The performance of a complex time-sharing system was monitored under actual operating conditions during a period in which changes in system configuration (both hardware and software) took place. Various techniques for assessing the impact of those changes on performance are discussed.

Performance criteria and measurement for a time-sharing system

by Y. Bard

Characterizing the performance of a time-sharing computer system is a vexing problem. There are two main difficulties: the extremely variable and nonreproducible load that is usually placed on such a system, and the lack of an agreed-upon performance criterion that could be optimized. Because of these factors, it is often difficult for the system designer to ascertain how changes in either hardware or software have affected system performance. It is also difficult to determine what future changes are most likely to improve performance.

Although lacking a clear-cut performance criterion, we intend to describe and evaluate system performance under conditions of variable and mostly unknown loads. By collecting large volumes of data over extended periods, we expect to make up for the variations in load, and by evaluating separately various components that should enter into any reasonable performance criterion, we hope to present the system designer with information on which he can base his decisions, even if we cannot dictate this decision in an objective manner.

The primary performance components discussed in this paper are overhead, average throughput, and maximum throughput. Their utility is demonstrated by application to three different major system configurations, and also to a specific change in a single module. Another primary performance measure, namely response time, is not discussed in this paper.

Table 1 Variables by DUSETIMR

Variable	*Description*	*Unit*
Date		Day, month, year
Time of day		Seconds since midnight
Wait time	CPU time in wait state	Seconds since start up
CP time	CPU time in supervisor state	Seconds since start up
Dispatch**	CP time not charged to any user; mostly dispatch time	Seconds since start up
No. users	Users signed on at moment of measurement	Number
VSIO	SIO instructions issued by users to virtual selector channels; mostly disk and tape I/O operations	Number since start up
VMIO*	SIO instructions issued by users to virtual multiplexer channels; mostly terminal I/O operations and spooling to virtual peripheral devices	Number since start up
RIO	SIO instructions to disks or drums originated by CP; includes paging and spooling I/O operations	Number since start up
P. in	Pages read in	Number since start up
P. out	Pages swapped out	Number since start up
P. steal*	Pages that belonged to users in the active queues and that were overwritten by incoming pages	Number since start up
Free-fret*	See section on free-fret	
User data	For each user signed on at moment of measurement:	
I.D.	User identification	
Tot. time	CPU time accounted to user	Seconds since sign-on
Prob. time	Problem state time accounted to user	Seconds since sign-on

*These variables were measured only in the later part of the study.
**This variable is called "overhead" in CP, but we have not used that name to avoid confusion with the fact that all CP time is overhead.

Data collection and assembly

The data used in this study were collected by monitoring an IBM System/360 Model 67 computer running under the CP-67 time-sharing operating system.[1,2] (CP-67 is described briefly in the Appendix.) A program entitled DUSETIMR collected the data. Running of the program was initiated automatically at nominally five-minute intervals, and the program was run from the operator's virtual machine with its data stored in the operator's disk space. About once a day the data were transferred from the disk on to magnetic tape. The program induces CP (the control program) to supply it with the current values of various counters that CP maintains. These counters are mostly cumulative and are reset to zero only at system start-up. Counters applying to individual users are started each time the user logs on.

Table 2 Derived variables

Variable	*Description*	*Unit*
Δt	Time between successive observations	Seconds
Problem state time	Total time − (CP time + wait time)	Seconds
Page I/O	P. in + P. out	Number
Spool I/O	RIO − Page I/O	Number
Active users	Number of users who have used some problem-state time during the last observation period	Number

Table 3 Data base periods

Designation	*Study period*	*Description*	*Number of data points*
II-2	9/69-11/69	CP version II, 2 core boxes	1,386
II-3	12/69-3/70	CP version II, 3 core boxes	7,890
III-3	5/70-6/70	CP version III, 3 core boxes	10,508

Table 1 contains a list of the recorded quantities. A full description of the data collection method is given by Adair and Bard.[3]

Before being subjected to analysis, these data were augmented with additional variables computed from the primary measured variables. Table 2 lists those derived variables that were found useful in the analysis. In addition, most variables were reduced to a per-second basis by taking the differences between successive measurements and dividing by the time elapsed between them.

Measurements used in this study were taken over a period of ten months. During that time, two major changes occurred in the system configuration: first, more main storage in the form of a third core box was added to the initial two; second, version III of CP-67 replaced version II as the operating system. Many additional small software and hardware changes occurred during this period, but their effects were judged to be negligible compared to the effects of the major changes, and so the data were divided into three subsets as detailed in Table 3. The data within each subset were treated as being homogeneous.

measurements

The increase in number of data points from the first to the third period is due partly to the fact that II-2 consists entirely of first shift measurements, II-3 contains much second shift and some third shift data, whereas around-the-clock operation and recording were the rule under III-3. Since high-load, first-shift data are

of primary interest, the disparity in useful information content between the three sets is much smaller than would appear at first.

CP overhead regression

By "CP overhead" we mean the central processing unit (CPU) time spent by CP in servicing the users' requests for system resources of various kinds, e.g., CPU time, main memory space, and I/O operations of various types. This should not be confused with the measured variable generally called "overhead", which we call dispatch here (see Table 1). Dispatch constitutes only a small part of the total CP overhead.

Our approach to analyzing CP overhead is the following: Let us number the five-minute periods for which we have measurements $\mu = 1, 2, \cdots, n$, and let the measured CP time in the μth period be t_μ. Let the length of the μth period be denoted by T_μ. Suppose we are dealing with m different types of requests that CP must service. Let us designate by $n_{\mu i}$ $(i = 1, 2, \cdots, m)$ the measured number of requests of type i serviced during the μth period. Suppose θ_i is the average time required by CP to service one request of type i. Then the total CP time spent in the μth period is given approximately by

$$t_\mu = \theta_0 T_\mu + \sum_{i=1}^{m} \theta_i n_{\mu i} \tag{1}$$

where θ_0 is the average amount of time spent by CP during each second of real time in performing functions other than those accounted for by the explicitly mentioned requests. In practice, we divide Equation 1 by T_μ to obtain

$$\tau_\mu = \theta_0 + \sum_{i=1}^{m} \theta_i x_{\mu i} \tag{2}$$

where $\tau_\mu = t_\mu / T_\mu$ is given in milliseconds of CP time per second of real time, and $x_{\mu i} = n_{\mu i} / T_\mu$ is the rate per second of requests of type i. It is now possible to estimate $\theta_0, \theta_1, \cdots, \theta_m$ by fitting Equation 2 to the observed values of τ_μ. This is done by the method of least squares, i.e., we determine $\theta_0, \theta_1, \cdots, \theta_m$ so as to minimize the sum of squares of the residuals

$$S(\theta_0, \theta_1, \cdots, \theta_m) = \sum_{\mu=1}^{n} r_\mu^2 \tag{3}$$

The μth *residual* r_μ is the "unexplained" portion of τ_μ, i.e.,

$$r_\mu = \tau_\mu - (\theta_0 + \sum_{i=1}^{m} \theta_i x_{\mu i}) \tag{4}$$

goodness of fit

If the r_μ are relatively small, then Equation 2 is considered to give a good fit to the data. An objective criterion for judging the

goodness of fit is obtained by comparing the sum of squares of the residuals to the sum of squares of the deviations of the measurements from their average value. Let $\bar{\tau}$ be the average value of the τ_μ. Then our goodness-of-fit criterion is R^2, the square of the multiple correlation coefficient

$$R^2 = 1 - \left\{ \left[\sum_{\mu=1}^{n} r_\mu^{\,2} \right] \Big/ \left[\sum_{\mu=1}^{n} (\tau_\mu - \bar{\tau})^2 \right] \right\} \tag{5}$$

Clearly, if Equation 2 fits the data perfectly, then all residuals vanish and $R^2 = 1$. Conversely, if the variables $x_{\mu i}$ contribute nothing to the understanding of τ_μ, then 2 degenerates simply into $\tau_\mu \simeq \theta_0 = \bar{\tau}$, and hence, $r_\mu = \tau_\mu - \bar{\tau}$ and $R^2 = 0$. Thus, if the assumptions under which we derived Equation 2 are at all valid, then we expect to find a value of R^2 close to 1.

Another quantity of interest in judging the goodness of fit is the *standard deviation of the residuals*, i.e., the root mean square

$$s = \left(\frac{\sum_{\mu=1}^{n} r_\mu^{\,2}}{n - m} \right)^{\frac{1}{2}} \tag{6}$$

The smaller s is, the better the fit of the model to the data. In attempting to fit the CP time data, we found that most residuals were reasonably small, but some were very large in magnitude compared to s. Observations that have relatively large residuals are called *outliers*, and their presence indicates either gross errors in the measurements, or the existence of important unmeasured CP activities. It is interesting to note that almost all CP time outliers had large *positive* residuals, indicating that CP time was spent in unaccounted-for activities. Following standard statistical practice, we dropped the outlying observations from the analysis, and Equation 2 was refitted to the remaining observations. The criterion for dropping an observation was that $|r_\mu|$ exceeded $2.5s$.

regression procedure

The crucial question that arises when Equation 2 is fitted to the data is which variables $x_{\mu i}$ should be chosen for inclusion in the equation. In our analysis, we were guided by the stepwise-forward selection regression procedure: First choose the variable capable of giving the greatest reduction in the sum of squares S; then add the variable capable of giving the greatest additional reduction and so on until no further significant reduction can be achieved. The details of the computations for this, as well as for all the other model-fitting calculations, can be found in standard texts on regression analysis.[4]

A forward selection regression program was applied to various subsets of the data. In all cases, it turned out that the program selected only variables included in the following list:

Table 4 Data set III-3 coefficients—effect of VMIO omission

	Without VMIO	*With VMIO*	*Unit*
θ_0	8.9	8.9	msec of CP time per sec of real time
θ_1	8.1	7.9	msec of CP time per VSIO
θ_2		2.6	msec of CP time per VMIO
θ_3	2.2	2.0	msec of CP time per page I/O
θ_4	16.5	4.6	msec of CP time per spool I/O
θ_5	138.0	134.1	msec of CP time per sec of problem state time

- VSIO (virtual selector I/O operations)
- VMIO (virtual multiplexer I/O operations)
- Page I/O operations
- Spool I/O operations
- Problem state time

The final analysis of the entire data was performed by including in the regression these variables and no others.

The variable VMIO was available only in the III-3 data set. The effect of not including this variable was tested by fitting the III-3 data both with and without VMIO. The coefficients are shown in Table 4. It is evident that omission of VMIO grossly distorts the spool I/O coefficients and slightly affects the others. For this reason, an adjustment was made to the coefficients for data sets II-2 and II-3 on the assumption that the inclusion of VMIO would have affected them in the same ratio as in III-3. The original and adjusted coefficients, as well as the number of dropped outliers and the value of the goodness-of-fit criterion R^2, are given in Table 5.

results

The estimated θ_i would possess various desirable statistical properties if the usual regression-analysis assumptions were fulfilled. These assumptions are:

1. The coefficients θ_i are constants.
2. The $x_{\mu i}$ are measured exactly.
3. The observed τ_μ satisfy the equation

$$\tau_\mu = \theta_0 + \sum_{i=1}^{m} \theta_i x_{\mu i} + \epsilon_\mu \tag{7}$$

where the ϵ_μ ($\mu = 1, 2, \cdots, n$) are independent, identically distributed, random variables with zero means.

If these assumptions were satisfied, then not only would the least-squares method provide estimates for the θ_i, but it would also give confidence limits that measure the accuracy of the estimates. This information would enable us to predict the

amount of variation to be expected in the estimates obtained from different samples taken from the same system configuration, and to assess the significance of differences in the estimates obtained from samples taken under different system configurations.

Unfortunately, assumptions 1 and 3 do not hold in our case for two reasons. (1) The time taken by CP to service a VSIO, for instance, depends on the complexity and length of the required I/O operation. Similarly, the time per each page I/O operation depends on how long CP has to search for the page to be overwritten. (2) Since not all functions performed by CP have been accounted for by the measured activities, the ϵ_μ tend to have positive means. Furthermore, their distribution changes from one time period to the next, depending on the nature of the particular work load present.

In view of the invalidity of the usual regression assumptions, it is not surprising that the regression model does not behave in the text-book manner. The presence of predominantly positive outliers has already been mentioned. More serious is the result that when each data set is broken up into smaller subsets, and each subset is fitted separately by the least squares method, the estimated coefficients show considerable variation, much greater than predicted by the regression model. Hence, we are at present unable to make meaningful statements concerning the accuracy of the estimated coefficients, nor are we able to test whether or not separate estimates for the same coefficients are significantly different. The following remarks are perforce of a subjective nature. With the aid of additional measurements, we expect that more realistic statistical models will be constructed from which more objective deductions can be made.

coefficients

Since II-2 and II-3 represent the same version of CP, it can be expected that the coefficients would be about the same in both periods. As shown in Table 5, this is indeed the case for θ_1 and θ_3. Because VMIO is not measured during these periods, and because of the large effect of VMIO on spool I/O operations, the changes in θ_4 between II-2 and II-3 need not be taken seriously. The large changes in θ_0 and θ_5 are somewhat mystifying. However, it should be remembered that θ_0 and θ_5 represent rather ill-defined functions of CP, and changes in these coefficients may be due simply to changes in the user environment.

As expected, all III-3 coefficients are smaller than their II-3 counterparts, reflecting the improved efficiency of the coding in version III of CP. This subject will be further discussed in the section on the free-fret module.

The coefficients of Table 5 represent not only the average time taken by CP to service a request of a certain kind, but also other

Table 5 Results of CP-time regression analysis*

Data Set	*Number of rejected outliers*	*Number of remaining points*	R^2	θ_0	θ_1	θ_2	θ_3	θ_4	θ_5
II-2**	33	1353	0.88	26.2	10.8	–	6.6	18.1	25.0
II-3**	238	7652	0.90	12.5	10.0	–	6.6	21.5	186.7
III-3**	253	10255	0.86	8.9	8.1	–	2.2	16.5	138.0
II-2†	–	–	–	26.2	10.4	?	6.1	5.1	24.0
II-3†	–	–	–	12.5	9.7	?	6.1	6.0	181.3
III-3††	226	10282	0.88	8.9	7.9	2.6	2.0	4.6	134.1

*Units of θ_i designated as in Table 4
**VMIO not included
†Coefficients adjusted for estimated effect of VMIO
††VMIO included

overhead associated with this request. For instance, when a user requests a page, it is necessary for CP not only to fetch the page, but it must also find another user who is runnable and dispatch him. In an attempt to isolate this extraneous time, an equation similar to 2 was fitted to the dispatch time data of period III-3 (the only one for which this variable was measured). The estimated coefficients, designated $\phi_0, \phi_1, \cdots, \phi_5$, are given in Table 6. Also affecting the value of each θ_i is the time spent by CP in servicing functions on which no measurements were taken, provided such functions were often correlated with the measured ones.

The differences between dispatch times associated with different CP functions arise from the following two reasons: (1) the number of times CP has to go through dispatch as a result of each function is variable; (2) the fraction of times that CP may redispatch the current user after performing some function varies. The time required for redispatching the current user is much smaller than the time required for finding a new user. The exact interpretation of the values in Table 6 must await further data.

servicing time

It is of interest to determine what fraction of its total time CP spends on the average in servicing the various functions. Let $\bar{x}_i$ be the average value of the $x_{\mu i}$, i.e., $\bar{x}_i = (1/n) \sum_{\mu=1}^{n} x_{\mu i}$. Then $\theta_i \bar{x}_i$ is the average number of milliseconds CP spends in function i per second of real time, and $100\ \theta_i(\bar{x}_i/\bar{\tau})$ is the average percentage of CP time spent in this function. The relevant data appear in Tables 7 and 8. Note that the progressive decreases in the averages of Table 7 are due to increasing amounts of second and third shift data for periods II-3 and III-3. From Table 8 it is evident that servicing VSIO requests is by far the most time-consuming activity that CP engaged in, at least at the installation under discussion.

Table 6 Dispatch regression coefficients for III-3 data

	Coefficients	*Unit*
ϕ_0	0.7	msec of dispatch time per sec of real time
ϕ_1	0.6	msec of dispatch time per VSIO
ϕ_2	0.3	msec of dispatch time per VMIO
ϕ_3	0.7	msec of dispatch time per page I/O
ϕ_4	1.1	msec of dispatch time per spool I/O
ϕ_5	1	msec of dispatch time per sec of problem state time

Table 7 Average request rates

	Data set			*Unit*
	II-2	II-3	III-3	
$\bar{\tau}$	0.254	0.201	0.120	sec of CP time per sec of real time
$\bar{x_1}$	11.2	9.6	8.7	VSIO per sec
$\bar{x_2}$	*	*	4.5	VMIO per sec
$\bar{x_3}$	10.8	4.8	3.3	page I/O per sec
$\bar{x_4}$	1.04	1.02	0.65	spool I/O per sec
$\bar{x_5}$	0.212	0.210	0.160	sec of problem state time per sec of real time

*Not measured

Table 8 Percentage of CP time devoted to various request types

Data set	*VSIO*	*VMIO*	*Page I/O*	*Spool I/O*	*Problem state time*
II-2	46	*	26	2	2
II-3	46	*	15	3	19
III-3	57	10	6	3	18

*Not measured

Table 9 Percentage of CP time devoted to various request types with dispatch time separated for III-3 data

VSIO	*VMIO*	*Page I/O*	*Spool I/O*	*Problem state time*	*Dispatch*
52	9	4	2	17	8.5

In Table 9, we have removed the dispatch time from the various requests in the III-3 data, i.e., we have replaced θ_i with $\theta_i - \phi_i$ in computing the percentages. In addition, we report the percentage of CP accounted as dispatch in the III-3 period.

Table 10 Number of observations in different groups

Number of Active users	*Data set* II-2	II-3	III-3
1,2	35	1082	2049
3,4	43	1413	3598
5,6	74	820	1485
7,8	135	802	753
9,10	162	727	569
11,12	220	838	534
13,14	247	873	514
15,16	207	653	463
17,18	158	383	263
19,20	70	168	148
21,22	26	82	85
23,24	9	38	34
25,26	0	10	10
27,28	0	0	3

Average throughput as function of active users

The throughput of the system may be loosely defined as the amount of work performed per unit time. The amount of work itself is a vector with several components. Some of these represent "useful" work, i.e., work directly requested by the users, e.g., VSIO, VMIO, and problem state time. As a general rule, changes in the system are considered desirable if useful throughput is increased, or if the overhead required for a given amount of useful throughput is decreased.

Overall average throughputs for the three periods in question have already been given in Table 7. These figures do not, however lend themselves to meaningful comparisons because they represent system response to widely varying loads. One cannot blame version III of CP for servicing, on the average, fewer VSIO's than version II, when this is due purely to the fact that fewer VSIO's were requested. What is needed, then, is some measure of the load placed on the system. Among the available variables, the only one that seems to be a measure of load, rather than of response, is the number of signed-on users, or, better yet, the number of active users as defined in Table 2. Our procedure was to group the available observations by number of active users, and compute separate averages for each group. Table 10 shows the number of observations available in each group, and Figures 1 through 5 plot average problem state time, CP time, wait state time, VSIO, and the page I/O operation rates as functions of the number of active users. No plots were made for VMIO because of lack of data, nor for the spool I/O operation because of its meager contribution to overhead (see Table 8). Because of the rather small number of observations with more

Figure 1 Average problem state time

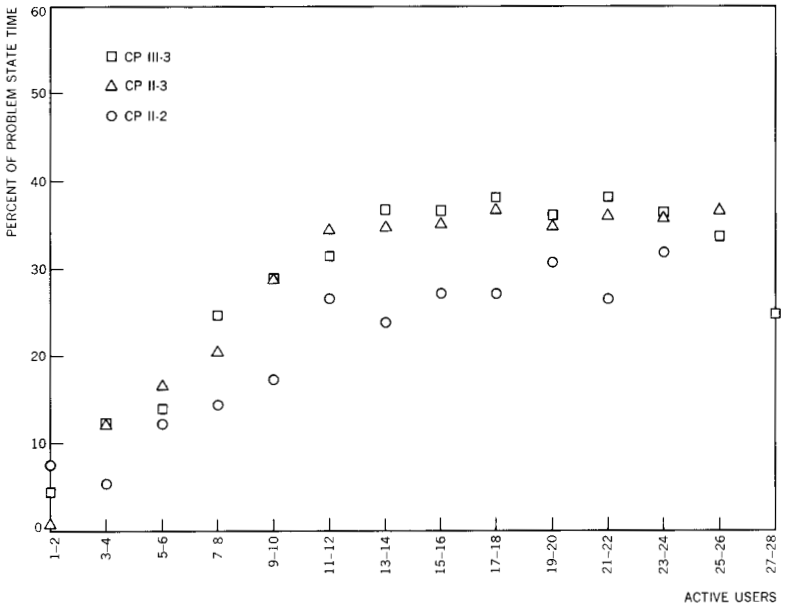
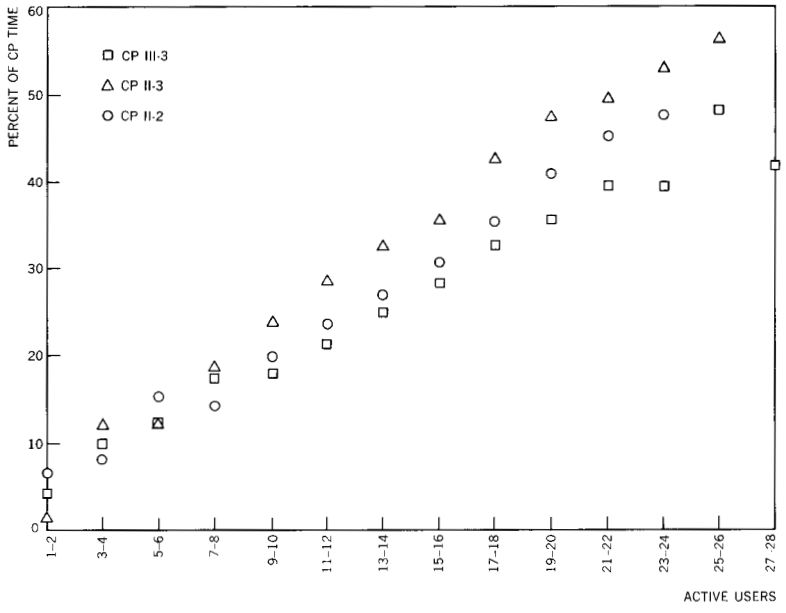


Figure 2 Average CP time

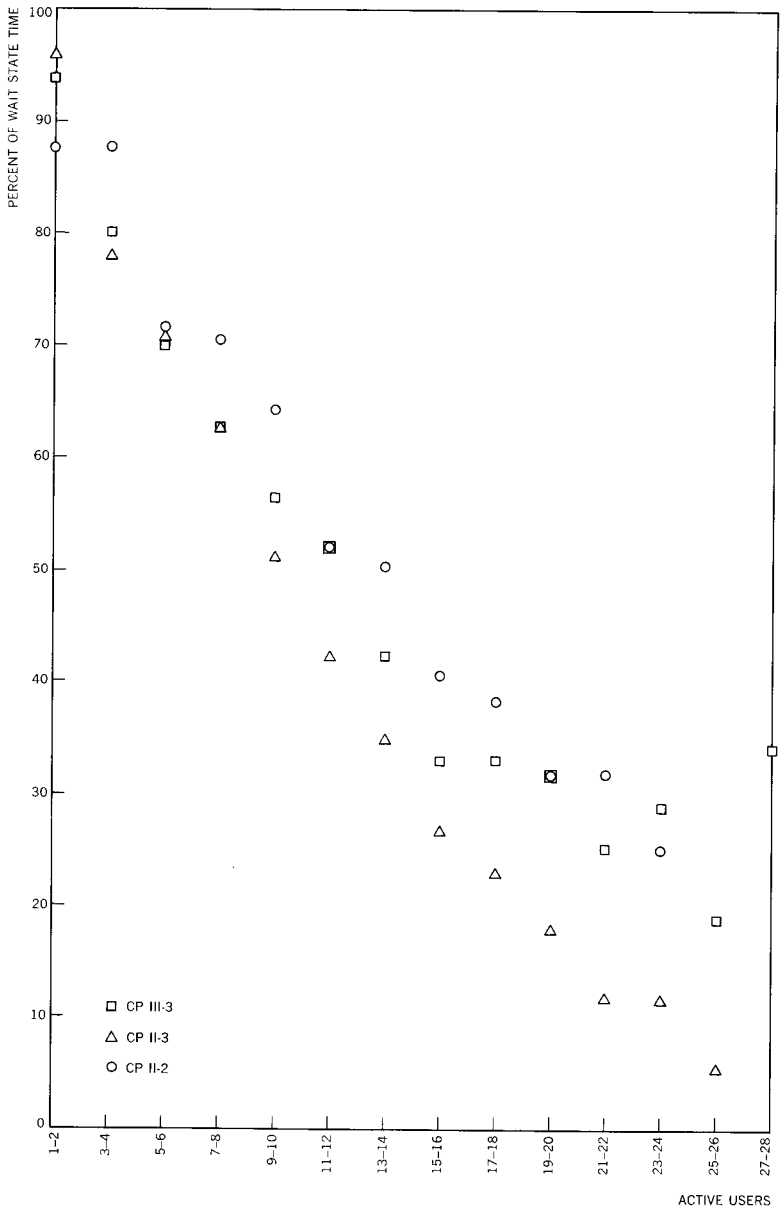


than 22 active users, not much reliance can be placed on the high ends of the curves.

study of figures

Certain facts emerge from a study of Figures 1 through 5. They are discussed in the following paragraphs.

Figure 3 Average wait state time



Figures 1 and 4 show that a definite increase in useful throughput was realized by going from two to three core boxes. This is presumably due to the fact that jobs had to spend less time waiting for pages, since more pages could be held in main storage at one time. There is no conclusive evidence to show whether average useful throughput has been increased by going from version II to version III.

Figure 4 Average VSIO rate

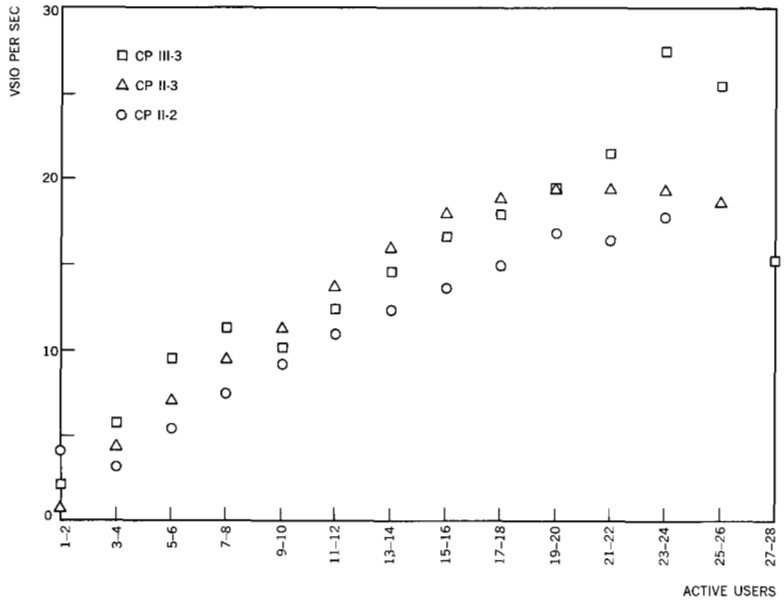


Figure 2 shows that the increase in useful throughput in going from II-2 to II-3 caused a corresponding increase in overhead. On the other hand, the more efficient coding of version III caused a drastic reduction in overhead when going from II-3 to III-3.

Figure 3 demonstrates what could have been concluded from Figures 1 and 2. Namely, in going from II-2 to II-3, increased problem state and CP times caused a reduction in wait state, and in going from II-3 to III-3, decreased CP time combined with unchanged problem state time result in increased wait state time.

Figure 5 shows how the paging rate increases exponentially with the number of active users. As expected, a decrease in paging is experienced when the core box is added. If the throughput had otherwise remained unchanged, the decrease in paging would have been much more noticeable. In practice, the system was able to increase useful throughput (problem state time and VSIO) while still maintaining a reduced paging rate.

Saturation

In the preceding section, we have examined the average system throughput under varying load conditions. It is shown that the reduction in overhead due to going from version II to version III

Figure 5 Average paging rate

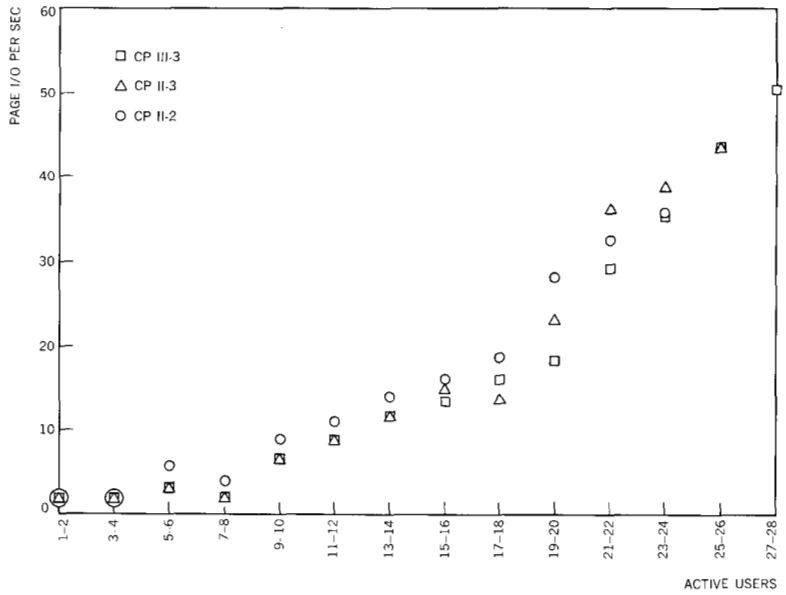
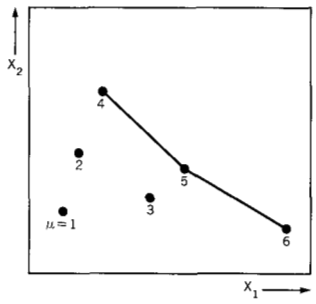


Figure 6 Illustration of Pareto-maximality



has not, on the average, resulted in increased throughput. This is a reflection of the fact that most of the time, system resources were not taxed to the limit; hence, reduction of overhead merely increased the slack (wait time). In principle, however, a system with lower overhead should be able to provide greater throughput when the load approaches the limiting capacity of the system. To verify this proposition, we attempted to isolate those observations that appeared to represent conditions of maximal throughput.

Since throughput is measured by several variables simultaneously, it is not meaningful to talk of *maximal throughput per se*. It is not very interesting to know that, say, the maximum observed problem state time was 95 percent. What we are interested in are conditions of *saturation*, in which the rate of one throughput cannot be increased further except at the expense of some other throughput. Suppose we have m throughput variables, whose values at the μth observation are $x_{\mu 1}, x_{\mu 2}, \cdots, x_{\mu m}$. The notation $\mathbf{x}_\mu$ denotes the vector whose components are these values. Each observation can be regarded as a point in m-dimensional Euclidean space. All the points fall in a certain feasible region of this space; a hypothetical collection of six points for $m = 2$ is depicted in Figure 6. Now we say that point μ *dominates* point ϕ if no component of $\mathbf{x}_\phi$ exceeds the corresponding component of $\mathbf{x}_\mu$, i.e., if $x_{\phi i} \leq x_{\mu i}$ for $i = 1, 2, \cdots, m$. In Figure 6, point 1 is dominated by 2, 3, 4, and 5, point 2 by 4, and point 3 by 5. Those points that are not dominated by any other points (4, 5, and

6 in Figure 6) are called *Pareto-maximal points*. These are the points that are of interest to us: if $\mathbf{x}_\mu$ is Pareto-maximal, then we know that at no time did we observe throughputs that exceeded $\mathbf{x}_\mu$ in all components; any increase in some components of $\mathbf{x}_\mu$ was accompanied by a decrease in at least one other component. Any observations taken during periods of system saturation must be Pareto-maximal points. In principle, a Pareto-maximal point need not be a saturation point; it may simply happen that during the period of observation the load placed on the system did not happen to reach saturation conditions. However, when many observations are available, it is likely that the Pareto-maximal points (particularly when they fall on a fairly smooth curve) do indeed represent conditions close to saturation. This supposition was further verified as follows:

If no saturation occurs, then one expects the curves joining the Pareto-maximal points to shift outwards gradually as the number of observations is increased. In fact, it was found that as new points were encountered, these tended to fill out the curves joining the old points, without much affecting the position of the curves.

Note that since observations were taken at about five-minute intervals, only saturation periods lasting at least five minutes can be detected in the data. To obtain fuller information on saturation conditions, it is necessary to take observations more frequently. Note also that it is no doubt possible to concoct job mixtures under which the system will sustain loads heavier than observed in our measurements.

The algorithm discussed here selects the Pareto-maximal points. We process the observations $\mathbf{x}_\mu$ ($\mu = 2, 3, \cdots, n$) one at a time, and at the time of this processing, we have already built up a list of candidates that is the collection of Pareto-maximal points relative to the first μ-1 observations. Initially, the list of candidates consists of the observation $\mathbf{x}_1$. We compare $\mathbf{x}_\mu$ to all the candidates. If some candidate dominates $\mathbf{x}_\mu$, we proceed to $\mathbf{x}_{\mu+1}$. Otherwise, $\mathbf{x}_\mu$ is added to the list of candidates, and all previous entries which are dominated by $\mathbf{x}_\mu$ are deleted. After the last observation has been processed, the list of candidates is identical to the set of Pareto-maximal points.

analysis

We chose the three variables, percentage of problem state time, page I/O per second, and VSIO per second, for the Pareto-maximal analysis. Since it is difficult to present three-dimensional results graphically, three separate analyses were carried out on the three pairwise combinations of these variables, and the results are plotted in Figures 7 through 9. To avoid confusion, smooth curves were fitted by eye to the Pareto-maximal points. The three-dimensional picture can be reasonably approximated by

Figure 7 Saturation: Pages vs problem state time

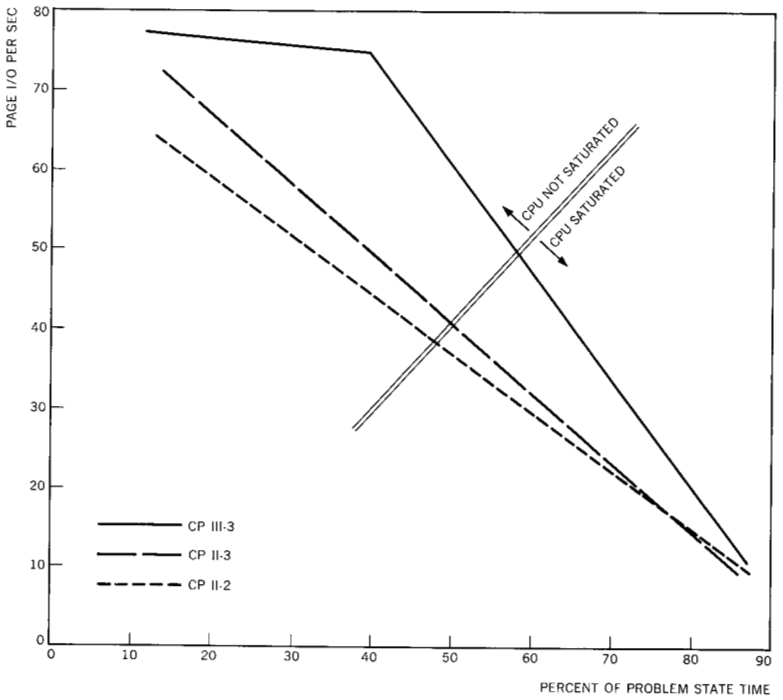


Figure 8 Saturation: VSIO vs problem state time

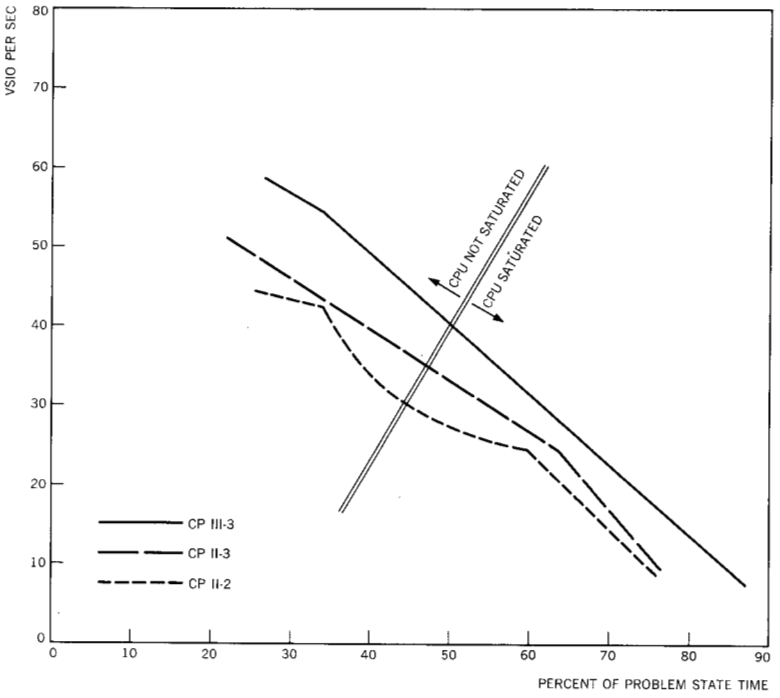
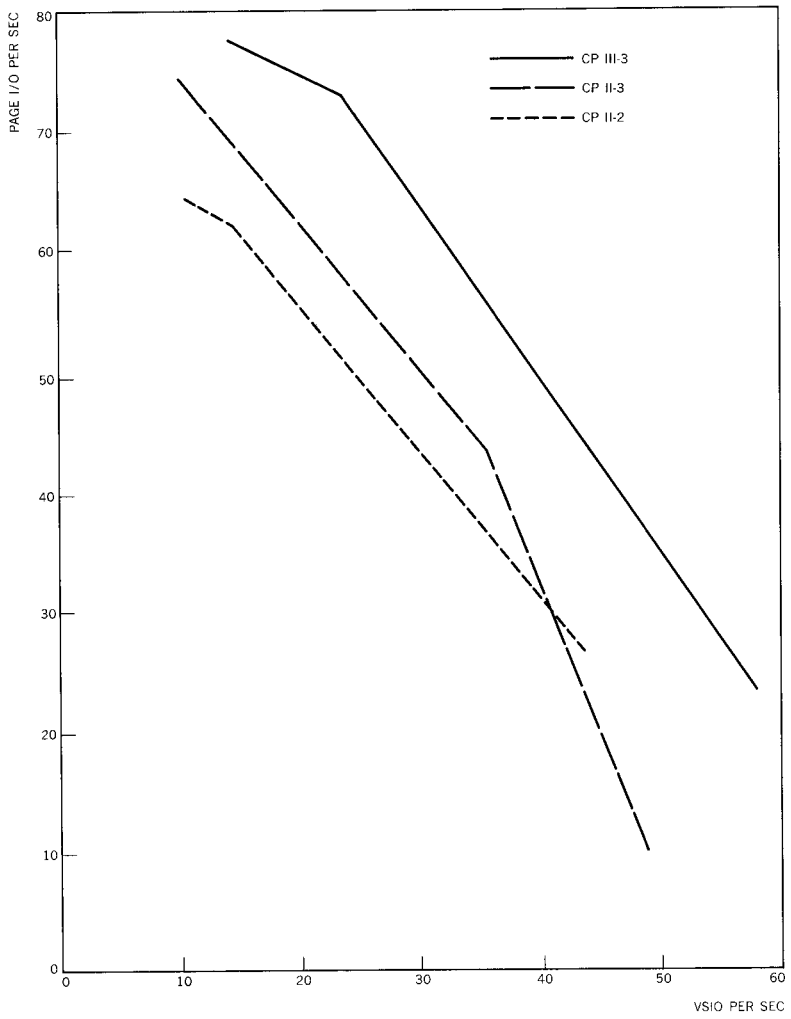


Figure 9 Saturation: Pages vs VSIO



constructing nearly planar surfaces whose intersections with the coordinate planes are the curves of Figures 7 through 9.

Figures 7 through 9 show that version III of CP has consistently provided higher peak-load throughput rates than version II. In all cases, the "frontier" of the observed operating region was pushed outwards. We are therefore tempted to conclude that version III can indeed sustain higher loads than version II before becoming saturated.

CP merely services VSIO requests as they come along. But through its dispatch and paging algorithms, CP exerts more direct control over problem state time and page I/O operations. Therefore, it is

particularly interesting to study Figure 7 which shows the peak load relationship between these two variables. The III-3 curve can be broken into three parts: (1) When problem state time is less than 40 percent, the paging rate is about constant at 75 to 80 pages per second. This is close to the average maximum paging rate possible on the drum under the paging algorithm used by these versions of CP. Hence, we conclude that the limiting factor here is the paging mechanism. (2) By referring to the DUSETIMR records, we found that in all Pareto-maximal points falling in the region to the right and below the double line in Figure 7, CP accounted for all the remaining CPU time, with practically no wait state time observed. Hence, CPU capacity is the limiting factor in this region. If CP coding were made even more efficient, performance could perhaps be improved in this region. (3) When problem state time is between 40 and 55 percent, neither the CPU nor the paging mechanism is saturated. It seems, therefore, that the dispatch and paging algorithms of CP may be responsible for the slack, and that revision of these algorithms could result in increased throughput.

Under version II of CP, paging saturation never occurred (at least not over five-minute periods), and CPU saturation occurred when problem state time exceeded 50 percent.

When we begin to assess saturation by VSIO requests (Figures 8 and 9), we are hampered by the fact that these are quite variable in length, complexity, and distribution among available channels and devices. Therefore, a simple count of requests does not permit us to tell whether or not I/O channel capacity was being taxed at any given time. Figure 8 does show, however, the region of CPU saturation where no appreciable wait time was observed. CPU saturation occurs at a somewhat lower level of percentage of the problem state when VSIO rate is high than when page I/O rate is high, undoubtedly because CP time per VSIO is larger than CP time per page I/O (see Table 5).

The information contained in the saturation curves could benefit system users by informing them of trade-off rates at saturation between the different resources. These can be estimated by taking the negative slopes of the saturation curves. Approximate rates are as follows:

11 milliseconds problem state time = 1 page (CP II) or
1.4 pages (CP III)
12 milliseconds problem state time = 1 VSIO
3 pages = 2 VSIO

We will use an example to show how these figures should be interpreted. Suppose a programmer wants to rewrite a program in such a way that fewer VSIO's will be required but at the expense of increased CPU time. For instance, intermediate results may be

recomputed whenever needed instead of being written out on secondary storage and later retrieved. If the increase in CPU time is less than 12 milliseconds for each VSIO eliminated, then the performance of this program under heavy load conditions can be expected to improve (i.e., more such programs will be runnable in a given time period). The application of these principles to changes in page requirements is less straightforward since it is difficult to predict these requirements in an arbitrary environment.

Free-fret module

One of the goals of any measurement technique is to evaluate the effects caused by specific changes in the system. These changes are usually not as gross as the addition of a core box or the installation of a new version of CP, and the task of determining their effects is accordingly more difficult. As a test of what can be done in this area, we attempted to assess the effects of changes made in the free-fret module of CP.

The free-fret module manages the free storage areas required by CP. When CP is called upon to perform some transaction (e.g., read in a page), it requires a few words of temporary storage and calls on the FREE routine, which allocates the required storage area from a storage pool. When CP completes the transaction, it calls on the FRET routine to return the storage area to the pool. An investigation by Margolin et al.[5] showed that it was possible to improve considerably on the free-fret algorithm included in version II of CP, and the new algorithm was included in version III. The effect of this change was thereby confounded with the effects of other changes incorporated in version III.

To isolate the effects of the change in the free-fret algorithm, an experiment was run over a two-week period. On Monday and Thursday of the first week, and Tuesday and Wednesday of the second week, the computer was run under the normal version III of CP. On Tuesday and Wednesday of the first week, and Monday and Thursday of the second week, the computer was run under version III modified to include the old free-fret algorithm. This design was selected so as to balance out any changes in load between the first and second week, or between different days in the same week. Data specific to the free-fret module were collected. Obtained among other things were counts of all calls to FREE and FRET and the time spent in executing these subroutines. These data will be reported on elsewhere.[5] Suffice it to say here that the average CPU time spent in servicing a FREE call went down from 363 (old) to 47 (new) microseconds, the average FRET, from 245 to 45 microseconds, and the fraction of total CP time spent in free-fret decreased from 16.5 to 2.9

Table 11 Effect of free-fret algorithm on CP coefficients

	CP version II*	CP version III*	CP version III†	Unit
θ_1	9.7	9.7	7.9	msec of CP time per VSIO
θ_2		2.3	2.6	msec of CP time per VMIO
θ_3	6.1	3.0	2.0	msec of CP time per page I/O
θ_4	6.0	5.7	4.6	msec of CP time per spool I/O

*Old free-fret algorithm
†New free-fret algorithm

Table 12 Average throughputs during free-fret test period

Variables	*Old free-fret*	*New free-fret*
Percentage of problem state time	19.8	22.0
Percentage of CP time	21.3	18.1
VSIO per sec	11.7	12.4
VMIO per sec	8.5	8.5
Spool I/O per sec	1.1	1.2
Page I/O per sec	6.7	7.0

percent.[6] Our primary concern here is with the question of how these changes affected overall system overhead and performance.

A regression analysis similar to the one discussed earlier was carried out on the data for version III running with the old free-fret algorithm. The results are shown in Table 11. For comparison we have included from Table 5 the results pertaining to II-3 and III-3.

It is immediately evident that practically the entire improvement in version III over II in the handling of VSIO and spool I/O operations is due to the new free-fret algorithm. In the handling of page I/O operations, other changes in the coding contribute to a 50 percent improvement and the free-fret algorithm contributes an additional 16 percent.

The volume of data available was insufficient to follow the procedures previously described in the sections on average throughput and saturation. However, Table 12 shows the average throughputs under the two algorithms during the eight-day test period. As shown in Table 12, there is an increase in throughput under the new algorithm, but we do not have as yet a sufficiently good statistical model that would permit us to determine whether or not this increase is significant.

The results of the free-fret experiment were used to verify the θ_i estimates in the following manner. We consider an equation similar to Equation 1, but instead of t_μ, we put in N_μ, the number of calls to FREE in the μth period. (The number of FRET calls is almost identical.) The same regression procedure is used to estimate the coefficients, whose meaning now is "number of FREE calls per VSIO", etc. It turned out that the number of calls per VSIO, VMIO, page I/O, and spool I/O operations were, on the average 3.5, 0, 2, and 2, respectively. Since the saving per pair of calls is $363 + 245 - 47 - 45 = 516$ microseconds, it follows that the time per VSIO, VMIO, page I/O, and spool I/O operation should be reduced by 1.8, 0, 1.0, and 1.0 milliseconds, respectively. Table 11 shows reductions of 1.8, −0.3, 1.0, and 1.1 seconds, in excellent agreement with the predicted results.

Summary comment

We conclude from the discussions in this paper that measurements taken over a sufficiently long period on a system in full operation can lead to a meaningful assessment of system performance under various configurations. For instance, we have shown that addition of a core box has increased average throughput and that reduction in overhead by recoding some CP modules has resulted in increased peak-load capacity.

At present, rather long observation periods are required, but the following steps should increase the sensitivity of our techniques, and hence reduce the amount of data needed:

1. Decrease the time between measurements at peak use times. The chance of a saturated condition lasting one minute is much higher than its lasting five minutes.
2. Establish statistical models that better account for the variability in the data and that enable us to test the significance of observed changes from one system configuration to the next.
3. Obtain more detailed and extensive measurements. For instance, we should not only count VSIO's but also measure the amount of data transmitted, channel utilization, and channel queue lengths.

Some of these steps are being implemented currently and will subsequently be reported in future papers.

The data and statistical estimates derived from the parameters, which reflect the performance of a certain CP-67 system, may not be representative of the performance of other CP-67 installations. In particular, the system measured was an interim development system that experienced many changes during the measurement period and did not reflect any official release. A number of per-

formance improvements included in version III were not incorporated in this system during the measurement period.

ACKNOWLEDGMENT

The author wishes to thank other members of the IBM Cambridge Scientific Center for their assistance and participation in the project. In particular, B. H. Margolin, T. I. Peterson, and M. Schatzoff made many contributions to the project, and R. J. Adair and R. Parmelee have been very helpful in explaining the workings of CP-67 and contributing the data-gathering programs, Adair especially, for having written the DUSETIMR program. Also, thanks are due to members of the group under R. A. Meyer, and of the group under M. Fleming, for their cooperation in the running of the free-fret experiment.

A paper on similar material was presented at the ACM SIGOPS Workshop on System Performance Evaluation in Cambridge, Massachusetts, April 5–7, 1971.

CITED REFERENCES AND FOOTNOTE

1. R. A. Meyer and L. H. Seawright, "A virtual machine time-sharing system," *IBM Systems Journal* 9, No. 3, 199–218 (1970).
2. *CP-67/CMS Version 3 System Description Manual*, Form No. GH20-0802-1, International Business Machines Corporation, Data Processing Division, White Plains, New York (1970).
3. R. Adair and Y. Bard, *CP-67 Measurement Method*, Report No. G320-2072. International Business Machines Corporation, Cambridge Scientific Center, Cambridge, Massachusetts (1971).
4. N. R. Draper and H. Smith, *Applied Regression Analysis*, John Wiley and Sons, New York, New York (1966).
5. B. H. Margolin, R. Parmelee, and M. Schatzoff, private communication.
6. These figures include the time required for collecting the data. Hence, in the absence of measurements, the reduction in the time spent by CP in free-fret would be from 14.6 to 2.3 percent.
7. *CP-67 Program Logic Manual*, Form No. GY20-0590-0, International Business Machines Corporation, Data Processing Division, White Plains, New York (1970).

Appendix: Description of CP-67

To further the understanding of those readers who are not familiar with CP-67, we give a brief overview of the system. The following description applies to versions II and III.

CP-67 is a control program that manages the resources of an IBM System/360 Model 67 in a time-sharing environment.[1] It creates for each user a virtual machine that (except for timing considerations) appears to the user as a complete stand-alone System/360. The user's terminal serves both as the operator

console, or panel, of the virtual machine and as an I/O device attached to a virtual multiplexer channel. The user may employ any appropriate operating system (OS, CMS, etc.) to run his virtual machine. The configuration of the virtual machine (i.e., main storage size, I/O devices, etc.) is established by means of entries in the user directory, and bears no relationship to the configuration of the real machine on which CP is running.

principal functions

CP has six principal functions. One of those functions is to schedule CPU time (dispatch), i.e., determine which user should run at any given time. CP selects a subset of users who are assigned to queues. Queue sizes may not exceed fixed limits, which usually depend on the size of real main storage. A runnable user is selected from one of the queues and is given control of the CPU. He will run until his time slice is up, until he causes a paging exception (see below), until he attempts to execute a privileged operation (e.g., I/O), or until some other interruption occurs. CP will then attempt to satisfy whatever request the user is making and dispatches another user (or possibly the same one if still runnable). When a user in a queue has accumulated a certain amount of CPU time, he is dropped from the queue, and some other user with currently higher priority is admitted in his place. Priorities are set so that the user waiting the longest usually has the highest priority.

Another function allocates main storage (paging). Main storage is divided into 4096-byte pages. When a running user refers to one of his pages that is not currently in main storage, a paging exception occurs. The user is interrupted, and CP brings the required page into main storage from the backup store (usually a magnetic drum). This page must be written over some page currently in main storage. CP first attempts to overwrite a page not belonging to a user in a queue; if no such pages can be found, a page is "stolen" from a user in a queue. If the page to be overwritten has been changed since it was last brought in, then it must be written out on the back-up store before the new page can be brought in. Thus, a paging exception will sometimes cause two page I/O operations (page out followed by page in), and sometimes one page I/O operation (page in only).

A third function interprets and simulates virtual I/O operations. CP intercepts all attempts by virtual machines to execute I/O operations. CP translates these operations from the virtual devices recognized by the users to the real devices attached to the machine. It then executes the required operations and returns to the user the appropriate condition codes.

A fourth function performs I/O spooling operations. CP will read card decks through the real card reader and file them in disk areas, where they can be found by the users' virtual card readers.

Similarly, CP stores on disk the output from the users' virtual printers and punches and eventually transfers it to the real output devices. Upon request, CP can transfer one user's virtual punch output to another user's virtual reader.

Handling privileged instructions of virtual machines is a fifth function. When a virtual machine issues a privileged instruction, CP gains control. It then decides whether the instruction should be interpreted and simulated (as in the case of I/O instructions) or reflected to the virtual machine's operating system.

A sixth function performs miscellaneous operations including bookkeeping, directory maintenance, and various other service functions.

For further details the reader is referred to the CP-67 Program Logic Manual.[7]