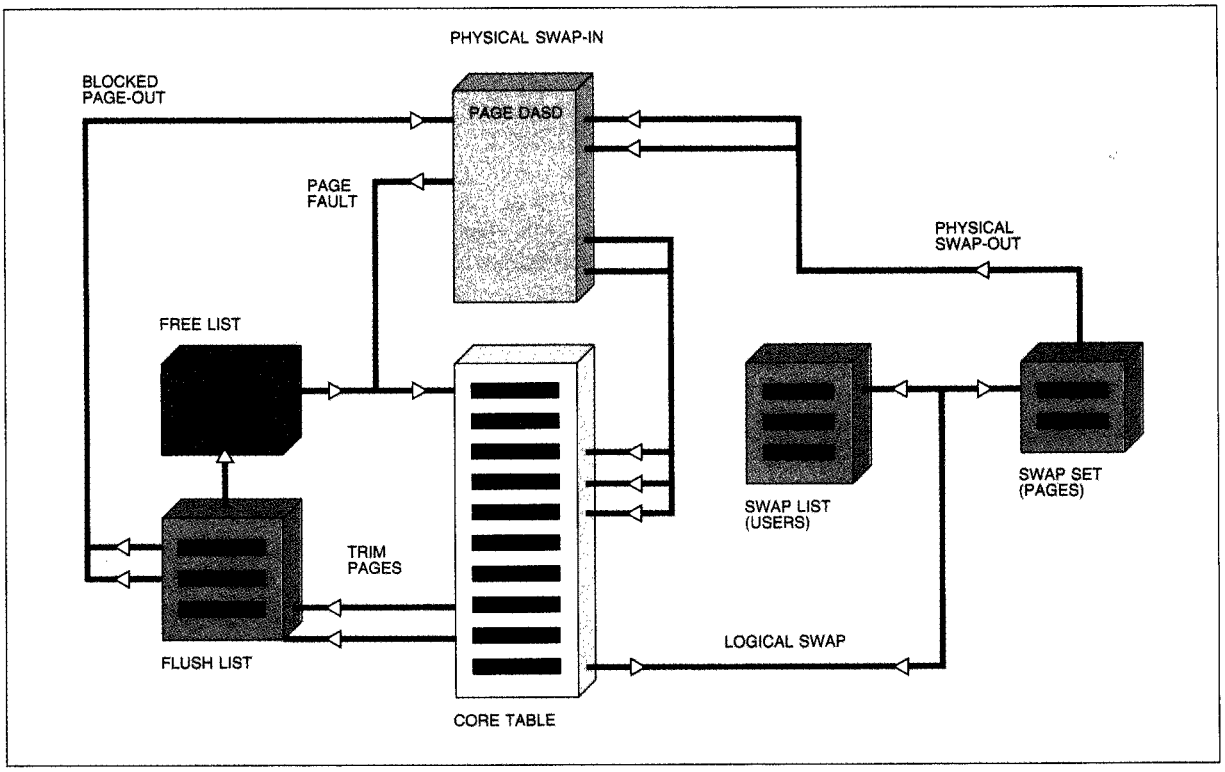
Prepage prototype. Figure 5 shows the paging schema in the prototype system. A brief overview of the schema is given first, followed by a detailed discussion of some aspects. The left side of the diagram is very similar to the demand paging subsystem in VM, illustrated in Figure 1. When a page fault occurs and the page resides on a paging DASD, a frame is obtained from the free list; then a page-in operation for one page takes place in the normal fashion into this frame. Page-outs are performed in a similar fashion, but now they tend to be blocked, in contrast to VM.

The right side of the diagram represents the new swapping operations. When a user reaches the end of its queue-slice and is queue-dropped, the user's working set pages are "logically swapped," which means that under normal circumstances they are retained in main storage. Pages not in the working set are "trimmed"; i.e., they are moved onto the flush list, which is the first source of frames with which to replenish the free list. Thus, in the experimental system, flushing is a normal, expected operation.

Obviously it is not possible to maintain the working set of all users in main storage forever. When a page frame shortage is detected, the honeymoon of a logically swapped user comes to an end, and physical swap-out takes place, releasing some of the user's frames for other use. In other words, physical swap-out is then used to replenish the free list.

Physical swap-out means that the next user on the swap list is identified, and some of his working set

Figure 5 The experimental prepage system schema



pages are swapped out. The next physical swap-out operation required because of frame shortages forces more of the working set pages of this user to be swapped out. A user is always totally swapped out before another user is selected from the swap list as a candidate for physical swap-out.

Users are placed on the swap list in FIFO order as they drop from the active queue. However, there is a separate swap list for Q2 and Q1 dropped users, and Q2 users are physically swapped out before Q1 users, thereby providing preference to Q1 users.

Sometime later a swap-in operation will sequentially read back the swapped-out working set pages of the user.

Relationship to scheduling. In the earlier paging systems most of the concern for the management of page frames centered around identifying working set frames for a running program. In interactive systems a slowly reacting (at least in machine terms) user appears as an I/O device to the system. Since it has become clear that working set pages span transactions in VM, it is desirable to have the scheduler provide additional hints to the paging system.

The scheduler needs to tell the paging subsystem when transactions start and stop, when the scheduler temporarily suspends them, when time slices end, and whether they are interactive or long-running. These hints are used by the paging system in managing real memory, in the page-replacement algorithm, and in determining working set pages. In order to provide these hints a number of scheduler changes had to be effected.

Working set identification. At queue-drop the experimental system identifies all nonshared pages referenced during the previous queue-stay. These pages can be considered to be the real working set pages[10] with a fixed delta tau (using Denning's terminology) equal to the time slice. The reference bits of these pages are reset. It is immaterial whether these referenced pages are changed or unchanged, and by our definition these are "needed pages." These pages are

logically swapped, whereas the "unneeded" pages are trimmed to the flush list.

In order to identify working sets on a transaction basis, a significant change was made to the management of page reference bits. It is important that reference bits not be reset by an asynchronous function (the core table scan) during the processing of a trivial transaction. Such resetting would make pages appear to be unreferenced at the end of the transaction, thus removing them from the working set. The system was changed to cause reference bit resetting at queue-drop time instead, as a part of working set determination.

Swap-out operations. Physical swap-out means that the next user on the swap list is identified, and one or more of his swap sets is swapped out.

The so-called swap sets are formed by grouping the logically swapped (working set) pages of a given user in order of virtual addresses into sets of pages. Thus, swap sets are related by *virtual address* and *time of reference*, since all the pages in a swap set were referenced during the same queue-stay. The pages in a swap set are written together on a paging device. In most cases the allocation yields consecutive slots on the device. The size of a swap set is an adjustable system parameter. A swap set exists on a paging device until it is swapped in, after which the affinity of the pages is lost.

After writing a swap set, the user will have one or more swap sets on the paging device, but the user may still have other frames in main storage, still in logical swap status. Thus, a user can be *partially swapped.*

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.

Swap-in operations. When an interactive user becomes ready again, preliminary to being queue-added, this condition normally causes a page fault to take place, usually for the first page in the user's address space. If the page fault occurs to a page in a swap set, the entire swap set is swapped in. This procedure, called "swap faulting," provides the blocking advantages described earlier. The swap-in operation will sequentially read back all the pages in the swap set into nonadjacent frames of main storage. At this time the locations of the slots on the paging device are made available for reuse. The process of bringing in all pages of a swap set is a form of prepaging. It is assumed that the time of last reference and the virtual address affinity of the pages predict a likely future use of the other pages. Should any of the pages not be referenced, they are trimmed at the next queue-drop. If a swap set is not brought into main storage during a queue-stay, its identity can be retained across multiple queue-stays. In most instances, however, especially for interactive users, a swap set exists only between consecutive queue-stays. Swap sets are really *big* pages, since they are always written and read together, and a page fault to any one, a swap fault, causes the entire swap set to be read.

The process of bringing in all pages of a swap set is a form of prepaging.

After a swap-in operation the referenced bits of the pages swapped in are reset in order to avoid misleading reference indications. The user's continued execution will later cause setting of most page reference bits, but not all pages swapped in are subsequently referenced. The unreferenced pages in referenced swap sets are called prepage errors. These are page reads that a demand paging system would not have done. They represent the change in content of working sets through time. At the next queue-drop the unreferenced pages will be flushed and paged out.

Logical swapping. At queue-drop the user's working set pages are "logically swapped out"; i.e., his page tables are invalidated, and the user is placed on the swap list. Under normal circumstances the user's, especially an interactive user's, working set pages therefore are retained in main storage, even though the page tables are invalidated. The totality of such working set pages belonging to interactive users represents the interactive buffer.

The most important source of pages for a transaction is those pages that have been retained in main mem-

ory from the last transaction; i.e., they were logically swapped out. When a transaction starts, all the user's logically swapped-out pages are immediately returned to him with the reference bits turned off; i.e., these pages are logically swapped in. The logical swap-in, unlike physical swap-in, returns all of the prior working set to the user. In a well-tuned system, most transactions complete without the need for any physical swap-ins. At the next queue-drop any remaining unreferenced pages are paged out. These pages were present in the prior working set, but are not a part of the current working set.

Multiprogramming level control. In VM, users were queue-added if the projected working set of the user plus the sum of the working sets of in-queue users (SUMWSS) amounted to less than the Available Frame Count (APAGES). Q1 users were given preference because APAGES was increased by a factor of 1.25 for Q1 users; thus they rarely noticed the reality of main storage constraint. The calculation ignores the requirement for an interactive buffer. Some installations attempted to obtain that by artificially lowering APAGES.

We concluded that VM tended to place an excessive number of Q2 users into the multiprogramming set. The prototype was used to evaluate the possible benefit associated with regaining the frames of some of these Q2 users. In order to regain the frames, a fixed upper limit on the number of Q2 users was established. The number was experimentally set to a point that reduced the number of Q2 users requiring resident pages but still prevented throughput reduction because of a lower Multi-Programming Level. If this artifice was found to be effective, a feedback mechanism would be created in the future.

Minimum working set. A default systemwide minimum working set size of two swap sets is established for users. This means that if the user has at least two swap sets' worth of pages at queue-drop time, his working set will not be trimmed below that value, and at least this much will be physically swapped out if physical swap takes place. Normally noninteractive users have many more pages than two swap sets' worth (i.e., 16 pages if a swap set size of eight is used), so the effect of this value is to guarantee less trimming for interactive users, which translates into more physical swapping and less demand paging.

Replacement algorithm. The effect of the various modifications is to convert the page-replacement algorithm from a global LRU page replacement to a hybrid. Swap pages of dropped users and trim pages are handled in FIFO queues, and are thus global LRU. The page-replacement algorithm for in-queue users becomes oriented to the working set, since at each reexamination of a user by the scheduler, the unreferenced pages from the last examination are trimmed. All reference bits are then reset so that trim and working set pages can be determined on the next examination.

Prepaging at queue-add. In order to take advantage of the *parallelism* of swap paths, the experimental system provides for a minimum swap-in size of two swap sets for interactive users. This means that when an interactive user becomes ready, two of his swap sets are prepaged, i.e., swapped in. The experimental system does no prepaging for noninteractive users. It is important to limit prepaging: If all the swap sets were always swapped in or out at once, a large user, e.g., a Multiple Virtual Storage (MVS) guest, could easily overwhelm the paging subsystem.

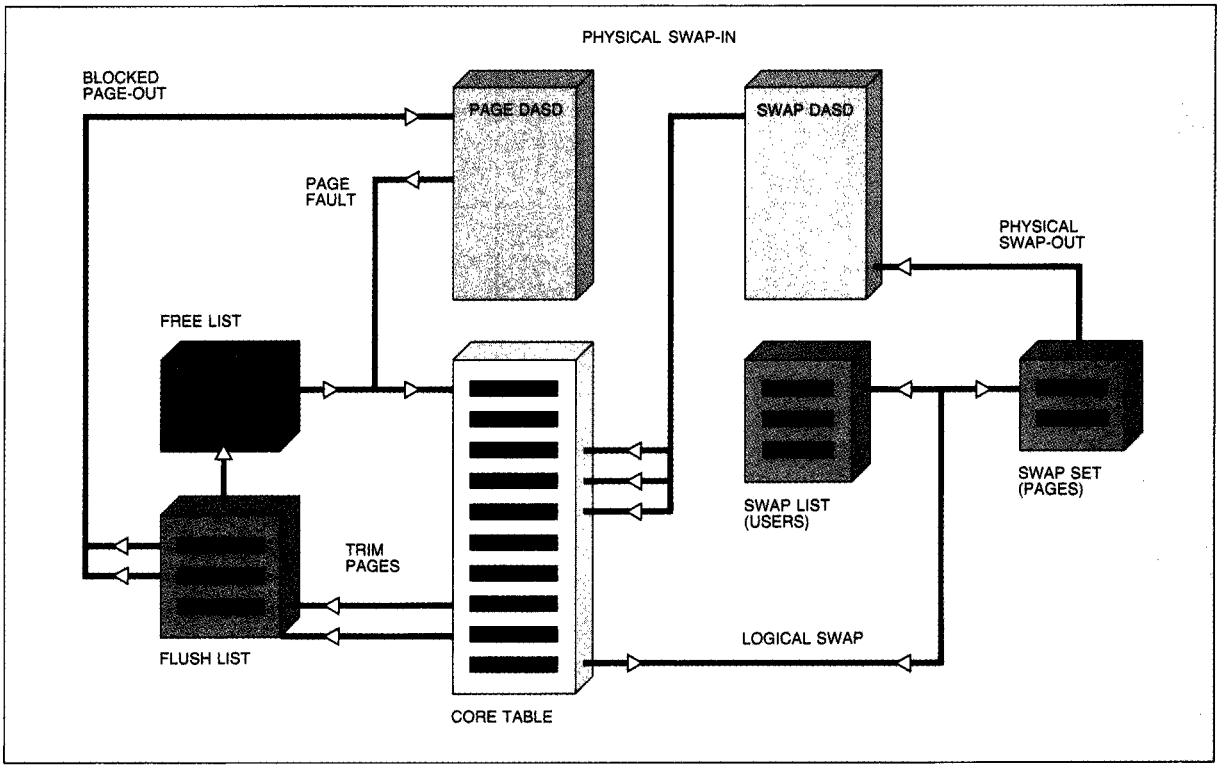
Free list replenishment. During early runs with the prototype we observed that the free list was frequently empty. This problem is serious, because it leads to waiting for page-out completion before page-ins can be scheduled. The correction is to increase the minimum size of the free list.

Full prepaging experiment. An experiment was done in which the working sets of all Q1 and Q2 users were *fully* prepaged at queue-add. The system showed very high swap rate and poor performance. This amount of prepaging brought in pages much earlier and brought in whole swap sets that were ultimately not referenced at all. Such unnecessary use of main memory caused contention for memory which resulted in high swap rates. We observed that users with large working sets would fully consume the free list in order to prepage all of the swap sets. Thus, the idea of full prepaging was abandoned.

Results. In the experimental system, physical swap-out, not core table scan as in VM, is the major source of frames for the free list. If for some reason swap-outs do not provide enough frames to replenish the free list, user page "stealing" (via core table scan) remains as the mechanism of last resort. If core table scan is invoked frequently, as demonstrated by a high steal rate in VMMAP, the swap mechanism is not functioning properly or the system is running out of memory below the 16-megabyte line.

Since most of the time swap-in operations require more than one frame at a time, the free list had to

Figure 6 The experimental system paging schema



be made larger than before by requiring a higher minimum.

This prototype was characterized by an increased paging rate and a dramatic reduction in trivial response time. In addition, there was a small increase in CPU time in supervisor state, due to increased management of paging and the higher paging rate.

Contiguous block page prototype. The contiguous block page prototype was characterized by the addition of special paging areas for the placement of swap sets. This addition was made in order to ensure contiguous allocation and a single SIO for each swap set.

Figure 6 shows a conceptual schema of paging in the experimental Swap Prototype system. The diagram is very similar to the demand paging subsystem in VM, illustrated in Figure 5. The special data sets for contiguous allocation of swap pages have been added to the diagram.

Two new algorithms were added to this prototype. The first was for management of swap areas, and the second was for demand-paging certain pages that are not subject to swapping.

Contiguous page datasets. Contiguous page datasets, or swap areas, are used to contain working set pages. The format is the same as for any other paging area in the system. The moving-cursor algorithm is used for allocation. The big difference is that the number of pages allocated at one time is equal to the size of a swap set. Thus, a single SIO is all that is ever needed to read or write a single swap set.

Garbage collector. Pages belonging to virtual machines are handled by logical swap, trim, and physical swap. There are pages that do not belong in this category, such as

- Shared segment pages
- Control-program-owned pages

Figure 7 Trivial transaction response

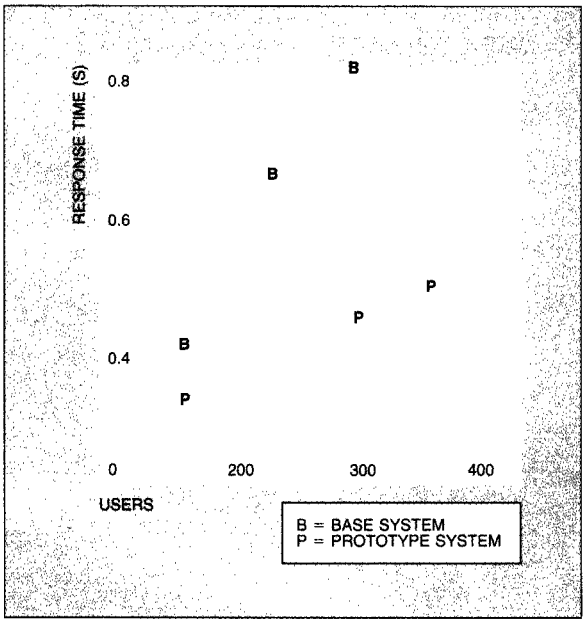


Table 2 Measurements on a 3081D

VM System	Base	Prototype
Users	310	310
Main memory	16 MB	16 MB
CPU busy (per CPU)	100%	100%
VIO rate	377	412
Paging SIO rate	240	136
Percent page wait	14	6
Page I/O time	20	16
Swap I/O time	—	42
Q1 seconds	0.78	0.36
Q1 drop rate	19.4	20.5
Page read per Q1	11.6	1.4
Q2 seconds	10.3	8.6
Q2 drop rate	4.8	4.4
Users in Q1	15	10
Users in Q2	55	40
Swap paging rate	—	720
Demand paging rate	425	50

- User pages not in working set (obtained without queue-add and not trimmed)
- Spool buffer pages

In VM the "core table scan" disposed of these pages. In the experimental system, core table scan is not normally invoked (except as a last resort); thus a new mechanism is required to handle "disposable" pages of this type. A *periodic* core table scan is introduced which looks for these pages. It scans one eighth of the dynamic paging area every four seconds, resets the reference bits of pages of this type, and disposes of unreferenced "nonuser" pages by putting them on the flush list, i.e., effectively establishing a working set for this type of page and trimming the disposable ones. Clearly all such pages are examined in 32 seconds, and the unreferenced page lifetime is 48 seconds on average for these kinds of pages.

Results. This swapping mechanism provides at least two blocking advantages. Since all the pages written by a single SIO are contiguous, device busy time is reduced. Also, the blocking means that the number of SIOs executed is reduced, and thus many CPU cycles are saved. Both a SIO reduction and a CPU reduction (significantly below either the prior prototype or the base system) were found in this prototype. This result was also translated into an increase in throughput. The prototype also showed a yet greater improvement in the interactive response time.

Performance results

Measurements were made on a 16MB 3081D processor with VM/SP HPO 1.0 and compared with the experimental system prototype. In this measurement, the number of users was varied between 125 and 360, and a synthetic benchmark was used with a randomly generated "user think time" averaging 15 seconds.

The paging configuration used was six IBM 2305 Fixed Head Storage actuators on three channels for the VM (base) case.

The paging configuration for the experimental system prototype was as follows:

Swapping:
  Six 3380 actuators on three channels
Paging:
  Two 2305 drums on one channel

There was no other configuration change between the two runs.

The prototype contained code to limit the number of Q2 users to a fixed number. False queue-drops make the average Q1 transaction appear to have

much shorter response time than it really has. The HPO 1.0 system was modified in a way similar to the prototype to avoid false queue-drops in order to make transactions (and measurements) consistent.

Figure 7 indicates little difference between the trivial responses for the cases when the number of users was kept low. For the case with 310 users, the experimental system prototype provided much better response time than the base case, whose response time became unacceptable. The experimental system could still support 360 users, whereas the base case could not.

Table 2 shows a number of important improvements. The most important result is the reduction in Q1 response time (known as Q1SECS in VMMAP). The number of page reads (demand paging) shows a dramatic reduction, which also largely accounts for the reduction in percent page waits. There is a slight improvement in throughput, as indicated by the virtual input-output (VIO) rate and Q1 drop rate increases. The reduction in Q2 drop rate can probably be explained by Q2 transactions becoming Q1 transactions because of less paging. There is a significant reduction in paging SIOs and interrupts, resulting in the CPU saving which led to the throughput improvement. The overall paging rate increased significantly but did not present a problem since the 3380 paths could handle the load. The 3380 disk devices could handle the swap paging rate without any difficulty. The reduction in the number of 2305s did not hurt performance, and perhaps their presence was not essential.

Two sets of measurements were made on a 4MB 4341-2 CPU. User think times were randomly generated to obtain the desired think time averages. In one case, 10 seconds of think time was used with 50 users; in the other case, 15 seconds of think time was used with 75 users. Note that each combination of think time and number of users results in the same system transaction rate.

In all of the measurements, six IBM 3350 DASD actuators on three (nondedicated) channels were used for paging. In the prototype case, three actuators were used for swapping and three for paging. There was no other configuration change of any kind.

Table 3 shows the range of improvements when measurements of VM/SP 1.1 are compared with the prototype. Trivial response time improved significantly, and the throughput improved slightly on the prototype when compared with the base. The throughput improvement can be attributed to the reduction in paging SIOs and interrupts. The overall paging rate increased but did not present a problem.

Figure 8 Response time for the 4341 experiments

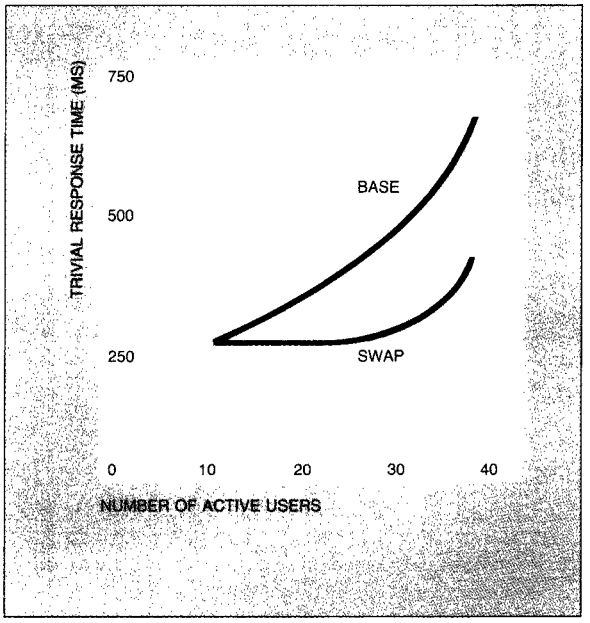


Table 3 Measurement results on 4MB 4341-2

	Improvement
Virtual CPU increase	3 to 4%
VIO increase	4 to 6%
Trivial rate increase	4 to 5%
Scripts completed	4 to 6%
I/O interrupt decrease	26 to 35%
Trivial response improvement	28 to 36%
Paging SIO decrease	52 to 60%
Dispatch decrease	11 to 12%
Paging rate increase	41 to 60%

Figure 8 shows a plot of response time versus *active users*. This plot combined the results of all the IBM 4341 processor measurement runs. The plot shows that response time remained essentially unchanged with the prototype while it rapidly increased in the base case. The plot would indicate that the same response time can be obtained with 20 active users in the base and 30 users in the prototype. This result seems to indicate a capability to support 50 percent more users with the same response time.

Observations based on the prototype measurements. All of the measurements indicate that the experimental system (1) can support a larger number of users than the base, or (2) can provide improved response time, or (3) can offer some combination of both.

Things learned

The experiments with the prototype provided some useful observations:

1. Demand Swap (swap fault) works. There is an indication that it is worthwhile *not* swapping in all of the swap sets at the same time, since occasionally there are swap sets that remain unused. It was beneficial to limit the automatic prepaging at the beginning of a transaction. It limits the loading effect on the swapping subsystem that would be caused by the immediate prepaging of a large number of swap sets. Further, the potential storage pressure is also reduced.
2. Page Steal Rate is a sensitive indicator. When free list value was not set high enough, significant (higher than one percent) stealing occurred. Clearly, in a swapping system stealing is undesirable.
3. Multi-Programming Level (MPL) control is important. When the prototype was initially run without explicit MPL control, too many Q2 users were queue-added. Thus, the prototype was changed to run with a "fixed" Q2 MPL control. Furthermore, the VM/HPO Release 3.4 product was designed to provide dynamic MPL control using the interactive buffer concept.
4. Overall paging rate increases, but demand paging rate is much lower. The paging rate increases (most of paging is due to swapping) partly because of the fact that the increased throughput requires more paging, partly because some of the pages swapped in are not really required, and partly because prepaging causes pages to occupy frames before they are actually referenced, which may increase page residency time. However, in all observed cases, demand paging rate was dramatically decreased, thereby reducing (or eliminating) the need for 2305s.
5. 3380 devices are excellent for swap paging. In the 3081D measurements a paging rate of up to 800 pages per second was easily supported by 3380s. Thus, since the high data rate dominates for swapping load, 3380s are eminently suitable devices.
6. Prepaging large (Q2) users may be destructive. In an early version of the prototype the entire working set was swapped, and the 3350s used could not support the heavy instantaneous paging load that resulted. Also, total swapping created an instantaneous requirement for a large number of free page frames. Thus, swap faults were invented, and prepaging control was introduced.
7. The old size of the free list is inadequate in a swap environment. Prepaging requires that many free pages be available suddenly, and the old free list could not supply frames fast enough.
8. Blocked page-outs are helpful. The round-turkey prototype has shown significant improvements just by blocking page-outs without any other change.

Conclusions

The experimental system paging enhancements improve interactive response time by improving the efficiency of the paging subsystem. The efficiency was improved in several ways. The system now makes better use of real memory and therefore makes some demand paging unnecessary. In many cases multiple contiguous pages are transferred following one DASD access, reducing average access time and DASD busy time.

The experimental system is only an adjunct to large real memory.

The experimental system is only an adjunct to large real memory. Ample real memory remains a requirement for large systems. The experimental system provides the best benefit in heavy paging environments. Although the prototype was aimed at interactive users, it works acceptably for long-running tasks.

The experimental system paging enhancements improve interactive response time, but they also improve throughput because the supervisor CPU time is reduced. The CPU time is reduced as a side effect of the page-blocking techniques which reduced the

frequency of SIOs and I/O interrupts. Also, the reduction in the number of user page-wait conditions reduces scheduler and dispatcher invocations.

Relationship to the HPO Release 3.4 product

The experiences gained in doing this series of prototypes led directly to the paging enhancements in Release 3.4 of HPO.[5,11-13] There are several differences between the prototype and the product.

- The operational and system generation characteristics were improved in the product by making many hard-coded constants into installation-specifiable parameters.
- In the absence of defined swap areas, the product pages out pages that would otherwise have been swapped. At the time of a reference to such a page, the product brings in the particular page, whereas the prototype brought in all of the pages that would have been swapped in. The product has the advantage of behaving in a way more consistent with prior releases in the absence of swap areas.
- The prototype made no change to the selection of members of the multiprogramming set but did put a fixed limit on the size. The product has much more sophisticated algorithms, with appropriate feedback, for determining the size and content of the multiprogramming set.
- The product contains a new parameterized algorithm to select interactive and noninteractive users for physical swap-outs.

Acknowledgments

We acknowledge the contributions of the following people to this prototype: L. Wheeler, R. Bos, G. Lothringer, T. Meggenhofen II, J. Mitchell, and J. O'Connell.

Cited references and note

1. W. J. Doherty and R. P. Kelisky, "Managing VM/CMS systems for user effectiveness," *IBM Systems Journal* **18**, No. 1, 143–163 (1979).
2. A. J. Thadhani, "Interactive user productivity," *IBM Systems Journal* **20**, No. 4, 407–423 (1981).
3. W. Tetzlaff and T. Beretvas, "A new approach to VM performance analysis," *1982 Computer Measurement International Conference XIII*, CMG, Inc., 6397 Little River Turnpike, Alexandria, VA 22312 (1982), pp. 339–350.
4. T. Beretvas and W. Tetzlaff, *Paging in VM/SP*, Washington Systems Center Technical Bulletin, GG22-9319, IBM Corporation (May 1983); available through IBM branch offices.
5. R. J. Miles, "VM/HPO scheduler overview," *CMG Transactions*, No. 53, 41–44 (Summer 1986).
6. *VMMAP General Information*, GC34-2164-1, IBM Corporation; available through IBM branch offices.
7. L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal* **5**, No. 2, 78–101 (1966).
8. The multiprogramming level is the number of in-queue users, i.e., those in Q1 and Q2 (including Q3), shown as Q1 + Q2.
9. *Virtual Machine/System Product System Programmer's Guide*, SC19-6203, IBM Corporation; available through IBM branch offices.
10. P. J. Denning, "The working set model for program behavior," *Communications of the ACM* **11**, No. 5, 323–333 (May 1968).
11. T. Beretvas, *Page/Swap Configurations*, G320-0053, IBM Corporation; available through IBM branch offices.
12. S. Friesenborg, *DASD Path and Device Contention Considerations*, Washington Systems Center Technical Bulletin, GG22-9217-0, IBM Corporation (March 1981); available through IBM branch offices.
13. T. Beretvas and W. Tetzlaff, *Paging Enhancements in VM/SP HPO 3.4*, Washington Systems Center Technical Bulletin, GG22-9367, IBM Corporation (May 1984); available through IBM branch offices.

William H. Tetzlaff *IBM Research Division, Thomas 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 Research Division. He has done research in the areas of information retrieval, system performance, capacity planning, and paging subsystems. He 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 Senior Manager of VM Restructure and File Systems in the Computer Sciences Department.

Thomas Beretvas *IBM Data Systems Division, P.O. Box 390, Poughkeepsie, New York 12602.* Mr. Beretvas received the B.Sc.(Eng), the M.S.E., and the E.E. (Prof.) in electronic engineering from University College of London, Princeton University, and Columbia University, respectively. He joined IBM in 1964 and has been working in the operating systems performance area since 1970. He specialized in paging and DASD performance and received an IBM Outstanding Contribution Award for a prototype page-swapping subsystem for VM. Currently, he is a senior programmer in the MVS systems area. Mr. Beretvas is a frequent speaker at SHARE, ECOMA, and CMG meetings. He has published more than thirty papers and monographs. He has been a member of ACM and IEEE since 1960 and has been an ACM National Lecturer since 1983. He was elected a director of CMG in 1985.

William M. Buco *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Mr. Buco is manager of Systems, Operations, and Services at the Research Center location in Hawthorne, New York. He joined IBM in 1974 as a VM systems programmer at the Yorktown Heights location of the Research Division. He worked on improving the performance and reliability of VM/370. In 1977 he became the manager of the VM/370 Systems Programming project. In 1983, he became the technical assistant to the I/S Director of the Research Division. He was promoted to his present position in September 1984. Mr. Buco was the recipient of an IBM Outstanding Contribution Award for a prototype page-swapping subsystem

for VM. He received a B.A. in mathematics in 1974 from Northeastern University and an M.S. in computer science in 1977 from Columbia University.

Jerry Greenberg *IBM Data Systems Division, P.O. Box 3332, Danbury, Connecticut 06810.* Mr. Greenberg is currently manager of Architecture and Design in Advanced Systems Development. In this position he is responsible for the design of high-performance transaction processing systems. His prior work in IBM included numerous management and technical positions in software development, business planning, and engineering development. He has in addition served on group staff and as technical assistant to senior IBM management. Mr. Greenberg received his degree in electrical engineering from the University of Florida. He received an IBM Outstanding Contribution Award for a prototype page-swapping subsystem for VM.

David R. Patterson *IBM Federal Systems Division, 6410 Rockledge Drive, Bethesda, Maryland 20817.* Mr. Patterson received an A.B. in mathematics from Duke University in 1967. He joined the Federal Systems Division (FSD) in 1968 as a programmer and worked as a programmer and systems programmer until 1978. He then joined the Washington Systems Center doing VM Field Support. His involvement with the original paging task force began during this assignment. In 1982, he went to White Plains, New York, to become a Product Administrator for high-end VM Products in the former National Accounts Division Headquarters. In 1983, Mr. Patterson became the Manager of VM Product Performance in the Data Systems Division laboratory in Kingston, New York, with responsibility for performance evaluation of VM/SP High Performance Option releases, including Release 3.4, which contained the code described by the paper in this issue. In 1985 he became the manager of VM Technical Requirements in Kingston. In 1986, he returned to FSD, where he is now involved in solving industrial automation problems for commercial customers. He is a Senior Systems Engineer in the Complex Systems Organization. He received an IBM Outstanding Contribution Award for a prototype page-swapping subsystem for VM.

Gerald A. Spivak *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Mr. Spivak joined the Computer Sciences Department at the Research Center in 1977. Prior to joining IBM, he worked in the operating systems development group at National CSS from 1972 to 1977. Since then he has been involved with various operating systems projects and is currently a member of the Control Program Prototype group. His interests are in operating systems performance, with particular attention to storage management and scheduling. Mr. Spivak received an IBM Outstanding Contribution Award for a prototype page-swapping subsystem for VM. He received a B.S. in mathematics in 1967 and an M.S. in computer science in 1969 from Rensselaer Polytechnic Institute. He is currently on assignment to Academic Information Systems Development in Palo Alto.

Reprint Order No. G321-5295.