Sampling the state of interactive computer users of hardware and programming in a time-sharing system leads to an understanding of delays to users caused by contention for resources. This paper discusses user state sampling by means of a program called VM/Monitor on the interactive time-sharing system, VM/370, although the methodology is applicable to other time-sharing systems. Also discussed are system bottleneck detection and secondary tuning after bottlenecks have been found. Possible extensions of the technique are also presented.

State sampling of interactive VM/370 users

by W. H. Tetzlaff

A fundamental change is taking place in the economics of computing systems as they interact with people: The cost of the hardware is decreasing, while labor costs are increasing. In time-sharing systems, the larger expense is also shifting from the hardware to the logged-on users.[1] This shift motivates installation management to understand the user characteristics in order to maximize the efficiency of the enterprise as a whole, rather than the computing hardware alone.

Computing systems typically cost hundreds of dollars per hour to maintain, whereas individual system user costs are usually tens of dollars per hour. For a large number of users on an interactive system, the cost for all users can be thousands of dollars per hour. If the key user-effectiveness bottlenecks in a system can be located and broken, the gain can be very large.

Additions to a computing system that can break bottlenecks and greatly add to the user productivity often add only tens of dollars per hour. When a key bottleneck is found, dramatic throughput changes may be realized by augmenting the critical path with such hardware as more main storage, additional input/output (I/O) paths, faster I/O devices for important data, or additional access arms. Calculation of the incremental cost of a change is normally an easy task, although measurement of the incremental user throughput improvement is more difficult. Both topics are discussed in this paper.

It is important to be able to solve system-performance problems as quickly as possible. When performance problems are investigated by analyzing large amounts of data, as they usually are, computer programs are needed to process the data. The writing of these programs is a costly effort, and the programming process is placed in the analysis process. This is particularly time consuming because the development of one program is seldom enough. The completion and running of one data reduction program may provide some insight into a problem, but such a program usually raises further questions that require more data reduction.

Even when a large number of data reduction programs have been created, they have never been sufficient to handle all future problems because many system variables change with time. Operating system changes, for example, redefine the meaning of data items. Operating system changes also cause new data to be collected. New problems require new views of the data. The best hope of dealing with a problem is to mechanize the process of writing the data reduction programs that deal with it. Problems are solved faster, because new programming is no longer a part of the process. A program generator would have the potential of being easy to adapt to changing data collectors and operating systems.

Data collection and reduction

At the IBM Thomas J. Watson Research Center at Yorktown Heights, NY, a program called VM/Monitor[2] collects data during the first shift for two VM/370 systems. VM/Monitor has long been the primary VM/370 data collection tool[3] at Yorktown, as the Research Center is familiarly called. VM/Monitor collects system performance and resource utilization data by means of sampling and trace techniques, and writes the data collected on tape or (under Release 5) on a spool file. Some of the captured events—terminal input, for example—are caused by the activities of system users. Other events correspond to the use of resources, such as individual Direct Access Storage Device (DASD) accesses. Still other events are caused when scheduling decisions are made: for example, moving a user from one scheduling queue to another. Sampling is initiated by the expiration of an interval of time. The time-driven events are used to cause information about each individual user to be recorded, as well as to cause information about DASD and tape device utilization to be written periodically.

During normal operation, we use VM/Monitor options that cause the recording of summary information at one-minute intervals. This procedure minimizes overhead, yet gives a good picture of system performance. A single logical record is written that summarizes system-wide activity during the last interval of time. Another logical record is written that summarizes device activity on

all DASD and tape devices. Finally, logical records are written for all logged-on users to summarize their activities during the previous interval.

generalized reduction of information

Data reduction of the VM/Monitor data is accomplished through the use of a special data reduction package produced at Yorktown and known as the generalized reduction of information program. This program is a program generator that has been designed to increase the productivity and effectiveness of performance analysts by allowing analysts to spend more time on analysis and less time on either defining or writing data reduction programs. Mechanizing the job of writing data reduction programs has greatly improved the speed of problem analysis. A program generator frees analysts from dealing with the details of input and output formatting, and it automatically performs many useful forms of data summarization.

It is important to consider the nature of performance data before attempting to write a processor for it. In the case of system performance data, there are a number of characteristics to be kept in mind. The quantity of data can be very large. Traces of frequent operating system events can produce enormous but still useful quantities of data. A data reduction package must be able to process large quantities of such data without consuming an unacceptable amount of resources. In many cases, the data are used only once, and often they are not used at all. Elaborate data base preprocessing for performance data that are used only once, or not at all, is not sensible.

Typically the basic structure of a system performance data base is stable, and the content of the data base records changes only modestly. In most cases, the type of change is the addition of new data items. The collected data usually have a hierarchical structure that may not be as evident in the physical layout of the data as it is in the logical relationships among the data items.

The types of reports that performance analysts need contain many common features. Therefore, the frequently needed output formats should be easy to create. Record formats and data-item formats may be very complex, and are very tightly coded so as to reduce the record sizes. This is because collectors are written to minimize collection overhead. Data collectors do little reformatting of the data found in system control blocks, which means that such data items are frequently in inconvenient forms for higher-level languages.

During the fall of 1972 and the early part of 1973, the external features of the generalized reduction of information program were defined. During the summer of 1973, the specifications and the initial coding of the program were completed and then became

available at Yorktown. The first version of the data base definition for VM/Monitor data was made during the summer of 1974, and the program continued to be used substantially in that form for data reduction for VM/Monitor data for several years. In 1976 the author became more involved with VM/370 performance, and rewrote the data base definition and extended the library of data reduction requests.

The same information collected by the program can be obtained in a number of other ways. For example, the IBM Field Developed Program, VM/Statistics Generating Package[4] can be used to present all information shown in the examples in this paper. The IBM Field Developed Program, VM/370 Performance Monitor Analysis,[5] accesses the same data and presents reports that are similar to many of the examples. One can, of course, write programs to read VM/Monitor data and produce similar reports.

State samples of user

At regular intervals, the VM/Monitor records state data about each user. By examining the data, it is possible to classify each user into one of several states that are useful for analysis. The states are summarized as follows:

- IDLE— keying, thinking, or absent
- RUNNABLE— in storage and ready to use the CPU
- RUNNING— using the CPU
- I/O WAIT— waiting for user I/O to complete
- PAGEWAIT— waiting for a page-in to complete
- ELIGIBLE— waiting for storage to become free

Analysis of state data helps locate bottlenecks in the system, and allows analysts to measure the effect of bottlenecks on the users.

The IDLE state indicates that the system is waiting for a response from the user by showing that a user's virtual machine is idle. All the other states represent active states of the virtual machine. In the case of the CPU, it is possible to differentiate between the state of using the CPU and waiting for it. In the cases of paging and I/O, it is not possible from the VM/Monitor data to distinguish between the state of waiting for a physical resource and that of using it. A user in the ELIGIBLE state has been temporarily removed from the dispatchable set because there is insufficient main storage for all active users. For more information on VM/370 states and the transitions among states, the reader is referred to Reference 6. Once all of the state data snapshots are available, we can use them in a number of ways by postprocessing them, as we do at Yorktown. It is also possible to state-sample a running system and produce the same reports on the fly. One way to use the state data is to

Table 1 Summary of user states

User status name	*Status count*	*Logged-on time*	*Active time*
		(percent)	
IDLE	16216	94.9	
RUNNABLE	267	1.5	30.7
RUNNING	98	0.5	11.3
I/O WAIT	357	2.0	41.1
PAGEWAIT	145	0.8	16.7
ELIGIBLE	0	0.0	0.0

count all the states, calculate percentages of occurrences of each state, and tabulate them. By aggregating the data for all users, we observe the time spent by all users or by an average user in each state. The percentages of all states provide a picture of how the logged-on time of the users is spent. It is expected that most of the logged-on time on a time-sharing system is spent in the IDLE or user-response state. Even when a user is working intensely at the terminal, the user response time—which includes keying time and thinking time—dominates the elapsed time. The number of terminals, placement of terminals, local habits and any forced log-off procedures also influence the idle time. This is primarily because these factors influence the number and length of long periods of time when a user appears to be idle.

The difference between the user's idle time and logged-on time, or the sum of the active time, becomes an effective measure of the quality of the service provided. This time is the part of the user's logged-on time during which he is forced to wait for the system to respond and may be regarded as the percentage of time that the system intrudes upon the user. A lower than normal idle percentage indicates that poorer service is being provided.

A determination of normal idle and active percentages of logged-on time can be done only for a particular system. Comparisons from one system to another system are not usually possible because of the many factors that influence long idle periods previously mentioned. However, as shown later in the paper, it is possible to refine idle- and active-time measures in a number of ways that make it possible to compare systems.

The active-time measure is directly related to the mean response time and can be calculated by multiplying the logged-on time by the fraction of state samples that represent active states. Mean response time may be calculated by dividing the active time by the number of transactions. This makes it possible to calculate mean response through the use of state samples and a transaction count, instead of through the more expensive timing of individual transactions.

Table 2 An example I/O bound system

User status name	*Logged-on time*	*Active time*
	(percent)	
IDLE	84.4	
RUNNABLE	0.9	5.7
RUNNING	3.7	23.8
I/O WAIT	9.3	59.6
PAGEWAIT	1.1	7.3
ELIGIBLE	0.7	3.6

The system-wide summary of user states gives insight into the relative importance of I/O, CPU paging, and main storage. In the user-state summary in Table 1, there is no clearly dominant state in the system. The users have spent about forty-one percent of their time using or waiting for the CPU, and about the same time waiting for or using I/O devices. They have spent about five percent of their logged-on time waiting for the system, which in this case is quite good.

Users whose states are summarized in Table 1 have never been in the ELIGIBLE state, meaning that there has always been enough main storage to add them to the multiprogramming set when they were ready to run. This suggests that the main storage was sufficient. The low percentage of the time spent waiting for pages suggests that paging was not a problem.

The forty-two percent of the time that users spent using or waiting for the CPU would be reduced by a faster CPU. An Attached Processor (i.e., an additional CPU that uses that same main storage as the primary CPU and is controlled by the same VM/370 operating system) would not reduce the running time, but it would reduce the RUNNABLE time, which was thirty-one percent of the active time. Reorganization of the existing DASD or the addition of more I/O-related hardware would attack the forty-one percent I/O waiting time.

In the example in Table 2, I/O is clearly the dominant factor. The users have been waiting for the system 15.6 percent of their logged-on time, which may indicate poor service. These data suggest taking a further look at the I/O subsystem to find bottlenecks or to increase the I/O capacity.

One of the problems with aggregating all users is that different types of work may be inappropriately reported together. A way to deal with this situation is to group the data by type of work. At Yorktown this is done on the basis of User Identification (USERID). There is a mapping between USERID and a group name, which makes possible separate reports of state information for

Table 3 State summaries by groups of users

Group name	*Elapsed time* (h)	*Logged-on time* (h)	*ELIGIBLE*	*I/O*	*RUNNING*	*RUNNABLE*	*PAGE-WAIT*
			(percent)				
A	8.73	1321.9	0.1	42.9	11.2	31.8	14.0
B	8.67	10.2	0.0	41.2	11.8	41.2	5.9
C	7.98	8.7	0.0	0.0	0.0	0.0	0.0
D	8.53	8.6	0.0	40.0	20.0	40.0	0.0
E	8.53	8.6	0.0	0.0	0.0	0.0	0.0
F	8.58	72.2	0.0	40.7	16.8	32.7	9.7
G	8.58	65.3	0.0	43.7	11.3	18.3	26.8

interactive users, system functions, service virtual machines, batch virtual machines, etc. This may allow the identification of a service problem for a particular class of work, when there appears to be no overall service problem. Table 3 gives an example of printing out of state summaries by groups of users.

Expansion factors

Expansion factors are used to evaluate service relative to the shortest time within which work could have been completed. One way to find the run time for a unit of work in a contention-free environment is to repeat the work under such conditions. This is, of course, not possible when trying to measure users at terminals doing real work. An alternative is to measure the resources used and estimate the period of time the work would have taken without contention. We use the term *minimum time* to designate the minimum elapsed time within which a given body of work could be completed without contention. The CPU component of minimum time is the easiest part to estimate accurately, since it is simply the measured CPU time.

VM/Monitor provides a count of the number of I/O operations done by each user. At Yorktown, we know from other measurements that most of our users are CMS users who do predominantly CMS sequential I/O steps. If we assume a random rotational position of the disk, we would expect a one-half rotation time to find each item of data and a small time to read it. If the I/O were sequential and immediate, it would take one rotation to obtain the next sequential data record, plus a small data transfer time. A reasonable estimate of the elapsed time of one I/O operation is one disk rotation. Thus for the Yorktown VM/370 systems a coefficient (C) of 0.017 seconds is appropriate because IBM 3330 disk units are used.

Similar arguments lead to calculations for the elapsed time to accomplish spooling operations that involve the virtual card reader, card punch, and printer. Knowledge of spool device characteristics and blocking factors allows for the calculation of the spool coefficients C_1, C_2, and C_3. The minimum time formulas are the following:

$$\begin{aligned}
\text{Minimum CPU} &= \text{Measured CPU time} \\
\text{Minimum I/O} &= \text{I/O count} \times C \\
\text{Minimum spool} &= \text{Cards in} \times C_1 + \\
&\quad \text{cards out} \times C_2 + \\
&\quad \text{lines out} \times C_3 \\
\text{Minimum time} &= \text{Minimum CPU} + \\
&\quad \text{minimum I/O} + \\
&\quad \text{minimum SPOOL}
\end{aligned}$$

The minimum time assumes that there is no overlap of input/output operations and CPU processing. Although some programs make specific use of large virtual storage for storage management, it is assumed that all paging is done for the convenience of the system. As a result, there is no paging component to minimum time. Minimum times can also be used to measure the system load caused by an individual user. Minimum times are linear combinations of CPU, I/O, and spooling usage. This is very similar to the service-unit concept used in MVS. In VM/370 the coefficients (C, C_1, C_2, and C_3) are calculated in order to give service units the dimension of elapsed time.

The next step is to measure the actual *elapsed time* during which a user's work has completed, which can be done by using state samples. To estimate the time spent in a particular state, multiply the fraction of the samples in that state by the logged-on time. The time spent in any one of several states can be calculated either by adding the state counts together or by calculating the elapsed times and adding them together. The following formula gives elapsed-time estimates:

$$\text{Elapsed time} = \frac{\text{Samples in state}}{\text{Total samples}} \times \text{Logged-on time}$$

Given methods for estimating elapsed time and minimum time one can calculate expansion factors. An *expansion factor*, calculated by the following equation, is the ratio of the elapsed time to the minimum time. Thus, for example, an expansion factor of 3 indicates that the work required three times the elapsed time that would have been required without contention.

$$\text{Expansion factor} = \frac{\text{Elapsed time}}{\text{Minimum time}}$$

The most important expansion factor is calculated from active time and minimum time. Other expansion factors can also be calculated by considering only the I/O or CPU component of both the active time and the minimum time.

There are a number of ways that expansion factors can be presented. If all the data that apply to one user are used, an expansion factor for that user can be calculated. If all the data during successive time periods are used, changes in expansion factors through time can be shown. Table 4 shows expansion factors calculated for all interactive users at one-hour intervals for one day. This becomes a time-of-day service measure. Other expansion-factor reports could be created for classes of users or particular users for hours, days, or longer.

Further refinement of the expansion factor is necessary to permit comparison of one system with another. A faster processor returns work to a user faster than a slower one, when the expansion factors are the same. A way to correct this is to normalize the minimum time to a particular processor and type of I/O device. Then the normalized expansion factor becomes relative to the selected "normal system."

User productivity

In order to talk about a user's productivity it is necessary to estimate the length of time the user is actually at the terminal doing work. Historically, the first estimate used for this was logged-on time, a measure with many inaccuracies. For example, the availability of terminals in private offices allows persons to remain logged on while not working, or to initiate long-running tasks and then leave the terminal. On the other hand, terminal rooms and shared terminals force LOGOFFs after inactivity. Restricted access causes continuous activity during logged-on time.

user-active time

As a better approximation to the true user-present time we calculate a *user-active time*, which is the sum of short elapsed-time intervals during which there was some system activity by a user. We compute user-active time by summing the short time periods (with a one-minute default) during which there is activity. During the remainder of the user's connect time it is almost certain that he was not present.

It is also possible to estimate the sum of the system response times from the state samples. System times are shown in Table 5 under the column headed "Active system."

user-present time

The user is probably present for only a part of the system response time, but for nearly all the transactions, whereas he prob-

Table 4 Expansion summary by time of day

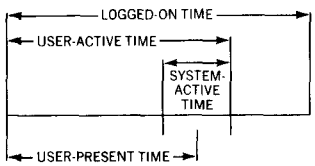
Start time	*Expansion factors*			
	Overall	*I/O*	*CPU*	*Paging*
11:00	4.83	2.59	7.29	2.13
12:00	3.83	2.36	4.97	2.40
13:00	2.04	1.55	1.67	4.23
14:00	3.44	2.48	4.16	2.09
15:00	3.22	2.46	2.64	3.01
16:00	2.85	2.27	2.78	1.77
17:00	4.11	2.40	5.29	2.71

Table 5 Activity summary by time of day

Start time	*Average users logged on*	*User active* (h)	*Active system* (h)
11:00	152.5	81.7	10.0
12:00	153.6	64.7	6.8
13:00	151.5	51.4	3.4
14:00	158.0	72.8	6.7
15:00	183.9	81.9	6.8
16:00	195.0	95.2	5.6
17:00	153.7	67.1	8.4

ably lets long-running transactions remain active at the terminal while he does other work. Thus user-active time is the upper limit of true *user-present time*, and user-active time minus system-active time is a lower estimate of user-present time. This is shown in Figure 1. In most systems, the system-active time is five to fifteen percent of the user-active time. Fairly accurate estimates of user-present time can be made on the basis of these considerations.

Figure 1 Estimating user-present time

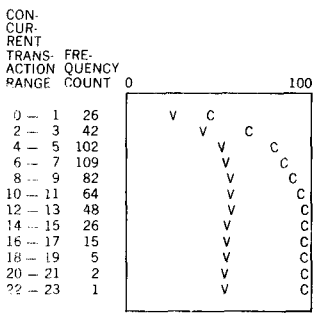


A way to further improve the estimate of user-present time is to determine whether the system has received terminal input from the user during the previous interval of time. One of the VM/Monitor options causes a time-stamped recording of all terminal inputs to the system. If the user enters an input line to the system during the current interval, he is assumed to be in a present state. If he has not entered an input, he is considered not present. Although this method refines the estimate of user-present time considerably, it tends to underestimate it slightly because during extended periods of time when a user is observing the output of a program he does not interact with the system and is considered not present.

Provided up to this point is sufficient information to calculate a system-independent measure of service based on state samples.

The preferable measure is the fraction of user-present time that the user waits for the system. We approximate that measure by using the fraction of user-active time that the user waits for the system. The measure should be used to compare similar work only. Thus it is necessary to group USERIDs as shown earlier in this paper.

Figure 2 Productive CPU use versus contention

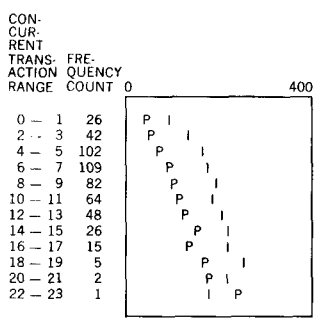


At Yorktown we produce a number of reports that show the many resources consumed by each system user. Table 6 shows a sample report that has been valuable in tracking large users and in understanding problems of particular users.

CPU monitoring

In the graphs in Figures 2 and 3, several resource utilization indicators are plotted relative to the number of concurrent transactions. The number of *concurrent transactions* is the number of commands that have been entered into the system but have not yet been completed. Each line on the graph summarizes all one-minute intervals that have ended with a particular number of concurrent transactions. As the number of concurrent transactions increases, contention for page frames increases, with the result that the paging rate increases. CPU utilization also goes up until it reaches one hundred percent. The percentage of time in problem state declines at higher levels of contention because more supervisor time is consumed in managing the resources.

Figure 3 I/O and paging versus contention



The frequency counts are crucial in interpreting these figures. When the VM/Monitor default monitoring interval is used, state samples are produced every minute. Thus each data point represents one minute of elapsed time. If most of the data points are in the range of effective CPU utilization (i.e., below the point of declining problem state CPU time), the system is not overloaded. The higher contention part of the curve indicates occasional occurrences of overload that can be tolerated. The higher part of the curve also suggests what is expected to happen as the load grows. If the bulk of the data points are in the area where problem state CPU is declining and paging is growing too high, the system is already overloaded. For further discussions of CPU overload see References 3 and 6.

If the CPU is overloaded, two courses of action are possible. One is to increase the capacity of the CPU, which usually involves adding hardware, though it may involve tuning in order to make more of the present CPU available. The other possible action is to reduce the load. Table 7 shows resource consumption by USERID. This report is sorted in decreasing order by minimum time; thus the larger resource consumers are at the top of the table. Two columns show the percentage of all CPU time and I/O operations

Table 6 Individual user status

USERID	*Logged-on* (min)	*Active time* (min)	*Active ratio*	*User status RUNNABLE*	*User status PAGEWAIT*	*User status I/O WAIT*	*User status RUNNING*
				(percent)			
USER 1	97	96	0.99	15.0	5.0	80.0	0.0
USER 2	280	181	0.65	4.0	0.0	90.0	4.0
USER 3	250	152	0.61	68.0	2.0	2.0	26.0
USER 4	263	166	0.63	31.0	3.0	56.0	9.0
USER 5	165	158	0.96	33.0	11.0	44.0	11.0
USER 6	280	198	0.71	68.0	6.0	21.0	3.0
USER 7	293	252	0.86	17.0	17.0	57.0	8.0
USER 8	148	148	1.00	81.0	4.0	4.0	9.0
USER 9	214	130	0.61	25.0	0.0	66.0	8.0
USER 10	287	124	0.43	25.0	0.0	75.0	0.0
USER 11	280	280	1.00	20.0	40.0	0.0	40.0
USER 12	293	115	0.39	50.0	2.0	20.0	26.0
USER 14	286	117	0.41	46.0	6.0	33.0	13.0

Table 7 Resource consumption by user

USERID	*Logged-on time* (min)	*Active time* (min)	*Minimum time* (min)	*Total CPU* (percent)	*Virtual CPU per I/O*	*Virtual CPU per page in*	*Total I/O* (percent)
USER 1	172	171	14	3.32	0.003	0.033	5.58
USER 2	170	168	10	2.38	0.001	0.011	3.95
USER 3	122	102	10	1.21	0.001	0.022	5.11
USER 4	173	170	9	1.79	0.002	0.005	3.87
USER 5	172	116	8	3.12	0.002	0.002	2.06
USER 6	95	95	8	1.33	0.002	0.014	3.70
USER 7	173	144	8	1.37	0.001	0.004	3.63
USER 8	172	143	7	1.95	0.006	0.019	2.43
USER 9	173	172	6	2.65	0.003	0.004	1.16
USER 10	160	146	5	2.42	0.010	0.037	1.16
USER 11	173	149	5	1.14	0.003	0.016	2.13
USER 12	105	105	5	1.58	0.005	0.012	1.51

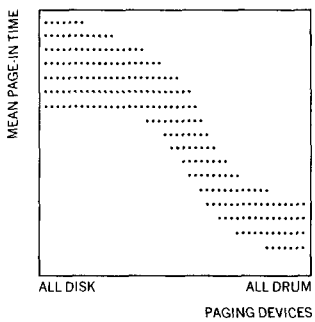
for each user. Two other columns show the average virtual CPU time between page-in operations and I/O operations. These columns help identify the degree to which particular users are I/O bound or paging bound. This type of report may be used to locate the large resource users on the system who might be candidates to be moved to another CPU. Another alternative would be to tune particular programs of high CPU users.

Paging subsystem monitoring

On VM/370, the ability to do demand paging very quickly is key to interactive response time because each new terminal transaction

initiates the processing of a command with a different working set of pages. This makes the effectiveness of the users highly dependent on the ability to fetch new pages from external storage. Thus the paging subsystem should be monitored on a regular basis, which can be done very simply. VM/Monitor writes records that summarize the current paging activity, with a default time interval of one minute. The average time delay for each page read into main storage (page in) can be calculated in almost the same way that the paging expansion factor is calculated.

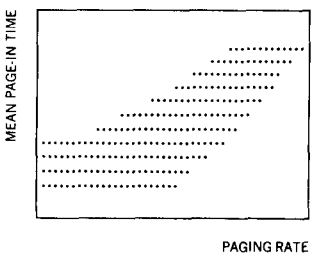
Figure 4 Page-in time as related to drum fraction



Problems that seem to be in the paging subsystem may have their origins in other areas. If there is insufficient main storage available for paging, one might first see paging as a problem. In the case of insufficient storage for paging, the users are in page-wait state longer than they should be. The system also reacts to the shortage by placing users in the eligible state. The need to do more paging might also cause a high paging rate, and that would overload the paging subsystem, thereby causing further delays in paging. Insufficient high-speed paging space could result in long delays to the users. Table 8 shows the mean page-in time (in milliseconds) as a function of time of day. This table also shows the fraction of pages that the *active users* (i.e., some resource consumption in the last minute) have on drum.

Ideally nearly all the pages should be on drum if both disk and drum paging are used. A program known as the *System Extension Program Product—Resource Manager* attempts to keep the most active pages on drum by copying inactive pages onto disk periodically. In the scatter plot in Figure 4, each point, which represents one hour of system activity, can be of great help in understanding the relationship of drum use and page-in time. The area in which the data points fall depends upon the load and the hardware configuration. This method of presenting the data shows the system reaction to different levels of overcommitment of the drum storage. The placement of the points on the vertical scale indicates the degree of degradation, while point placement on the horizontal scale indicates the elapsed time during which the system is degraded. Trend lines can also be created by averaging the points that fall within ranges of the drum-fraction variable and plotting the averages. This plotting technique is similar to that used for the CPU graph in Figure 2.

Figure 5 Page-in time as related to paging rate



The scatter plot in Figure 5 has a different characteristic curve and shows the relationship between paging rate and page-in time. At lower paging rates, the page-in time is relatively insensitive to the paging rate. At higher paging rates, the page-in time rises rapidly as the paging rate goes up. The general shape of the region depends on the particular system configuration. Channel contention, control unit contention, disk arm contention, speed of paging devices, etc. influence the shape of the point distribution.

Table 8 Page-in time by time of day

Start time	*Average users logged on*	*Mean page-in time* (ms)	*Drum fraction*
11:00	152.5	17	0.68
12:00	153.6	19	0.66
13:00	151.5	34	0.68
14:00	158.0	17	0.70
15:00	183.9	26	0.66
16:00	195.0	34	0.64
17:00	153.7	22	0.73

Table 9 Storage-constrained (2-megabyte) system

User status name	*Status count*	*Logged-on time*	*Active time*
		(percent)	
IDLE	3652	84.9	
RUNNABLE	12	0.2	1.8
RUNNING	250	5.8	38.6
I/O WAIT	200	4.6	30.9
PAGEWAIT	101	2.3	15.6
ELIGIBLE	84	1.9	12.9

The important thing to observe is whether most of the data points lie within the region of lower and consistent page-in times. If they do not, the cause should be investigated.

Main-storage monitoring

Adequate main storage is very important in a paging system so that there is a high enough multiprogramming level to make full use of the CPU and I/O devices. Insufficient storage may be recognized by some combination of I/O and page-wait delay on the part of the CPU, i.e., users spending excessive time in the page-wait and eligible states, and a high paging rate.

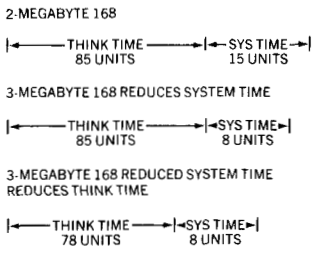
Table 9 shows user-state data on a storage-constrained system. The large amount of active time suggests that users are being significantly intruded upon in their work by long responses. The 15.6 percent of the time that users are in a page-wait state is a bit high, and the 13 percent of the time that users are in the eligible state is much too high.

Table 10 Non-storage-constrained (3-megabyte) system

User status name	*Status count*	*Logged-on time*	*Active time*
		(percent)	
IDLE	1315	91.9	
RUNNABLE	13	0.9	11.3
RUNNING	56	3.9	48.6
I/O WAIT	37	2.5	32.1
PAGEWAIT	9	0.6	7.8
ELIGIBLE	0	0.0	0.0

When more storage is added to the system, as shown in Table 10, the time that users wait for the system goes from 15.1 to 8.1 percent of their terminal session. The improvement results from a lowering of the time that users wait on the eligible list for storage and the time they wait for pages while in storage. Reducing storage contention indirectly increases the CPU time available for users by lowering the CPU time used for paging. Having more dispatchable users in storage increased the CPU utilization. These actions have resulted in a dramatic improvement from the users' standpoint.

Figure 6 Estimating user time savings



Under the assumption that the users continue to use eighty-five units of time to prepare the work that the system completes in eight units of time, the potential savings in user time is seven percent, as shown in Figure 6. Reference 7 suggests that users may actually save an additional seven units of time through their own performance, as indicated by the lower bar in Figure 6.

The cost of the additional megabyte to this System/370 Model 168 is a very small incremental cost compared to the user productivity benefit. The overall throughput has increased as a result of reducing CPU paging overhead and using CPU time that had previously been wait time. User productivity could have increased as much as fourteen percent. Most impressive is the improvement as the user sees the system. Work that had previously taken fifteen units of time now takes eight units of time, for a reduction in system response time of forty-seven percent.

Concluding remarks

Measurement of interactive systems should be a continuous process. It need not be done on a twenty-four-hour basis, but it should be done regularly, such as by partial measurements every day or complete measurements every few days. Collection during all hours when large numbers of users are present is most desirable.

Under Release 5 of VM/370, it is possible to have VM/Monitor data written into a spool file that is placed in the virtual reader of a particular machine when collection is completed. VM/Monitor can also be turned on and off at specific times of day without the intervention of the operator. These two features make continuous use of VM/Monitor very easy. One of the primary reasons for regular monitoring is to build up a body of information on system performance with its regular workload under normal conditions. Without such a history, it is very difficult for computing center management to understand the changes that have taken place when a problem arises. A lack of information about normal system running reduces the usefulness of a measurement tool to that of an uncalibrated thermometer.

Another important reason for regular monitoring is to allow the collection of historical load and service data. This then becomes part of the input to be used for load projection and capacity planning.

During the beginning states of monitoring, the system programmer who checks the daily reports gains in knowledge of the system. Gradually the reports may be restructured so that they report exceptional conditions only. Even if exception reporting is not done formally, as a part of the programs that produce the reports, it is done informally by the system programmer. The programmer learns which items require daily inspection and the normal ranges of values for those items.

This methodology can be applied to both operating systems and special-purpose application systems. When starting to measure a system, one should determine the significant states for users of that system and how to test for them. The methodology presented in this paper could then be applied. Potential states in other systems might include waiting for spooling, waiting for a particular lock, waiting for a data set, waiting for a shared DASD volume, waiting for program loading, waiting for the CPU, and using the CPU.

Sampling the status of users on an interactive system has proved to be a valuable tool for understanding the factors that delay response. The state information provides insight into the relative importance of the bottlenecks in a system. Elapsed times may be calculated from state data without resorting to event timing. Because the technique is inexpensive—sampling with VM/Monitor takes a fraction of one percent of the CPU time—sampling can be done on a continuous basis.

ACKNOWLEDGMENTS

Many people have played a part in making the generalized reduction of information program available initially and then applying it

to VM/Monitor data. The author especially acknowledges the following persons. H. W. Lynch, J. A. Cooperman, and P. H. Callaway influenced the features in the data reduction program. L. Flon was responsible for much of the design and programming of the program generator. D. H. Potter, an early user of the program generator, provided valuable feedback. L. Flon and P. H. Callaway created the initial VM/Monitor data base definition. P. H. Callaway and W. M. Buco used the VM/Monitor data for several years and developed a library of data reduction programs. W. M. Buco's knowledge of the VM/370 was vital in interpreting the VM/Monitor data. W. J. Doherty made many helpful suggestions and offered useful observations about the significance of the VM/Monitor output. W. J. Doherty, H. Serenson, and R. P. Kelisky provided management support and a fertile environment in which the work could be done.

CITED REFERENCES

1. W. J. Doherty and R. P. Kelisky, "Managing VM/CMS systems for user effectiveness," *IBM Systems Journal* **18,** No. 1, 143-163 (1979, this issue).
2. *VM/370 System Programmer's Guide*, Document GC20-1807, IBM Corporation, Data Processing Division, White Plains, New York 10604.
3. P. Callaway, "VM/370 performance tools," *IBM Systems Journal* **14,** No. 2, 134-160 (1975).
4. *VM/SGP Program Description and Operations Manual*, Document SH20-1550, IBM Corporation, Data Processing Division, White Plains, New York 10604.
5. *VM/370 Performance Monitor Analysis Program Description and Operations Manual*, Document SB21-2101, IBM Corporation, Data Processing Division, White Plains, New York 10604.
6. Y. Bard, "Performance analysis of virtual memory time-sharing systems," *IBM Systems Journal* **14,** No. 4, 366-384 (1975).
7. S. J. Boies, "User behavior on an interactive computer system," *IBM Systems Journal* **13,** No. 1, 2-18 (1974).

Reprint Order No. G321-5091.