The evaluation of a time-sharing page demand system

by JUAN RODRIGUEZ-ROSELL*

The Royal Institute of Technology
Stockholm, Sweden

and

JEAN-PIERRE DUPUY

IBM France Scientific Center
La Tronche, France

INTRODUCTION

The techniques of multiprogramming originated in an attempt to better utilize a computer system’s resources. Multiprogramming supervisory systems are usually rather complicated and their performance is still poorly understood. Some of the reasons why performance should be analyzed are given in Reference 1. It is possible to monitor the system with hardware devices, but analysis must often wait several hours or days before it can be performed. Also, there are quantities that are impossible to reach with conventional hardware monitoring devices. In particular, process identities are lost. This means that some kind of software monitoring method must be envisaged.

A number of articles 2, 3, 4, 5 have discussed different approaches to software measuring. Since operating systems must compute great quantities of information during their operation, it is clear that the best informed piece of software of the system must be the supervisor itself. In the case of CP-67, 10 the system used at the Institute of Applied Mathematics of the University of Grenoble, a simple approach to monitoring can be used.

CP-67 runs on an IBM System 360/67, which is a machine having the dynamic address translation feature. CP-67 creates the time-sharing environment by generating virtual machines. The real resources of the system (CPU, main memory, channels, etc.) are shared among the users concurrently logged-in onto the system. After a virtual machine is generated the user loads in his virtual computer the operating system of his choice, e.g., CMS, OS/360 or even CP-67. Only the real CP-67 runs in supervisor, non-interruptible mode. All the generated virtual machines run in problem state, and the computer is enabled for I/O and timer interrupts while they are active. Privileged instructions executed by a virtual machine cause a program interrupt to CP-67 which analyzes it and determines the action to be taken.

Of special interest are the paging and dispatching algorithms employed by CP-67. 10 For our purposes it suffices to mention that when processes request the system resources (for example, CPU and main memory) they are divided into two sets, interactive and non-interactive, depending on their type of activity. Those members of each set who have been allocated main memory belong to the active class. When a process causes a page fault, the paging algorithm tries to find a page not in use by someone in the active class. If no page is found satisfying this criterion it will choose the first page it finds, in what amounts to essentially random replacement.

Virtual machines communicate with CP-67 by means of the diagnose instruction. 10 Thus, it is possible for a virtual measuring machine to demand the information contained in special locations within CP-67. The information is updated by CP-67, but the data collection and treatment is done by the measuring machine. The treatment includes writing the data in a disk file and displaying the more important information in a CRT or a console.

The transfer of information by the diagnose instruction and the data treatment are imputed to the measuring machine, which is viewed as just another process.

* Presently a visitor at IBM France Scientific Center.
by the supervisor. Furthermore, the virtual machine is dormant waiting for a timer interrupt during the comparatively long intervals between two measurements. The supervisor overhead in running this measuring system is updating the information kept in CP's counters and the process management associated with the virtual measuring machine.

The variables chosen to be observed can be classified as follows:

1. Variables of identification which identify each measure. They are: date, time of day, ordinal number of the present measure.
2. Instantaneous variables. They have a meaning only at the moment at which the measure is taken. There is no relationship between values taken at different times. They are: number of virtual pages in drums and in disks, number of users logged-in, interval between two measures.
3. Cumulative variables. When the operating system is loaded these variables are set to zero. They are incremented by the system. The difference between two values is divided by the time interval between two measures. They are: time in problem state, time in supervisor state, time in wait state, number of pages read per second, number of pages written per second, number of privileged instructions per second.
4. Variables of integration. These are variables whose instantaneous value, or even the difference between two values, lacks significance. Instead it is their time average value that is relevant. For example, the number of processes in the active lists may be the same at the moment of two successive activations of the measuring machine. Yet, during this time interval the multiprogramming level has changed often, reflecting the fact that user processes block and become active again. It has been found necessary to form the time integral of the number of users in the active class. This is the only case where relatively many instructions (about 30) have been added to the system's code. The quantities that must be treated this way are: lengths of queues of the active class, lengths of channel I/O request queues.

The results presented correspond to measures taken during the month of July 1971. The system configuration included 512 K 8-bit bytes. Also, two drums used solely for paging, both on the same channel, and two disk units each consisting of 8 disk packs. Each disk unit has its own selector channel. The system measured was CP-67 version 3 level 0. The computer system is run under control of CP-67 from 8:00 to 12:00 and from 15:00 to 18:00. The time interval between measures was set to 100 seconds of virtual time. Because virtual clocks run slower than real clocks this gives real time intervals of between 100 to about 140 seconds. About 2500 sets of measures were collected in the four-week period analyzed in this report.

It is very difficult to give a characterization, even approximately, of the system load. Heavily used components are PL/1, Fortran, Assembler, Lisp and various other languages. Also context editors and debugging aids that run interpretively. Most virtual memories have either 64 pages or 128 pages. Peak loads correspond to about 40 terminals simultaneously logged in, though several of these virtual machines may be running multi-access systems of their own, such as APL or (virtual) CP-67.

DISCUSSION OF RESULTS

Direct, on-line visualization has proved invaluable in discovering anomalous system behavior. The first lesson we learned is that the system has a tendency to thrash when the number of logged-in consoles exceeds a certain value. The symptoms are a low CPU efficiency and a large number of pages stolen from the members of the active class. Thrashing has been amply discussed in Reference 9 and is well-known to be caused by increasing the multiprogramming level beyond a certain
point. This causes an increase in the page fault rate, which brings an increase in the traverse time (the time necessary to bring a missing page in memory). This leads to a decrease in total processing time. As far as we know, no experimental data has previously been published on this phenomenon.

There are other conditions whose influence on system behavior would have been very difficult to determine without the CRT display. The supervisor time grows when a virtual machine writes on its console, or, more generally, on a virtual device attached to the virtual multiplexor channel. The supervisor time can easily reach 70 percent or more of the total time when programs write large amounts of data using the multiplexor channel. Another effect which is easily discernible is the degradation in performance brought about by insufficient drum space for paging.

Among the interesting results concerning the system is the low CPU problem state utilization, about 25 percent. Another result is the time the machine spends in supervisor state, which is about 35 percent of the total time. These values are well in agreement with values obtained by a hardware monitoring device, and other values reported in Reference 6. This means that the system, although unbalanced, is not CPU bound.

Some of the results concern thrashing and thrashing conditions. Explanations of thrashing have to do with number of processes actively competing for the resources. Paging rates have been given in References 6 and 7, yet they are given as a function of the number of logged-in users, not as a function of the multiprogramming level.

Figure 1 is a plot of \( m_r \), the number of page faults per second of real time, as a function of \( N_\text{AP} \) the mean number of active processes, i.e., the mean multiprogramming level. Figure 2 is a plot of \( m_v \), the number of pages stolen from users in the active class, per second of real time. There is nothing dramatic about \( m_r \), certainly not the steep change that should take place when a certain multiprogramming level is exceeded. Rather, the curve keeps its nearly constant slope in the region shown. Figure 2 indicates that as the multiprogramming level is increased, more and more pages are stolen from the processes in the active class. It has also been found that 70 percent of the pages chosen to be replaced have been modified and must first be written on the drum.

Figure 3 shows the percentage of real time spent in problem state, which we shall call efficiency, as a function of \( N_\text{AP} \). The maximum efficiency is attained for values of \( N_\text{AP} \) between 2.5 and 4. The efficiency decreases as the multiprogramming level is increased beyond 4. This is well in agreement with thrashing behavior, and it suggests that thrashing sets in at this point. Figure 2 confirms this impression.
From Figures 1, 2 and 3, Figure 4 can be deduced. Knowing that the CPU can execute 600 000 instructions per second, Figure 4 gives \( m_p \), the expected number of page faults per instruction executed. This curve is the thrashing curve of Denning. The sudden change in slope occurs for values of \( N_{AP} \) between 3.5 and 4. Before these values, \( m_p \) varies comparatively little. A change in multiprogramming level from 4 to 4.5 brings about a change in \( m_p \) four times as great as a change from 3 to 3.5. The lower curve gives \( m_{vp} \), the number of pages stolen per instruction executed. If we subtract \( m_{vp} \) from \( m_p \) it is apparent that the number of page faults per instruction executed would grow less rapidly if no pages were stolen.

We can conclude that, from the standpoint of paging rate behavior, the system follows what amounts essentially to a working set policy before it gets into memory saturation and thrashing. Afterwards two combined effects take place. One of them is the increase in paging rate because there are, on the average, less pages available in memory for each process. The second effect is due to the paging algorithm that takes the decision of stealing pages which will soon be referenced again. This will make the paging rate still greater.

Any discussion concerning system efficiency must perforce take into account the fact that programs request I/O operations during their execution. If a program executes \( V \) virtual time units it will need a total real time of

\[
T = V + (V m_p) T_p + (V m_{io}) T_{io}
\]

where the quantities \( m_p \) and \( T_p \) are respectively, the number of page faults per unit of virtual time and the transfer time of a page fault expressed in virtual time units. A complete discussion of these quantities is found in Reference 9. Similarly, \( m_{io} \) and \( T_{io} \) are the number of I/O operations per unit virtual time demanded by the process and the time necessary to process one such request. The discussion of the factors affecting these two parameters parallels that of \( m_p \) and \( T_p \). It is interesting, however, to notice that \( m_{io} \) depends only on the process being considered.

Hence, the process efficiency will be

\[
\eta = \frac{V}{T} = \frac{1}{1 + m_p T_p + m_{io} T_{io}}
\]

In order to simplify the discussion we can assume that \( T_p \) and \( T_{io} \) are composed only of seek time and/or rotational delay plus data transfer time. We can also include the time spent in the queue waiting for the request to be processed. The overhead in \( T_p \) and \( T_{io} \) is supposed to be small. Measurement of the paging and disk file channel queues indicate that before thrashing no queues are formed at the I/O channels. Unfortunately this condition does not last long, for when \( N_{AP} \) grows the transfer time \( T_p \) grows, because longer queues form. There is, in addition, the increase in \( m_p \) due to thrashing. In the limit we will have

\[ m_p T_p \gg 1 + m_{io} T_{io} \]

Before thrashing the system efficiency grows linearly with \( N_{AP} \)

\[ \eta_{sys} = \frac{N_{AP}}{1 + m_p T_p + m_{io} T_{io}} \]

As the multiprogramming level is increased longer queues form at the paging channel. If the drum is organized as a FCFS drum, like in CP-67, there will eventually be \( N_{AP} - 1 \) requests in the queue, each request requiring \( T_p \) time units to be served. If \( T_D \) is the time the channel is engaged serving one request we would have

\[ T_p = (N_{AP} - 1) T_D + T_D = N_{AP} T_D \]

The quantity \( m_p \) can be expressed as \( A + B N_{AP} \), \( A, B \) constant.

Consequently,

\[ \eta_{sys} \approx \frac{1}{AT_D + BT_D N_{AP}} \]
And the efficiency decreases hyperbolically with $N_{AP}$. We can then distinguish three regions in the response of the system's efficiency to an increase of the multiprogramming level. In the first region it behaves nearly linearly, in the third region it decreases hyperbolically and in the in-between region it will grow slowly until reaching the maximum possible efficiency (see Figure 3).

The influence of the multiprogramming level on system efficiency has been discussed in Reference 11. However, that model is exceedingly simplified and neither the variation in paging rate nor I/O operations are taken into account. But it is pointed out that some optimum multiprogramming level must exist. Some experimental data are given in Reference 8 but the CPU is near saturation at the measured points.

Proceeding in the same manner as for $m_p$ we find that $m_{io}$ is practically constant. Its value has been found to be

$$m_{io} = 0.1 \times 10^{-3} \text{ requests per instruction}$$

Typically the time necessary to process an I/O request is about 50,000 instructions. Since no queues are ever found at the disk channels the product $m_{io}T_{io}$ can be assumed to remain essentially constant.

Figure 5 shows the experimental values for $n_{WD}$, the number of requests waiting to be served by the drum channel, as a function of $N_{AP}$. To obtain a mathematical model of the system is a more difficult question. Most feedback queuing models assume exponentially distributed service times, which is not the case of the drum. Another model is given in Reference 12 where the drum's service time can be a constant or a random variable. However, in this model file I/O operations are not taken into account. The model discussed in Reference 13 does take into account disk I/O operations, but the I/O request sequence and operations are fixed. Thus it is of no avail in our case. In view of the paucity of existing mathematical models we feel justified in applying the simple $M/G/1$ queuing model to the drum. The data transfer time of the drum is 3.5 msec for a page and a complete revolution is made in 17 msec. Using the Khinchine-Pollaczek formulae we can find the theoretical curve plotted also in Figure 5. The curves diverge after $N_{AP} = 3$, which is about the point where the system begins thrashing.

Figure 6 compares the experimental values obtained by using the results of the $M/G/1$ model for $T_p$ and the measured values for $m_p$. For small $N_{AP}$ the curve is very sensitive to $m_{io}T_{io}$. Further measurements are necessary in order to obtain better estimates of program behavior. As $N_{AP}$ grows, the number of requests actually waiting in the drum queue is greater than the calculated value and their difference causes the theoretical curve to be an upper bound on the system efficiency.

Figure 7 shows the cumulative distribution function for $N_{AP}$. The average multiprogramming level never
reached 7 in this case. All values between 0 and 2.5 have the same probability of occurrence. The most probable value is the same as the mean value and it is 3.0. Assuming the thrashing threshold to be at $N_{AP} = 3.5$ we find that the probability of thrashing is 40 percent. There is only a 25 percent probability of being in the optimum region ($2.5 \leq N_{AP} \leq 3.5$). Of course the multiprogramming level depends on what the users are doing. It seems that 35 percent of the time the multiprogramming level is less than 2.5, meaning that the users are not active enough to impose a heavier load on the system. From the efficiency standpoint it seems that, should the service demand be large enough, it is better to tolerate some thrashing (e.g., $N_{AP} = 4.0$) than to be overly anxious to avoid it, imposing, for example, $N_{AP} = 2.0$.

CONCLUSION AND FUTURE WORK

The system described permits evaluation of the behavior of CP-67 in a given environment. The limiting factor was found to be main memory, since neither the CPU nor the I/O channels were saturated. Main memory has been increased to 1024K bytes, and the measurement system has permitted evaluation of the new configuration.

Since the original dispatcher does not prevent thrashing, it has been modified to follow a working set policy. Data are being collected to analyze its behavior. In particular, more information is needed about process resource demand and I/O behavior.

ACKNOWLEDGMENTS

The authors are indebted to Mr. M. Peltier and Mr. C. Hans, of IBM France Scientific Center, for their guidance and stimulating criticism. Professor B. Arden, presently visiting Grenoble University, has been kind enough to suggest some modifications. A. Fusi, of IBM Italy Scientific Center, and J. P. LeHeiget, of Grenoble University, collaborated in the initial design of the measurement system. Many friends have reviewed this report and the anonymous referees have provided constructive suggestions.

The work of the first author has been supported by a research grant from IBM Svenska A.B. and a scholarship from the French Government.

REFERENCES

1. D J CAMPBELL, W J HEFFNER
   Measurement and analysis of large operating systems during system development
   FJCC 1968
2. H N CANTRELL, A L ELLISON
   Multiprogramming system performance measurement and analysis
   SJCC 1968
3. A KARUSH
   Two approaches for measuring the performance of time sharing systems
   Second Symposium on Operating Systems Principles Princeton University 1969
4. B W ARDEN, D BOETTNER
   Measurement and performance of a multiprogramming system
   Second Symposium on Operating Systems Principles Princeton University 1969
5. J H SALTZER, J W GINTELL
   The instrumentation of Multics
   Second Symposium on Operating Systems Principles Princeton University 1969
6. Y BARD
   CP-67 measurement and analysis: Overhead and throughput
   Workshop on System Performance Evaluation Harvard University 1971
7. S J MORGANSTEIN, J WINograd, R HERMAN
   SIM/01. A simulation measurement tool for a time shared demand paging operating system
   Workshop on System Performance Evaluation Harvard University 1971
8 S R KIMBLETON  C G MOORE
A probabilistic framework for system performance evaluation
Workshop on System Performance Evaluation Harvard University 1971

9 P J DENNING
Resource allocation in a multiprocess computer system
Tech Report MAC-T -50 MIT Project MAC

10 CP program logic manual
Form GY20-0590-0 IBM Corp Technical Publications Department

11 V L WALLACE  D L MASON
Degree of multiprogramming in page-on-demand systems
CACM June 1969

12 Y C CHEN  G S SHEDLER
A cyclic queue network model for demand paging computer systems
IBM Research Report RC 23498 1969

13 G S SHEDLER
A queuing model of a multiprogrammed computer system with a two level storage system
IBM Research Report RJ 836 1971