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.

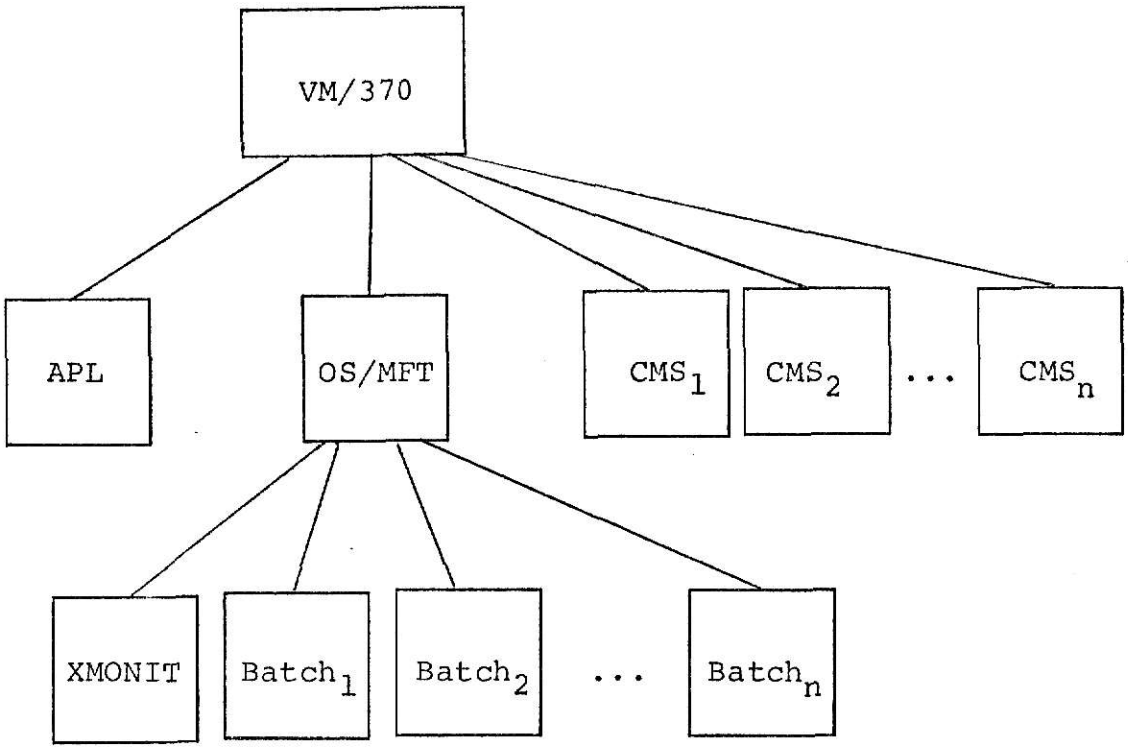
VM/370 also provides a virtual machine-oriented time sharing system called the Conversational Monitor System (CMS). CMS is actually a stand-alone single-user time sharing system designed to be run in a virtual machine. Thus, every CMS user has his own virtual machine which runs the CMS operating system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