PERFORMANCE OF A SYSTEM WITH MULTIPROGRAMMING VIRTUAL MACHINES

by

ROBERT ANDREW YOUNG

B. S., Kansas State University, 1975

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1976

Approved by:

Major Professor

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

Acknowledgements

The research reported in this document was sponsored by the Kansas State University Computing Center. I would like to thank the Computing Center, and particularly Dr. Tom L. Gallagher and Mr. Michael H. Miller, for this support.

I would also like to express my appreciation to Dr. Virgil E. Wallentine who served as my major professor and provided valuable input into both the research and this document. I would also like to thank the other members of my supervisory committee--Dr. Tom L. Gallagher and Dr. Richard F. Sincovec--for their assistance.

INTRODUCTION

Introduction

This paper deals with performance of a third generation computing system running a multi-level operating system. Specifically, the operating system in use is a conventional multiprogramming non-virtual third generation operating system (OS/MFT[1]) run in a virtual machine under the control of a virtual machine monitor (VM/370[2]). This virtual machine runs concurrently with other virtual machines which perform other services.

The particular area to be addressed is real storage management. As is recognized both in the current literature[3-7] and in commercially available virtual memory implementations[8,9], along with a global demand paging system such as is used by VM/370, there must be a facility to limit the level of multiprogramming to avoid the phenomenon known as thrashing. This control generally exists either in the form of a working set scheduler or in the form of more explicit controls such as process deactivation.

In this particular environment additional complications arise. The level of the operating system which has the greatest control over the multiprogramming level knows nothing about the management of real storage. Also, the level which performs management of real storage has no

mechanism to completely control the multiprogramming level. Furthermore, the communications between these levels is generally restricted by the pre-defined virtual machine architecture.

Various evaluations of performance which have been made previously in this environment are documented. Included is a discussion of an extension to the virtual machine architecture designed to remove restrictions on the effective level of multiprogramming which occur due to the multi-level structure of the system.

Following this discussion, a mechanism for extending the virtual machine architecture to allow control of the multiprogramming level is discussed. This facility has been implemented, and the results of performance evaluation studies of the implementation follow.

Chapter I of this paper documents in greater detail the specific hardware and software configuration for which the study was made. It also describes the techniques used for performing the performance evaluations.

Chapter II describes prior work which has been done in the area of performance of this system. Included is a description of the Accelerator,[10] a facility to remove restrictions on the multiprogramming level inherent in the multi-level structure of the system. Also included are results of benchmarks of extended

hardware configurations and of use of existing performance adjustment options.

Chapter III discusses the PAGEMON facility for controlling the multiprogramming level of the system. It includes the theoretical basis for the facility, the actual implementation, and the results of performance evaluations of the facility.

Chapter IV contains conclusions from the study and suggestions for possible extensions and areas of related future research.

Chapter I

SYSTEM ENVIRONMENT

Section 1 - Introduction

This chapter documents the specific hardware and software configuration for which this study was made. Included in this discussion are the service goals for the installation under study. Following that discussion is an initial description of the performance problems encountered and a description of the environment under which the performance evaluation benchmarks whose results are reported later were run.

Section 2 - Hardware and Software Configuration

Prior to December 1973 the Kansas State University Computing Center operated an IBM 360/50 with 128K bytes (K denotes the multiplicative factor 1024) of processor memory and 1M bytes (M denotes the multiplicative factor 1,048,576) of IBM 2361 Large Capacity Storage (LCS). This system was run using the MFT version of the OS/360 operating system[1] with the Houston Automatic Spooling Priority (HASP) system for spooling. Three basic services were supported by this system. The IBM APL/360 program product supported interactive terminals for problem solving activities. A KSU-developed facility called XMONIT provided fast-turnaround access to student-oriented in-core compilers such as the University of Waterloo's WATFIV and WATBOL, and Cornell University's PL/C. This service was limited to small jobs (45 seconds of CPU time and ten pages of printed output) and the users loaded the jobs and obtained the output themselves. The third service was batch processing of OS jobs.

In December 1973, the IBM 360/50 was replaced by an IBM 370/158 with 1M bytes of memory and five spindles of IBM 3330 disk storage. This change provided an approximate factor of four increase in CPU performance

for jobs which previously executed in the IBM 360/50's processor storage, and a factor of eight for those which previously executed in LCS. The IBM 3330 disk drives also provided faster accessing and more overlap during accessing than did the IBM 2314 disk drives attached to the IBM 360/50.

At that time there were several options available as to which operating system should be used for the IBM 370/158. Running OS/360 standalone was ruled out since the decrease in memory size would make it difficult to run more than one batch partition along with both APL and XMONIT services. OS/VS1[8] was ruled out since it did not support HASP, and the large number of local modifications to HASP made such a conversion extremely expensive. OS/VS2 release 1[9] was a major contender since it was functionally very similar to OS/MVT which was sufficiently similar to OS/MFT to make such a conversion feasible.

However, one of the primary goals for the IBM 370/158 system was to reduce the requirement for dedicated machine time for system software development and maintenance. For this reason, it was instead decided to utilize IBM's Virtual Machine Facility/370 (VM/370)[2]. VM/370 provides a virtual machine system which replicates the IBM 370 architecture to multiple virtual machines. This choice allowed the batch and XMONIT services to be

provided from within a virtual machine running an OS/MFT system with HASP which was identical to that run on the IBM 360/50. The only difference was addition of support for new types of peripherals which were installed on the IBM 370/158. The APL service was provided from a separate virtual machine running a DOS version of APL/360.

Theoretically, this should have significantly reduced the need for dedicated system development and maintenance time since a second virtual machine running a separate copy of OS/MFT and HASP could be run during production to test changes to OS and HASP. Also, a virtual machine could be used to run a copy of VM/370 to test changes made to VM/370 itself. Unfortunately, such development and maintenance procedures are very expensive in terms of CPU requirements, real storage requirements, disk space requirements, and requirements for other peripheral devices. As a result, this goal has not been met.

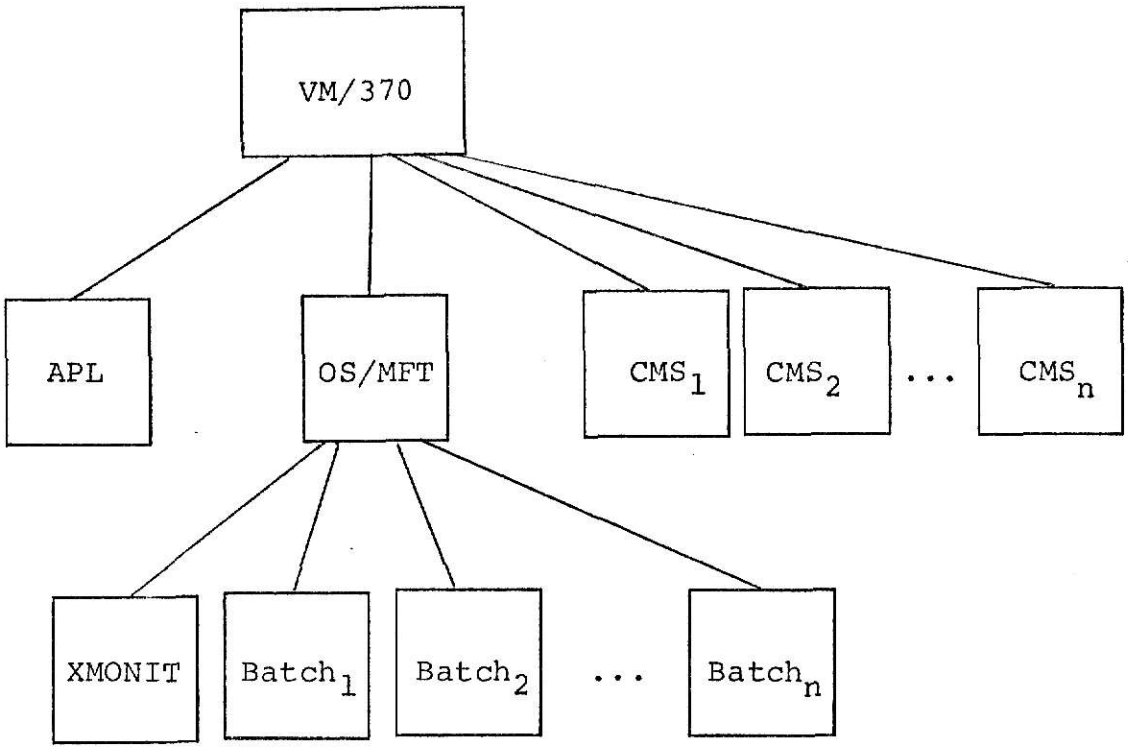
VM/370 also provides a virtual machine-oriented time sharing system called the Conversational Monitor System (CMS). CMS is actually a stand-alone single-user time sharing system designed to be run in a virtual machine. Thus, every CMS user has his own virtual machine which runs the CMS operating system which then runs the user's programs and processes the user's

commands. Since CMS provided time sharing functions which were desired as part of the new system but which could not be provided using the APL system, it was made available to the general user community after several major enhancements to provide more flexible handling of user file storage. Although the primary intended use of CMS is editing, job submission to the OS/MFT batch virtual machine, and perusal of output from the OS/MFT virtual machine, it does have the ability to execute user programs interactively.

In summary, the current system as shown in Figure 1 consists of the VM/370 control program running several virtual machines: the multi-user APL virtual machine, the OS/MFT virtual machine which provides batch and XMONIT service, any active virtual machines used for system development and maintenance, and a CMS virtual machine for each active (signed-on) CMS user.

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.



Multi-Level System Structure

Figure 1

Section 3 - Initial Performance Problems

As CMS use began, both internally within the Computing Center and within the general user community, performance problems became noticeable. The performance impact of these users showed up in such ways as slow response to OS operator commands, slow turnaround of XMONIT jobs (which usually complete in 10-30 seconds on the IBM 370/158), periodic pausing of peripheral devices controlled by the OS virtual machine, and slower-than-usual execution of batch jobs. APL users experienced poor response as did CMS users. These occurrences provided the impetus to study the performance of the system and isolate the limiting resources.

It was known at the time the system was installed (principally based on benchmarks run by IBM during the selection process) that in batch-only benchmarks OS/VS2 release 1 stand-alone performed significantly better than did OS/MFT running under VM/370. This fact also added to the interest in studying the system's performance once it was installed.

At this time there were very few methods available to determine information concerning system performance. The usual methods of measuring wall-clock time for running a batch job stream and measuring response time for a CMS or APL terminal session were available. CPU

utilization information for virtual machines was available as a by-product of VM/370's accounting for virtual machine resource use as was the number of paging I/O operations for each of the virtual machines. The system paging rate could be obtained by issuing an operator command.

The only software monitor available for VM/370 at that time was IBM's internal VM/PT.[11] This monitor runs in a CMS virtual machine and uses a special interface to examine counters and accounting data which is maintained in real storage by the VM/370 system. However, because it runs in this manner, it has both a high CPU and a high real memory overhead. Also, it is not available to customers and can be run at customer installations only with IBM personnel present. These two factors greatly limited its usefulness at KSU.

Some information about the performance of the OS virtual machine was gained by running an OS software monitor called SLACMON[12] in the OS virtual machine. This monitor measures how OS is performing in the virtual machine, but tells nothing about how the OS virtual machine is performing in the VM/370 system. Thus, such considerations as real storage management and real I/O scheduling cannot be measured by SLACMON.

Section 4 - Performance Evaluation Techniques

When making performance evaluations, three items are used. A batch job stream which consists of two copies of a nine-job stream is used to simulate the batch job load. This job stream was obtained from the benchmark used during system evaluation for the acquisition of the IBM 370/158. These jobs are loaded prior to the start of the timed run, and they are allowed to execute but not print during the run. The queued printed output is then purged after the run is complete.

CMS users are simulated by a facility called TPSIM which was developed at KSU.[13] TPSIM consists of a virtual machine which, through a special interface with VM/370, can simulate the terminal activity of multiple CMS users concurrently. This virtual machine has as input a "script" for a terminal session which consists of the responses to be provided each time a read is issued to the simulated terminal. During benchmark activities, all of the CMS users are allowed to sign on prior to the start of the timed run. After the signon is complete, the users are suspended by TPSIM when a read is issued for a CMS command. These suspended users are then released, one every 30 seconds, during the timed portion of the run. They are allowed to proceed

through the terminal session and then sign off. In general, all of the CMS users will have signed off before the eighteen batch jobs have completed.

TPSIM maintains a time-stamped log of the terminal input and output for the simulated users. This log can be used to determine response times as well as a response index. This response index which is used as the comparison criteria between benchmarks involving CMS users is the amount of wall-clock time between the restarting of the CMS user by TPSIM and the time the user signs off. This index is provided by the VM/370 accounting for virtual machines since a VM/370 ACNT command is issued by TPSIM immediately prior to restarting the virtual machine. This command prints accounting data and resets the accumulations of CPU and connect time for the virtual machine. Thus, the connect time shown at sign-off is the response index. This represents the time required to simulate all of the terminal I/O for the terminal session (which is constant) plus the sum of the response times for all of the commands and other input from the terminal. Therefore, it is a good indicator of overall response time.

TPSIM requires seven pages of real memory (which are locked in real memory to avoid page faults for the TPSIM virtual machine which would affect the timing of

the CMS user simulation) and approximately 20 seconds of CPU time for simulating ten CMS users running a terminal session which takes between eight and ten minutes.

XMONIT service is simulated by manually loading XMONIT jobs periodically throughout the run. The XMONIT stream consists of three copies of a stream consisting of two WATFIV jobs and one LISTDECK job (which merely lists a deck of cards). APL service has not been simulated in any of the benchmarks.

A summary of the batch jobs along with the CPU requirements of each step may be found in Appendix A. A listing of the CMS simulated terminal input may be found in Appendix B.

Chapter II

PRIOR PERFORMANCE EVALUATIONS AND SYSTEM CHANGES

Section 1 - Introduction

This chapter describes prior work which has been done on the system in the area of performance. It begins with an initial hypothesis of the cause of the problems described in the previous chapter. Then the Accelerator,[10] an enhancement to the virtual machine architecture to remove limitations on the multiprogramming level which are inherent in the multi-level system design, is discussed. Information is given on the IBM-supplied software monitor, its use in performance evaluation, and its confirmation of the initial hypothesis. Results of benchmark runs on extended hardware configurations are given. An evaluation of the reserved page frames option is given. Finally, a description of the study of the scheduling and page management algorithms, and of a change made to the page management algorithm is given.

Section 2 - Initial Hypothesis

Using the facilities available at the time, it was hypothesized that the major performance impact of CMS users was contention for real memory. This contention results in high amounts of paging activity and high wait times. These wait times are due to the fact that whenever a page fault occurs while executing a virtual machine, that virtual machine must be marked non-dispatchable until the referenced page has been brought into real memory. Then, the instruction which caused the page fault is re-executed and execution of the virtual machine continues. This wait time is known as page wait.

Since there is no equivalent to such a page fault if the virtual machine were executing on a real machine instead of a virtual machine, the VM/370 control program has no alternative but to mark the virtual machine non-dispatchable. Other virtual memory system such as OS/VS1 and OS/VS2 can provide the option to notify a task of a page fault and allow it to continue execution elsewhere until the referenced page has been brought into memory instead of marking the task non-dispatchable until the page is available.

This hypothesis as to the cause of the performance problems was formed based on the fact that the CPU

utilization (as determined from the billing for virtual machine CPU utilization) decreased when the CMS users were added and the system paging rate increased.

Section 3 - The Accelerator

The inability for a virtual machine to continue execution when a page fault occurs presents a potential performance problem. For example, when a page fault occurs in the OS virtual machine it may very well be possible to execute some other OS task while the paging operation is being performed for the page fault. However, there is no facility in the IBM S/370 architecture to allow this. This problem was overcome by a facility called the Accelerator developed by Southern Illinois University.[10] Southern Illinois University experienced large amounts of page wait for their OS/MVT virtual machine which was running the TSO time sharing facility as well as several batch regions.

The Accelerator effectively extends the S/370 architecture in a virtual machine to allow a virtual machine to continue execution in the event of a certain type of page fault. If a page fault occurs while executing virtual machine code which is enabled for virtual interrupts, an external interrupt is generated by the VM/370 control program using a new interrupt code. Upon receiving this external interrupt, the OS system marks the current task non-dispatchable due to page wait and enters the OS dispatcher to select another

task for execution. Then, when the referenced page is available, another external interrupt using a second new external interrupt code is generated. Upon receiving this external interrupt code, the OS system marks all tasks which were previously in page wait as now being dispatchable (unless they were non-dispatchable for other reasons as well) and enters the dispatcher to select the highest priority ready task to continue execution. Since these interrupts do not identify particular tasks or pages, it is necessary to reset the page wait status of all of the tasks which are in page wait.

The theoretical effect of this modification is to allow overlap of paging I/O for the OS virtual machine with execution of OS tasks rather than having these activities occurring sequentially. It localizes the effect of a page fault from the lower-level process (virtual machine) to the higher-level process (OS task within a single virtual machine). It also increases the effective level of multiprogramming for the OS virtual machine since low priority OS tasks would not get a chance to execute if higher priority tasks were encountering page faults.

Southern Illinois University reported very significant performance improvements through use of this facility. When it was implemented at KSU (after

conversion from MVT to MFT) it showed only minor improvements in both batch-only and batch-and-CMS benchmarks. Furthermore, in the benchmarks with CMS users the improvement in batch throughput was at the expense of CMS response time. This was due to the greater contention for real memory which resulted from the higher effective level of multiprogramming in the OS virtual machine and thus for the entire system. Specific benchmark results at KSU will be discussed in a later section.

Section 4 - VM/370 Software Monitor

In March 1975 IBM supplied the data collection portion of a software monitor as an integral part of the VM/370 control program.[2,14] The PERFORM option of the monitor provides sampled collection of data from over 100 system counters including such items as system page wait, system I/O wait, system idle wait, system problem state time, system paging rate, and counts of privileged instruction simulations by instruction. The USER option provides sampled collection of data for each signed-on virtual machine including problem state time, real supervisor state time, paging activity, I/O counts, current status, resident page count, and projected working set size. The DASTAP option provides sampled collection of data for each disk and tape device including I/O counts and device status. For further information on available data refer to the appropriate IBM literature.[2] Additionally, the SCHEDULE option provides trace data from the scheduler at the end of each virtual machine's in-queue time slice which includes data about the time slice including CPU utilization, resident page count, referenced page count, I/O count, page read count, and projected working set size.

Unfortunately, the data reduction portion of the monitor available from IBM[15,14] is a relatively expensive

program product. Furthermore, other installations reported that the data reduction package was extremely slow and cumbersome to use. Therefore, the data reduction package was not obtained for use at KSU. Instead, several data reduction programs were written to provide reports of the information considered most useful.

It was later determined that further information was needed regarding paging activity, so the data collection portion of the monitor was extended to add a PAGING option which provides trace data whenever a page fault occurs, a page-in I/O operation completes, a page is stolen, etc.. This data identifies the particular real and/or virtual page involved and allows the flow of virtual pages in and out of real memory to be reconstructed from the trace data. This allows analysis of page residence by virtual page as well as a more accurate computation of page wait by virtual machine than was possible from the sampled user status.

The following data items which will be referenced later in discussion of results of specific benchmarks are provided in the output from these data reduction programs. The first item is the run duration computed by the monitor. This will not exactly match the reported resident time for the batch job stream since the monitor must be started and stopped manually,

causing additional small delays. However, the error introduced here is approximately 0.10 minutes (which appears as idle wait time) and is relatively constant.

The output then gives system wait time broken down into idle wait, I/O wait, and page wait. Idle wait exists for the system when all users (virtual machines) are in an idle wait state. A virtual machine is in idle wait when it is in a virtual wait state and is not waiting for paging I/O or other disk or tape I/O to complete. A specific virtual machine is considered to be in page wait if that virtual machine is waiting for paging I/O to complete to service a page fault. A specific virtual machine is considered to be in I/O wait if it is not in page wait, but it is in a virtual wait state and has disk or tape I/O active or scheduled. The system is considered to be in page wait if there are no dispatchable virtual machines and over half of pageable real storage is occupied by pages from virtual machines which are in page wait. The system is considered to be in I/O wait if there are virtual machines in either page wait or idle wait, but the above condition for system page wait does not hold and there are no dispatchable virtual machines. In general, since the OS virtual machine uses over half of the approximately 180 pages of pageable real storage, the system will be in page wait if the system is in a wait state and the

OS virtual machine is in page wait, and it will otherwise be in I/O wait if the OS virtual machine is in I/O wait. However, the system page wait will be less than the OS page wait since OS page wait may be overlapped with activity in other virtual machines.

The system problem state time gives the amount of time the real machine was executing in problem state. The supervisor state time, representing the VM/370 overhead, can be computed by subtracting the three wait times and the problem state time from the total time for the run reported by the monitor.

In the data from the PAGING option of the monitor is a section on user paging statistics. Included in this data is the computed page wait for each virtual machine based on data from the paging supervisor. In this case, page wait is defined as the time from occurrence of a page fault until the page has been brought into real memory and the virtual machine is once again dispatchable.

As discussed in a previous section, the CMS response index is computed from the accounting data printed on the log created by TPSIM of the simulated CMS terminal sessions.

Section 5 - Initial Analysis of Data from Software Monitor

Benchmarks were run during June 1975 using the system software monitor to learn more about the performance of the system. Running only the eighteen job batch job stream and the nine XMONIT jobs, the batch job stream took 10.35 minutes of resident time as shown in Table 1. Of this time, 0.75 minutes were in page wait with a paging rate of 3.33 pages/second, 2.09 minutes of the run were in I/O wait, and the remaining 7.51 minutes of the run was CPU utilization. This corresponds to a CPU utilization of 73%.

Running this same benchmark along with ten simulated CMS users increased the batch resident time to 18.66 minutes (a 44% increase). OS page wait rose to 7.76 minutes (accounting for 42% of the total duration of the run) and I/O wait dropped to 1.23 minutes. This yields approximately 53% CPU utilization. The paging rate rose to 19.08 pages/second. This very clearly supports the hypothesis that the major impact of CMS users is the contention for real memory which results in increased page wait. The 7.01 minute increase in OS page wait accounts for nearly all of the increase in the batch job stream resident time.

THE FOLLOWING PAGE WAS BOUND WITHOUT A PAGE NUMBER.

THIS IS AS RECEIVED FROM CUSTOMER.

CMS Users	0	10
Batch Resident Time	10.35 min	18.66 min
CMS Response Index	-	10.79 min
OS Page Wait	.75 min	7.76 min
OS I/O Wait	2.09 min	1.23 min
Paging Rate	3.33 pg/sec	19.08 pg/sec
CPU Utilization	73%	53%

Impact of CMS Users

Table 1

Section 6 - Extended Configurations

There are several ways in which the problem of contention for real memory can be reduced. One is to add more real memory. In fact, current literature indicates that this is the only real long-term solution to a shortage of real memory.[3] Testing such an enhanced configuration at KSU is impossible. Another way the problem's impact can be reduced is to increase the ability to handle a paging load. This was done in a series of benchmark runs which used the disk drives which were normally used for mounting removeable volumes containing user datasets, private user volumes, and private volumes used for system software development and maintenance. These drives which are on the same controller and channel as the other drives were used instead for dedicated paging volumes. Normally, paging occurs on volume VM370 which also contains the CMS temporary disk space and the CMS system residence volume. The results of these benchmarks can be found in the first part of the benchmark results contained in Appendix C. They are also summarized in Table 2.

Adding a single dedicated paging volume decreased the batch resident time from 18.66 minutes to 16.40 minutes. Similarly, the CMS response index decreased from 10.79 minutes to 9.44 minutes. The paging rate

Dedicated Paging Vols	0	1	2	
Batch Resident Time	18.66	16.40	14.98	min
CMS Response Index	10.79	9.44	9.07	min
OS Page Wait	7.76	5.20	3.59	
System Paging Rate	19.08	20.23	25.49	pg/sec

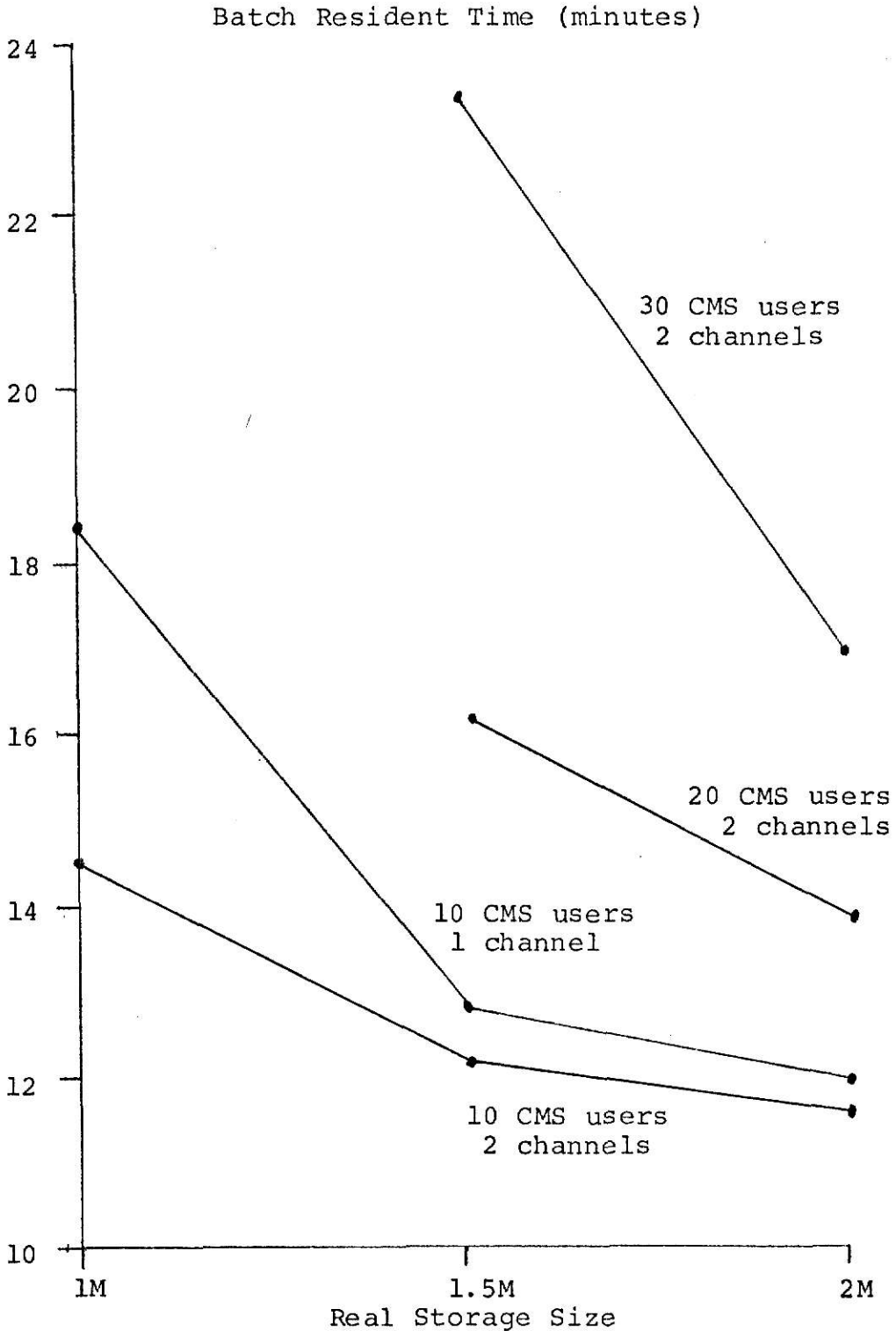
Extended Configuration Benchmark Results

Table 2

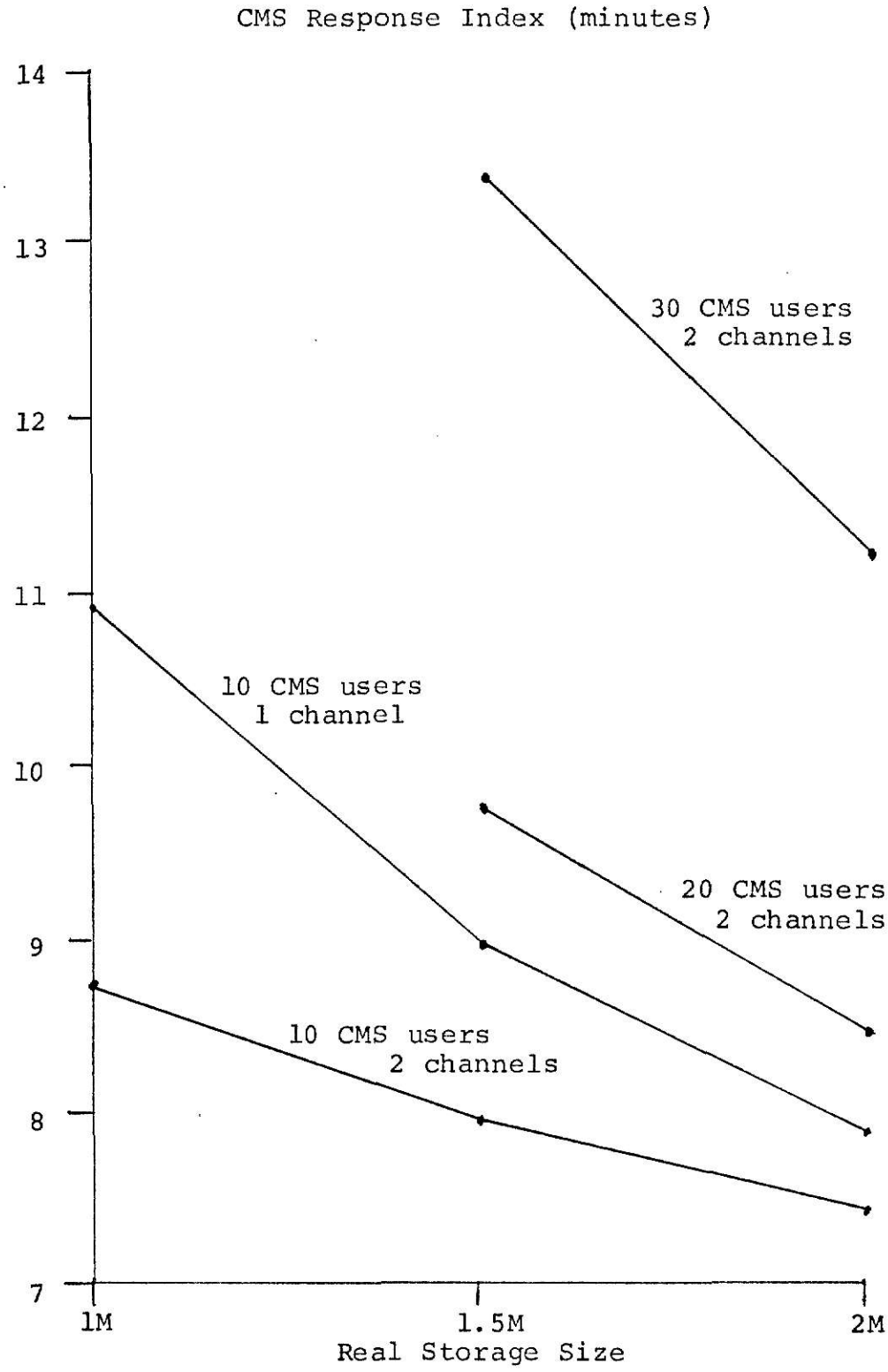
increased from 19.08 pages/second to 20.23 pages/second due to the increased paging capability as a result of reduction of device contention on the paging volume. Adding a second paging volume further reduced the batch resident time to 14.98 minutes and the CMS response index to 9.07 minutes. The paging rate rose to 25.49 pages/second. Using two volumes for paging allows overlap of both seek time and rotational delay due to the Rotational Position Sensing feature[16] of the IBM 3330 disk drive. Unfortunately, however, the use of dedicated volumes for paging is not an acceptable solution without adding more disk drives to the system.

To obtain further information on system performance, benchmarks were run during June and July of 1975 at another state agency which has an IBM 370/158 with two channels of IBM 3330 disk drives and 2M bytes of real memory. A validation run provided both a comparison base for those benchmarks and a verification that the results were comparable to those achieved in benchmarks at KSU. The results of these benchmarks which show combinations of memory sizes of 1M bytes, 1.5M bytes, and 2M bytes, use of one and two channels, use of dedicated paging volumes, and simulation of ten, twenty, and thirty CMS users are included in tabular for in Appendix C and graphical form in Figures 2 and 3.

Adding an additional .5M bytes of memory caused



Extended Configuration Benchmark Results
Figure 2



Extended Configuration Benchmark Results
Figure 3

the batch resident time to drop from 18.38 minutes to 12.62 minutes, and the CMS response index to drop from 10.89 minutes to 8.93 minutes. Increasing storage to 2M bytes with only ten users decreased batch resident time to 11.53 minutes and the CMS response index droppeed to 7.79 minutes.

Adding the second channel containing a single dedicated paging volume caused the batch resident time in 1M bytes of storage to drop from 18.38 minutes to 14.50 minutes, and the CMS response index to drop from 10.89 minutes to 8.78 minutes. Recall, however, that adding only the dedicated paging volume on the same channel caused decreases of the batch resident time and the CMS response index to 16.40 minutes and 9.44 minutes respectively. The remaining decrease is attributable to the separation of the paging activity onto the second channel. The table in Appendix C shows the results when adding a second paging volume on the second channel, and when adding a second channel with 1.5M bytes and 2M bytes of memory. It also shows the performance of twenty and thirty CMS users in 1.5M bytes and 2M bytes with two channels.

Although these benchmarks definitely indicate the need for additional memory and additional I/O capability, financial considerations make these changes unfeasible. Thus, it is necessary to look elsewhere

for solutions to the problem.

Section 7 - Reserved Page Frames Option

In VM/370 there is a facility called the reserved page frames option which allows an installation to reserve a certain minimum number of page frames for one specific virtual machine. Other virtual machines then cannot steal pages from the virtual machine with reserved pages unless the reserve limit has been exceeded. Before benchmarking this facility, it was modified to remove some initialization problems and to allow pages to be stolen from the reserved virtual machine according to the least recently used algorithm when the reserve limit has been exceeded. The IBM-supplied code marked a specific set of page frames reserved, and the specific pages of the virtual machine which happened to be placed in those frames would be the ones which were not eligible to be stolen by another virtual machine. They could be replaced only by another page from the reserved pages virtual machine.

It was anticipated that if the reserved page frames option were used for the OS virtual machine that the effect would be to improve OS performance at the expense of CMS response time. Such a benchmark, whose results are included in Table 3, was run with the reserved page frame count set to 110 pages.

Reserved Frames	0	110
Batch Resident Time	18.15 min	20.82 min
CMS Response Index	8.82 min	13.57 min
System Page Wait	4.83 min	7.18 min
System I/O Wait	1.87 min	1.20 min
System Page Reads	13104	20147
OS Page Faults	7056	6735
OS Page Wait	7.35 min	9.08 min
CMS Page Faults	520	1147
CMS Page Wait	.88 min	2.45 min

Reserved Page Frames Benchmark

Table 3

Normally during benchmarks with ten CMS users, the OS virtual machine has an average of 118 pages resident out of the 170-180 page frames available for paging. As anticipated, CMS response was significantly degraded. The CMS response index increased from 8.82 minutes to 13.57 minutes. However, the batch resident time increased from 18.15 minutes to 20.82 minutes. The number of OS page faults did in fact decrease from 7056 to 6735. This indicates that the reserved frames option was functionally working in that it reduced the number of OS page faults. The average number of OS pages resident increased from 113.30 to 117.99. However, the OS page wait grew from 7.35 minutes to 9.08 minutes. As expected, the total number of page faults for the system grew from 13104 to 20147 with the CMS user average page wait growing from 0.88 minutes to 2.45 minutes.

The unanticipated batch degradation is due to the decrease in efficiency of handling OS page faults due to the larger system load of paging activity. This can be seen by examining the OS "page fault response time" which is computed as the total amount of page wait divided by the number of page faults which required a physical page-in I/O operation. The OS page fault response time increased from 0.077 seconds without the reserved page frames option to 0.101 seconds with it.

Thus, this particular method of favoring the

batch facility at the expense of CMS response in order to improve batch performance resulted in degrading both services significantly.

Section 8 - Study of VM/370 Scheduling and Page Management

Studies were made of the VM/370 scheduling and dispatching algorithms to obtain additional insight into the operation of the system. Basically, VM/370 has a working set scheduler.[4,5] A projected working set size is maintained for each virtual machine. When a virtual machine is ready to run, it is placed on an eligible list where it must wait until there is sufficient room in real memory to hold its working set. At that time the virtual machine is removed from the eligible list and placed in the in-queue list of users (virtual machines). The dispatcher then selects a virtual machine from the in-queue list of users which are currently runnable and runs it.

In examining the system software monitor output (particularly the scheduler activity trace data), it was observed that in general all virtual machines were being moved directly from the eligible list to the in-queue list of users. This, among other things, made the scheduler bias factors and installation-assigne user priorities--which are used in computing the eligible list priority--ineffective. This led to a closer examination of the working set size projection algorithm. This resulted in finding an error which caused incorrect calculation of the number of nonresident

referenced pages for a virtual machine larger than 1M bytes. This error was corrected and, although it had no effect on performance or the effectiveness of the eligible list, the correction did make the output from the software monitor look more reasonable.

The unmodified VM/370 page management system maintains three lists of page frames: the free list, the user list, and the flush list. The free list contains frames which are immediately available for reassignment. The page management routines try to maintain a certain minimum number of pages on the free list to avoid having to wait for a page-out operation to complete before a page frame can be assigned to handle a page fault. The user list contains pages belonging to active users (virtual machines) and is ordered based on the "second chance" page replacement algorithm[17] which is an approximation to the more familiar "least recently used" algorithm. Pages on the flush list are those for users which were active but have been dropped from the scheduler's list of active virtual machines either due to entering an idle wait state or exceeding a time slice limit. The flush list is ordered first-in-first-out.

VM/370 has a favoring option whose documented purpose is to ensure that the specified virtual machines will never wait on an eligible list (i.e., their

working sets are always considered to fit in storage). This option is used for the OS virtual machine due to significant degradation which was experienced otherwise. This option has the additional side effect that it prevents the favored virtual machines' pages from being moved from the user list to the flush list at each time slice end or whenever an idle wait state is entered.

Moving pages to the flush list causes several problems. First, during page replacement pages are taken from the flush list rather than the user list whenever possible. This results in pages on the flush list being taken before pages which were referenced longer ago, but which belong to either a favored virtual machine or one which is still in the scheduler's list of in-queue users and has not encountered a time slice end. Furthermore, the pages are added to the flush list in ascending virtual address order rather than the order in which they appeared on the user list as established by the page replacement algorithm. Thus, the reference information collected by the replacement algorithm is lost. This occurs even if the user is immediately added back to the queue of runnable users. In that case, the pages are added back to the user list, but they are added in virtual address order.

Since the OS virtual machine was (and still is)

favored, its pages did not participate in this process. This caused the page replacement algorithm to be biased against CMS users. This resulted in unnecessarily stealing recently referenced pages from CMS users when less recently referenced pages belonging to the OS virtual machine existed. This in turn resulted in unnecessary paging operations which increased the average time required to service a page fault, and thus impacted both OS and CMS performance. This problem has been eliminated by disabling the code which moves pages from the user list to the flush list. Now, all pages are taken by the replacement algorithm from the user list rather than from the flush list, and the page replacement algorithm's ordering of the pages and manipulation of the reference bits is not disturbed.

Section 9 - Conclusions

The real solution to the performance problem is additional hardware in the form of more real storage and increased I/O capacity. However, since that is not feasible, it is necessary to turn to the software to attempt to improve the performance of the existing hardware.

Attempts to improve OS throughput by biasing the system against CMS users such as the reserved page frames option were not effective. The change which was effective was removing an inadvertant bias against CMS users in the page replacement algorithm. By improving the CMS performance by avoiding unnecessary paging activity for CMS users, the batch performance improved as well.

Chapter III

PAGEMON IMPLEMENTATION AND EVALUATION

Section 1 - Introduction

In this chapter we discuss the theoretical basis for implementation of the PAGEMON facility to control the level of multiprogramming to reduce the contention for real storage. A previous attempt at implementing such a facility is then described. Then, the actual implementation of the current PAGEMON facility is discussed in detail. This is followed by results of the benchmark runs used to evaluate the facility.

Section 2 - Theoretical Basis

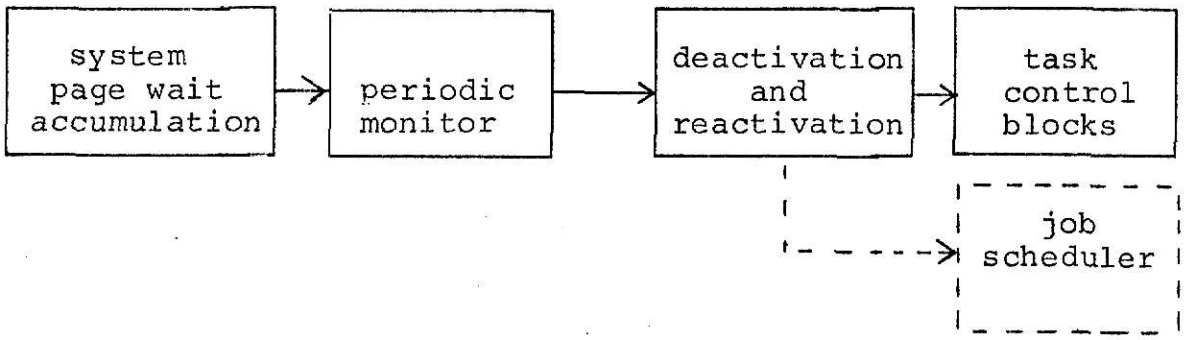
The previous results led to the hypothesis that performance could be improved generally if the multiprogramming level could be dynamically controlled based on the number of active CMS users and their activity. The necessity of a mechanism for controlling the multiprogramming level in a global virtual memory system is recognized in the literature.[3-7] This is what VM/370's working set scheduler attempts to do for the virtual machines. Although the working set scheduling algorithm may be effective in an environment of many similar virtual machines (a large system running only CMS users for example), it cannot satisfactorily deal with a single large virtual machine such as a multiprogramming OS system due to the manner and frequency with which its characteristics change. Furthermore, even if the scheduler could project the OS virtual machine's behavior, its control is limited by the fact that it can control only entire virtual machines. It has no knowledge of or control over tasks within a single virtual machine.

Mechanism for adjusting the multiprogramming level dynamically based on system activity do exist in other systems. Batch-oriented virtual memory operating systems such as IBM's OS/VS1 and OS/VS2[8,9] have task deactivation

functions such as shown in Figure 4 which monitor system indicators of the contention for real memory, and when high contention resulting in thrashing occurs or is imminent, temporarily deactivate a low priority task to reduce the multiprogramming level. This deactivation reduces the contention for real memory and relieves the thrashing condition. Then, at some other appropriate time, the deactivated task is restarted. In some systems this is extended so that the job scheduler will not schedule any new jobs for execution if currently executing jobs are deactivated.

One could not utilize such a scheme directly in a virtual machine environment to control contention for real memory because a virtual machine, although it can control its own tasks, is unaware of the existence of other virtual machines and of the distinction between real and virtual memory. It cannot obtain information to monitor for high levels of real memory contention so that deactivation of tasks within the virtual machine can be performed.

It is therefore proposed that the virtual machine architecture be extended to supply to a specified virtual machine information regarding performance of the real system so that the virtual machine's operating system (OS in this case) can take an appropriate action.



Sample Task Deactivation
in Single-Level Environment

Figure 4

Section 3 - Initial Implementation

A crude version of this type of a facility was implemented which resided completely within the OS virtual machine.[18] Since a special interface already exists to allow virtual machine such as VM/PT to examine the contents of real storage, that interface was used to periodically examine the accumulated page wait time maintained by the VM/370 dispatcher. Then, based on various threshold values which were installation-controlled parameters, it would attempt to deactivate or reactivate a task. A special OS interface used by the SLACMON software monitor for OS[12] was used to allow this facility to run in supervisor state and key zero where it could manipulate the status bits in the Task Control Blocks (TCBs) for the OS tasks.

Since the monitoring function ran in OS, it had high overhead and was greatly affected by the performance of the OS virtual machine. In particular, the monitoring intervals would be lengthened significantly during periods of high page wait, which are the periods during which fast response is most desired. Also, since the monitoring function used OS timing and task management facilities, it referenced a fairly large part of the OS supervisor during its operation, thus making it a further contributor to the working set of the OS

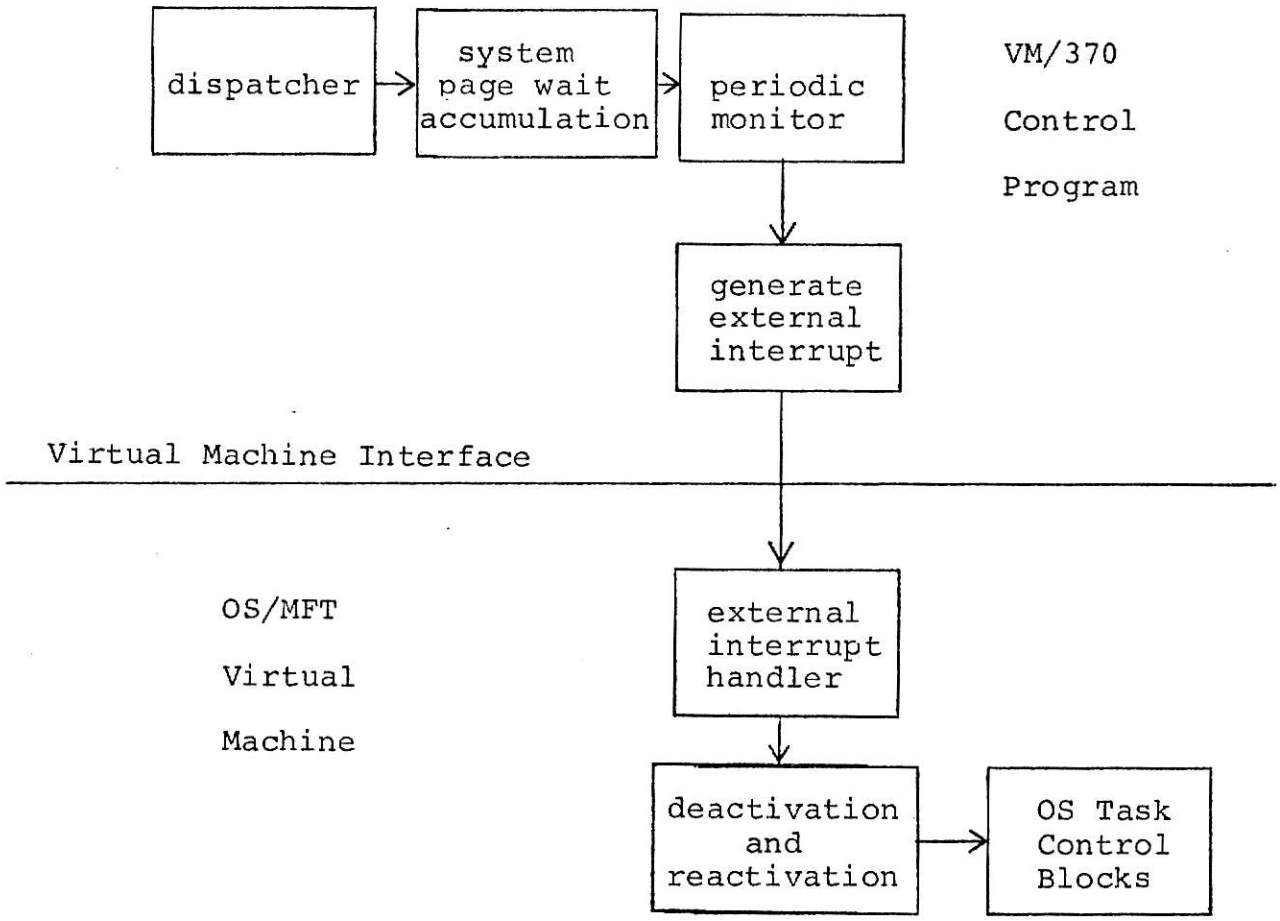
virtual machine.

This facility is referred to as PAGEMON version 1. The new facility which was designed and implemented to avoid the problems experienced by PAGEMON version 1 is known as PAGEMON version 2 and will be referred to as merely PAGEMON throughout the remainder of this document.

Section 4 - PAGEMON Implementation - Overview

PAGEMON version 2 (hereafter referred to as PAGEMON)[19] consists of four major parts. These parts correspond to the monitoring function, the deactivation and reactivation function, and operator communications routines for these two functions. The functions have been distributed between the VM/370 control program and the OS virtual machine's supervisor as shown in Figure 5.

Since the system data concerning page wait and other performance-related data is collected by the VM/370 control program and known only to the VM/370 control program, the monitoring function will reside in the VM/370 control program. This allows direct low-overhead access to the data required to monitor the system performance. Furthermore, since operation at this level inherently has less CPU overhead than communication with and operation within a virtual machine, the decision routines will also exist at this level as a part of the monitoring. Thus, the virtual machine is not involved in any monitoring or decision activity up to the point that it is required that a task activation or deactivation occur. At that point, the activation request or deactivation request is communicated to the specified virtual machine.



PAGEMON Operation

Figure 5

The virtual machine's operating system (OS in this case) contains the code to, upon request, deactivate or activate a task if possible. It is up to the virtual machine's operating system to determine eligibility for activation or deactivation and to select an appropriate task from those eligible for the requested function. The virtual machine can ignore a request if there are no tasks eligible for the requested function, but should not arbitrarily ignore reactivation requests if there are deactivated tasks.

The communication of these requests is, like the Accelerator, through the use of external interrupts defined in an extension of the virtual machine architecture. Interrupt codes X'0004' and X'0008' have been chosen for deactivation and reactivation requests respectively. These codes were previously used by the direct control feature, which is neither used nor supported by the VM/370 control program. The interrupts obey the usual rules for external interrupts except that they have equal priority and are queued first-in-first-out rather than in separate priority classes. These interrupt codes are valid operands of the VM/370 CP EXT command which allows a virtual machine's operator to simulate generation of external interrupts. This facilitates testing of the OS modifications to handle the interrupts and also allows recovery from synchronization errors should they occur.

An operator communications routine for the VM/370 portion of the facility exists in the form of the VM/370 PAGEMON command. This command allows the facility to be started and stopped. It allows setting of various parameter values for the monitoring and decision routines, and inquiry as to the current settings of these parameters. It ablows resetting all parameters to their default values. It allows inquiry as to the current system page wait as collected by the monitor function, and allows inquiry as to the net count of deactivation requests. It also allows a trace facility to be started and stopped. This trace facility sends a console message to the virtual machine which is receiving the activation and deactivation requests each time a request is made. This message includes the current net deactivation request count.

The communications routine for the OS portion of the facility exists in the form of a program called PAGEMON2. This program, which can be started from the console or via a job, allows setting of parameters for the task eligibility determination and task selection routines. It also allows inquiry as to the settings of these parameters and as to the current state of the system. It allows specific jobs to be forced into an activated or deactivated state, or their forced status to be rest. The communications routine

communicates with the operator via the OS Write-to-Operator-with-Reply (WTOR) function. The communications routine can be started and stopped without affecting the operation of the deactivation/reactivation routine, and it is not required to start it before starting the VM/370 monitoring function.

Section 5 - PAGEMON Implementation - Structure

The modifications to the VM/370 control program consist of a small modification to the routine which periodically computes CPU utilization for the VM/370 CP IND LOAD command (in module DMKSCH), the addition of an entry for the PAGEMON command in the CP command decoding table (module DMKCFC), the addition of the PGMBLOK macro (PAGEMON communications block), the addition of a new non-pageable module DMKPGM (PAGEMON monitoring and decision routines), and the addition of a new pageable module DMKCPM (PAGEMON command processor).

The modification to DMKSCH modifies the existing routine which periodically computes the system CPU utilization for the VM/370 CP IND LOAD command. The modification causes a call to DMKPGM to be made when the system page wait exceeds a certain threshold value. Thus, when the PAGEMON functions are not needed there is a very minimal monitoring overhead which is a part of existing system monitoring.

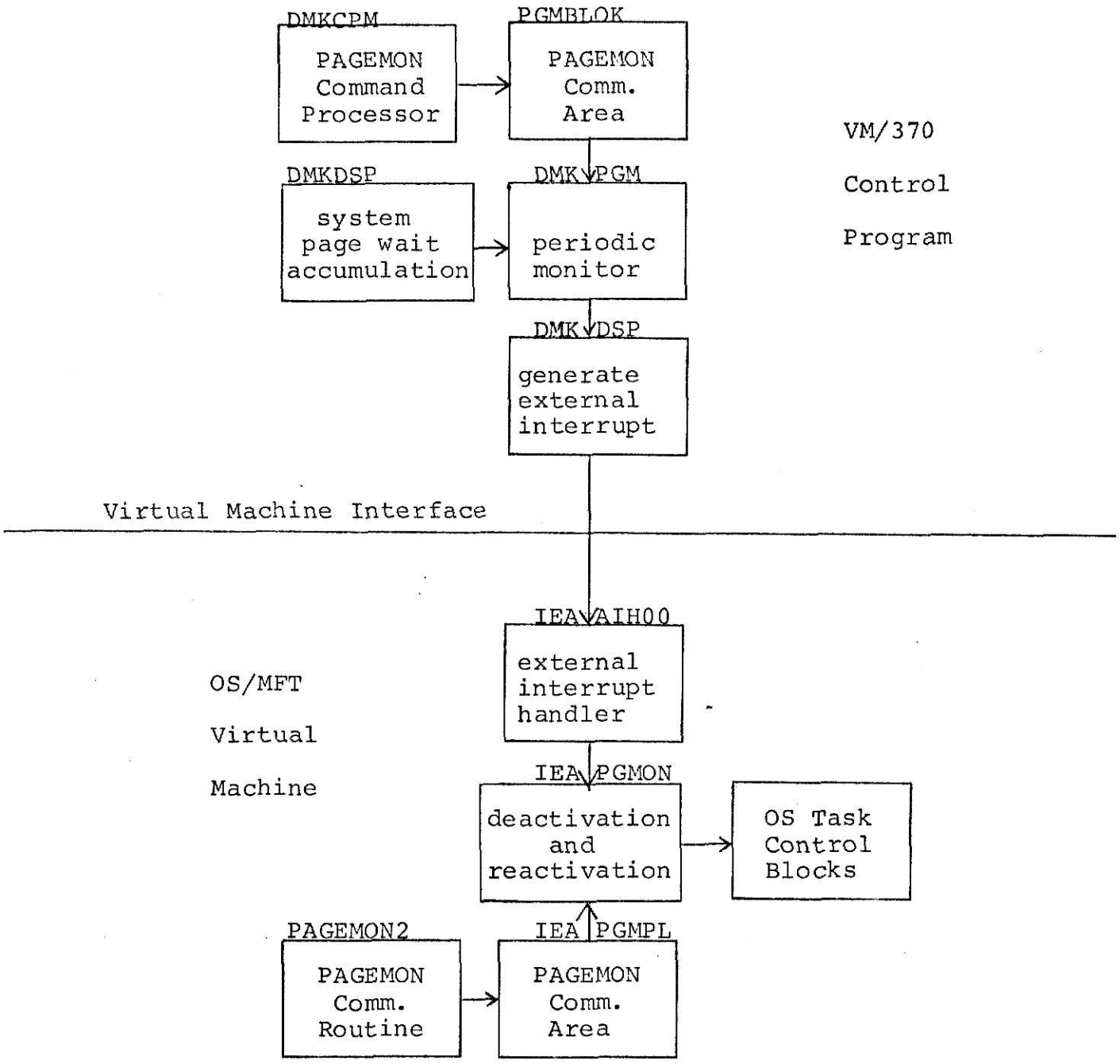
Module DMKPGM performs the monitoring once the threshold value has been exceeded. This module also makes the decisions as to whether or not a deactivation or reactivation is required and interfaces with the VM/370 dispatcher (DMKDSP) to generate the appropriate external interrupt to the specified virtual machine

if necessary. Figure 6 gives a pictorial representation of this structure.

Module DMKCPM processes the VM/370 PAGEMON command and modifies the PGMBLOK which is the communications area between DMKCPM and DMKPGM. This block contains all of the parameter values as well as statistics concerning operation of the PAGEMON facility.

The OS modifications consist of adding code to the external interrupt handler (in MFT module IEAAIH00) to recognize the two new external interrupt codes and pass control to the new task deactivation and reactivation routine IEAPGMON when they are encountered. IEAPGMON is a nucleus-resident routine which determines task eligibility, selects tasks, and performs task deactivation and reactivation. It contains within it a communications area IEAPGMPL which holds the parameters which are modified by the operator communications routine and which are used in determining eligibility for deactivation and reactivation.

The communications routine PAGEMON2 is a separate program which resides in any protected library. It is executed by the PAGEMON2 procedure which may be invoked via an operator START command or via JCL in a job which is authorized to access the library which the program is stored in. It uses the SLACMON SVC to get into supervisor state so that it may modify the parameter



PAGEMON Structure

Figure 6

list stored in IEAPGMPL and so that it may disable interrupts to obtain mutually exclusive access to the Task Control Block (TCB) chain. It also uses the KSU general purpose user SVC which has been modified to provide a function which returns the address of IEAPGMPL which is contained within the OS nucleus.

Section 6 - PAGEMON Implementation - Algorithmic Description of VM/370 Functions

The parameters for the monitoring and decision routine are given in Table 4. They will be referenced by name in the following discussion of the monitoring and decision algorithms.

Every thirty seconds the VM/370 CPU utilization computation routine in DMKSCH computes the smoothed system page wait and compares it against the MONLIM parameter value. If the system page wait exceeds this value, PAGEMON has been activated with a PAGEMON START command, and PAGEMON is not currently in "intensive monitor mode", then DMKPGMON is called to start intensive monitor mode.

In intensive monitor mode, the system page wait is recomputed every MONINT seconds. It is then smoothed based on the SMOOTH parameter value yielding

(SMOOTH*old + (100-SMOOTH)*current)/100

where old is the previous smoothed value (or the value computed by the DMKSCH routine if this is the first monitoring interval after entering intensive monitor mode) and current is the current page wait for the last monitor interval.

Task deactivation will be requested if MAXDEACT will not be exceeded and the system page wait (as smoothed above) has exceeded HIPAGE for at least HICNT

Name	Default	Function
USER	OS	User who will perform deactivations
MAXDEACT	4	Maximum number of deactivated tasks
LOPAGE	20%	Low system page wait threshold for reactivation
HIPAGE	30%	High system page wait threshold for deactivation
LOCNT	1	Number of LOPAGE excessions for reactivation
HICNT	1	Number of HIPAGE excessions for deactivation
MONINT	1	Monitor interval in seconds
MONCNT	30	Number of monitor intervals system page wait must be below MONLIM threshold to terminate intensive monitor mode
SMOOTH	0	Page wait smoothing constant
WARN	40%	Page wait limit for system operator warning message
MONLIM	10%	Page wait threshold to enter intensive monitor mode

PAGEMON SET Parameters

Table 4

consecutive monitor intervals. The deactivation is requested by generating the appropriate external interrupt to the virtual machine specified by the USER parameter.

Task reactivation will be requested if the current count of deactivated tasks is positive and if the system page wait has been less than LOPAGE for at least LOCNT consecutive monitor intervals.

Two consecutive deactivations may not occur less than HICNT monitor intervals apart, and two consecutive reactivations may not occur less than LOCNT monitor intervals apart.

A warning message is sent to the system operator if the system page wait exceeds WARN. This warning will be sent at most once every thirty seconds.

If an activation or deactivation is requested, a message containing the new net count of deactivation requests is sent to the user specified by the USER parameter if the PAGEMON TRACE ON command is in effect.

Intensive monitor mode will be terminated if the smoothed system page wait does not exceed MONLIM for at least MONCNT consecutive monitor intervals and the net deactivation request count is zero.

The PAGEMON QUERY SET command gives the current values of all of these parameters, and PAGEMON SET changes them. All except USER can be changed when the

PAGEMON function is active. The PAGEMON QUERY STAT command gives the current page wait (PGWT) as maintained during intensive monitor mode, and the current net deactivation request count (DACNT).

The PAGEMON RESET command resets all of the parameters to their default values after stopping PAGEMON if it is active. PAGEMON START starts PAGEMON which will then allow the routine in DMKSCH to call DMKPGMON to enter intensive monitor mode if the MONLIM threshold is exceeded. PAGEMON STOP causes intensive monitoring to stop. PAGEMON STOP or RESET also causes external interrupts to be generated to reset the net deactivation request count to zero (and hus to reactivate all deactivated tasks).

Appendix D contains a reference summary of the PAGEMON command operands.

Section 7 - PAGEMON Implementation - Algorithmic Description of OS Functions

The new OS routine IEAPGMON is entered whenever one of the two external interrupt codes used by PAGEMON is detected by the OS external interrupt handler. If the request is for deactivation, an eligible task is selected for deactivation (if there is one), it is marked non-dispatchable due to deactivation, and the OS dispatcher is entered if the deactivated task is the current task. If a reactivation request is received, a deactivated task which is eligible for reactivation is selected, the non-dispatchability due to deactivation is reset, and the dispatcher is entered to select a new task for execution since the reactivated task may be of a higher priority than the interrupted task.

The task selected for deactivation will be the lowest priority eligible task. Eligibility is determined by the following criteria: task was created with a non-zero protect key (i.e., the task is a user task), task is not executing in must-complete mode, jobname associated with the task can be determined, task limit priority (maximum value the task dispatching priority can attain) does not exceed the HIPRIOR parameter of PAGEMON2's SET command, the jobname associated with the task has not appeared in an ACTIVATE jobname command

which is in effect, an ACTIVATE ALL command is not in effect, and the task is not the PAGEMON2 communications task.

The task selected for reactivation will be the highest priority deactivated task which is eligible. Eligibility is determined by: the jobname associated with the task has not appeared in a DEACTIVATE jobname command which is in effect, and a DEACTIVATE ALL command is not in effect.

The operator is provided, via the communications routine, with a variety of commands for the OS part of the facility. A reference summary of these commands is contained in Appendix E. The SET HIPRIOR command allows the maximum value a task's limit priority can have and still be eligible for deactivation to be set. A QUERY SET command allows the value of that parameter to be displayed. A RESTART command causes all tasks to be reactivated and the tables of forced activations and deactivations to be cleared. The ACTIVATE jobname command causes the specified job to be activated if it is deactivated, and causes that job to be ineligible for deactivation. Similarly, a DEACTIVATE jobname command causes the specified job to be deactivated if it is eligible, and causes the job to be ineligible for reactivation. The RESET jobname command causes the effects of the ACTIVATE jobname and DEACTIVATE

jobname commands to be reset. The QUERY FORCE command displays forced activations and deactivations. The ACTIVATE ALL command causes all deactivated jobs (except those deactivated by the DEACTIVATE jobname command) to be activated and to be ineligible for deactivation. The DEACTIVATE ALL command causes all eligible tasks to be deactivated and to be ineligible for reactivation. The RESET ALL command resets the effects of these two commands.

The QUERY DEACT and QUERY ACTIV commands display the jobnames of deactivated and activated tasks respectively. The QUERY STAT command causes various statistics to be displayed. These are: DACTCNT - count of currently deactivated tasks; DACTREQ - count of deactivation requests (net); DAREQCNT - count of deactivation requests (total); RAREQCNT - count of reactivation requests (total). The STOP or END command terminates the communications routine without additionally altering task status or parameter values.

The communications routine will accept commands through SYSIN if supplied prior to going to the console. This allows PAGEMON2 to be executed during system startup with SYSIN referencing a dataset containing the appropriate commands to set the parameter values. Furthermore, the OS communications routine will accept VM/370 CP commands (such as PAGEMON) and pass them to

VM/370 for execution. Thus, this dataset could contain the commands to set both the OS and VM/370 parameters and to issue the PAGEMON START command during OS startup.

Section 8 - PAGEMON Evaluation

After implementing PAGEMON version 2, benchmarks were run to determine whether it had a positive or negative effect upon system performance, and the extent of that effect. Only the two batch partitions were eligible for deactivation. The results of these benchmarks are contained in Appendix F. Under the heading of PAGEMON status, various codes are used to indicate the PAGEMON parameter values. The code 'mi' specifies the monitor interval in seconds. If not specified, a value of 1 was used. The code 'sm' specifies the SMOOTH parameter. If not specified, 50 was used. The code 'c' specifies the values of the HICNT and LOCNT parameters which were always specified identically. If not specified, a value of 1 was used. The code 'max' specifies the value of the MAXDEACT parameter. If not specified, the value 3 was used. The code 'thresh' specifies the LOPAGE and HIPAGE threshold values respectively. It defaults to 20/30. MONLIM was always set to 10 and MONCNT to 30.

Before studying the effects of PAGEMON, several other comments about the benchmark results should be made. First, these benchmarks showed a greater level of non-repeatability of results than has been experienced in the past. For example, runs H1 through H8 shown in

Appendix F were all made during one five hour period. The system was IPLed (initialized) prior to running H1, and then it was not IPLed again until run H8 was made. Note that runs H1 and H2 showed nearly identical results, but that run H7 which was configured identically with H1 and H2 ran almost a minute longer in terms of batch resident time.

The additional minute of resident time appears to come from at least two factors. One is that the VM/370 supervisor overhead seems to increase as the system continues to run after an IPL. This could account for up to 30 seconds of the difference. The other potential factor is the operation of the OS in-core job queue. The OS job queue, which contains various non-resident control blocks about jobs, is resident in virtual storage rather than on disk. This provides a performance improvement since paging can be done in 4K byte blocks whereas job queue I/O is done in 176 byte blocks using a single buffer. However, the disk space allocation algorithm is first-in-first-out, but the disk space is released in a different order than it is allocated. This results in job queue references having some degree of locality until the allocation has gone completely through the dataset once and wraps around to the beginning. It appears to take approximately two benchmark runs for

this to occur. Then, since the blocks of the job queue were not freed in sequential order, they are not assigned sequentially. This results in more random reference patterns which cause the working set of the OS virtual machine to increase. This could account for part of the increase in resident time between runs H1 and H7.

As a result, it is felt that relatively little information about PAGEMON can be gained from further examination of runs H1 through H8. The second unusual item that may be noticed from this data is that in comparing these results with those from the other benchmarks reported in previous sections, the batch resident time has dropped from approximately 18 minutes to 15-16 minutes, and the CMS response index has dropped to 8-9 minutes. A small part of this may be due to several CMS commands in the simulated terminal session which previously executed correctly but now generate error messages for missing files. However, a more significant contribution is felt to come from changes in disk pack organization. In particular, the VM/370 spool area (which contains both paging and spooling space) was cleared recently. This may have resulted in more optimal allocation of the space used for paging within this area, and thus resulted in improved performance.

Runs H9-H11 and H12-H18 were made at separate times. The runs with PAGEMON inactive (runs H1, H2, H10, H12, H14, and H17 in Appendix F) which were made immediately after an IPL appear to be relatively consistent.

The best illustration of the value of PAGEMON is run H15. This run has the PAGEMON parameters set to provide a somewhat dampened response to changes in page wait. The smoothing constant is set at 75% and the LOCNT and HICNT parameters are set to 4 which with MONINT at 1 means that the threshold values must be exceeded for four consecutive one second monitoring intervals before an activation or deactivation will occur. The threshold values in use were HIPAGE of 30% and LOPAGE of 20%. A comparison of runs H12 and H15 is shown in Table 5. The batch resident time has decreased by 37 seconds (4% decrease). Similarly, the CMS response index has decreased slightly (2%). As expected, the page wait and the number of page faults for the entire system, for OS, and for CMS have all decreased. The system I/O wait has increased by 0.18 minutes. This increase represents either periods during which I/O was previously being overlapped by paging operations (and page wait), or the increase in I/O wait due to the reduction in the multiprogramming level of the OS virtual machine and thus the decrease

	Normal	PAGEMON	Change
Batch Resident Time	15.83 min	15.27 min	-4%
CMS Response Index	8.78 min	8.53 min	-2%
System Page Wait	3.81 min	2.99 min	-22%
System I/O Wait	1.59 min	1.77 min	+11%
Total Page Reads	10756	9619	-11%
OS Page Faults	5607	4886	-13%
OS Page Wait	5.39 min	4.14 min	-23%
CMS Average Page Faults	472	431	-9%
CMS Average Page Wait	.75 min	.66 min	-12%
Run	H12	H15	

PAGEMON Performance Analysis

Table 5

in potential overlap of execution and non-paging I/O between OS tasks. However, the effects of this run with PAGEMON are still all positive.

Run H11 shows a case where the decrease in page wait as a result of PAGEMON was matched by an increase in I/O wait. Comparing run H11 with run H10 in Table 6, although there was a significant improvement in CMS response, there was a smaller improvement in batch throughput. In this case, the smoothing constant was set to 50% and the LOCNT and HICNT parameters were set to 1. The same threshold values and monitor interval length were used as in run H15. In this case the decrease in page wait of 0.82 minutes has been almost completely offset by an increase in I/O wait of 0.63 minutes. However, the CMS page fault counts have decreased, thus causing the decrease in the response index even below that of run H15. In this particular run the deactivations decreased the paging load caused by the OS virtual machine. This improved the CMS response time and the OS and system page wait times. However, the deactivations also increased the OS I/O wait due to the decreased level of multiprogramming which reduced the potential overlap of execution and I/O activity within the OS virtual machine.

Run H16 in Appendix F is another example where the parameter settings were such that the I/O wait

	Normal	PAGEMON	Change
Batch Resident Time	16.12 min	15.95 min	-1%
CMS Response Index	8.82 min	8.20 min	-7%
System Page Wait	3.64 min	2.82 min	-23%
System I/O Wait	1.72 min	2.35 min	+37%
Total Page Reads	10299	9243	-10%
OS Page Faults	5391	4843	-10%
OS Page Wait	5.26 min	4.06 min	-23%
CMS Average Page Faults	453	406	-11%
CMS Average Page Wait	.71 min	.57 min	-20%
Run	H10	H11	

PAGEMON Performance Analysis

Table 6

increased to such an extent that it canceled out the effects of PAGEMON in reducing the page wait. In this case, the parameter change was to lower the HIPAGE threshold from 30% to 25%.

The general conclusion from these comparisons is that PAGEMON is effective, although not to the degree which was hoped. This is true providing that the parameters are properly set. Improperly setting the parameters, although it probably would not degrade CMS response, can degrade the batch performance by reducing the multiprogramming level to such a point that the I/O wait becomes excessive.

Run H18 was made to obtain current data on the performance of the Accelerator. It shows a decrease in batch resident time which exceeds that obtained with PAGEMON. However, it does this at the expense of CMS response time which has increased to over 9 minutes. The data which is summarized in Table 7 shows that system page wait has decreased to 2.16 minutes. However, with the Accelerator, system page wait will be recorded only when page wait occurs due to a page fault occurring in disabled code (which the Accelerator cannot handle). Page faults which occur in enabled code will result in increased I/O wait. The I/O wait has in fact increased slightly, but the overall effect is to reduce total system wait time. Since the effect of

	Normal	Accel	Change
Batch Resident Time	15.72 min	15.07 min	-4%
CMS Response Index	8.67 min	9.15 min	+6%
System Page Wait	3.45 min	2.16 min	-38%
System I/O Wait	1.70 min	2.14 min	+26%
Total Page Reads	10252	10912	+6%
OS Page Faults	5322	5454	+2%
OS Page Wait	5.02 min	4.43 min	-12%
CMS Average Page Faults	445	492	+11%
CMS Average Page Wait	.70 min	.95 min	+36%
Run	H17	H18	

Accelerator Performance Analysis

Table 7

the Accelerator is to increase the effective multi-programming level, the working set and thus the memory requirements of the OS virtual machine increase. This is shown by the increase in total page reads, and in both OS and CMS page faults. The OS page wait value reported is not meaningful with the Accelerator since the after-the-fact simulation of the page management activity to produce this value cannot tell whether the Accelerator was able to handle a particular page fault or not. However, the CMS page wait value is accurate and is the highest value observed in this series of runs whose results are documented in Appendix F.

Due to its adverse effects upon CMS users, use of the Accelerator is not preferred over use of PAGEMON. However, PAGEMON should not be used without some monitoring to ensure that the parameter settings are reasonable for the environment under which it operates.

Chapter IV

CONCLUSIONS

Section 1 - General Conclusions

The first conclusion that can be reached is that some further investigation into the lack of reproducability of benchmark results, particularly when not immediately preceded by an IPL, is required. Although a system this complex has a sufficiently large number of uncontrollable factors to make perfect reproducability impossible, it is possible that identification of major causes of lack of reproducability may provide a potential mechanism for improving system performance. In particular, the allocation algorithm of the OS in-core job queue should be examined and, if feasible, made more appropriate for this environment.

Secondly, the implementation of PAGEMON is a success, although not of the magnitude which was hoped. It is recommended that PAGEMON be implemented for use during production at KSU with appropriate monitoring of its activities. Operation in a dampened mode appears to be most appropriate.

Thirdly, more complete documentation of performance impacts of system changes should be maintained. This, in addition to being aesthetically desireable, may provide information on seemingly unlikely ways to improve system performance, or at least unlikely factors which influence it.

Section 2 - Future Research

For future research in this same area, there are many possibilities.

The current PAGEMON implementation is limited to a single virtual machine. This limitation is satisfactory for the environment at Kansas State University. However, one area for future development is the extension to supporting multiple virtual machines. Distribution of requests across the various virtual machines is still an open issue.

Study should be made concerning use of PAGEMON in conjunction with the Accelerator. Although they have seemingly opposite effects, it is possible that working together they could achieve better results than each achieves individually. If they were used together, the Accelerator would simply have the effect of removing an artificial restriction on the multiprogramming level. PAGEMON would have the responsibility for controlling the multiprogramming level. However, the mechanism used for accumulating system page wait time would have to be adjusted for use with the Accelerator in order to provide an accurate indication of the page wait which the Accelerator hides as I/O wait.

No feedback into the working set projections has

been provided in this implementaion of PAGEMON. This is partially due to the fact that the working set scheduler is relatively ineffective at KSU. Some study should perhaps be done into whether such feedback would be useful and how it would be done. Determining the magnitude of the projection adjustment is a particularly difficult problem.

PAGEMON currently relies solely on the OS dispatching priorities as the selection criterion for deactivating tasks. The interaction of PAGEMON with the HASP dynamic dispatching algorithm[20] should be studied. This algorithm preiodically alters dispatching priorities based on past CPU utilization to give I/O-bound tasks higher priority, and thus maintain a higher level of multiprogramming. Since deactivation appears as wait time rather than CPU utilization, it appears intuitively that the current selection of tasks for deactivation within the dynamic dispatching group is somewhat random, but biased towards deactivating CPU-bound tasks. It may be necessary to gather additional information to either ensure a fairer distribution of the deactivation, or to make some attempt to identify which task is most likely to improve the system performance when deactivated. The prospects for being able to perform the latter of the two tasks are not encouraging.

This reasearch as well as all of these areas mentioned above for future research should help to improve the understanding of performance of systems with multiprogramming virtual machines, and of multi-level operating systems in general.

REFERENCES

1 IBM Corp., VM/370 Control Program Logic, Y20-0880-3, Burlington, Mass. (June 1974).

2 IBM Corp., OS/360 MFT Guide, C27-6939, Poughkeepsie, New York (March 1972).

3 Denning, P. J., "Virtual Memory", Computing Surveys 2 No. 3, 177-183 (Sept. 1970).

4 Denning, P. J., "Working Set Model for Program Behavior", Comm. ACM 11, No. 5, 323-333 (May 1968).

5 Rodriguez-Rosell, J., and Dupuy, J., "The Design, Implementation, and Evaluation of a Working Set Dispatcher", Comm. ACM 16, No. 4, 156-163 (April 1973).

6 Hoare, C. A. R., and McKeag, R. M., "A Survey of Store Management Techniques", from Hoare and Perrott, Operating Systems Techniques, Academic Press, New York, 139-146 (1972).

7 Brinch Hansen, P., Operating System Principles, Prentice-Hall, Englew-od Cliffs, New Jersey, 184-191 (1973).

8 IBM Corp., OS/VS1 Features Supplement, C20-I752, White Plains, New York (July 1975).

9 IBM Corp., OS/VS2 Features Supplement, C20-1753, White Plains, New York (March 1974).

10 Hendricks, D., "Operating Systems in a Virtual Machine", Proceedings of SHARE XLV, SHARE Inc., Chicago (August 1975).

11 IBM Corp., VM/PT Users Guide, Z20-2693, Poughkeepsie, New York (1974).

12 Naples, J.,"SLACMON BOF Project Report", Proceedings of SHARE XLVI, SHARE Inc., Chicago (March 1976).

13 Young, R., TPSIM, Kansas State University Computing Center, Manhattan, Kansas (1974).

14 Callaway, P. H., "Performance Measurement Tools for VM/370", IBM Systems Journal 14, No. 2, 134-160 (1975).

15 IBM Corp., VM/SGP Program Description/Operations Manual, H20-1550, White Plains, New York (1975).

16 IBM Corp., Reference Manual for IBM 3830 Storage Control and IBM 3330 Disk Storage, A26-1592, White Plains, New York (April 1972).

17 Hoare, C. A. R., and McKeag, R. M., "A Survey of Store Management Techniques", from Hoare and Perrott, Operating Systems Techniques, Academic Press, New York, 139-141 (1972).

18 Young, R., PAGEMON Version 1, Kansas State University Computing Center, Manhattan, Kansas (1975).

19 Young, R., PAGEMON Version 2, Kansas State University Computing Center, Manhattan, Kansas (1976).

20 IBM Corp., The HASP System, 360D-05.1.014, White Plains, New York, 190-192 (June 1972).

APPENDICES

Job	Step	CPU Time (sec)
AB01	Fortran Compile	18.70
	Fortran Execute	21.49
AB02	COBOL Compile	2.60
	COBOL Execute (w/sort)	3.65
AB03	PL/I Compile	3.75
	PL/I Execute	1.23
AB04	Fortran Compile	4.52
	Fortran Execute	30.48
AB05	Card-to-Disk Copy	.60
	PL/I Compile	10.47
	PL/I Execute	6.81
AB06	Fortran Compile	3.74
	Fortran Execute	1.93
AB07	PL/I Compile	4.22
	PL/I Execute	3.12
AB08	WATFIV Stream	3.37
AB09	WATBOL Stream	.35
Total		121.03

Batch Job Summary

Appendix A

```
CP MSG OP RESTART
FMSET C OFF
ERASE W WATFIV
FMERASE W WATFIV
EDIT W WATFIV
INPUT
$JOB
      DO 1 I=1,3201,400
    1 WRITE (6,2),(I-1+J,19*(I-1+J),I-1+J+50,
     X19*(I-1+J+50),J=1,50)
    2 FORMAT('1',57X,'MULTIPLES OF 19'/'0'/
     X(1X,8(I4,I7,5X)))
      STOP;END
$ENTRY

TOP
T *
TOP
L/DO
C/400/100/
C/3201/401/
N
C/),(/)(/
N
C/8/2/
TOP
T *
FILE
L * *
TYPE PROFILE EXEC          (file not found)
SP PUN *
PUN W WATFIV
Q R ALL
READ *
L * *
Q V ALL
Q N
Q DISK *
COPY PROFILE EXEC * PROF2 EXEC A     (file not found)
L * *
```

CMS Simulated Terminal Session Input (Part 1 of 2)

Appendix B

```
FMR PROB2 WATFIV (USER VMPB4      (2 second WATFIV job)
L * WATFIV
CP MSG OP START WATFIV
CP IND USER *
CP SP PRT TO BENCHMK
WATFIV PROB2 (PRINT
CP CL PRT NAME PROB2 OUTPUT
CP IND USER *
CP MSG OP END WATFIV
ERASE PROB2 WATFIV
LOGOFF
```

CMS Simulated Terminal Session Input (Part 2 of 2)

Appendix B

Run	Mem (MB)	Chan	Ded Page Vols	CMS Users	Batch Res (min)	CMS Resp (min)	OS Page Wait (min)	Sys Page Rate (pg/sec)
C	1.0	1	0	10	18.66	10.79	7.76	19.08
D	1.0	1	1	10	16.40	9.44	5.20	20.23
E	1.0	1	2	10	14.98	9.07	3.59	25.49
A2	1.0	1	0	10	18.38	10.89	7.27	19.57
A3	1.0	2	1	10	14.50	8.78	3.70	23.67
A4	1.0	2	2	10	13.92	8.29	2.13	27.36
A5	1.5	1	0	10	12.62	8.93	1.10	12.99
A6	1.5	2	1	10	11.97	7.92	.60	15.01
A7	2.0	1	0	10	11.53	7.79	.17	7.95
A8	2.0	2	1	10	11.37	7.32	.20	9.95
B6	1.5	2	1	20	15.97	9.69	2.82	22.72
B7	1.5	2	1	30	23.08	13.29	8.94	23.77
B8	2.0	2	1	20	13.62	8.34	.38	17.98
B9	2.0	2	1	30	16.70	11.12	1.29	22.86

Extended Configuration Benchmarks

Appendix C

```
PAGEMON     START

            STOP

            RESET

            QUERY       STAT
                        SET

            TRACE       ON
                        OFF

            SET         USER        userid
                        MAXDEACT    value
                        LOPAGE      value
                        HIPAGE      value
                        LOCNT       value
                        HICNT       value
                        MONINT      value
                        MONCNT      value
                        SMOOTH      value
                        WARN        value
                        MONLIM      value
```

VM/370 PAGEMON Command Summary

Appendix D

RESTART		
ACTIVATE	jobname ALL	
DEACTIVATE	jobname ALL	
RESET	jobname ALL	
SET	HIPRIOR	value
QUERY	SET STAT FORCED DEACT ACTIV	
CP	command	
STOP		
END		

OS PAGEMON Command Summary

Appendix E

Run	PAGEMON Status	Group	IPL	Batch Res (min)	CMS Resp (min)
H1	inactive	1	yes	15.75	8.75
H2	inactive	1	no	15.77	8.85
H3	mi=2,sm=50,c=1	1	no	16.17	8.33
H4	H3, max=1	1	no	16.17	8.65
H5	inactive, 1 init	1	no	16.70	8.33
H6	max=1,sm=75,c=4	1	no	16.35	8.67
H7	inactive	1	no	16.75	8.83
H8	max=1,sm=75,c=4	1	yes	15.45	8.58
H9	max=2,sm=50,c=1	2	yes	16.38	8.37
H10	inactive	2	yes	16.12	8.82
H11	max=2,sm=50,c=1	2	yes	15.95	8.20
H12	inactive	3	yes	15.83	8.78
H13	max=1,sm=75,c=4	3	yes	15.42	8.67
H14	inactive	3	yes	15.95	9.02
H15	max=2,sm=75,c=4	3	yes	15.27	8.53
H16	H15, thresh=20/25	3	yes	15.92	8.67
H17	inactive	3	yes	15.72	8.67
H18	inactive (Accel)	3	yes	15.07	9.15

PAGEMON Benchmark Results

Appendix F

Run	Batch Res (min)	CMS Resp (min)	Sys Page Wait (min)	Sys I/O Wait (min)	Sys Page Reads	OS Page Flts	OS Page Wait (min)	CMS Page Flts	CMS Page Wait (min)
H1	15.75	8.75	3.63	1.64	10573	5434	5.31	463	.75
H2	15.77	8.85	3.76	1.53	10699	5590	5.46	462	.77
H3	16.17	8.33	2.88	2.38	9698	5055	4.30	419	.65
H4	16.17	8.65	3.33	2.04	10131	5245	4.79	440	.70
H5	16.70	8.33	2.25	3.73	8236	4309	3.47	360	.51
H6	16.33	8.67	3.30	1.86	10124	5294	4.70	445	.69
H7	16.75	8.83	3.98	1.55	11317	5959	5.73	477	.76
H8	15.45	8.58	3.09	1.93	9633	4965	4.33	430	.69
H9	16.38	8.37	2.89	2.45	9440	4987	4.28	414	.60
H10	16.12	8.82	3.64	1.72	10299	5391	5.26	453	.71
H11	15.95	8.20	2.82	2.35	9243	4843	4.06	406	.57
H12	15.83	8.78	3.81	1.59	10756	5607	5.39	472	.75
H13	15.42	8.67	2.85	1.98	9547	4826	4.06	430	.68
H14	15.95	9.02	3.77	1.43	10830	5521	5.29	477	.77
H15	15.27	8.53	2.99	1.77	9619	4886	4.14	431	.66
H16	15.92	8.67	3.12	2.04	9447	4987	4.32	419	.66
H17	15.72	8.67	3.45	1.70	10252	5322	5.02	445	.70
H18	15.07	9.15	2.16	2.14	10912	5454	4.43	492	.95

PAGEMON Benchmark Results

Appendix F

PERFORMANCE OF A SYSTEM WITH MULTIPROGRAMMING VIRTUAL MACHINES

by

ROBERT ANDREW YOUNG

B. S., Kansas State University, 1975

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1976

ABSTRACT

This document deals with performance of a third generation computing system running a multi-level operating system. Specifically, the operating system to be considered is a conventional multiprogramming non-virtual third generation operating system (IBM's OS/MFT) run in a virtual machine under the control of a virtual machine monitor (IBM's VM/370). This virtual machine runs concurrently with other virtual machines which perform other services.

The particular area to be addressed is real storage management. As both the current literature and available virtual memory implementations recognize, along with a demand paging system such as is used in this system, there must be a facility to limit the level of multiprogramming to avoid the phenomenon of thrashing. This control generally exists either in the form of a working set scheduling algorithm or in the form of more explicit controls such as process deactivation.

In the particular environment under consideration additional complications arise. The level of the operating system which has the greatest control over the multiprogramming level knows nothing about the management of real storage. Also, the level which performs management of real storage has no mechanism

to completely control the multiprogramming level. Furthermore, the communications between these levels is generally restricted by the pre-defined virtual machine architecture.

Various evaluations of performance which have been made previously in this environment are documented. Included in this is discussion of an extension to the virtual machine architecture designed to remove restrictions on the effective multiprogramming level which occur due to the multi-level structure of the system.

Following this discussion a mechanism for extending the virtual machine architecture to allow control of the multiprogramming level is discussed. This facility has been implemented, and the results of performance evaluation studies of the implementation follow. Finally, extensions of this implementation and related areas for future research are discussed.