A methodology for performance-tuning a virtual storage system is discussed. This methodology encompasses performance measurement, workload characterization, performance evaluation, and planned experimentation. Use of the methodology is illustrated by describing results of a case study involving the IBM *Research Division's* TSS/360 *system.*

Tuning a virtual storage system

by H. A. Anderson, Jr., M. Reiser, and G. L. Galati

The performance objectives of different computer installations are diverse. Therefore an operating system should provide a flexible and easily modifiable mechanism for implementing system resource management policies.[1,2,3] Specifying a policy for a particular computer installation is an iterative process, in which there are five logically distinct steps:

- Understanding the resource management components of the system
- Recording workload and system activity
- Characterizing the workload and identifying the system variables that most significantly affect performance
- Experimenting with various resource management policies
- Determing the policy that best satisfies the installation's performance objectives.

The intent of this paper is to illustrate these steps by means of a case study of a virtual storage system, the TSS/360 (Time Sharing System/360) installation at IBM'S T. J. Watson Research Center in Yorktown Heights, New York. Emphasis is on the methodology of the five steps rather than on the TSS installation itself. We also stress our use of data analysis techniques to evaluate the effects of system changes on performance. A benefit of our methodology, with regard to modeling system performance, is a simplification of the system description. Our objective is to improve system responsiveness to a typical user transaction, which we define later in this paper.

System resource management

In a virtual storage computer system, resource management consists of three basic operations: I/O load balancing, the scheduling and dispatching of tasks, and the allocation of page frames in main storage and page slots in external disk storage. Typically, system resource management algorithms derive some of their control information from parameters contained in system control blocks which are specified at system generation time. These system parameters aid in implementing a resource allocation policy that is appropriate for a particular computer installation. In our installation, the scheduling, dispatching, and storage management mechanisms provided effective controls for adjusting system responsiveness. (I/O load balancing presented no problem.) An understanding of these controls is the first step toward interpreting the results of system measurements.

scheduling and dispatching

In TSS/360, as in TSO, a conversational task is created when a user logs on and is deleted when he logs off. Task scheduling and dispatching are controlled by a table-driven scheduler[4] which provides for detailed specification of scheduling and dispatching policies. The scheduler is driven by a system control block called the scheduling table, which is initialized at system generation time. It consists of a set of scheduling table entries (STEs), each having 25 parameters. We can classify the parameters as follows:

- *Scheduling parameters* assign a priority to an eligible task (i.e., an active task awaiting allocation of main storage) and set a deadline for the next time slice dispatched to that task.
- *Dispatching (or time slice) control parameters* limit the system resources that can be allocated to a dispatchable task (i.e., an active task that is being multiprogrammed). Examples of such parameters are the length of a time slice and the maximum number of page frames allowed to be allocated.
- *Time-slice-end exit parameters* specify the set of possible STE assignments that control a task's next time slice.

Scheduling and dispatching algorithms are defined by specifying the STEs, a function that is similar to specifying the Installation Performance Specifications of the System Resources Manager in OS/VS2.[2] A task is always assigned to an STE and undergoes an STE transition at the end of a time slice, when it is being rescheduled. Most queue-driven operating systems are built around three queues: an *inactive queue* of tasks not ready for service (e.g., tasks waiting for terminal response or the mounting of tapes or disk packs), an *eligible queue* of tasks that are ready for service but have not been allocated main storage, and a *dispatchable queue* of tasks that have been allocated main storage and share the processor.

The scheduling table contains parameters used by the algorithm that controls the flow of tasks from the eligible queue to the dispatchable queue. When an inactive task becomes active it is enqueued in the eligible queue, and a deadline is computed for its next time slice by using the scheduling control parameters assigned to the task. Eligible tasks are ordered by their respective priorities (which are also determined by the scheduling table), and they are ordered within their priorities by their time slice deadlines. Thus the eligible queue is a set of priority queues.

The scheduler searches the eligible queue to find the highest-priority, farthest-behind-schedule task that satisfies the main storage entrance criterion, then enqueues that task in the dispatchable queue. The main storage entrance criterion controls the level of multiprogramming. In TSS/360 the criterion is

$$P_{max} < P_{eav}$$

in which P_{eav}, a system variable, is an estimate of the number of available page frames, and P_{max} is the STE dispatching control parameter that limits the number of page frames that can be allocated to the task when it is in the dispatchable queue. A similar entrance criterion exists in other virtual storage systems.

The scheduler tracks the size of a task's working set by assigning an STE transition at the end of a time slice. This transition may increase or decrease P_{max}, depending on the task's page frame allocation at time slice end. P_{eav} is updated when a task is enqueued in the dispatchable queue according to the formula

$$P_{eav} \leftarrow P_{eav} - P_{max}$$

and again when the task is dequeued from the dispatchable queue:

$$P_{eav} \leftarrow P_{eav} + P_{max}$$

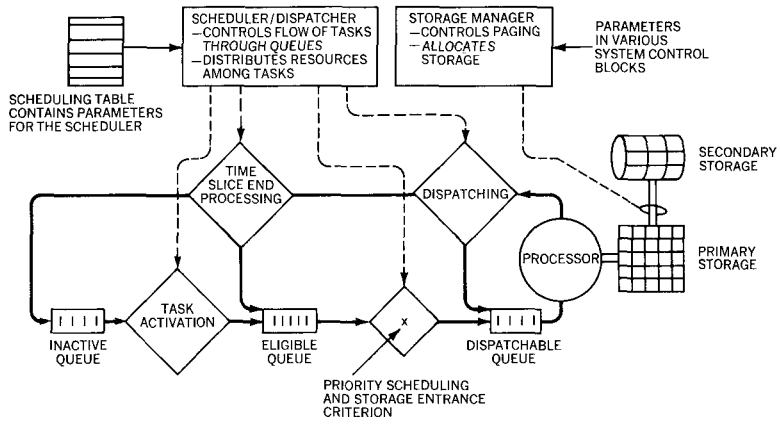
P_{eav} accounts for the page frames allocated to the supervisor and the pages of the dispatchable tasks. A task remains dispatchable until the end of its current time slice (which is subdivided into quanta). If the task is still active at time slice end, it is enqueued in the eligible queue. If it is not active, it is enqueued in the inactive queue (see Figure 1).

The TSS/360 scheduler distinguishes among conversational tasks, nonconversational tasks, and system tasks. The primary objective of our study was the scheduling of conversational tasks to improve responsiveness.

memory management

There are two types of pages, sharable and private. Examples of programs residing in sharable pages are the command analyzer and the access methods. User coded programs generally reside

Figure 1 Task queuing in TSS/360



in private pages. Each page type is managed differently. The page frames allocated to a dispatchable task contain the private pages referenced by the task during its present time slice. At the end of a time slice, a task's page frames are released and made available. A task's STE also indicates whether a page stealing (replacement) algorithm is in effect for it or not.[4] The page frames that contain sharable pages (sharable page frames) are not allocated to tasks.

Usually, when a page fault occurs, service of the faulting task is halted and the task incurs a page wait while the faulted page is loaded into a newly allocated page frame. Sometimes there is still a copy of the faulted page in an available page frame, in which case that page frame is reclaimed without making the task wait for an unnecessary page transfer.

The storage manager attempts to maintain an inventory of available page frames. When a page fault occurs and the inventory is below an upper threshold, the reference bits of the sharable page frames are reset. If the inventory is below a lower threshold, the sharable page frame reference bits are scanned to find a preestablished number of unreferenced sharable pages to replace from main storage. If sharable page replacement does not increase the number of available page frames to a quantity above the upper threshold, the storage manager requests the dispatcher to reduce the level of multiprogramming by selecting a dispatchable task to be swapped out of main storage. A task swap-out request is the way the scheduler is informed of excessive contention for storage.

In an interactive virtual storage system, the management of sharable pages and user pages in auxiliary storage directly affects performance. The TSS/360 storage manager controls the migra-

tion of sharable and private pages. Copies of the most frequently referenced pages are maintained on the paging drums, the less frequently used pages being deferred to paging disks. The main objectives of these algorithms are to equitably share the paging drum among the active users and to prevent the fast paging devices from becoming cluttered with sharable pages.

Instrumentation

The evaluation of interactive system performance requires basically two kinds of data: (1) a record, or log, of user transactions (i.e., the interactive workload), and (2) multivariate measurements of system activity. A user transaction is a sequence of user-system interactions, starting with the user's initial request for service and ending when the system prompts him for more work. A user transaction log, consisting of timing measurements and records of computing resource demand, can be produced by an event tracing monitor.[5] To characterize the TSS/360 workload, we used an event tracing monitor that measured the following variables for each task:

- Processing time spent in virtual storage (problem state)
- Data base reads and writes
- Page reads and writes
- Average number of page frames allocated during a time slice
- Lines of terminal output
- Number and types of commands.

system activity

To record system activity measurements, we used a sampling monitor that sampled the contents of event counters and event timers at a specified rate, forming a multivariate time series. The quantities measured can be classified as follows:

- Paging demands
- Traffic between main storage and external storage (data set activity measurements)
- Processor utilization
- Causes of time slice ends
- Computing resources allocated to tasks
- Scheduling and dispatching activity.

Our sampling monitor sampled system activity once a minute. In a virtual storage system, some variables representing internal congestion and storage usage can fluctuate considerably over a minute, and it is desirable to have the system maintain smoothed (or filtered) values of these variables. In our situation, however, filtered values were not maintained by the system itself. Therefore, with the event tracing monitor, we sampled selected variables many times per minute and time-averaged them off-line. These filtered values were then merged with the measurements of the sampling monitor.

Table 1 Selected system variables that significantly affected the responsiveness of the IBM Research Division's TSS/360 system

$*N_t$	average number of tasks
$*N_e$	average number of eligible tasks
$*N_d$	average number of dispatchable tasks
p_v	processor utilization due to executing in virtual storage (problem state)
p_o	processor utilization due to executing of the resident supervisor
$*P_{eav}$	average number of available page frames as estimated by the scheduler
$*P_a$	average number of available page frames
$*P_s$	average number of sharable page frames
r_s	sharable page reclaim rate (pages per second)
r_p	private page reclaim rate (pages per second)
r_{pf}	system page fault rate (pages per second)
TSE_{mp}	time-slice-end rate due to a task's exceeding its P_{max} (TSEs per second)
TSE_{mq}	time-slice-end rate due to a task's exceeding its virtual storage processing limit (TSEs per second)
TSE_{so}	time-slice-end rate due to forced swap-outs—i.e., the forcing of tasks off the MP queue to free up page frames (TSEs per second)

*system variables for which time averaging was required

The data we analyzed consisted of a minute-by-minute summary of the activity represented by over 70 system variables and the averaged values of response times for the typical transaction (Table 1). A similar approach was used by Anderson and Sargent to produce the performance data analyzed in evaluating the performance of an APL/360 system.[6]

The workload

The formulating of system resource management policies requires a classification of the workload based either on the type of service requested or on resource demands. The quality of service received by transactions in a given workload class depends on the response time or throughput criteria established for that class, the statistical properties of the computing resource demands, and system congestion at the time of service. We classified the IBM Research TSS/360 interactive workload according to the statistical properties of the computing resource demands.[7] As a result of this classification we identified a certain class of transactions, which we term *typical transactions*, that require less than 0.2 second of virtual storage (problem state) processing time and make no more than ten read/write accesses to a data set. We found 70 percent of all transactions to be typical. Our objective for performance improvement was to decrease the average response time of the typical transaction.

Table 2 Examples of the positively skewed cumulative distribution functions (CDFs) of the resource demands of transactions

Virtual storage processing time		*Data set reads*		*Data set writes*	
Seconds	*CDF*	*Reads*	*CDF*	*Writes*	*CDF*
0.1	0.745	0	0.622	0	0.622
1.0	0.958	10	0.815	10	0.915
10.0	0.991	20	0.905	20	0.945
100.0	0.998	30	0.940	30	0.965
Avg. 0.75 s		Avg. 9.65 reads		Avg. 6.4 writes	
Std. 12.3 s		Std. 50.3 reads		Std. 60.0 writes	

Table 3 Relative frequency of data set reads vs. data set writes

Data set reads	*Data set writes*			
	0	*1 to 10*	*11 to 20*	*21 to 30*
0	0.622	0	0	0
1 to 10	0.164	0.034	0.010	0.001
11 to 20	0.028	0.019	0.014	0.013
21 to 30	0.001	0.010	0.001	0.002

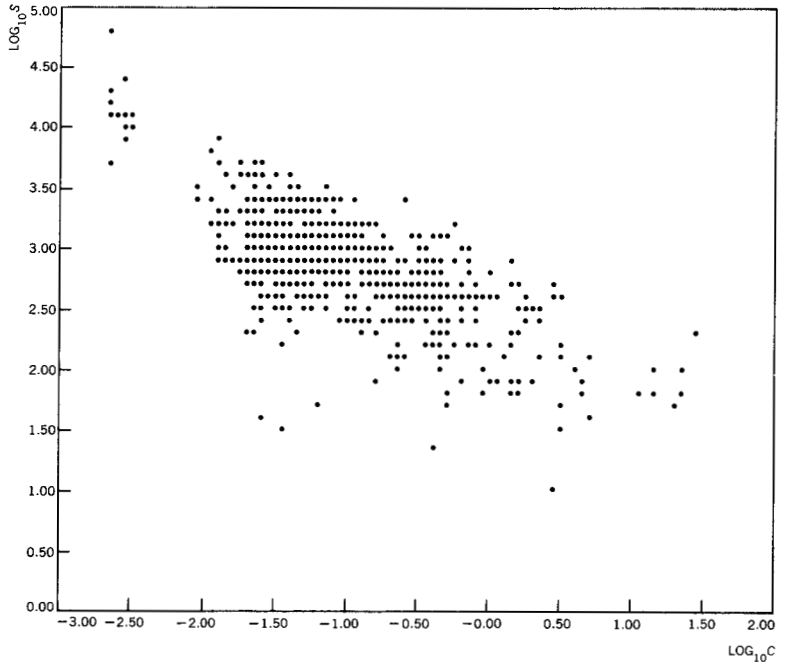
Typically, the cumulative distribution functions (CDFs) of the computing resource demands of conversational transactions are positively skewed (Table 2). Similar observations have been made for other interactive systems, leading Bryan[8] to remark, "We rarely see the average, we usually see the typical". For example, the CDF of the virtual storage processing time has a mean of 0.75 seconds, a median of 0.05 seconds, and a coefficient of variation of 16.5. Foreground-background scheduling is appropriate for systems with such skewed resource demand distributions.[6]

Interesting relationships can be observed among the workload variables. For example, in Table 3 the relative frequency of data set reads compared with data set writes indicates the existence of three classes of transactions: those that require no transfer of data from or to external devices, those that require data to be loaded into virtual storage, and those that require data to be loaded into and unloaded from virtual storage. This classification is still relatively coarse. We have found finer divisions by means of principal component analysis.[7]

paging

The performance of the system depends greatly on the paging rate. For example, the correlation coefficient is 0.82 between the paging rate and the average response time of a typical transac-

Figure 2 Paging rate (S) for TSS/360 transactions as a function of virtual storage processing time (C)



tion. This is the highest coefficient of correlation between the response time and any workload variable. This high correlation shows the significant effect that the page replacement algorithm has on response time.

The paging demand had other interesting characteristics. When there was an increase in the virtual storage processing time, C, demanded by a transaction, we observed two relationships: the average number of allocated page frames increased (Table 4), and the paging rate, S, decreased rapidly (Figure 2). The plot of log S vs. log C in Figure 2 shows a linear relationship. Transactions requiring 0.002 second of virtual storage processing may require that 20 pages be referenced, and thereby have S = 10,000 page reads per second. Processing those transactions is costly in terms of paging overhead. On the other hand, transactions requiring extensive virtual storage processing time, such as compilations, tend to have low paging rates, of the order of $S < 100$ page reads per second. Tasks activated by such transactions typically use their whole time slice each time they are dispatched. After a brief initial transient phase, consisting of loading their working set, they settle down and relatively seldom give rise to page faults.

The system paging activity is quite sensitive to the mix of tasks, as shown by the great disparity in the paging rate, S. Multipro-

Table 4 Relative frequency of average page frame allocations vs. virtual storage processing times

Virtual storage processing time (seconds)	*Average number of page frames allocated*			
	1 to 10	*11 to 20*	*21 to 30*	*31 to 40*
0 to 0.01	0.001	0.11	0.005	0
0.01 to 0.1	0.02	0.58	0.071	0.002
0.1 to 1	0.001	0.059	0.107	0.009
1 to 10	0	0.003	0.014	0.007

gramming explains some of the variability of S for a given value of C. High levels of multiprogramming cause S to increase for a transaction because the effect of the page replacement algorithm becomes significant. System modifications designed to improve performance, such as repackaging frequently referenced programs, can be evaluated in terms of the changes they cause in the relationship between S and C.

Performance evaluation

To understand the performance of a computer system, one has to translate a logical description of the system into a behavioral one. An understanding of system behavior is obtained by identifying the system variables that have the most significant effect on performance. These variables, called performance factors, can be identified through statistical data analysis techniques such as scatter plots, correlation analysis, and stepwise regression analysis. The goal is to identify the relationships among the numerous system variables and between the system variables and different measurements of system responsiveness. Identifying the performance factors also helps in defining the appropriate level of detail in the system description (the system model).

Data analysis of the kind indicated above led us to conclude that system performance has to be analyzed in terms of the system variables maintained by the scheduler, dispatcher, and storage manager. The scheduler and dispatcher are concerned with controlling the three queues of Figure 1. The scheduler maintains its own estimate of available page frames. The dispatcher sequences the servicing of the dispatchable tasks and limits the computer resources allocated to them during their time slices. The storage manager maintains an inventory of available page frames. Its main task is to allocate and deallocate page frames.

Table 5 Sample correlation coefficients that exhibit intercorrelations between the performance factors and the logarithm of the response time, R^*

		N_e	N_d	p_v	p_o	P_{eav}	R^*
N_t	avg. no. of tasks	0.4	0.2	−0.2	0.3	−0.2	0.2
N_e	avg. no. of eligible tasks		0.5	−0.2	0.5	−0.6	0.4
N_d	avg. no. of dispatchable tasks			0.1	0.6	−0.8	0.3
p_v	processor use due to executing in virtual storage				0.3	0.0	0.0
p_o	processor use due to executing the resident supervisor					−0.7	0.3
P_{eav}	avg. no. of available page frames as estimated by scheduler						−0.5

		P_s	r_s	r_p	r_{pf}	R^*
P_a	avg. no. of available page frames	0.6	−0.5	0.5	−0.5	−0.3
P_s	avg. no. of sharable page frames		−0.5	0.6	−0.6	−0.4
r_s	sharable page reclaim rate (pages/sec.)			−0.6	0.8	0.2
r_p	private page reclaim rate (pages/sec.)				−0.6	−0.3
r_{pf}	system page fault rate					0.3

		TSE_{mq}	TSE_{so}	R^*
TSE_{mp}	time-slice-end rate due to a task's exceeding its P_{max} (TSEs/sec.)	−0.2	0.0	0.1
TSE_{mq}	time-slice-end rate due to a task's exceeding its virtual storage processing limit (TSEs/sec.)		−0.2	0.0
TSE_{so}	time-slice-end rate due to forced swap-outs (TSEs/sec.)			0.1

The most significant system variables, the performance factors, are defined in Table 1. We measured performance in terms of the average response time of the typical transaction, denoted by R and defined by the sum

$$R = \frac{1}{N} \sum_{i=1}^{N} R_i$$

Figure 3 Average numbers of dispatchable tasks (N_d) and eligible tasks (N_e) as functions of the average number of logged-on tasks (N_t)

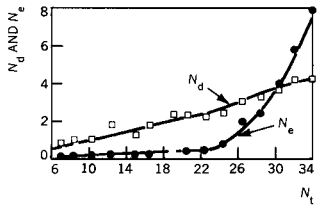


Figure 4 Average response time (R) as a function of logged-on tasks (N_t)

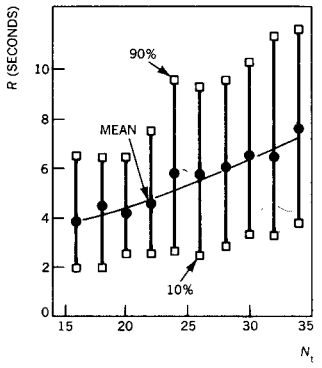
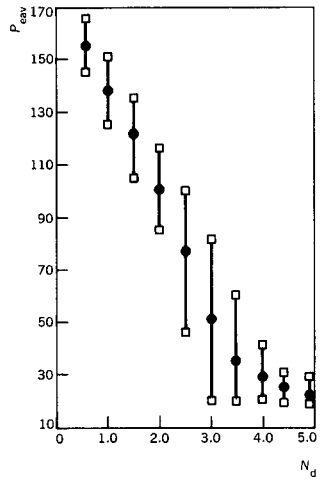


Figure 5 Scheduler's estimate of available page frames (P_{eav}) as a function of dispatchable tasks (N_d)



where R_i is the response time of the ith typical transaction of the N typical transactions that were completed during the sample interval, which in our case was one minute. Some of the performance factors were identified when we developed a statistical model for the logarithm of the response time, R^*, using stepwise regression analysis. R^* is defined by

$$R^* = \frac{1}{N} \sum_{i=1}^{N} \log R_i$$

The logarithmic transformation was used to account for the apparent nonlinear relation of R to numerous system variables and also because of the higher correlation of R^* with other system variables (Table 5). For example, the performance factors P_{eav} and P_s (number of sharable page frames), had the highest correlation with R^*. Other performance factors were identified by analyzing changes in system behavior that resulted from experimenting with different scheduling algorithms. The objective was to bring the system into new operating regions, such as higher levels of multiprogramming.

The performance factors are correlated as indicated by the correlation coefficients in Table 5. The correlations result from the interdependence among scheduler, dispatcher, and storage manager.

The behavior of a virtual storage system is often considered complex because of the high number of possible interactions among the system's components. However, if the performance of such a system is observed for a few days with an uncontrolled workload, patterns of behavior become apparent even though there is wide variation in the system's behavior. These behavior patterns can be explained by determining which factors of the system resource management policy have a dominant effect on performance. When this is done, a relatively simple description of system behavior is possible. The following description of the behavior of our system serves to illustrate this point. (When our measurements were made, the IBM Research TSS/360 computer configuration consisted of a System/360 Model 67 with one megabyte of main storage, two paging drums, and three paging disks.)

The capacity of main storage, and the storage demand imposed by the workload, determine the average number of dispatchable tasks, N_d. When the average number of logged-on tasks, or users, N_t, exceeds 30, N_d approaches four tasks (Figure 3). Initially, the average number of eligible tasks, N_e, increases slowly with increasing N_t, then increases rapidly as the system becomes saturated. A similar form of performance degradation, but in terms of average response time, was observed for CTSS

(the Compatible Time Sharing System) by Scherr[9] and explained with a simple queuing model by Kleinrock.[10] For our system, however, a plot of the average response time of a typical transaction vs. N_t (Figure 4) does not reveal the onset of system saturation as clearly as the plot of N_e vs. N_t (Figure 3).

System saturation results from the combined page frame demand of the dispatchable tasks. When the average number of dispatchable tasks, N_d, approaches five, the scheduler's estimate of the average number of available page frames, P_{eav}, is less than 24 (Figure 5). Therefore, eligible tasks often fail the entrance criterion. Even tasks activated by typical transactions are adversely affected, since their storage management parameter, P_{max}, was set to 24 page frames.

The threshold effect of the entrance criterion is illustrated in Figures 6 and 7. There is a sizable increase in the number of eligible tasks, N_e, when P_{eav} falls below 24 page frames for a protracted period, causing a corresponding increase in the average response time, R. The degradation in system responsiveness caused by the entrance criterion is dramatized in Figure 8 by the substantial change in the shape of the response time histogram that accompanies an increase in R. A similar degradation in response time was predicted by a central server queuing model of the MULTICS system developed by Sekino.[11]

As the number of dispatchable tasks, N_d, becomes greater, there is a decrease in the average number of available page frames, P_a, and sharable page frames, P_s, as shown in Figure 9, because private page frame allocation increases. In order to increase P_a, tasks are forced off the dispatchable queue—that is, swapped out of main storage. In our system the rate of swapping was low, since P_a rarely averaged below the inventory threshold of ten page frames. The system was able to maintain an inventory of available page frames. The number of sharable page frames needed to support more than four dispatchable tasks, N_d, was approximately 64 (i.e., $P_s = 64$ page frames). The decrease in P_s with increasing N_d is accompanied by an increase in the sharable page reclaim rate, r_s, as sharable page thrashing begins (Figure 10).

In Figure 11, the plot of the reclaim rate of private pages, r_p, vs. N_d provides another graphic demonstration of how system performance degrades with increasing congestion. When N_d increases beyond three tasks, r_p approaches zero. In other words, it becomes increasingly unlikely that any of the private pages referenced by an eligible task in its previous time slice will still be in main storage when the task meets the entrance criterion and becomes dispatchable. Therefore during congested periods, tasks that require more than one time slice will often be delayed

Figure 6 Number of eligible tasks (N_e) as a function of the scheduler's estimate of available page frames (P_{eav})

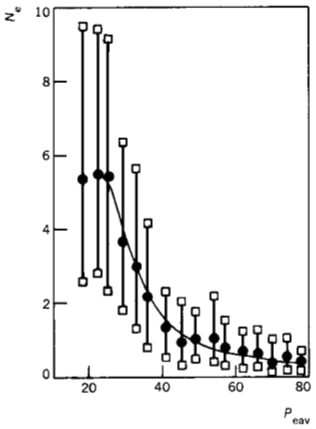


Figure 7 Average response time (R) as a function of the scheduler's estimate of available page frames (P_{eav})

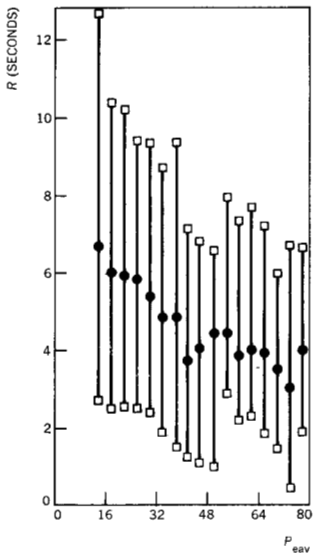
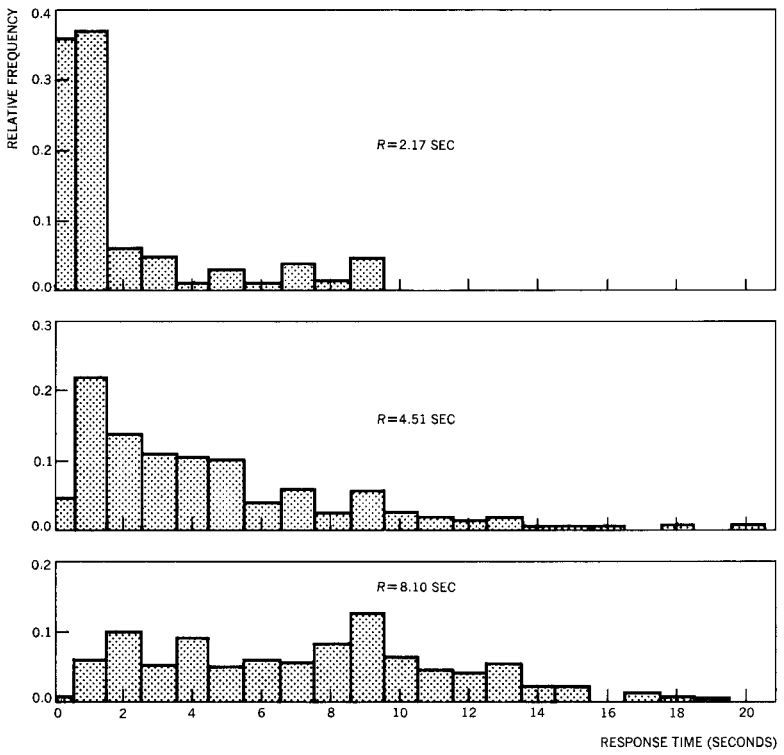
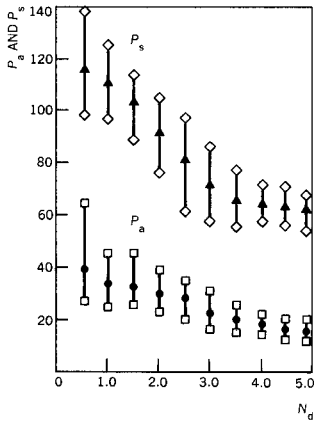


Figure 8 Degradation in system responsiveness with increasing average response time (R)



because of page waits. This mechanism explains the high correlation between response time and the paging rate, as discussed on pages 252 and 253.

Figure 9 Available page frames (P_a) and sharable page frames (P_s) as functions of dispatchable tasks (N_d)



Our objective, as stated earlier, is to schedule typical transactions for improved responsiveness. Considerable paging overhead is incurred in servicing typical transactions because of their high page fault rate. During congested periods, therefore, it is difficult to overlap computing and paging to make efficient use of the processor and still be responsive. We investigated this problem by analyzing processor utilization, which can be divided into three components.:

- p_v, the percentage of time spent executing programs in virtual storage (i.e., in the problem state),
- p_o, the percentage of time spent executing the resident supervisor (overhead),
- $1 - (p_v + p_o)$, the percentage of time the processor is idle.

If we plot both p_v and p_o against the number of dispatchable tasks, N_d (Figure 12), we observe that as N_d exceeds 2.5, p_v no longer increases, while p_0 continues to increase. When $N_d > 3$, the processor is primarily servicing tasks activated by typical

transactions. In other words, typical transactions are the only ones allowing a level of multiprogramming of 4 or even more than 4. There is no increase in p_v in the range $N_d \geq 4$ because there is little overlap of computing with paging. On the other hand, when $N_d < 4$, p_v can become quite large because it is possible for compute-bound tasks in the dispatchable queue to be serviced frequently, thereby efficiently overlapping paging with computing.

A bottleneck develops in main storage when $N_d \geq 4$. It is reflected in the plot of overhead, p_o, vs. N_d (Figure 12), which levels off around 60 percent, indicating that the processor spends most of its time managing storage. The linear relationship between p_o and the page fault rate (Figure 13) illustrates that p_o is accounted for by paging activity.

Figure 10 Sharable page reclaim rate (r_s) as a function of N_d

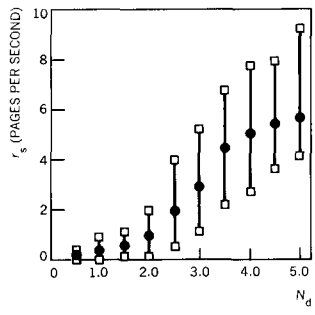
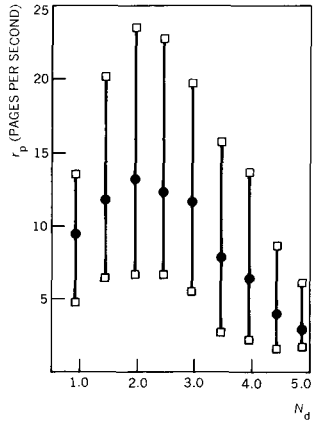


Figure 11 Private page reclaim rate (r_p) as a function of N_d



Improving system performance

Having gained insight into the factors affecting performance, we evaluated system changes expected to improve performance and concluded that system responsiveness probably could be improved by modifying the scheduling algorithm. In addition, we were interested in determining whether we could reduce P_s, the average number of page frames allocated to sharable pages. The existing scheduling algorithm essentially divided eligible conversational tasks into two major classes: newly activated tasks awaiting their first time slice, and tasks whose activation required more than one time slice. The former class had priority over the latter, leading to a form of foreground-background scheduling. Tasks in the latter class were subdivided according to their page frame allocation in their most recent time slice.[1]

Our change in the scheduler was aimed at overcoming a system deficiency requiring that a typical transaction be dispatched as many as three times. We hoped to improve system responsiveness by changing the scheduler to distinguish between eligible conversational tasks that had received less than four time slices and those that had received more. That is, a task would circulate up to three times through the high priority foreground queue before being enqueued in the appropriate background queue. We felt it would be worth evaluating a change in the sharable page replacement algorithm so that sharable page replacement would be initiated not only when the average number of available page frames, P_a, dropped below its lower threshold value, but also when P_{eav}, the number estimated by the scheduler, dropped below a similar threshold. Our objective was to find out whether more page frames could be allocated to the private pages owned by the dispatchable tasks by reducing the average number of sharable page frames, P_s, during periods of high congestion.

Figure 12 Processor utilization due to overhead (p_o) and virtual storage time (p_v) as functions of N_d

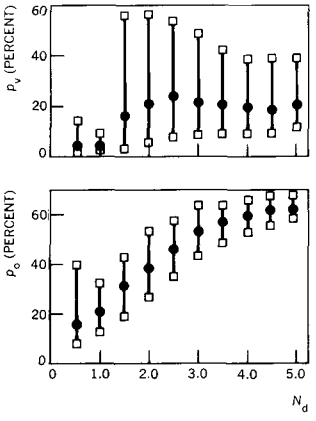
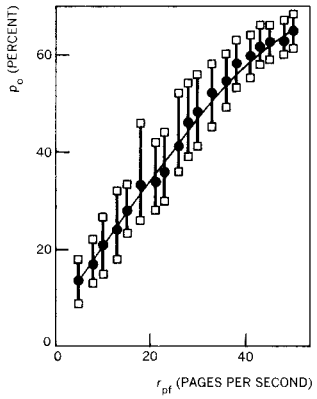


Figure 13 Processor utilization due to overhead (p_o) as a function of the page fault rate (r_{pf})

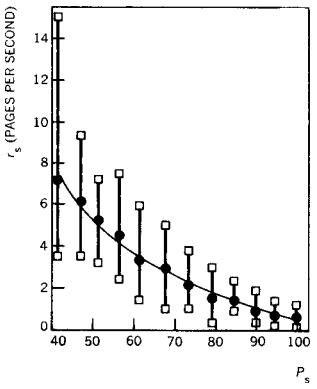


Evaluating these two system changes required a 2^2 factorial design of experiments. The two factors were the scheduling algorithm (old and new) and the sharable page purging algorithm (old and new). Factorial design experiments have been successful in quantifying complex phenomena. Typically they are used to evaluate the effects of a number of variable factors on some measurable response of a process.[12]

Our plan was to conduct each of the four experiments on three different days at nine randomly selected 15-minute periods in the afternoon, when the average number of logged-on tasks, N_t, would exceed 26. We measured in the afternoon because the behavior of N_t was most stable then. The average values of the performance factors, computed for each replication, quantified the uncontrolled workload and the effects of the two experimental factors on different aspects of system behavior. Table 6 presents, for each experiment, average values of the response time and some of the performance factors.

Results of the experiments indicate that the scheduling change reduced the average response time, R, from five to four seconds, and that the change in the page purging algorithm degraded performance slightly. In fact, the page replacement change reduced the average number of sharable page frames, P_s, to the point where the sharable page reclaim rate, r_s, increased significantly, indicating thrashing (Figure 14). Decreasing P_s simply increased the average number of available page frames, P_a, decreasing an already low task-swap-out rate (TSE_{so}) from 0.08 to 0.04 tasks per second. In Table 6, the values of the performance factors indicate that no unexpected or unexplainable changes in system behavior or workload occurred during the experiments.

Figure 14 Sharable page reclaim rate (r_s) as a function of available page frames (P_s)



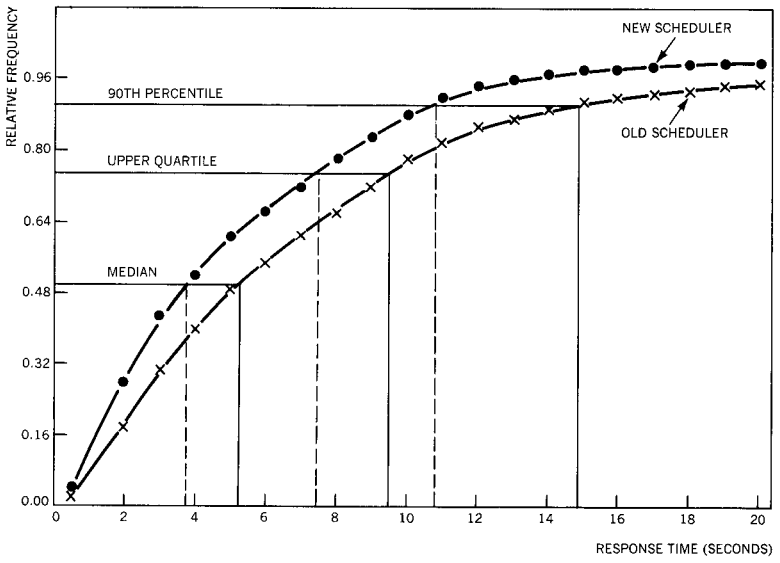
Although our change in the scheduler definitely improved the system's responsiveness, we were concerned that this achievement might be attributed to differences in the uncontrolled workload. Therefore we wanted to test the effects of the two experimental factors after eliminating the effect of workload differences that occurred between replications of the experiments. This was accomplished by using the analysis-of-covariance technique,[12] which eliminates the effects of uncontrolled variables that enter into an experiment and are correlated with the response measurement. The uncontrolled workload can best be represented by N_d, the average number of dispatchable tasks, and N_e, the average number of eligible tasks. Both are correlated positively with the average response time, R.

Table 7 summarizes the analysis of variance for testing the significance of the scheduling and page replacement changes on R, after N_d and N_e have been eliminated. The effect of the scheduling change was found to be significant. The significance was eas-

Table 6 Results of the 2^2 factorial design experiments, showing the effects of the two factors on system responsiveness and system activity

System variables		*Old schedule and old replacement algorithm*	*New schedule and old replacement algorithm*	*Old schedule and new replacement algorithm*	*New schedule and new replacement algorithm*
R	response time	5.38	4.23	5.49	4.39
N_t	avg. no. of tasks	31	32	32	31
N_e	avg. no. of eligible tasks	4.4	4.3	5.6	5.6
N_d	avg. no. of dispatchable tasks	3.7	4.0	4.1	3.9
p_v	processor use due to executing in virtual storage	0.18	0.19	0.18	0.18
p_o	processor use due to executing the resident supervisor	0.59	0.61	0.63	0.62
P_{eav}	avg. no. of available page frames as estimated by scheduler	36	31	29	29
P_s	avg. no. of sharable page frames	67	64	56	55
P_a	avg. no. of available page frames	19	19	26	25
r_p	private page reclaim rate (pages/sec.)	5.3	5.8	7.1	5.7
r_s	sharable page reclaim rate (pages/sec.)	1.7	1.8	4.4	5.3
r_{pf}	system page fault rate	39	42	42	47
TSE_{so}	time-slice-end rate due to forced swap-outs (TSEs/sec.)	0.1	0.1	0.0	0.0

Figure 15 CDFs of response time for the two scheduling algorithms



ily appreciated when we compared the differences in the CDFs of R for the two scheduling algorithms (Figure 15). We concluded that our scheduling change improved system responsiveness, and that our change in the page replacement algorithm was not worth while.

Concluding remarks

Contemporary operating systems provide system resource management mechanisms that permit the installation manager to define his system performance objectives. The installation manager thinks of these objectives in terms of response time or throughput, but in reality they have to be translated into system internal terms such as time slice deadlines or service units.[1,2] This is not an easy translation to make, since it requires experimentation with different system resource management policies. Identifying the appropriate system resource management policy requires a methodological approach. Such a methodology consists, first, of performance measurement, workload characterization, and performance evaluation, followed by a series of performance tuning experiments. We found that by using such a methodology, we were able to characterize the workload and the behavior of the IBM Research Division's TSS/360 system, then verify that our proposed scheduling change positively improved system responsiveness.

Table 7 Analysis of variance for the results of the 2^2 factorial design experiments after the effects of N_d and N_e have been eliminated

Source of variation	*Degrees of freedom*	*Sum of squares*	*Mean sum of squares*	*F-ratio*
Replacement algorithm	1	1.04	1.04	1.5
Scheduling algorithm	1	5.03	5.03	7.2*
Interaction	1	0.68	0.68	1.0
Error	29	20.1	0.67	

*significant at the 5 percent significance level

ACKNOWLEDGMENTS

This work received the active support and cooperation of W. Doherty, D. Doner, and N. Pass. Our performance-tuning efforts were greatly aided by the numerous discussions we had with M. Ghanem and H. Kobayashi.

CITED REFERENCES

1. W. J. Doherty, "Scheduling TSS/360 for responsiveness," *AFIPS Conference Proceedings, 1970 Fall Joint Computer Conference* **37,** 97–111 (1970).
2. H. W. Lynch and J. B. Page, "The OS/VS2 Release 2 System Resources Manager," *IBM Systems Journal* **13,** 4, 274–291 (1974).
3. R. G. Munck, "A table driven scheduler for widely diverse requirements," *Proceedings of the 1971 IEEE International Computer Society Conference,* 183–184 (1971).
4. H. Katzan, *Operating Systems: A Pragmatic Approach,* Van Nostrand Reinhold, New York (1973).
5. W. R. Deniston, "SIPE: a TSS/360 software measurement technique," *Proceedings of 24th National Conference, ACM,* 229–245 (1969).
6. H. A. Anderson and R. G. Sargent, "Investigation into scheduling for an interactive computing system," *IBM Journal of Research and Development* **18,** 2, 125–137 (1974).
7. H. A. Anderson, G. L. Galati, and M. Reiser, "The classification of the interactive workload for a virtual memory computer system," *Proceedings of Computer Science and Statistics: 7th Annual Symposium on the Interface,* Iowa State University, Ames, 30–40 (1973).
8. G. E. Bryan, "JOSS: 20,000 hours at a console—a statistical summary," *AFIPS Conference Proceedings, 1967 Fall Joint Computer Conference* 31, 769–777 (1967).
9. A. L. Scherr, *An Analysis of Time Shared Computer Systems,* MIT Press, Cambridge (1967).
10. L. Kleinrock, "Certain analytical results for the time-shared processors," *1968 IFIP Congress Proceedings,* 838–845 (1968).
11. A. Sekino, *Performance Evaluation of Multiprogrammed Time-shared Computer Systems,* MAC TR-103 (Project MAC Technical Report), MIT, Cambridge (1972).
12. B. J. Winer, *Statistical Principles in Experimental Design,* 2nd ed., McGraw-Hill, New York (1971).

Reprint Form No. G321-5013