The performance of VM/370 systems is analyzed in relation to the multiprogramming level and the user work load. Saturation conditions are examined, and methods for locating bottlenecks in the CPU, main storage, paging, and I/O subsystems are given. The paper also describes the data requirements, along with techniques for data collection and reduction. The techniques are illustrated with data from an actual case study.

Performance analysis of virtual memory time-sharing systems

by Y. Bard

The technology of system performance measurement is sufficiently well-developed so that one is generally able to gather unlimited quantities of data on almost any aspect of the system's operation. It is not always clear, however, what should be done with the data once they are gathered. What is often lacking is a straightforward way of answering specific questions relating to system performance. This applies especially to virtual memory systems, and in particular to VM/370 (Virtual Machine Facility/370), which is not as well-known and understood by as many people as the more familiar DOS or OS for the System/360 and System/370.

The performance questions that may be raised range all the way from the most specific (How long will a 15-card assembly take on Monday at 10 AM?), to the most general (Is my system performing adequately?). In this paper, we present a methodology for answering performance questions of the general type. We shall attempt to answer questions such as: Is the system saturated? Under what user load does it become saturated? Which components form bottlenecks? How can these bottlenecks be removed? How are average response time and resource utilization related to the user load? The paper emphasizes practical methods for summarizing and interpreting performance data gathered on systems running in a production environment.

The methodology presented here is based on performance measurement and analysis techniques described in previous papers.[1,2] Those papers emphasized the detection of performance improvements caused by changes in the system. No attempt was made to analyze the causes for improvement or lack of it, nor to relate these to the internal structure of the system. What we shall try to accomplish in the present paper is to relate the trends in the data to the workings of the system, and thus gain an insight into what might cause the system to saturate and its performance to degrade.

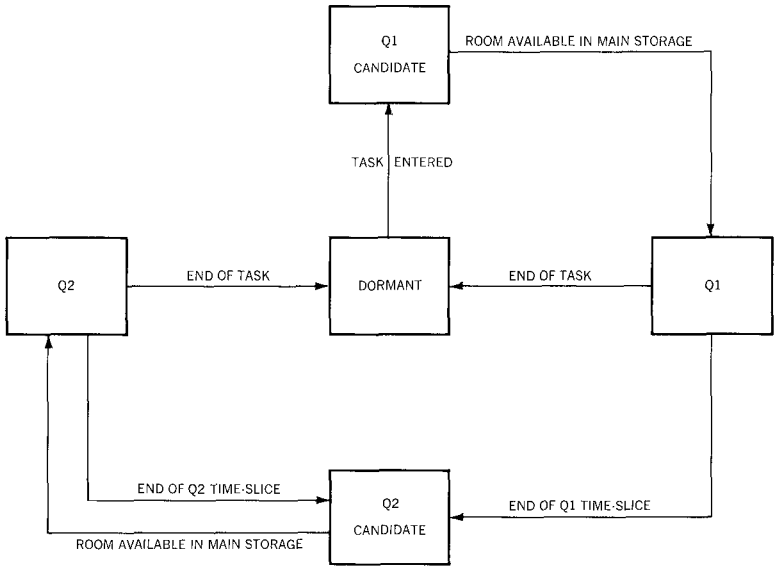
The performance of the VM/370 system depends in a fundamental way on the multiprogramming level maintained by the system. This concept is defined in the first section, which follows, and its relation to system performance is explored in the next section of the paper. The multiprogramming level, however, is an internal variable, and we really want to relate performance to external load variables. Therefore, one needs to determine the relation between the load and the multiprogramming level, which is done in the third section. With the succeeding sections, we start exploring possible system bottleneck areas, including the CPU, main storage, paging, and I/O subsystems. We then specify the performance data required for the analyses, and then describe data collection and reduction methods. Following that is an actual case study that illustrates the use of the previously described methods. Finally, some problems associated with large batch virtual machines are discussed. The conclusion highlights some of the performance questions that were not dealt with elsewhere in the paper. While we have emphasized the analysis of VM/370 performance, the methods described here could be adapted easily to any virtual memory time-sharing system, such as an OS/VS2 system running primarily TSO.

System description

We describe here only those aspects of the system that are relevant to our performance analysis. More comprehensive descriptions may be found in the reference manuals.[3,4]

VM/370 is, for our purposes, simply a virtual-memory, time-sharing system. Each user of the system may enter tasks, usually from a remote terminal. The system shares its resources among these tasks. The flow of user tasks through the system is depicted in Figure 1. A user is in the "dormant" state until he has completed the entering of a task. Until proven otherwise, the task is assumed to be "interactive", i.e., to require fast response while making only slight demands on system resources. While receiving service, such tasks are said to be in "Q1," but before being admitted to this state they are called "Q1 candidates." If a

Figure 1 User states and principal transitions



Q1 task does not terminate before consuming a certain amount of CPU time (roughly 400 milliseconds), it loses its "interactive" status. It now becomes a "Q2 candidate" and is eligible to be admitted into "Q2," which is the set of noninteractive tasks currently being serviced. There is also a limit (about five seconds) on the amount of CPU time that a task may receive during one stay in Q2. A task requiring more CPU time may cycle several times between the "Q2 candidate" and "Q2" states.

We have not shown all possible state transitions, but the ones shown suffice for our purposes.

multiprogramming level

The only tasks that may actually receive CPU time at any moment are those in the Q1 and Q2 states. These are called "in-Q" tasks, and their number is the *multiprogramming level* (MPL). The Q1 and Q2 candidates are tasks that are ready to run but are not allowed to do so at the moment because the system does not wish to overcommit its resources. These tasks are said to be *eligible*. Admissions from eligible to in-Q status is in order of task priority, which is based on a combination of a permanent directory entry, the time when the user had last received service, and optional penalties depending on the user's previous resource demands.[5]

In-Q users' main storage requirements are met dynamically through a demand paging mechanism.[6] The system maintains an estimate of each user's storage requirements; this estimate is referred to as the user's projected *working set*. Admission is

based principally on the availability of main storage space to accommodate the user's projected working set. Pages belonging to users not in Q are always marked available for replacement.

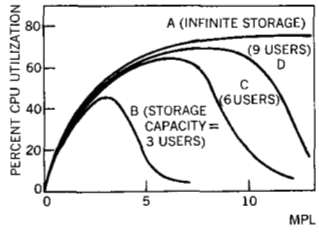
The VM/370 system always runs users' CPU instructions in the problem state and its own CPU instructions in the supervisor state. The amount of time that the CPU spends in the problem state is, therefore, a good measure of "useful" work done, or throughput attained, by the system. Traditionally, attempts to make VM/370 and its predecessor, CP-67, perform well have centered around maximizing percent problem state time, while maintaining acceptable response times for interactive (Q1) tasks.

While a task is in Q, it requires system resources, e.g., main storage, file I/O, and paging I/O, in addition to CPU cycles. The rates at which these resources are being utilized thus provide additional performance measures, as do the response times to tasks of various types. A true picture of system performance can be gained only if all these factors are considered simultaneously.

Relation between MPL and performance

As the MPL goes up, the system is increasingly able to overlap the use of its various components, consequently improving their utilization. Soon, however, one or more components approach 100 percent utilization, so that no further increase is possible. The system is then said to be *saturated.* There is no benefit in increasing the MPL beyond the saturation point, which is determined primarily by the system configuration. Thus, a fast CPU with many I/O channels saturates at a higher MPL than a slow CPU with few channels. The saturation point is also affected by the nature of the work load: if the work is unevenly spread among the system components, saturation will occur sooner than if the load is well-balanced.

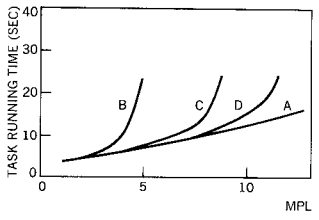
Figure 2 Relationship between throughput and multiprogramming level



At any rate, if the effects of paging are ignored, then curve A in Figure 2 shows a typical relationship between performance (as measured by CPU utilization) and the MPL. Saturation is approached at an MPL of six.[7] This curve might be an idealized representation of a VM/370 system with essentially infinite main storage capacity (hence, negligible paging activity).

In a real system, main storage capacity is finite. As the MPL increases, the amount of storage available to each program decreases, hence the paging rate increases. This increase becomes drastic when the storage allocated to each program falls below its "working set." Soon the paging channel becomes saturated, and further increases in the relative paging rate force down the

Figure 3 Relationship between running time and multiprogramming level

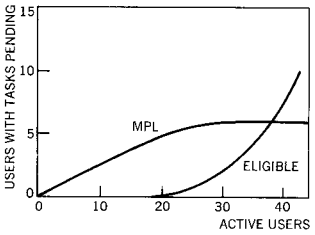


productive utilization of other components, such as the CPU.[8] The results for various storage capacities are also shown in Figure 2. In curve B, storage capacity can accommodate the working sets of three programs. Hence, utilization peaks at an MPL of three. Similarly, curves C and D depict storage capacities of six and nine programs, respectively. Observe that the maximum utilization in curve B is considerably below the infinite-storage saturation level, but is very close to it in curves C and D. It appears that only a limited gain in performance is attainable by increasing storage capacity much beyond accommodating six programs, which is the infinite-storage saturation point. (See the section on main storage for further discussion).

The main function of the system scheduler is to maintain the optimal MPL for the given system configuration and user work load. The scheduler performs its task by estimating each user's working set, and by admitting into Q only as many users whose working sets can be accommodated in main storage.

We are also interested in determining how the running time of a given task is affected by the MPL. Suppose a task requires one second of CPU time. Let $u(n)$ be the CPU utilization at MPL $= n$. If system resources are shared fairly among users, the task receives, on the average, $u(n)/n$ seconds of CPU time per second of elapsed time, and hence its running time may be estimated as $n/u(n)$ seconds. The running time is plotted versus MPL in Figure 3 for the various configurations shown in Figure 2.

Figure 4 Relationship between user load an multiprogramming level



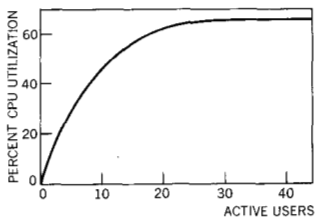
Relation between load and performance

Although it is important to understand the relation between MPL and performance, what one is really interested in is the relation between performance and the load placed on the system. We shall take the number of active users (i.e., those logged-on users who are not taking a coffee break) as the primary measure of system load. The measure is open to much valid criticism—no two users are alike; how can you compare a user who is editing a file under CMS to one who is using the initial-program-loading procedure for a large OS/VS2 virtual machine; etc. Yet measurements taken at many installations have shown that performance variables averaged over reasonable periods of time (one week, say) present very consistent patterns when plotted against number of active users. These plots will be our primary performance evaluation tools.

Suppose the system storage capacity suffices to accommodate an MPL of six. Suppose, further, that the typical user spends nine seconds to enter a task which, under moderate load conditions, takes three seconds of real time to complete. Thus, on the aver-

age, one out of every four users will have a task pending at any one time. As long as the total number of users does not exceed 24, there will be no more than six tasks pending, and these can usually be multiprogrammed (admitted into Q) as soon as they arrive. If the number of users exceeds 24, more than six tasks will be pending, and these must wait in the eligible set before being serviced. These trends are depicted in Figure 4: up to the saturation point (24 users), the MPL builds up, and the eligible set is almost empty. Beyond that, the eligible set grows linearly, while the MPL stays constant.

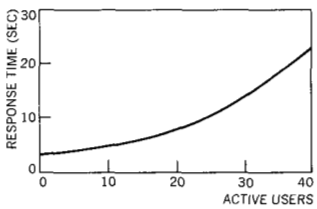
Figure 5 Relationship between throughput and user load



The implications for system resource utilization are obvious. For each number of active users, we enter Figure 4 to find the MPL, and then we enter Figure 2 to find the utilization. The result is shown in Figure 5: utilization climbs up to the saturation point and then levels off. In practice, utilization actually starts to decrease somewhat beyond the saturation point, due to increased contention and increased consumption of storage space by system control blocks.

Observe that if the storage capacity was sufficient to accommodate nine tasks, Figure 5 would remain essentially unchanged. In Figure 4, however, saturation would occur at a user level of 36, rather than 24. Thus, the MPL plot by itself is not a good indicator of how many users will saturate the system.

Figure 6 Relationship between response time and user load



We can also determine how task response time is affected by system load. Suppose we have a specific task in mind, requiring one second of CPU time to complete. The response time is composed of the running time, obtainable from Figure 3, and the waiting time in the eligible set, which is proportional to the number of tasks in that set. The proportionality factor is easily determined: If the average task required T seconds of CPU time, and the MPL $= n$, then a task leaves Q1 or Q2 on the average of every $T/u(n)$ seconds. Hence, if there are m eligible tasks, the expected waiting time is $mT/u(n)$. Thus, Figures 3 and 4 can be used to generate Figure 6, which shows the total response time. It is possible to verify that Figure 6 would be relatively unaffected by increasing storage capacity beyond the six-user level. Figure 6 shows the essential effect of saturation on response time. Below the saturation point, the system possesses unused resources and is able to serve additional users without severely impacting response time. When saturated, additional users can be accommodated only at the cost of reduced service to all.

Our analysis applies only to a "typical" task, i.e., one whose various resource requirements are roughly proportional to those of the overall work load. Radically different types of tasks, e.g., compute-bound or I/O-bound ones, may present completely dif-

ferent response profiles. A system whose I/O channels are saturated may have enough leftover CPU power to maintain excellent response to a compute-bound task, and vice-versa.

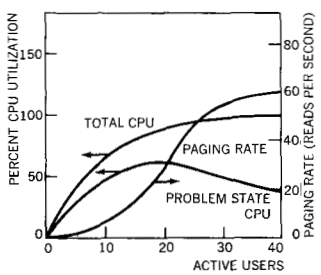
Bottlenecks

Plots such as Figures 5 and 6 can be used to spot saturation conditions and determine the user load under which they occur. If there is no evidence of saturation, all is well. However, if saturation is detected, we must next determine the system component that is causing the saturation. This component, which we shall call the *bottleneck*, is the one whose capacity must be increased first before overall system performance can be improved. Such improvement would be felt in two ways: An increased number of users could be served before the onset of saturation, and system throughput at saturation would be increased.

The main hardware components of a VM/370 system are the CPU, main storage, the paging subsystem, and the I/O (other than paging) subsystem. We shall describe below how bottlenecks in any one of these components can be spotted, and what remedies are available to remove them.

The CPU

Figure 7 CPU saturated with paging overhead



The CPU is saturated when its utilization approaches 100 percent. A truly saturated CPU can be cured only by being replaced with a faster one. However, some further analysis may reveal different underlying causes for the saturation and suggest cures of a less drastic nature.

One case in point occurs when the CPU becomes saturated with overhead due to paging. The case is shown in Figure 7: total CPU utilization approaches 100 percent, problem state time declines, and paging rate climbs as the number of users increases. Such conditions prevail if, for some reason, the scheduler consistently underestimates working sets and thus maintains too high an MPL. Reducing the MPL will release some of the CPU time spent on paging, but whether or not the remaining MPL will be sufficiently high to maintain good throughput depends on the amount of storage available (see below). Increasing storage capacity while retaining the same MPL would also decrease the paging rate and release some CPU time for productive use.

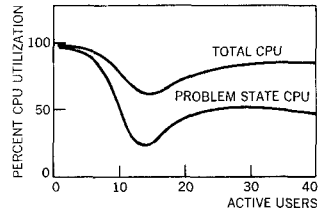
The presence of compute-bound jobs in the work load can result in very high CPU utilization. Some of the CPU time used by these tasks is really available to other users, should they need it. Thus,

if response time to the compute-bound tasks is not a primary concern, then we should not really consider the CPU to be saturated. Figure 8 is the profile of a system that is always running a fixed number of compute-bound jobs, while the remaining users are of a more "normal" type. Profiles such as Figure 8 appear quite commonly if data are plotted indiscriminately, since the lower portions of the curve may represent the off-shift running of large compute-bound jobs.

Main storage

Main storage is (or at least the scheduler thinks that it is) saturated when the eligible list is almost never empty. Nevertheless, a saturated memory is not necessarily a performance bottleneck. If paging is moderate and the CPU is fully utilized, then main storage capacity is adequate and will have to be increased only after a more powerful CPU is installed. If both paging and CPU utilization are light, then the scheduler is probably overestimating working sets and consequently maintaining too low an MPL. If the paging rate is high, productive CPU utilization (percent problem state time) is low, and the MPL is high, then the scheduler may be at fault. This so-called thrashing condition may be removed by inducing the scheduler to maintain a lower MPL. Only if the MPL is low, paging is heavy, and productive CPU utilization (percent problem state time) is low, is the saturated main storage a true bottleneck and in need of expansion.

Figure 8 Some users compute-bound



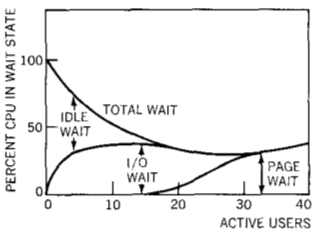
Generally, acceptable performance is achieved if storage is expanded only to the point where an adequate MPL can be maintained. However, as illustrated later in the case study, additional performance improvements are attainable by further increases of storage capacity above the saturation point. If excess storage capacity is installed, then a substantial number of pages belonging to interactive users can be held over from one interaction to the next. The total paging rate is thereby reduced, and CPU time previously spent on paging overhead is freed for productive use. Furthermore, response time to interactive tasks is improved. However, as the number of users on the system increases, the amount of excess storage required to hold temporarily inactive working sets increases proportionally.

Paging subsystem

The VM/370 system breaks up total system wait time into three components:

1. Idle wait, when no high-speed I/O requests are outstanding.
2. Page wait, when outstanding I/O requests are primarily for paging.

Figure 9 Paging-bound system

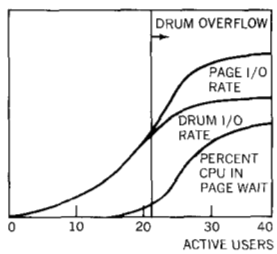


3. I/O wait, when outstanding I/O requests are not primarily for paging.

Typically, the idle-wait state predominates when the number of users is small. By the time saturation is reached, idle wait will have decreased to zero. Absence of idle wait in itself, however, is no proof of saturation: when the number of users is moderate, there may be enough work to keep either the CPU or an I/O path busy at any given time, but not enough to saturate either.

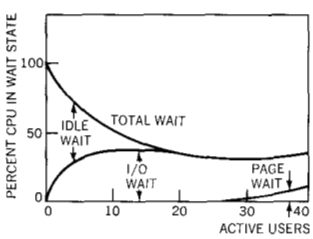
If the CPU is not the bottleneck, there will be a substantial amount of wait state even at saturation. This wait state may be due to poor overlapping of CPU and I/O activities, caused by main storage being insufficient to accommodate an adequate MPL. If this is not the case, however, and if page wait accounts for the major part of the residual wait time (see Figure 9), then the paging subsystem is probably at fault.

Figure 10 Paging drum overflow



Page wait may be experienced either because the paging rate is too high or because page transit time is too long. The first condition, caused by working sets being underestimated by the scheduler, has been dealt with above. The second condition occurs when either no high-speed paging devices are installed or their capacity has been exceeded. The system may normally page to a fixed-head storage device ("drum") such as the IBM 2305. But when the number of users is sufficiently large, the drum overflows and some paging will be to a slower device such as the IBM 3330. The point at which overflow becomes significant can be determined from a plot of total page I/Os and drum I/Os versus number of users (Figure 10). Various remedies, short of installing an additional drum, may apply. Freeing virtual pages that are no longer needed;[9] reduced size of virtual machines; applications reprogrammed to use less virtual storage; increased use of shared systems—all these may "stretch out" the drum capacity. These remedies are of no use when there is no drum to start with. In that case, one may possibly improve performance by spreading the paging areas over more devices. This reduces seek times and permits better overlapping of seek and transmission times. Also, other I/O activity (if any) should be removed from the paging channel.

Figure 11 I/O-bound system



I/O subsystem

A bottleneck in the I/O subsystem reveals itself in a manner analogous to the paging subsystem. If there is enough main storage to maintain an adequate MPL, and yet a significant amount of I/O wait time remains at saturation (Figure 11), a deficient I/O subsystem is indicated. It may be simply that the work load is of so I/O-bound a nature that no feasible expansion of the I/O facilities

Table 1. Selected VM/370 counters

Symbolic name	*System block*	*Type (See key below)*	*Description*
DMKSYSNM	SYSLOCS	3	Number of logged-on users
PROBTIME	PSA	2	CPU time in problem state
IDLEWAIT	PSA	2	CPU time in idle wait state
PAGEWAIT	PSA	2	CPU time in page wait state
IONTWAIT	PSA	2	CPU time in I/O wait state
PGREAD	PSA	1	Page reads into main storage
PGWRITE	PSA	1	Page writes out of main storage
DMKPTRSS	*	1	Pages stolen from in-Q users
DMKDSPNP	*	3	Page frames available for paging
DMKVIOCW	*	1	CCW translations
DMKHVCDI	*	1	DIAGNOSE I/Os[11]
DMKPAGPS	*	1	SIOs to paging device
RDEVIOCT	RDEVBLOK	1/D	SIOs to real device
VMUSER	VMBLOK	3/U	User ID
VMDSTAT	VMBLOK	3/U	User Q status
VMQLEVEL	VMBLOK	3/U	User Q status (continued)
VMVTIME	VMBLOK	2/U	Virtual CPU time accounted to user
VMTTIME	VMBLOK	2/U	Supervisor CPU time accounted to user
VMPGREAD	VMBLOK	1/U	Pages read by user
VMPGWRIT	VMBLOK	1/U	Pages written by user
VMIOCNT	VMBLOK	1/U	Nonspool I/Os by user
VMPAGES	VMBLOK	3/U	Number of user pages residing in main storage
VMWSPROJ	VMBLOK	3/U	User's estimated working set

*The addresses of these counters may be obtained at run time in the DMK SYM load map, as described in the Appendix.

Counter Types: 1 : Event counter
2 : Time accumulator
3 : Current status indicator
/D : One for each real I/O device
/U : One for each logged-on user.

will handle it. In this case, one might conclude that the CPU in use is too fast, and a slower one would suffice. More typically, some rearrangement and/or expansion of the I/O subsystem will cure the problem. It will be necessary to measure the utilizations, or at least the I/O rates, of the individual I/O channels and devices. Then, better-balanced loading can be achieved by moving physical packs from one channel to another, or by moving assigned mini-disk areas from one pack to another, by creating multiple copies of heavily used shared disk areas (e.g., the CMS system disk). One must caution, however, that the usage patterns of user mini-disk areas are often quite volatile. These areas should be moved about only if consistent patterns are detected.

Under extreme conditions, an I/O bottleneck may develop if the MPL is too *high*, rather than too low. As the MPL increases, so does the likelihood that several users sharing the same disk drive will find themselves in Q together, giving rise to high arm

contention. If one observes a coincidence between very high MPL and high I/O wait time, one should amend the scheduler so as to restrict the MPL to a reasonable level.

It is possible for both page and I/O wait to account for significant portions of elapsed time at saturation. In this case, improvements in either subsystem will be useful, but improvements in both may be required to eliminate the bottleneck entirely.

Data requirements

The foregoing analyses require that performance variables be measured over a certain time period, say, one week of routine operation. The measurements are made possible by a set of "counters" that are automatically maintained by the VM/370 control program. These counters are of three types:

1. Event counters, incremented each time the event occurs (e.g., a page read).
2. Time accumulators, incremented each time the system changes state (e.g., the CPU leaves the wait state).
3. Current status indicators (e.g., number of logged-on users).

Furthermore, there are separate counters for overall system events, for each logged-on user, and for each I/O device. Table 1 contains a list of those counters which are of most interest to us. Others are described in Reference 10.

Some items of interest are not measured directly but can be calculated easily from measured items. For instance:

CPU in wait state	= idle + I/O + page wait
CPU in supervisor state	= Elapsed time − problem state − wait state
User-initiated I/Os	= CCW translations + DIAGNOSE I/Os[11]
Drum I/Os	= Sum of I/Os over drum devices
Channel *n* I/Os	= Sum of I/Os over channel *n* devices

Typically, the counters will be sampled at more or less regular intervals (see next section on how this might be done), and differenced to obtain measures of system activity during each sampling interval. In most cases, these differences will be divided by the length of the time interval between measurements to obtain event rates or percentage utilizations. In addition, one will be able to calculate the values of several variables at the sampling instant:

MPL	= Number of in-Q users

Q Candidates = Number of eligible users (runnable but not in Q)
Storage Used = Sum of pages belonging to in-Q users
Storage Demanded = Sum of working sets of eligible and in-Q users

One can also compute:

Active Users = Number of users who have used CPU time during sampling interval

While sampling procedures give good estimates of cumulative system activity, this may not be the case with instantaneous values such as the MPL, which can fluctuate rapidly. This problem may be solved in two ways:

1. *Event-Driven Monitoring*—If a time-stamped record is made of each user state transition, it is possible to compute, say, the MPL at each instant. From that, the average MPL over any sampling period may be computed exactly.
2. *Integrating Counters*—Additional counters may be implemented in the control program. At each change in a variable of interest, its counter is incremented by the previous value of the variable multiplied by the time elapsed since the previous change. The counter is sampled periodically, and its increment divided by elapsed time gives the average value of the variable. This method requires less overhead than the preceding one.

We also need measurements of response time. If we wish to compute the average response times to, say, all Q1 and Q2 tasks, we again have the same two alternatives:

1. *Event-Driven Monitoring*—The time-stamped records of user state transitions can also be used to compute the time it takes for each round trip through the eligible list and Q1 or Q2. This is essentially the response time to a task requiring one trip through Q.
2. *Integrating Counters*—Here we need two counters for each user state. One counter accumulates the number of entries into that state, the other accumulates the lengths of the stays in that state. The ratio of the second to first counter gives the average stay.

A completely different method for measuring response time is to devise one or more benchmark programs and to run these at regular intervals throughout the data-gathering period.[2] These benchmarks are run in a separate virtual machine, while the system is also servicing its normal work load. They may be regarded as "thermometers" which are inserted into the system to

measure its "temperature"—in this case, its responsiveness to the types of requests characterized by these benchmarks, which may include trivial interactive, I/O-bound, compute-bound, or mixed tasks. Most conveniently, the benchmarks may be "synthethic" in nature, consisting of one subroutine that may be called with parameters specifying how much CPU time should be used, how much file and terminal I/O activity should be performed, and how many pages should be referenced. The subroutine contains CPU, I/O, and memory usage loops, and the number of executions of each loop is determined by the values of the calling parameters.

Measurement techniques

In this section, we describe two techniques that may be used to extract the information from the counters maintained by the system.

The data-gathering monitor may be implemented as part of the control program. The monitor is entered upon the occurrence of a timer interruption that is set for the required sampling period. The required data are moved to a buffer area, whence they are written onto tape or disk. Event-driven monitoring may be accomplished by inserting MONITOR CALL instructions in the appropriate places in the control program. By using different codes for different types of events one can establish flexible controls over the types of data to be gathered in any monitoring session.[12] Such a monitor is included in the VM/370 system starting with the Release 2 PLC 13 version.

A second type of monitor is one that runs in a virtual machine. The VM/370 system permits a privileged virtual machine to read the contents of specified locations in the control program's address space (see Appendix for details). A virtual machine may also "put itself to sleep" pending the arrival of a timer interruption. By using these facilities, the virtual machine may sample the system counters at specified intervals. Each time it wakes up, it may also run the synthetic benchmarks described in the preceding section, and record their running times along with the sampled values of the counters. This type of monitor cannot obtain event-driven records and, therefore, is restricted in the amount of detail it can provide. Its sampling period can be regulated only approximately.

The observations taken by this method will be biased by the fact that the measuring virtual machine must be in Q and running at the time the observations are taken. The variables most severely affected are the MPL and the storage used. Furthermore, the overhead imposed on the system by this type of monitor is quite

variable and difficult to account for. And, perhaps worst of all, the monitor itself becomes part of the work load that it attempts to measure. These effects may be minimized by running the monitor at a frequency commensurate with the power of the system being measured. However, the virtual machine monitor also offers some advantages: it requires no changes to the control program, it consumes no real storage space when not actually running, it may be written in a higher-level language, (except for a simple interface routine), and the data it obtains are immediately available in a virtual machine, so that they can be analyzed to provide performance diagnostics in real time. Furthermore, changes are easily implemented and tested.

Data reduction techniques

We shall assume that counter readings have been taken at intervals during the data collection period. We shall refer to each reading as an *observation* and to the time between successive readings as an *observation period.* It is necessary to reduce the data so that plots such as Figures 5–11 may be drawn. The reduction proceeds as follows:

1. Compute the number of active users for each observation period.
2. Group together all the observations for which the number of active users is one or two; similarly, all observations for which the number of active users is three or four, and so on. Coarser groupings may be required if the total number of observations is small. One would like to *average* at least 50 observations per group.
3. Compute the mean and standard deviation of each performance variable within each group of observations.
4. For each performance variable of interest, plot the mean within each group, against the number of active users for that group (use the median value; e.g., use 1.5 for the group containing observations with one or two active users).

When the plots are made, they may immediately reveal some of the trends illustrated in Figures 5–11. Sometimes, however, the trends will be masked by random fluctuations in the data. This is most likely to occur in those portions of the curves where relatively few observations are available—typically in the upper range of active user values. If random fluctuations predominate, it may be necessary to aggregate the groups further. For instance, instead of breaking the range up into 1–2, 3–4, . . . use 1–4, 5–8, . . . If even this fails to produce meaningful trends, more data are probably required.

If no clear trends are apparent from the plots, it might be worthwhile to determine whether this is due to inadequate data. For

Figure 12 Scatter due to insufficient data

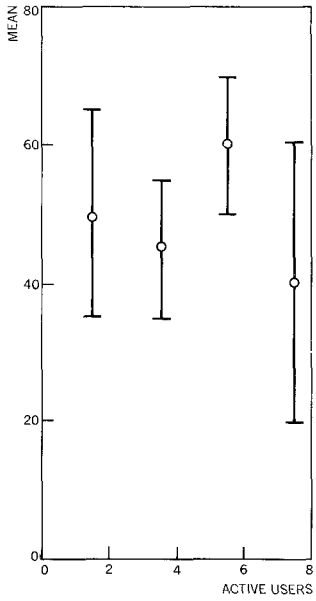
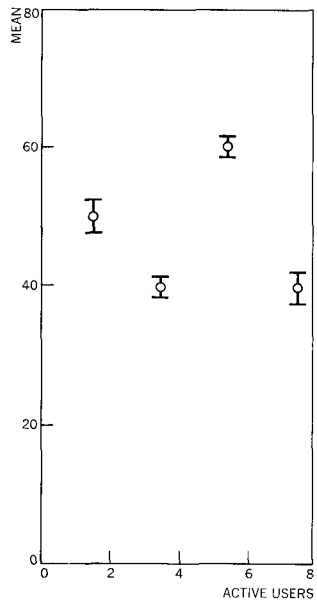


Table 2 Sample data

Active users	*Number of observations*	*Mean*	*Standard deviation*	*Standard error*
1–2	4	50	30	15 (= 30/ $\sqrt{4}$)
3–4	16	45	40	10
5–6	9	60	30	10
7–8	4	50	40	20

this purpose, compute the *standard error* of each mean value computed in Step 3 above. The standard error is simply the standard deviation divided by the square root of the number of observations in the group. Then, erect a vertical line through each point on the plot, extending from one standard error below the mean to one standard error above the mean. Sample data (rather artificially concocted) appear in Table 2, and the corresponding plot in Figure 12. If one looked at the mean values alone, one would think that percent problem state time fluctuated erratically as a function of the number of users. The standard errors, however, show that the fluctuations are probably insignificant, and that more data are required. Suppose, however, that one hundred times as many obervations were taken, without changing the means and standard deviations recorded in Table 2. The vertical lines would be reduced to one tenth their height (Figure 13). In this case, the fluctuations would appear to be significant, and some attempt to find their cause might be made.

Figure 13 Scatter due to inherent nature of data



For more rigorous methods of analysis, including tests of hypotheses and significance tests, the reader is referred to standard statistical texts.

The values of any performance variable within each group of observations possess a certain statistical distribution. So far, we have used the estimated mean and standard deviation of that distribution. Frequently, it will be preferable to deal with the percentiles of that distribution (for example, the 75 percentile of a distribution is the value below which 75 percent of the observations fall). There are two cases where use of percentiles is preferable:

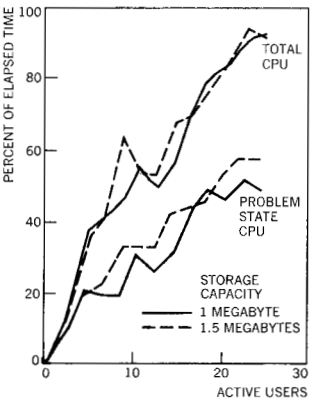
1. If the variable in question has occasional abnormally large observed values, then the mean and standard deviation are very much influenced by these outliers. The median (50 percentile), the 75 percentile, etc., are then much more representative of "typical" values. Variables most likely to fall in

this category are benchmark response times. We have found the 75 percentile of benchmark response time to be a particularly informative measure of performance. Higher percentiles (e.g., 90) are too sensitive to occasional abnormally long response times, whereas lower percentiles (e.g., the median) are not sufficiently indicative of user satisfaction with the service they are receiving.

2. For installation planning purposes, one would wish to know what system resource capacity would be needed to satisfy a certain percentile of requests. One might, for instance, wish to know how many users should be allowed on the system at any time so that the chance of saturation is only 25 percent, or one may wish to design storage capacity so as to accommodate an MPL of six at least 90 percent of the time.

Needless to say, the percentiles may be plotted against number of active users, producing curves similar to the ones obtained by plotting the averages.

Figure 14 CPU utilization (case study)



A case study

This case study deals with a VM/370 installation running on an IBM System/370 Model 155-II, with one megabyte of main storage, one byte-multiplexer channel, and five block-multiplexer channels. One of the latter was dedicated to the primary paging device, an IBM 2305-II fixed-head storage facility ("drum"). The work load was generally of a time-sharing nature, with most virtual machines operating under the Conversational Monitor System (CMS). The data presented by the solid curves in Figures 14–19 were derived from about 1000 observations taken over a week's normal running period by means of a virtual machine monitor. Figures 14–16 all demonstrate the onset of saturation somewhat below the 20 active-user level. Figure 17 shows that page wait accounts for an increasing fraction of total wait state in the saturated region, and Figure 18 shows the total paging rate increasing rapidly, with the paging drum beginning to overflow at the saturation point. Figure 19 shows that main storage is not saturated in the region of operation.

Figure 15 Average response time for interactive tasks (case study)

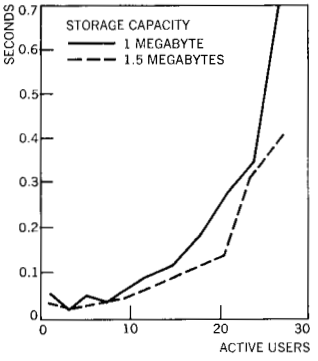
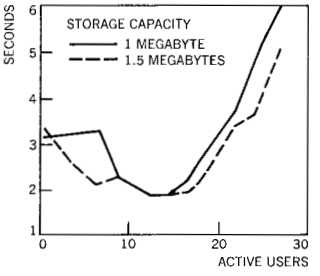
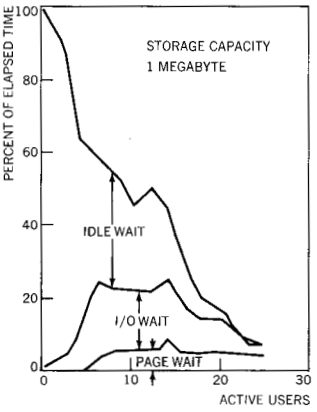


Figure 16 Average response time for noninteractive tasks (case study)



The data lead to the following conclusions: The main bottlenecks are (1) CPU overhead due to paging; (2) page wait due to drum overflow. The second factor could be eliminated by the installation of a second drum. The first factor, and the adverse effects of the second factor, would be mitigated by reducing the paging rate. This could be achieved by increasing main storage capacity to minimize the loss of users' pages between stays in queue. The second solution was adopted and the improved performance with 1.5 megabytes of main storage is indicated by the dashed curves in Figures 14–19.

Figure 17 Analysis of CPU wait state (case study)

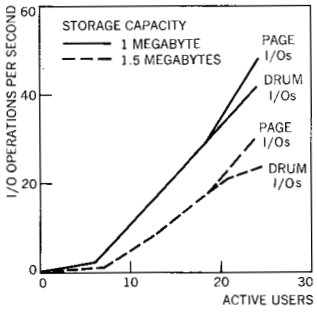


Nongranular systems

It has been assumed heretofore that the work load presented to the system possessed a certain amount of granularity. The "active user" was used as the unit of the granularity, and it was expected that the work demanded from the system increased monotonically, if not proportionally, with the number of users. This property was important because it permitted us to spot saturation conditions: if throughput does not increase in the face of increasing demand, then the system must be saturated. Once saturation has been established, identification of the bottleneck is no longer heavily dependent on the granularity of the work load.

The work load in some installations is nongranular in type. Specifically, we are referring to systems where the work load is composed primarily of one (or a few) batch virtual machine. In this case, performance of the VM/370 system depends so strongly on that of the batch system, that not much can be said of the former without knowing something of the latter. For instance, if the batch virtual machine is itself running a multiprogrammed operating system, then the true multiprogramming level cannot be determined from the VM/370 counters alone. For this reason, it is difficult to devise a systematic approach, and we shall have to content ourselves with a few random remarks.

Figure 18 Paging activity (case study)



If main storage capacity is sufficient, throughput may well increase if the batch job stream is broken up into multiple streams that are fed to multiple virtual machines, each running a copy of the operating system. Once this is done, the work load attains a degree of granularity that may permit performing the previously described analyses.

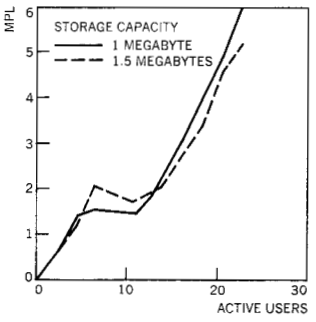
If the system is believed to be saturated, then looking at CPU utilization, breakdown of wait state, paging and I/O rates, etc., can isolate the bottlenecks, just as it did in the granular case. If the CPU is not the bottleneck, however, it may be difficult to determine whether or not the problem arises from having an inadequate MPL. Internal measurement of the virtual operating system may be required.

Conclusion

In this paper, we have dealt with some aspects of system performance, but we have left many questions unanswered. The analyses we have described will tell us, hopefully, whether the system is saturated and which components are at fault, but further analysis will be required to determine exactly how much additional capacity is needed to achieve a desired performance level.

Frequently, reasonable projections can be made on the basis of our plots, but in general, the evaluation of proposed configurations will require additional tools such as analytic[13] or simulation models. We have already alluded to the need to consider the entire distribution of requests, and not merely the averages, in the process of configuring a system.

Figure 19 Average multiprogramming level (case study)



Another type of question that we have left open relates to very specific performance questions. Even when 75 percent of all requests receive satisfactory service, some users may be complaining about poor response. Careful investigation may reveal that they are all competing for the same disk arm, or their working sets are enormous, or the operator is unresponsive to their tape requests. Or, by some odd coincidence, 10 users may have started long compute-bound tasks at once. An event-tracing monitor[12] is particularly helpful to reveal the underlying causes, but on-line reduction and summarizing of data sampled by a virtual machine monitor will often provide sufficient and timely information about what is going on. Reasonable familiarity with system internals is usually required for interpreting the data correctly.

CITED REFERENCES AND FOOTNOTES

1. Y. Bard, "Performance criteria and measurement for a time-sharing system," *IBM Systems Journal* **10,** No. 3, 193–216 (1971).
2. Y. Bard, "Experimental evaluation of system performance," *IBM Systems Journal* **12,** No. 3, 302–314 (1973).
3. *IBM Virtual Machine Facility/370, Introduction,* Form No. GC20–1800, IBM Corporation, Data Processing Division, White Plains, New York (1972).
4. *IBM Virtual Machine Facility/370, Control Program (CP) Program Logic,* Form No. SY20-0880, IBM Corporation, Data Processing Division, White Plains, New York (1972).
5. C. J. Young, *VM/370 Biased Scheduler,* IBM New England Programming Center Technical Report TR 75.0001, Burlington, Massachusetts (1973).
6. R. P. Parmelee, T. I. Peterson, C. C. Tillman, and D. J. Hatfield, "Virtual storage and virtual machine concepts," *IBM Systems Journal* **11,** No. 2, 99–130 (1972).
7. All specific numbers (except in the case study) appearing in this paper are for illustration only. They do not pertain to any specific VM/370 installation, nor are they claimed to be typical.
8. Overhead due to paging may actually increase *total* CPU utilization.
9. In VM/370 this may be done by using a DIAGNOSE instruction in the virtual machine.[4]
10. C. W. Endee and R. L. Goodman, *VM/370 CP Counters,* IBM New England Programming Center Technical Report TR 75.0004, Burlington, Massachusetts (1974).
11. A special type of low-overhead I/O operation permitted by VM/370.[4]
12. P. H. Callaway, "Performance measurement tools for VM/370," *IBM Systems Journal* **14,** No. 2, 134–160 (1975).
13. Y. Bard, "An Analytic Model of CP-67 and VM/370," *International Workshop on Modelling and Evaluation of Computer Architectures and Networks,* IRIA, Rocquencourt, France (1974). Also available as IBM Cambridge Scientific Center Technical Report No. G320-2101, Cambridge, Massachusetts (1974).

Appendix: Accessing VM/370 counters from a virtual machine

Suppose the contents of VM/370 counters which are known to have real addresses $A1, A2, \ldots, AN$ are to be stored in N consecutive words, starting at virtual address L. The following procedure would be used:

1. Store the addresses $A1, A2, \ldots, AN$ in N consecutive words starting, say, in virtual location M.
2. Load the address M into some general purpose register, say $R1$.
3. Load the number N into some other general purpose register, say $R2$.
4. Load the address L into general purpose register $R2 + 1$.
5. Issue the diagnose instruction X'83', R1, R2, X'0004'.
   Note: This diagnose instruction may be executed only by virtual machines having privilege class E. The diagnose instruction, the locations L through $L + 4 \times N - 1$, and the locations M through $M + 4 \times N - 1$ must all be in the same virtual page, i.e., their virtual addresses may differ only in the three low-order hexadecimal digits. It is permissible to have the data overwrite the addresses, i.e., to have $L = M$. Since the diagnose instruction and a branch instruction following it would take up two words, the total number of data words obtainable with one diagnose is 1022.

The addresses of most counters can be found in the system macroinstructions indicated in Table 1. The remainder may be found at run time in the DMKSYM load map, which may be read into the virtual machine's address space by issuing the diagnose instruction X'83', R1, 'X'00038', where register $R1$ contains the address of a page-aligned, 4096-byte buffer into which the load map is to be read. Each entry in the load map consists of 12 bytes: the symbolic name in EBCDIC (eight bytes), followed by the real address (four bytes).

Reprint Order No. G321-5022