Discussed in this paper is a computing center management methodology based on the premise that the computer user's time and work product are valuable. Experience in the use of interactive systems in a research environment from 1965 to the present time is presented. Current user experience and management of VM/CMS are emphasized. The use of computers as tools for extending users' powers of memory and logic and the development of new methods of managing VM/CMS are discussed in detail.

Managing VM/CMS systems for user effectiveness

by W. J. Doherty and R. P. Kelisky

This paper discusses the evolution of the Virtual Machine/Conversational Monitor System (VM/CMS) interactive computing services at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, and the necessary changes in viewpoint needed to manage interactive computing services effectively for the user community. (We use the familiar name "Yorktown" throughout this paper.) Computer users at Yorktown have available a variety of computing services. VM/370 is provided on two seven-megabyte System/370 Model 168's, one of which has an Attached Processor. MVS, i.e., OS/VS2, together with the Time Sharing Option (TSO) is provided on an eight-megabyte System/370 Model 168-3. Employees at Yorktown are urged to use these computing facilities to do creative and innovative work. Administrative personnel, as well as scientists, use computers where it is cost-effective for them to do so. Since nearly all computing services are accessed interactively from terminals in users' offices, terminal rooms, or, in some cases, home terminals, we have attempted to develop management policies and practices that make interactive computing services more effective for the user.

Interactive computing began as a service at Yorktown with the introduction of APL in 1965, and presented users with an interface fundamentally new and different from that of batch computing. Clearly, most of the strong acceptance of APL resulted from the APL language and the APL system, but these positive characteristics might not have sufficed if at the same time the APL group had not recognized that this new way of computing required a new way of managing computing services. That is, management recognized that interactive computing must be *available* in order to be useful. The user must be able to turn on the terminal at any time and, with a high degree of confidence, find the interactive system up and running. APL service was provided on a System/360 Model 50 essentially twenty-four hours a day, seven days a week. Preventive maintenance was limited to a few hours every three weeks, and because of the intrinsic reliability of the APL system and the machine, the user was able to assume that APL would be available whenever needed: day and night, weekends, and holidays. Although computing center management may not have realized it at the time, criteria for user effectiveness, system availability, and system reliability were being established for interactive computing services. These criteria have strongly influenced subsequent management practices of the Yorktown computing center.

In 1967, the Time Sharing System, TSS/360, was introduced at Yorktown on a System/360 Model 67. On TSS/360, we began to experiment with and understand the effect of system response time on user productivity, as well as the value of indicating to the user a measure of system load prior to his logging on, the importance of scheduling controls in order to distribute system resources equitably, and the need for "transparent" management of a growing on-line user data base. Although TSS/360 no longer runs at Yorktown, the experience gained has suggested new extensions to VM/370 and TSO, so that these systems might become more effective for the user.

In 1969, the Control Program, CP/67, the predecessor of VM/370, was introduced on a second System/360 Model 67 to serve as a software development facility for operating systems. An irresistible user demand for CMS, which was not offered at first to our computer users, showed the importance of providing an easy-to-use system for people without special knowledge of computers. An increased load on the system, resulting from demand for CMS, showed that it would be necessary to give the programmers who had responsibility for maintaining CP/67, and later VM/370, which replaced CP/67 in 1972, tools to enable them to understand what was taking place inside the system so that a few users could not usurp system resources to the detriment of system performance for the entire user community. We also learned that interactive system users are reluctant to give up a function or facility that had

proved useful on an earlier interactive system. For example, a facility equivalent to the ease of data management, as provided on TSS, had to be made available on VM/370.

Lessons learned

In later sections, we examine in more detail results of our experience with managing interactive systems, but first we present the most important lessons learned from experience.

- The computer user's time and work product are valuable. Therefore, interactive systems should be managed so that ability to work is enhanced with the least inconvenience to the user.

For most interactive computing, the aggregate user time is more costly than the computer time. Figure 1 shows the cumulative percentage of VM/CMS CPU cycles consumed at Yorktown by computer users *versus* the cumulative percentage of people using VM/CMS over a month. Similar curves characterize the distribution of computing usage in many other installations.[1] In general, less than ten percent of the people consume about seventy-five percent of the computing resource. Focusing on that top ten percent of the computer users, we find that a small fraction of their interactions dominates their demand for computing. Each user's own management, not the computing center, must determine whether the machine-intensive computing is technically justified. Such computing occurs for about five percent or less of the interactions. The remainder of the interactions involve immediate, ongoing communications between people and the system. To illustrate the economic implication, assuming a cost of $800 per hour for a large computer, the twenty-five percent of the system that accounts for ninety-five percent of the interactions costs $200 per hour. And assuming one hundred simultaneous users at $20 per hour, the user cost is $2000 per hour. These figures are only for illustration. At Yorktown, computer costs are less, users' time costs more, and there are normally many more than one hundred simultaneous users on each of two VM/370 systems. Therefore, for more than ninety-five percent of the computer interactions, the users' time is much more costly than that of the computer, and these costs are diverging. Thus, our aim is to provide an environment in which the computer user can make effective use of time and efficient use of the computing resources provided.

Figure 1 Yorktown VM/CMS

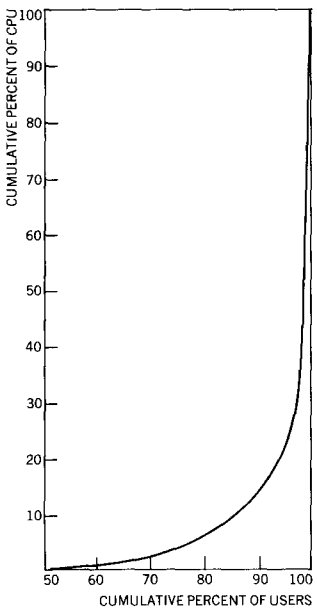
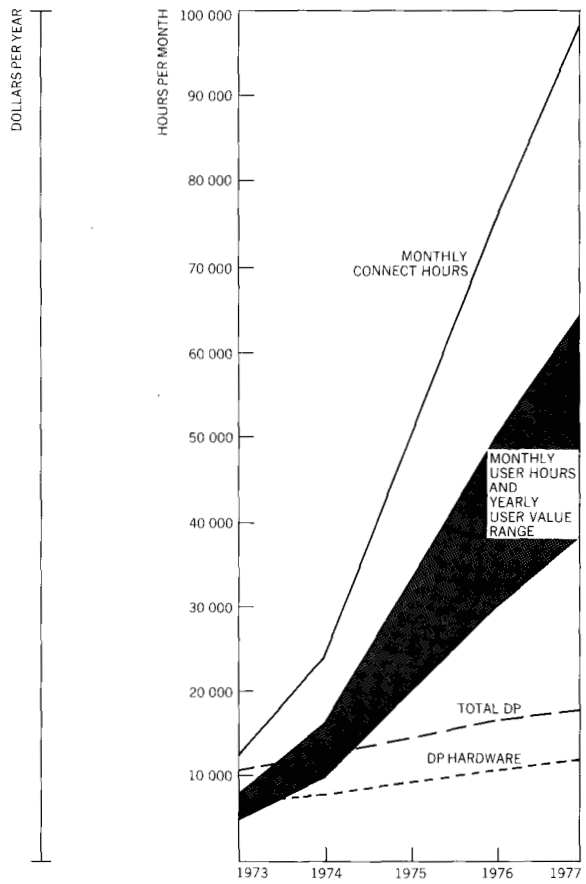


Figure 2 shows the growth in interactive computing at Yorktown from 1973 through 1977. The aggregate cost of the users' time is greater than computer costs, and this difference continues to grow. Therefore, we seek maintenance and enhancement strategies, within budget limits, that help users to do their work and

Figure 2 Comparative ranges of user hours, cost of user time, and computing costs at Yorktown



that tend to give highest priority to users' needs for service. This means that we isolate changes from one another and allow both old and new versions of functions to coexist, so that system improvements do not force the user to change his ways of working.

An essential tool for our maintenance and enhancement strategy is the virtual machine in VM/370. In virtual machines, we can test new versions of most subsystems or applications without disturbing existing versions. The fact that we can isolate many special functions within separate virtual machines provides the ability to tailor the appearance of a given function, and thereby simplify the user interface without fear of "contamination" from other irrelevant function. When CMS is the operating system in the virtual machine, we find that the system overhead for performing that function is small.

- Complexity in the man-machine interface is wasteful of users' time, and can be reduced by the judicious use of procedures.[2]

In April, 1978, the Yorktown computing center processed more than ten million *interactions*. An interaction is any user input and its accompanying system response. If the information content per interaction is too small, people may be unknowingly slowed down by excessive interaction. Viewed another way, a decrease in the amount of work the user has to do to derive a useful result from an interaction makes the system more effective for that user.

Often, computer users take great care in determining an effective combination of computing functions for a frequently performed task. Once they have done this, the combination of functions is given a name and stored on secondary storage, so that the person need not repeat those thought processes again. These procedures (called EXEC's) may include useful default values for parameters as well as logic to make the system adapt to the user's environment, rather than requiring the user to adapt to the machine. Once someone has constructed an EXEC, others can and do take advantage of it. For example, there are EXEC's to simplify the use of the IBM internal computer network, so that the user normally does not have to be concerned with the kind of data involved, the protocols, or the ways of addressing the person or group to whom data are sent or from whom they are received. The naive user need not even be aware of the existence of the commands or parameters inside EXEC's. By this process, complexity is reduced, typing errors are avoided, and the user-effectiveness of interaction increases.

From examining twenty percent of the users on one of the VM/CMS systems at Yorktown, we find that there are twice as many EXEC's as all other source programs in conventional programming languages combined.[3] Elsewhere, in 1977, the Stanford Linear Accelerator Center (SLAC) installation observed an average of twenty-five commands executed from EXEC's for each command typed at a terminal[4] when using WYLBUR, a highly sophisticated editing and batch submission subsystem developed by Stanford University about ten years ago. In 1971 at Yorktown, the number of commands executed per person, counted at execution time after procedure expansion, doubled in just five months. Thus, experience indicates that there is continued growth in function as users make the system adapt to their requirements. We can think of this growth as arising from "captured intelligence." In the early sixties the computing industry speculated on the notion of "artificial intelligence," whereby the computer would heuristically determine the solution to some problem. In practice, we find the concept of captured intelligence extremely useful in raising the man-machine interface to a level meaningful for many people.

- Display terminals are able to provide a user more information in a given time than typewriters. They can be especially valuable in helping one understand how to use the system.

Where justified by the user's work, in the view of the user's management, it is our objective to place an IBM 3277 display terminal on each user's desk. In general, we find terminal usage of an hour or more per day to be a prerequisite for such an installation. An example of the value of the display terminal in accelerating the user's work is provided by our documentation practices on EXEC's. It has become customary to include comments on the use of each EXEC directly in line with the EXEC code. Thus, each user can determine the functions each EXEC performs, its syntax, and the default values of parameters. These are obtained by entering the name of the EXEC followed by a "?". By having the comments included in line with the EXEC, these comments are then more likely to be changed whenever the EXEC is changed, which maintains currency between the documentation and the code. The Yorktown user community finds that these "self-defining" commands are much more useful than any other form of documentation or education. Such a facility is not feasible with the slower data rates of typewriter terminals.

- Since systems programming is costly, management seeks solutions to user problems that do not lead to growing commitments in systems programming time.

Systems programmers are a scarce and costly resource, and strategies that increase systems programming requirements are to be avoided. In general, a local change to an operating system carries with it an obligation to reevaluate that change every time there is another change to that part of the operating system. Such local changes represent a kind of promissory note on which one pays interest (system programming time) indefinitely. Yorktown computing center management seeks ways to reduce this cost. For example, we have selected a way of attaching new display terminals that avoids repeated reprogramming as the underlying operating system changes. This subsystem, called the Advanced Terminal Subsystem, attaches non-IBM terminals to an IBM System/7 that has been programmed so that the underlying operating systems (VM/CMS, MVS/TSO, and formerly TSS) on the System/370 Model 168's need not be changed. Change is localized to the System/7, and new terminal types are introduced by means of table entry rather than by modifications to MVS and VM/370.

By giving people special virtual machines and access to whatever source code they need, together with good tools for debugging changes and for finding out who else might be interested in any given function, we provide an environment in which changes take place but reliability remains high. Special libraries exist to allow users to make their changes available to others. By simply typing the command OWNER followed by a command name, any person can determine the current owner of a given version of a function, and communicate problems or new ideas for enhancement di-

rectly to that owner through the system itself. Once such functions have proved to have value to many people, they can readily be incorporated into the central library, with little effort by the systems programmers.

- Because the development of new computing services is costly, installation management develops new services incrementally in order to evaluate these services before the next step is taken.

We avoid development projects that must span several years before their benefits are available to the users. Not only are such projects costly, but also there is the great risk that the problem to be solved has changed or has disappeared during the development process. Because program development is a very complex process, we find that feedback is required continuously throughout the development cycle. By bringing the computer to the person who has the greatest knowledge of the problem to be solved, and providing adequate computing resources and an appropriate set of tools, we find that that person can build a prototype solution, try it personally, and ask others to try it. Many such iterations refine and perfect the prototype so as to yield the required function. We find that this is a fast way to produce high quality function for low cost, or perhaps for the lowest cost. We also find that even very skillful programmers absolutely require such feedback throughout the development cycle. There is accumulated evidence to show that most errors have occurred in the design stage,[5] and that individual skills are the major factor in determining the quality of programming projects of fewer than fifty thousand lines of executable source code.

By developing new services in an iterative fashion, we are more likely to be sure that we are solving the right problems, that the proposed solutions are indeed perceived by the users to be useful solutions, and that the function can be readily changed as future needs dictate. VM/CMS and the virtual machine provide the basic environment we need to work in this fashion. Our experience suggests that the iterative method of program development is a major improvement over the traditional programming development process.

- User need for batch processing grows proportionally to interactive computing, a fact that increases the need for the availability of well-planned specialized function.

The load distribution curve of Figure 1 is the same for interactive computing as it is for batch processing. Because of the growth in interactive computing that has continued for the past ten years, there has been pressure to detach work that is large in computing resource demand, or that has reached some logical stage of com-

pletion. If this were not done, a long-running process would act as a block at the terminal, and prevent a person from accessing other data while that long-running process is completing. By packaging special functions in special-purpose virtual server machines, it becomes easier to off-load work whenever appropriate. These machines need not run on the same real machine, and, in many cases, they do not. To facilitate communication among such special-purpose virtual machines, an experimental mechanism has been developed for communication among different virtual machines that run on the same real machine. We have used this mechanism in text processing, laboratory automation,[6] and in a VM/370-based Mass Storage System (MSS) data migration system. The Virtual Machine Communication Facility (VMCF) of VM/CMS is an outgrowth of this work. Network Job Entry and Network Job Interface software products have given us the necessary facilities for communication among different real machines. By being able to communicate with other machines, whether virtual or real, we can permit new and old functions to coexist. Our users communicate with old functions as needed rather than convert old functions to new environments. By insisting on having most of our terminals locally attached, we minimize response time delays, reduce security problems associated with remote access, and reduce the number of hardware subsystems required to be available simultaneously. Thus, "networking" data rather than people is a more effective way to promote collaboration and to avoid replication of work.

The following sections discuss management actions as they have affected availability and reliability, response time, expansion factors and visibility of service, data management, and specialized function.

Availability and reliability

Since computing services at Yorktown are available, insofar as possible, twenty-four hours a day, seven days a week, one may ask whether such a schedule is justified. Technical management has authorized more computing than can be accommodated during prime shift, but there are other reasons why such a schedule was established: the requirement for computer availability by experimentalists, the necessity that system maintenance not be done during prime shift, and the relatively low incremental cost to provide these added services. There are, however, impediments to such a schedule: software or hardware failures, scheduled software changes or tests, scheduled hardware shutdowns for engineering changes, preventive hardware maintenance, hardware relocation, and the unavailability of personnel to operate the systems on weekends and holidays.

software maintenance

As to whether such extreme availability is necessary, we note that researchers (computer scientists, chemists, physicists, engineers, etc.) expect and require a degree of flexibility in working conditions. Experiments cannot be confined to the normal eight-hour working day, and there are experimentalists who may extend or shift their working hours to coincide with the requirements of their experiments. There are also scientists who work more than eight hours per day, or who are simply more productive if given access to the tools they need in the middle of the night, on weekends, and on holidays. Finally, we realized that if users are to entrust all their programs, procedures, documents, and data to the interactive systems, that information must be available whenever they are working, in the same sense that the contents of a locked file in the user's office are available.

This policy is at times incompatible with the need to test new software at night, modify or repair hardware, or carry out necessary housekeeping tasks (such as making copies of the on-line data base as a safeguard against destruction). But, given a commitment to provide around-the-clock service, compromises can be worked out so that many hardware changes are postponed to third shift or to weekends. Together with our users, we have evolved a strategy for software testing so that changes to the software system are introduced on prescheduled days, usually during a shutdown at 18:00 lasting fifteen to twenty minutes. Then the period from 18:00 to 8:00 the next day is designated as a "user-risk session," so that users know that new (and possibly unstable) software is being tested. The computing center is allowed two system crashes, after which the standard prime shift system is restarted, and users know that no further software testing will be done that night. Housekeeping tasks can usually be designed to run in a time-shared mode, so that back-up of the data base can be done after 18:00, even though users are on the systems. Such tasks take longer to complete when executed in the time-shared mode rather than in the stand-alone mode. However, the stand-alone mode of the computer is employed very rarely because it limits user access to the systems.

When new software changes to the operating systems have been tested for about a month on second and third shifts, they are then introduced on prime shift, usually on a Friday, so that we have the weekend to solve new problems that may appear. In general, after one prime shift failure of the new software we return to the older standard system. If the new software runs for a week without prime shift failures it becomes the new standard system.

hardware maintenance

Hardware failures usually call for immediate repair in many computing centers. In general, our policy is to restart the system and maintain service, even when it is degraded. Only if failures become so frequent that users cannot proceed with useful work, or

if we cannot restart the system, do we give up the system for service on prime shift. Relocation of hardware and the addition of engineering changes are limited to second and third shift periods or weekends, excluding 8:00 to 18:00 on Saturday if possible.

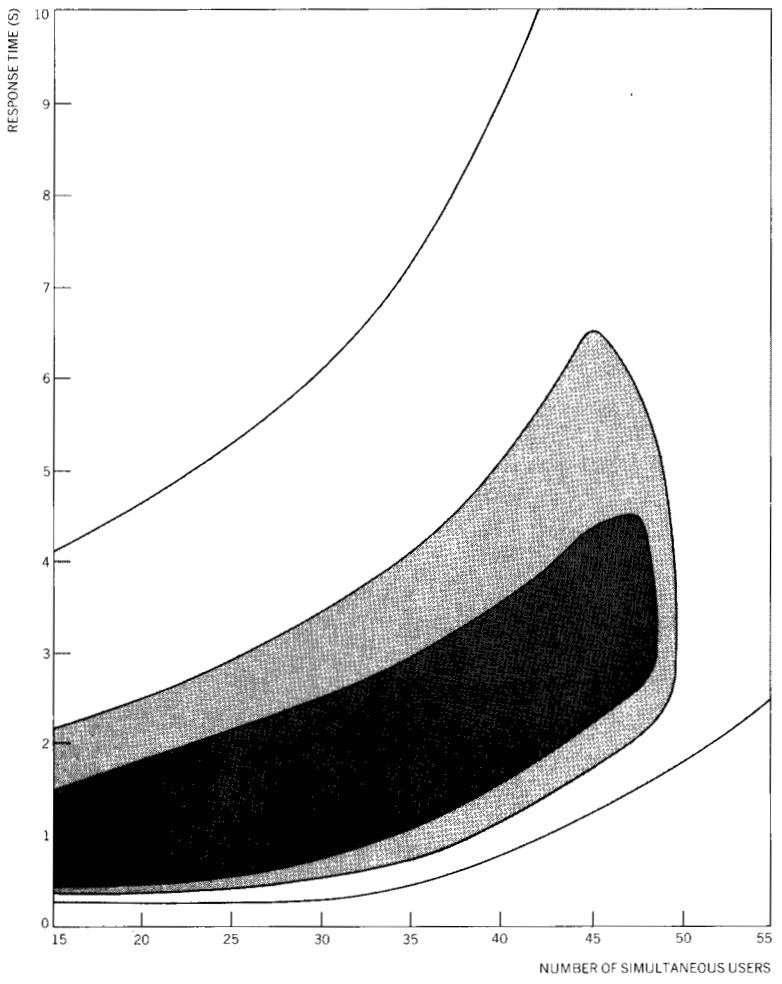
system operations

Despite our intentions to provide around-the-clock service, a major difficulty is that of operator availability. Because we operate under manpower limitations, we were led to seek new ways to increase operator availability with a fixed number of operators. We recognized that we could not operate each machine from its own console and continue to mount tapes and disks on demand with the available operations staff. Therefore, we designed a central "operations bridge" from which the three System/370 Model 168's can be operated by means of IBM 7412 terminals and 3277 display terminals, and, where necessary, we have also made changes to the operating systems so that they may be started and controlled remotely. Since all the operators must operate all systems, it is more difficult to develop operators who are highly skilled on those systems. Consequently, MVS/TSO/JES3 and VM/370 each have an operator team with a lead operator. Members of one team can operate the other system, but do not necessarily have specialized knowledge of it. This strategy allows the computing center to operate on weekdays with six operators on prime shift, five operators on second shift and three operators on third shift. On weekends and holidays there is one operator for each of two twelve-hour periods per day. On certain holidays, the interactive systems run unattended, with the understanding that private tapes and disks cannot be mounted. Nevertheless, there are persons who come into the laboratory on holidays to use the computers, or who, in a limited number of cases, dial in from home terminals.

management commitment

Another important aspect of system availability is that of management flexibility. Computing center management must be willing to make compromises in schedules in order to serve special needs of users. When computing center schedules are published each week, or when sudden changes to those schedules must be published by means of log-on messages to users, we invite users with special needs to contact us. If a user is attempting to finish a paper to meet a deadline, or has urgent work that is due at a certain time, the computing center tries to adapt to the user rather than insisting that users always adapt to the computing center. On the other hand, it is important to adhere to schedules that promise service at specified hours. It is extremely frustrating to come into the laboratory on a weekend or holiday, expecting to use the computer, and find that the computing center has changed the schedule, even though the reason may be a good one. There is also a special telephone number from which one can obtain a recorded message on the status of the systems. This message is changed and time-stamped whenever a system failure or other

Figure 3 Occurrences of particular combinations of response time and numbers of simultaneous users for TSS/360 at Yorktown in 1971



emergency changes the service schedule. The main disadvantage of the recorded message is that it is just not accurate enough for brief system failures, although it is useful for failures that last more than five minutes. Users report that recorded messages are accurate no more than seventy percent of the time, and we are seeking a better way of informing them of the status of the systems.

Performance management and visibility of service

System response time is the time the user waits, measured from his signal to the system that there is work to be done to the point at which the system begins to present those results to the user. Figure 3 is a representation of response time to a single small

program that ran every twenty seconds in the spring of 1971. Each twenty-second period, the program would awaken and record the number of logged-on users and the response time that it received from the system. Figure 3 might be considered a three-dimensional graph in which the *z*-coordinate is the number of times a specific combination of response time with a given number of simultaneous users occurred. Darker shading denotes more frequently observed operating conditions.

system expansion factor

As the number of simultaneous users increases, the response time steadily and sharply degrades. Thus, if we act so as to maximize the number of simultaneous users, the effect is to maximize the number of people to whom we are presenting a sharply degraded service. This means that the number of simultaneous system users is a misleading measure of system load. *System expansion factor*, rather than number of simultaneous users, is a better measure for the computing center and for the users. This factor is defined as the ratio of the actual time to do a unit of computer-limited work to the minimum time to do that work in a stand-alone environment, and is calculated by dividing the total system service time plus queuing time to perform some user-requested action by the CPU time plus the estimated I/O time. This assumes that computation time and I/O time for any one user are not overlapped. It also assumes that all paging and disk arm movement are overhead, induced by contention. Although this is not entirely true in all cases, it serves well enough as a first-order approximation of what actually happens.

response time

We find that significant performance improvements lead to a reduction in the number of simultaneous system users, perhaps because they do their planned work sooner. This is accompanied by an increase in the number of interactions processed, coupled with decreased overhead needed to do that work. Figure 4 shows the profound influence that system response time degradation can have on user behavior. If we break an interaction into two parts, a System Response Time (SRT), during which the system is processing a request for a user, and a User Response Time (URT), during which a user is keying in the next request to the system, each second of system response degradation leads to a similar degradation added to the user's time for the following request.[7] This phenomenon seems to be related to an individual's attention span. The traditional model of a person thinking after each system response appears to be inaccurate. Instead, people seem to have a sequence of actions in mind, contained in a short-term mental memory buffer. Increases in SRT seem to disrupt the thought processes, and this may result in having to rethink the sequence of actions to be continued. This phenomenon was studied at Yorktown in 1971 during normative studies of interactive computing, and is discussed in Reference 7. We recorded every interaction on TSS for three years, and this data base, which in-

Figure 4 Effect of system response time on user response time

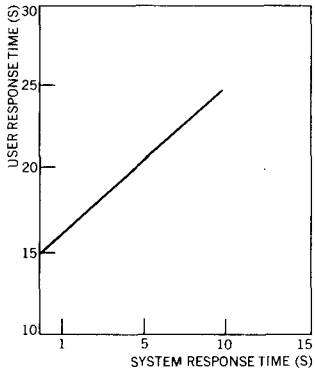


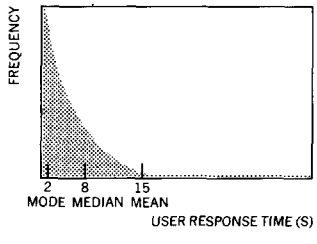
Table 1 Benchmark tasks on System/370 Model 158

	VM/370 with Resource Manager			*VM/370 without Resource Manager*		
Storage	2 Mbyte	1 Mbyte	2 Mbyte	2 Mbyte	1 Mbyte	1 Mbyte
Users	80	80	80	60	80	60
Response	0.146s	0.624s	2.2s	0.537s	1.6s	0.601s
Transactions	12,374	11,149	7,921	9,176	7,319	8,062
Problem CPU	60.4%	50.3%	33.6%	37.1%	23.8%	43.5%
Total CPU	98.5%	88.8%	66.6%	97.4%	51.2%	73.2%
Drum I/O	95%	95%	65%	65%	65%	65%
Disk I/O	5%	5%	35%	35%	35%	35%

cluded a wide range of human tasks, was used in the reported study. It is interesting to note that controlled behavior studies[8] going back to 1938 show that the following three things happen in a stimulus-response situation when the stimulus becomes both long and erratic: People slow down in their work, they become emotionally upset, and they make more mistakes. Given an interaction rate of nine million per month, a system response time delay of two seconds, and assuming the short-term human memory buffer model, there would be a calculated minimum of 36 million seconds per month of lost user time. That is 10 000 man-hours or 60 full-time persons lost for the month. Thus our objective is to keep 90 percent of our interactions to a response time of one-half second or less. Subsecond response time appears to be as important a human requirement as an economic one.

Figure 5 shows the distribution of User Response Time (URT), which has been known as "think time." The mode of this distribution is 2 seconds, the median is 8.5 seconds, and the mean is 12 to 15 seconds, depending when one terminates the data points. If all end points, such as the two-hour lunch break or the times when users forgot to log off, are included, nearly any average is conceivable. In the extreme case, the quantity measured is the length of time the system stayed up. By truncating after two minutes, we have found a consistent URT average of 12 to 15 seconds over the past several years.

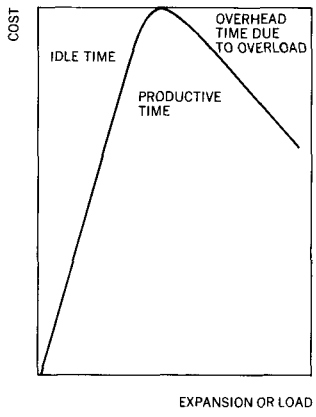
Figure 5 Frequency of user response time



resource management facility

Table 1 highlights the effect of scheduling on the man-machine interface. It shows the effect of the Resource Manager facility for VM/370 in a benchmark environment as it would affect users. The columns with VM/370 Resource Manager show the results of running with that facility in a heavily loaded environment. The columns without VM/370 Resource Manager show that same heavily loaded environment without that facility. Notice that in this environment, fifty percent more interactions can be handled by the system with the Resource Manager. And the response time at the ninetieth percentile is twenty times better. Notice also that the

Figure 6 Effect of expansion or increasing load on productive time



base system handles more interactions with sixty persons than with eighty. By allowing those extra twenty users onto the system in that overloaded environment, we observe that we have effectively lost the twenty people and slowed down the other sixty. Without a knowledge of the expansion factor, the number of simultaneous users served is a poor measure of the load on the system. The Resource Manager has been included as a part of the Systems Extension Program Product beginning with Release 5 of VM/370.

Figure 6 shows computer cost as a function of the expansion factor. If a machine is underused, most of its costs may become lost dollars. As the demand for computing service grows, the expansion factor also grows. Within a certain range of expansion factors, the system is optimally used, in the sense that as the load builds to still higher expansion factors, a significant portion of the hardware usage goes into the unproductive work of system overhead.

Figure 7 Effect of underload and overload on hardware cost

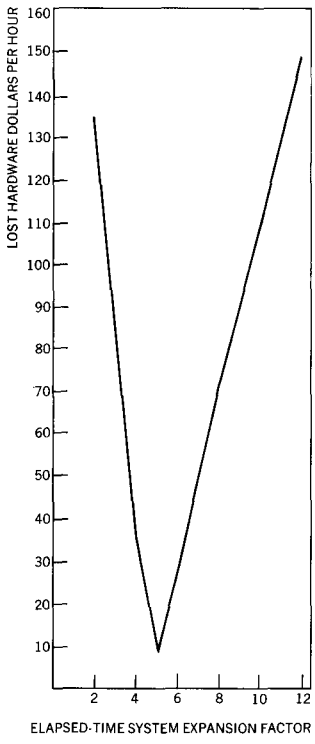


Figure 7 shows the lost hardware dollars per hour that result from underload and overload on one of the VM/CMS systems at Yorktown. The V-shaped curve is an envelope that contains the actual operating points. The expansion factor used here is an *elapsed-time expansion factor*, which is the period of time needed to process a unit of computer-limited work divided by the minimum time to do that work in a stand-alone environment.

Figure 8 shows the cost of the lost user time per hour *versus* the expansion factor. The expansion factor grows together with the number of simultaneous users. The number of persons being slowed down by the delivery of poorer service to each one also grows.

Figure 9 shows dollars lost due to a combination of hardware operating points and user costs. The graph suggests that there is indeed an optimal operating range, having expansion factors in the range of four to six, that depends on both the operating system and the machine speed. The cost of overload is far greater than the cost of underload when the user's time is considered.

We have software instrumentation called VM/Monitor to accumulate data on the daily use of the VM/CMS systems at Yorktown,[9] and we attempt to maintain the operating point not exceeding an expansion factor of six for any hour. Note that the definition of expansion factor that uses the ratio of elapsed time to stand-alone time differs by about a factor of three from that reported via the INDICATE command in VM/370.

performance

Techniques and programs discussed in Reference 9 gather data and analyze the service each user receives each day. These pro-

grams are run each night, and give us a clear indication of the quality of service rendered that day, the likely causes of problems, and the relative pressure on each system resource. These programs calculate the expansion factor for each hour of the day and make the business of tuning, capacity planning, and service analysis a fairly straightforward task. Reference 10 suggests that the performance of an entity (a function, a system, etc.) is the difference between the observed behavior of an entity and the observer's expectations of behavior. We have found that if the computer user can be given some notion of the current state of the system, the user's expectations are modified accordingly, and sensible decisions in planning work can be made. In TSS we introduced a numerical measure called the "THI" (borrowed from the Temperature-Humidity Index of human comfort) which was an averaged value of the time needed to perform a standard task. The THI was available to the user before he logged on, and could be interpreted in terms of the work the user was planning to do. For example, editing received reasonable response, even with a relatively high THI, whereas a PL/I compilation and execution might be unacceptably slow under the same conditions.

It was reasonable to transfer this experience with the THI to VM/370, but it was also desirable to give VM/370 users more information about the status of VM/370 than the simple THI of TSS. Accordingly, a special THI[11] was implemented for VM/370, and the user logs on to VM/370 to obtain this information. The components of the VM/370 THI are available to both the control program and a user program that is executing in virtual storage. Certain maintenance tasks, such as data base backup or data migration, can check for low system activity before proceeding, and thus minimize their effect on the user community. We have also found that information about system overloads derived from the THI enables the operator to ask dominant users to postpone or moderate their activities, and to detect program looping not apparent to the user. The Yorktown version of the VM/370 THI and other information-collecting facilities were made part of VM/370 Release 2, and are known as the VM/370 Measurement Facility. The VM/370 THI is now known as the Load Indicator.

Data management

In 1965, APL on the Yorktown System/360 Model 50 imposed rigid constraints on the size of a user's data base: 36K bytes of working space of which about 32K bytes were available to the user (who constantly protested against those limits). At that time, the only way to increase work space size in APL was to provide more main storage and/or reduce the number of work spaces in main storage simultaneously (thereby usually increasing response time). Until APL/CMS became available, the APL work space limitation was an

Figure 8 Cost of lost user time per hour as a function of system expansion factor

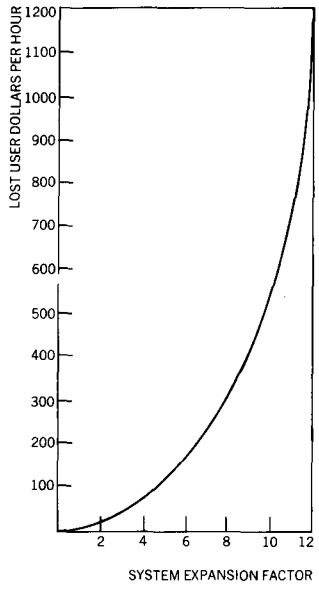
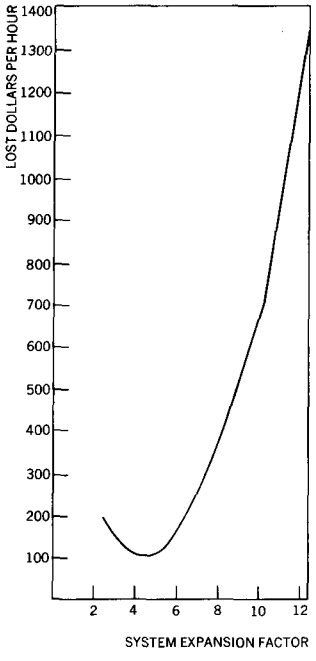


Figure 9 Dollars lost due to a combination of hardware operating points and user costs



annoyance to many users. On TSS/360, the situation was radically different because TSS appeared to its users to be a one-level store. To the limits of allocated disk space, the TSS user could create data sets, file them, and retrieve them by data set name. The system managed all direct-access storage space for users. Although we could not continue to add on-line disk storage to TSS, we nonetheless wanted to avoid requiring the user to spend time moving data sets to and from private tape or disk volumes. We therefore developed a data set migration facility for TSS that became operational in April 1969, and had the following characteristics:

data set migration

- Data sets were marked with the date last referred to.
- Data sets not referred to for more than a specified number of days (a number that was estimated to maintain a safe level of on-line space available) would be compressed and migrated from the on-line disk volumes to demountable disk volumes and stored off-line.
- If a user referred to a migrated data set, the system informed him that it had been migrated and told him how to obtain it.
- Users could list the names of all migrated data sets, voluntarily migrate data sets not needed in the near future, and erase unneeded data sets.

TSS Data Migration met many users' data management requirements, but it had some limitations. There was no mechanism by which users could specify that any migrated data set to which he referred be restored automatically without additional action. This was an annoyance even though the user was given a message that contained the name of the migrated data set. If the user knew that he must restore a migrated data set, he was still required to initiate the restoration process and wait until restoration was complete. Most users preferred that restoration proceed without locking the terminal, so that other work could continue, and then inform them when restoration had been completed.

With the introduction of CP/67 and later VM/370 at Yorktown, the interactive system user found that the burden of on-line space management was transferred back from the system to the user.

- Because each user is given a fixed allocation of disk space, the user might have to stop during work to decide which files to erase or move to tape, in order to make room for new files being created.
- As a user's files grew, the user had to try to persuade installation management to allocate more space. On the other hand, the user was being pressed by management to give up space not actually in use.
- When the user was given more space, the previous allocation had to be copied by the computing center into the new space. This was an overnight operation, and usually required that

many users' space allocations had to be copied, even though they might not be changed in size.

data management strategies

By late 1972, the computing center had more requests to add new users than available disk space for them. Since user disk space on VM/370 is frequently only partially used, and since on-line disk space is only fractionally active at any point in time, rather than invest heavily in additional disk storage devices, management explored two strategies. Least acceptable to the user community was an external schedule that distributed the available system time among the users by mounting only certain subsets of users' disk files at certain times. This not only constrained each user to specific times, but frequently made it impossible to access a colleague's files if those files were available only at a different time. This strategy also resulted in wasted space occupied by extra copies.

As a result, we began a one-man development effort of a second strategy to determine whether user files could be managed by the system so that only "active" files occupied on-line disk space. This design was very promising, and by 1973 we decided to transfer our experience from TSS to the minidisk concept in VM/370. The most convenient unit of space on VM/370 is not that of users' files (although we subsequently implemented voluntary file migration), but the minidisk, which is a specific number of contiguous cylinders of on-line disk space.

VM/370 data migration, as implemented at Yorktown, moves and compresses an entire inactive minidisk to a demountable disk, thereby leaving an empty slot on-line into which an active minidisk of that size can be moved on demand. As in the case of TSS, the decision to migrate a minidisk is made on the basis of a specified number of days elapsed since the minidisk was last accessed. The user whose minidisk has been migrated experiences a longer than normal delay during his first log-on, while the minidisk is being located, expanded, and moved into an empty on-line slot. Subsequent log-ons proceed normally, unless the user again fails to log on within a specified number of days.

AUTOLOGON

Since the minidisk migration task must be available at all times, we have implemented an AUTOLOGON facility that automatically logs on specified tasks after a system restart. AUTOLOGON has had value far beyond its role in minidisk migration. As a result of demands for asynchronous processing mentioned earlier in this paper, we find that there are many special tasks that carry out important services for our users. For example, the NETWORK task, which enables users to send work between systems, a function that previously required manual restart by an operator, is now started automatically at system load time. The file migration capability that we have added to VM/CMS enables users to migrate

files by file name from a minidisk to a Mass Storage System cartridge, making it possible to save and retrieve CMS files. A valuable consequence of VM/370 migration is the backup capability it affords the installation. On third shift, copies are made of all minidisks that have been accessed on the previous day. These minidisks are stored on a second set of on-line disks that are copied to tape. If a user inadvertently erases files, within four hours the computing center can restore a copy of the minidisk that is no more than twenty-four hours old. The computing center has rarely lost a user's data; nearly all data losses take place when one erases or inadvertently writes over his own files.

Another major benefit of the data migration facility is the reduction in the cost of managing on-line data. Just before the introduction of these facilities for VM/CMS in 1973, there were about three hundred fifty tape mounts per day and two hundred disk mounts per day throughout the computing center. By early 1978, when the use of computing had grown by a factor of eight times that of 1973, we were processing thirty tape mounts and five disk mounts per day. Thus an effective reduction of at least a factor of eighty has been achieved in the frequency of tape and disk mounting.

Specialized function

A user's virtual machine in VM/370 is well isolated from other virtual machines so that the user may cause only his own virtual machine to crash, but cannot normally affect others. Therefore, we believe that the virtual machine is an excellent vehicle for the development of special functions that can then be tailored for ease of use, ease of maintenance, and good performance. An important strength of VM/370 is the strong isolation provided by virtual machines.

Complexity in a user's interface is lessened because in VM/370 a particular function can be isolated from all other functions, and the user need learn only that function to use it. As a function evolves over time, tradeoffs in efficiency and ease of use can evolve together without being clouded by other, irrelevant issues.

Because users of interactive systems have become increasingly inventive and demanding in their uses of the system, interactive growth has resulted in a proportional increase in the demand to do computing that is separated in time from one's interaction with the system. This has usually been called "batch processing." By grouping special-purpose functions in special machines, we can apply better controls to those functions. This includes special scheduling controls, the grouping of requests so as to reduce the overhead of initialization, and the tailoring of functions over time in accordance with a user's evolving needs. Also, by giving users

the ability to submit work to virtual machines to be processed at a later time, we remove bottlenecks between a user's interface to the system and the user's data. It is important to note also that these special-purpose functions need not run on the same real machine as the one on which the user entered the function.

Thus the virtual machine concept applies directly to distributed processing. The gains to be had for load distribution may be real, but they are small in comparison with the gains in reduced complexity, ease of maintenance, and improved service to the user. Our experience with distributed specialized function leads us to the conclusion that we have now seen three distinct stages of computing center activities. The first occurred in the early sixties, when the computing center provided computing services and served as a pool of applications programmers who were allocated to help scientists develop programs for using computers to solve problems that were machine limited. At that time, computing was about one hundred times more costly, and so only a few applications were cost justifiable. There also existed at that time a lack of data management facilities and a primitive user interface.

The second stage was that of the systems programmer, when most computing centers grew rapidly in size of both hardware and software. Highly skilled people were required and change management grew increasingly complex. During this period, the computing center personnel were devoted to making the systems more manageable, and their time no longer went directly to the user. Scientists and administrative personnel became direct users of the systems.

Our current environment is the third stage, in which the systems personnel are primarily involved in basic systems changes and the management of tools for distributing the responsibility of change control to the users, who have the strongest need for the changes. In this way, users themselves can better control their own environment. Our networking capability allows our users to communicate freely with one another and exchange new functions directly in machine readable form. Systems programmers are seldom involved. Our users send an average of 3000 files per day in and out of Yorktown computing systems. The typical file size is 50 000 characters, which is equivalent to a twenty-page paper. That is a measure of the communication among users on physically different real machines. Communication among users of the same physical machine but different virtual machines is probably far greater.

Summary

Appropriate management actions can significantly enhance the effectiveness of an interactive system from the user's point of

view. During more than ten years' experience with interactive systems at the IBM Thomas J. Watson Research Center, we have made management decisions that extend the capability of these systems to work for the user by enabling sensible decisions to be made about the user's own time, and by selectively adding to the systems facilities that enable the user to spend less time compensating for system limitations and more time on the problem. Increased emphasis has been placed on using the computer as a tool to extend the users' memory as well as their logic power. Our computing systems are adapted by experienced users, via EXEC's, so that the effectiveness of man-machine communication increases with time. This is of special benefit to the inexperienced user who then takes advantage of the experienced user's evolutionary growth. The ways in which we have dealt with issues of availability, distributed change management, data management, in-line documentation, response time, expansion factors, specialized function, and effective user-to-user communication have been the key to our success. VM/370 and CMS are the primary vehicles that our users have found to be effective for their rapidly evolving interactive environment.

ACKNOWLEDGMENTS

The authors acknowledge the contributions of VM/CMS managers at Yorktown, who include W. M. Buco, A. N. Chandra, B. Lie, D. T. Mainey, N. J. Pass, and H. Serenson. P. H. Callaway and W. H. Tetzlaff have contributed to the performance management of VM/CMS. A. M. Katcher, W. R. Deniston, D. T. Mainey, and R. P. Carroll developed the data management facilities. L. H. Wheeler and W. M. Buco have developed scheduling and resource management strategies. C. J. Stephenson has been responsible for the evolution of the EXEC language since 1971. W. E. Daniels developed a set of EXEC's that are collectively called MAINTAIN. A. N. Chandra developed an experimental mechanism for communication among different virtual machines that run on the same real machine.

CITED REFERENCES

1. A. W. Luehrmann and J. M. Nevison, "Computer use under a free-access policy," *Science* **184,** 957-961 (1974).
2. W. E. Daniels and R. W. Ryniker, *EXEC 2, A Computer Language for Word Programming*, Research Report RC6292, IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598 (1976). (ITIRC AAA77A000736)
3. W. J. Doherty, "Human factors: Impact on interactive computing," *Proceedings of SHARE 50* **2,** 1244-1266 (1977).
4. T. Johnson (SLAC) and W. J. Doherty, private communication, February 1977.
5. B. W. Boehm, *Software Engineering*, Report SS-76-08, TRW, Incorporated, Redondo Beach, CA.
6. A. Guido and J. P. Considine, "Laboratory automation via a VM/370 teleprocessing virtual machine," *AFIPS Conference Proceedings* **46** (National Computer Conference, Dallas, Texas, June 13-16, 1977), 865-877 (1977).

7. S. J. Boies and J. D. Gould, "User performance in an interactive computer system," *Proceedings of the Fifth Annual Conference on Information Sciences and Systems*, Department of Electrical Engineering, Princeton University, Princeton, New Jersey (1971), p. 122.
8. R. S. Woodworth, *Experimental Psychology*, Henry Holt and Co., New York (1938).
9. W. H. Tetzlaff, "State sampling of interactive VM/370 users," *IBM Systems Journal* **18,** No. 1, 164–180 (1979, this issue).
10. W. J. Doherty, "Measurement and management of interactive computing," *Proceedings of SHARE XLIV* **3,** 1587–1598 (1975).
11. P. H. Callaway, "Performance measurement tools for VM/370," *IBM Systems Journal* **14,** No. 2, 134–160 (1975).

Reprint Order No. G321-5090.