A STUDY IN OPERATING SYSTEM PERFORMANCE MEASUREMENT AND MODELING

by

David S. Lindsay and Domenico Ferrari

Memorandum No. ERL-M550

13 October 1975

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A STUDY IN OPERATING SYSTEM PERFORMANCE MEASUREMENT AND MODELING[†]

David S. Lindsay

Federal Computer Performance Evaluation
and Simulation Center (FEDSIM)
Washington, D.C.

and

Domenico Ferrari

Computer Science Division
Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California at Berkeley
Berkeley, California 94720

October 1975

Abstract

This paper develops an analytic model of a CDC 6400 running under the CALIDOSCOPE operating system. The model's calibration with experimental data and its use to predict performance improvements are also described. The model calculates the mean throughput rate of the system as a function of the detailed I/O behavior of the jobs being processed. In particular, the model considers the possibility of each job overlapping its computations with buffered I/O. Despite the model's relative simplicity, it has proven quite accurate in predicting the throughput rate of a synthetic job stream under several different system I/O management policies.

[†]The research reported here was supported in part by the Computer Center of the University of California at Berkeley, and by National Science Foundation Grant Number DCR74-18375.

1. Introduction

In spite of the surprisingly fast progress made by analytic modeling techniques for the evaluation of computer system performance in the very recent past, there are still some aspects of system behavior which are not easy to model. One of these aspects is buffered I/O, which allows a program to overlap its computation with the execution of the I/O operations it has initiated. This feature, which is found in several systems and may have a non-trivial influence on their performance, cannot be easily represented in a queueing model, in which a customer being served at a service center cannot at the same time be queued up or receiving service at another center.

This paper describes a simple probabilistic, but not queueing, model which yields the mean throughput rate of a system with buffered I/O. The buffered I/O feature is explicitly taken into account. The use of the model in performance analysis and improvement, and its calibration and validation with experimental results, are illustrated.

2. The Problem

The purpose of the model developed in our study was to provide us with the ability to analyze the performance of the CDC 6400 computer running under the CALIDOSCOPE operating system at the Computer Center of the University of California, Berkeley. This ability was especially to be exploited in performance improvement investigations in which we wanted to predict the impact on performance of operating system modifications having to do primarily with disk I/O functions. The following types of modifications, all characterized by their relatively easy implementation, were specifically considered: the modifications of the

I/O buffer size, those of the system calling algorithm, and those of the central-processor reactivation policy. A brief description of the hardware-software system which was the object of the study is needed before these modifications, particularly the last two, can be explained in detail.

The hardware configuration of the system consists of the central processor (CP); a 64K 60-bit word central memory (CM); a 512K word extended core storage (ECS); ten peripheral processors (PP's), each with its own memory of 4K 12-bit words; and 12 I/O data channels which connect the PP's to the peripheral equipment. The local peripheral equipment includes 2 disk drives, 5 tape drives, 4 line printers, 1 card reader, and 1 multiplexor.

At any given instant, two hardware registers within the CP confine CP access to a block of contiguous addresses in CM. Each PP can modify the contents of these registers by executing an "exchange-jump" instruction, whose result is the switching of the CP from the processing of the current job to that of another job loaded in CM. Thus, the CM is divided at any given time into disjoint blocks, only one of which is accessible to the CP. These blocks are called control-points. The number of active control-points reserved for user jobs is the degree of multiprogramming of the machine. Under CALIDOSCOPE, the maximum number of user control-points is 7. Three additional control-points are reserved for the system. The PP's, some of which execute the operating-system code, have unrestricted access to the CM.

The CP cannot access the I/O channels directly. Communications with them occur via a circular buffer in CM and through the PP's. The buffer size must be at least as large as one physical record of the I/O device being used (for disk I/O, at least one disk sector, i.e., 64 words); however, it may be made arbitrarily larger. In the following description, we shall for clarity consider output operations only; input operations are performed in a similar way, except for the obvious modifications.

When the program running on the CP produces data to be output, it places them into the buffer. When the buffer is "full enough",[†] the CP program may choose to issue a call to the PP program CIO (Circular Input-Output). This call, like any other system call, is done by writing the name of the call plus some necessary additional information into the first word of the CM block where the running program is stored. One of the PP's scans this word once every millisecond and, if it finds a call there, dispatches ahother PP to service it. The decision to issue a CIO call is made by the CP program according to what was called above the <u>system calling algorithm</u>.

If the CP program cannot proceed in its execution due to the fact that the buffer has become full and CIO has not yet begun to empty it, one of two courses of action is to be taken: the CP program either hangs in a useless loop waiting for some space to be freed in the buffer, or relinquishes the control of the CP by issuing an RCL (Recall) system call. In the latter case, the CP is switched to another job and will eventually, after an unpredictable amount of time, return to the program which has relinquished control. To reduce this amount of time to a minimum, the PP program which actually controls disk transfers may issue a "reactivate CP" request to the supervisor as soon as the transfer of a sector is completed. This request causes the CP to be switched to the control-point which had issued the CIO. Since the provision for

[†]This notion will be clarified later.

reactivating the CP did not exist in CALIDOSCOPE when our study was undertaken, but its implementation was believed to be relatively easy, we wanted the model to be able to predict the effect of such a change. Thus, the CP reactivation policy (i.e., the presence or absence of this provision) was included in the list of the modifiable parameters.

The performance index selected for this study was the mean throughput rate of the system under a specific workload to be defined in Sections 4, 5, and 6.

3. The I/O Rate Concept

In this section, we discuss the problem of characterizing the I/O behavior of the system and its users. The amount of I/O performed by a job is not a satisfactory way to characterize this behavior, since it neither takes into account the real time nor the CP time consumed in producing the I/O. Another approach would be to use the CP time between I/O requests as the parameter of interest. This would be an improvement; however, as described in Section 2, the I/O on the Berkeley CDC 6400 uses circular buffers and is designed to enable concurrent operation of the CP and of the I/O devices. Thus, a sample job run twice might exhibit radically different numbers of I/O requests and CP times between requests during the two runs. For example, during one run the job might be able to keep control of the CP during some output operation, and therefore manage to fill the buffer while the disk is emptying it. In principle, the duration of such a process could be arbitrarily long. During another run, the same job might compete with another job for the CP and not be able to keep up with the disk, so that many more I/O requests would be required to output the same data.

As described in Section 2, the end goal of our study was to construct a model capable of predicting the performance of the system with various buffer sizes, system calling algorithms, and CP reactivation policies; thus, any I/O characterizing parameter which would be seriously affected by such changes had to be avoided.

A parameter which is more suitable for our objective is the ratio of the amount of I/O to CP time. This parameter is likely to be relatively insensitive to system changes and repeated runs, since every job will perform a specific task such as reading input, computing, and producing output, independently of what the system is doing. At any given point in a job, I/O is being consumed/produced at some rate (measured with respect to CP time) regardless of what machinations the system is performing. Thus the parameter chosen was the ratio:

$$r = \frac{s}{t_{cp}} \qquad (1)$$

where s is the number of sectors transferred, and t_{cp} is CP time consumed between consecutive transfers.

This parameter, called the I/O rate, can be measured by tracing each disk transfer performed by the system during a measurement session. The number of sectors transferred and the CP time consumed by each job must be recorded.

Figure 1 illustrates an imaginary job performing output as it uses CP time (solid line). The job's I/O subroutines call the disk processor when the buffer is full enough, and, some time later, the disk empties the buffer at its rate of 1 sector per millisecond (broken line). The parameter we measure, the average I/O rate for the interval, is the slope of the dotted line. Clearly, this is not an exact reproduction of the job's performance, but it is the best we can do with the information in the trace described above. Note that the output rate is defined as:

$$r_o = \frac{s_o}{t_{cp,o}} \qquad (2)$$

where s_o is the number of sectors transferred during the output operation, and $t_{cp,o}$ is the CP time consumed since the previous output operation.

The same considerations can be repeated for the input rate r_i. We must, however, replace index o by index i and define $t_{cp,i}$ as the CP time consumed until the next input operation.

4. The Model

In this section, we derive an analytic model which predicts the real time necessary to run a given set of jobs under various system conditions. The model is based on the I/O rate concept illustrated in Section 3, and is designed to be efficient rather than mathematically rigorous. That is, where effects are felt to be small, approximations are made which are designed to simplify the mathematics. Our model is based on the simplifying assumption of homogeneity: that is, we assume that each logical or physical device of a certain type (e.g., controlpoint, file, disk) is equally utilized. Instances of homogeneity will be mentioned in the sequel as they are encountered.

We now introduce the symbols to be used in deriving the model. They all refer to the real time period during which a given set of jobs is processed.

A - Average processor and mechanical delay per disk access. A includes the processor delay involved in the system response to a user CIO call, the average seek time, and the average latency time. For brevity, we may refer to A simply as "access time". A does not include queue waits, nor does it include actual read or write time. It is thus the average delay between the end of the queue wait (if any) and the beginning of the transfer.

C_j - Total CP time used by all user jobs.

C_s - Total CP time used by the operating system.

∂ - Average delay of a disk I/O request caused by queue wait.

D - Number of (single-spindle, single-arm) disk drives.

F - Number of files active at any one time. An active file is a file which is used by a job being executed. Thus, F is the number of files which are accessed by those jobs which, at a given time, are in execution. We assume that F is independent of time.

M - Mean degree of multiprogramming (in CALIDOSCOPE terminology, the number of user control-points which are active).

q - Probability that a file is being used by a job. A file is said to be "used" either during transfers of data belonging to it, or during processor or mechanical delays associated with accesses to it. Note that a file is <u>not</u> being used while a request for it is waiting in the disk queue.

R - Total real time consumed by the execution of the set of jobs under consideration.

T - Total transfer time consumed by each disk drive. T is assumed to be the same for each drive, and equally divided among control-points.

U - Time unavailable to the CP at each user control-point. U is the total time each user control-point spends in a blocked state. U is assumed to be the same for each user control-point.

X - Total number of transfers to/from each disk drive. X is assumed to be the same for each drive.

Let us now focus our attention on one of the M user control-points. Even though the control-point being considered is assumed to consume only $1/M$ of the total user CP time C_j, it is potentially able to make more use of the CP. In fact, the control-point is ready to use the CP whenever it is not blocked on a disk request. We use the term "ready" to mean that a control point is "able to use the CP", whether or not it actually has the CP. Thus, each user control-point is in a ready state unless it has issued an RCL request which has not yet been completely satisfied, i.e., for a fraction of the total real time $p = \frac{R-U}{R}$.

Under CALIDOSCOPE, the system control-points have higher CP priorities than the user control-points. CP scheduling of the system control-points is preemptive, so that, if some system control-point is ready to run, it does. If two or more system control-points are ready to run at once, other priority rules are invoked; but, for our purposes, this is irrelevant, and we only need to know that the system control-points, which are represented in our model by the whole system CP time C_s, outrank all of the user control-points.

To simplify our model further, we assume that the system control-points' ready states are uncorrelated in time with the ready states of the user control-points. Thus we have that, at any instant, the probability of a system control-point being in a ready state is C_s/R.

Assuming the user control-points block and become ready independently of each other, the probability of n user control-points being simultaneously ready is:

$$\Pr\{n\} = \frac{M!}{n!(M-n)!} p^n (1-p)^{M-n} , \quad \text{for } 0 \le n \le M . \quad (3)$$

Some user control-point will execute if at least one is ready and the system is not ready. The total user CP time C_j is thus given by:

$$\begin{aligned} C_j &= R \cdot \frac{R-C_s}{R} \cdot \Pr\{n > 0\} \\ &= R \cdot \frac{R-C_s}{R} \cdot [1 - (1-p)^M] . \quad (4) \end{aligned}$$

Substituting p with its expression, we have

$$C_j = (R-C_s)[1 - (\frac{U}{R})^M] , \quad (5)$$

or

$$R^{M+1} - R^M(C_s+C_j) - RU^M + C_sU^M = 0 . \quad (6)$$

We shall denote the polynomial in (6) by $P(R)$.

If we know C_j, C_s, M, and U, we can solve (6) for R, the real time. We shall now investigate some of the properties of the solution.

If $M = 1$ (uniprogramming), solving (6) yields

$$R = \frac{1}{2}[(C_s+C_j+U) \pm \sqrt{(C_s+C_j+U)^2 - 4C_sU}] . \quad (7)$$

Since in any real system C_j, C_s and U are non-negative, the radicand in (7) is non-negative. Furthermore, if $C_j > 0$, the radicand is positive. Thus, the positive sign in (7) is to be chosen to make

R positive, and a real solution exists in the range:

$$\tfrac{1}{2}(C_s + C_j + U) \leq R \leq (C_s + C_j + U) \ . \qquad (8)$$

The lower bound for R can be improved by observing that the radicand is larger than $(C_s+C_j+U)^2$. Thus, for a uniprogramming system we obtain, as could be expected,

$$C_s + C_j \leq R \leq C_s + C_j + U \ . \qquad (9)$$

Although equation (6) appears to be difficult to solve in the general case for arbitrary M, we can show that exactly one solution exists for $R > C_s + C_j$.

Evaluating the polynomial $P(R)$ at $C_s + C_j$, the first two terms cancel, leaving:

$$P(C_s+C_j) = -(C_s+C_j)U^M + C_s U^M = -C_j U^M < 0 \ , \qquad (10)$$

if both C_j and U are strictly positive (the only interesting case). However, since the coefficient of R^{M+1} in equation (6) is positive, we have $P(R) > 0$ for sufficiently large R. Thus at least one solution exists for $R > C_s + C_j$.

To see that no more than one solution can exist, we evaluate the second derivative of $P(R)$ and discover that $P(R)$ is concave upward for $R > \frac{M-1}{M+1}(C_s+C_j)$; therefore, since $P(R)$ is negative for $R = C_s + C_j$, positive for sufficiently large R, and concave upward, equation (6) has one and only one solution for $R > C_s + C_j$ (see Figure 2).

A solution based on Newton's method appears attractive. Since $P''(R) > 0$ for $R > C_s + C_j$, the first derivative is monotonic increasing. But we do not know when it becomes positive, since $P'(C_s+C_j) = (C_s+C_j)M - U^M$.

If the Newton's method initial value is at Q (see Figure 2), a wrong root will be found, since Newton's method uses the slope and function value at some point to determine the function value at the next iteration.

Note, however, that $P'(R)$ is positive in some half-infinite neighborhood including the desired solution. Since $P''(R)$ is positive for $R > C_s + C_j$, Newton's method, if started anywhere to the right of the function minimum (say at the point Q') will converge. Therefore, it is only necessary to choose an initial point Q' such that:

$$Q' > C_s + C_j \quad \text{and} \quad P'(Q') > 0 \, . \qquad (11)$$

If conditions (11) are satisfied and the values of C_j, C_s, and U are known, Newton's method will allow us to determine the real time R needed to execute the given set of jobs. Unfortunately, as we shall see in Section 5, U cannot be determined without knowing R. However, an iterative process which first determines R, then U, then repeats, can be made to converge. This process will be described in the next section.

5. An Iterative Solution of the Model

In Section 4, we constructed a model which allows us to compute the total real time needed by the CDC 6400 system to execute a given set of jobs. To obtain the total real time, equation (6) must be solved. Some properties of the only acceptable solution have been studied in Section 4. In this section, we show how this solution can be arrived at by an iterative method and how the values of parameters U, C_s and C_j, which appear in (6), can be calculated. We immediately note that the degree of multiprogramming M is a system parameter and can be assumed to be known; also known is the total CP time C_j consumed by the given set of jobs, which is a workload characteristic. A method for computing C_s in the CALIDOSCOPE system is described in the Appendix. That method, which requires the knowledge of X, the total number of transfers per disk to be calculated below, may not be generalizable to other systems. However, if $C_s << C_j$, an adequate accuracy may be achievable with only a rough approximation to C_s; in this case, which seems likely to occur, the lack of a generalizable model is not too serious.

To compute U, we consider a control-point which we arbitrarily call "ctlpt 1". We shall determine what fraction r_e of the ready time at ctlpt 1 will actually be spent computing, and then assume that ctlpt 1 executes continuously while it is ready, at a rate equal to r_e times the actual CP rate. The introduction of the <u>effective rate</u> r_e allows us to correct for the effects of other control points competing for the CP.

To compute r_e, we simply take the ratio of the CP time consumed by ctlpt 1, C_j/M, to its total ready time:

$$r_e = \frac{C_j}{M(R-U)} \, . \qquad (12)$$

At a certain instant a CIO call is placed by the job executing at ctlpt 1; we assume that ctlpt 1 only fills its output buffer (or empties its input buffer) at a rate r_e times the executing job's output (or input) rate at that instant. When the buffer becomes full (or empty), an RCL is issued, and then and only then does ctlpt 1 block. The real elapsed time until ctlpt 1 once again becomes ready contributes to U.

We note that all I/O algorithms which we use place a system call to CIO (the PP program which performs I/O) when a certain specified amount of data, which we shall call C, is in an output buffer, or when the same specified amount of free space exists in an input buffer. In either case, C is the amount of data which is to be transferred. For the rest of the discussion, we will refer simply to output for clarity, although the discussion applies equally well to input. Apart from the initial and final transients of a job, at least the amount of data C will be transferred per I/O operation. If the job being executed released the CP at the instant of the CIO call, the amount of I/O would be exactly C; but it does not: it keeps the CP until or unless the buffer becomes full. Once the buffer becomes full, control of the CP is relinquished until the system decides to restart that control-point. Evidently, if B is the buffer size, we have $0 < C \leq B$.

Let us first consider the case in which no "reactivate CP" requests are made by the disk driver PP program. In that case, once the buffer is full and the job releases the CP, no further CP action related to that job will occur until the I/O is complete. We assume that the CP continues to process at the rate r_e for as much of the queue wait ($\bar{d}$) and access time (A) as it can before the buffer fills. Thus, the real time between an output request and either the instant the buffer is full or the instant the transfer begins is

$$t_o = \min\left(\frac{B - C}{r_e \cdot r_o}, \bar{d}+A\right) \qquad (13)$$

where r_o is the output rate of the job at ctlpt 1 at that time; the first term is the real time required to fill the buffer; the second is the average time until the disk transfer begins.

The amount of data in the buffer when the transfer begins will then be:

$$C + t_o \cdot r_e \cdot r_o \ . \qquad (14)$$

If $t_o < \bar{d} + A$, an RCL will have been issued, so that no further CP action can occur; the amount of data transferred is thus B, the buffer size.

But if $t_o = \bar{d} + A$, the transfer begins when the buffer contains only an amount of data equal to $C + t_o \cdot r_e \cdot r_o < B$; thus, the CP keeps executing, adding data to the buffer at the rate $r_e \cdot r_o$ as the disk empties it at its rate r_d.

If $r_e \cdot r_o < r_d$, the buffer will empty at the rate $r_d - r_e \cdot r_o$, requiring a time $\frac{C + t_o \cdot r_e \cdot r_o}{r_d - r_e \cdot r_o}$. Since data is still being processed by the CP, the total amount of data transferred will be:

$$C + t_o \cdot r_e \cdot r_o + r_e \cdot r_o \cdot \frac{C + t_o \cdot r_e \cdot r_o}{r_d - r_e \cdot r_o} . \qquad (15)$$

But if $r_e \cdot r_o > r_d$, the buffer will fill, an RCL will be issued, and the CP will not be available until the disk transfer terminates. Since we know how many sectors are transferred at each rate, we know how many transfers occur at each rate; their sum is the total number of disk transfers X. If the job ever issues an RCL, the time until the disk transfer completes contributes to U.

The flowchart in Figure 3 details this process for any single value of the I/O rate. If the distributions of input and output rates for the given set of jobs are known, the resulting values of X and U for each I/O rate are to be weighted according to their probability before being summed. This is the task of the final box in the flowchart in Figure 3. Note that, in this figure, NSECT is the total number of sectors to be transferred by the jobs, hence a characteristic of the workload and assumed to be known.

In the case of the reactivate CP feature, we first note that jobs with sufficiently high I/O rates will be able to transfer enormous amounts of data per I/O request: if the rate at which the CP processes data is higher than the disk transfer rate, and if the "reactivate CP" requests from the disk driver always activate the CP, then the job can in principle continue transferring until it terminates. If we partition the I/O rate histograms into two parts corresponding to the rates lower than the disk transfer rate and to the rates higher than this rate, we can compute the probabilities of being in one of these two states and use them to weigh the resulting values of X and U for these two cases. The computation of X and U in presence of the "CP reactivate" feature is illustrated by the flowchart in Figure 4. Note that NMAX is the expected number of sectors transferred per request when the job is in the high-I/O-rate state. This is equal to 1/p, where p is the probability of leaving the high-I/O-rate state. Our measurements showed that about half of the data was transferred in each state, and that p was about 1.1%. Thus, the average number of sectors transferred per request in the high-I/O-rate state was about 90. Note also that, while in the high-I/O-rate state, a job only uses the CP part of the time. This is because it fills the buffer faster than the disk can empty it, and hence has to issue an RCL request. But the moment one more sector has been transferred by the disk driver, the CP is reactivated for that job. We make the approximation that the CP time consumed during this process is merely whatever is needed to process the data. The difference between the time used by the disk and the computed CP ready time will then contribute to U.[†]

The computation of U requires the knowledge of $\bar{d}$, the mean

[†]The CP times thus computed will be lower than the correct ones, since the computation neglects the CP time required to place the RCL requests and wait until the system's response.

waiting time of an I/O request in the disk queue (see Figures 3 and 4). To derive $\bar{d}$, we use a non-queueing-theoretic approach. In our simple, approximate model, we assume statistically independent and stationary accesses to disk files: thus, the probability of a disk file being accessed by one job is independent of time, and the probability of a disk being accessed by two jobs at once is merely the product of the probabilities of single accesses.

Under these assumptions, whose realism clearly decreases as queue length increases, the total queue wait is the sum:

$$R \cdot \sum_{m=2}^{F} (m-1)\Pr\{m\} \quad , \tag{16}$$

where m is the number of simultaneous accesses to the same disk. We also have:

$$\Pr\{m \text{ files simultaneously active on any one disk}\} = \frac{F!}{m!(F-m)!} q^m (1-q)^{F-m} \quad , \tag{17}$$

and

$$\begin{aligned} X \cdot A + T &= E[\text{total seek} + \text{latency} + \text{transfer time of one disk}] \\ &= R \cdot \sum_{m=1}^{F} \frac{F!}{m!(F-m)!} q^m (1-q)^{F-m} \\ &= R \cdot [1 - (1-q)^F] \ . \end{aligned} \tag{18}$$

Thus,

$$1 - (1-q)^F = \frac{X \cdot A + T}{R} \ , \tag{19}$$

or

$$q = 1 - [1 - \frac{X \cdot A + T}{R}]^{1/F} \ . \tag{20}$$

If $F = 1$, $q = \frac{X \cdot A + T}{R}$ as expected; and for any F, if $q << 1$

(i.e., $(X\cdot A+T) \ll R$), then $q \cong \frac{X\cdot A+T}{F\cdot R}$, also as expected.

The expected value of the total queue wait per disk is

$$\begin{aligned} \text{E[total queue wait per disk]} &= \sum_{m=2}^{F} R\cdot \Pr\{m \text{ files}\}\cdot(m-1) \\ &= R\cdot \sum_{m=2}^{F} \frac{(m-1)\cdot F!}{m!(F-m)!} q^m (1-q)^{F-m} \\ &= R[(1-q)^F - (1-q\cdot F)] \ . \qquad (21) \end{aligned}$$

Note that, if $F = 1$, E[total queue wait per disk] $= R[(1-q) - (1-q)] = 0$, as expected. Dividing the total delay given by (21) among the X transfers, we obtain

$$\bar{d} = \frac{R}{X}[(1-q)^F - (1-q)F] \ . \qquad (22)$$

Note that F is assumed to be known; thus, (22) allows us to compute $\bar{d}$ only if R and q are known.

An iterative procedure based on equations (6), (12), (20) and (22), and on the algorithms in Figures 3 and 4 can be used to compute R. Such a procedure is illustrated in Figure 5 and basically consists of choosing initial values for $\bar{d}$, C_s, R, U and X, and of applying the relationships listed above in a relaxation fashion.

The value of q is the most critical value in the whole iterative process. In some test cases performed on the model, while the iterations were converging to the final value for R, R was changing by a few per cent from one iteration to the next, while q was only changing in the seventh decimal place. Conversely, a small change in q (introduced manually) caused the computation for R to oscillate.

Since in these cases the computation of q based on equation (20) overcompensates for the errors of the previous iteration, we have achieved good results by using a damping technique: instead of merely computing q from equation (20), we used only half of the change in q given by (20) with respect to its previous value. However, in cases where the real time is much larger than the CP time, i.e., when $R >> C_j + C_s$ (by a factor of 2 or more), the solution is near a singularity, and this damping technique is not sufficient to substantially speed up the convergence of the iterations. This problem, when it arises, can be cured by giving an initially large but increasingly smaller increment to the value of q instead of applying the damping technique described above.

6. The Experiments and the Results

The performance comparisons needed in improvement studies are meaningful only if the modified and unmodified systems are compared under the same workload. Since controlling the production workload is practically impossible, an executable workload model was constructed, consisting of a number of synthetic jobs to be produced at every experiment by a synthetic job generator [1]. The choice of a set of synthetic jobs was also dictated by the desire to use the workload model in the experimental calibration and validation of our system model. Thus, flexibility was an important requirement of the workload model. A further requirement was that the synthetic jobs should adequately represent the real jobs in terms of their individual CP-I/O behavior [2].

To achieve the desired degree of representativeness, a very detailed trace of almost 8 hours of normal production processing was recorded on February 23, 1974. The trace was subsequently analyzed and two histograms of input rates and output rates (see Section 3) were extracted from it. The intervals of the values taken by the I/O rates were then subdivided into ranges, and the relative frequency of each range computed from the histograms.

Each synthetic job is at any given time of its execution in an input rate state and in an output rate state corresponding to two of these ranges, one for input and one for output. The probabilities of these states are equal to those derived from the trace. Each I/O rate state is assumed to be statistically independent of all the other I/O rate states. The synthetic jobs could thus be programmed identically, using a random number generator to select the states. The only difference is that each job initializes the random number generator with a different value. The ratios r_i and r_o (see (1)) corresponding to the various I/O rate states are reproduced by executing a CP loop whose duration, added to the time required to generate a random number, equals $1/r_i$ or $1/r_o$, and by transferring one sector of data at the proper time. Each new random number generated is used to select the next I/O rate state. Input and output are performed by each job simultaneously and independently, and this cycle is repeated 1,000 times.

A synthetic job stream consisting of 20 of these jobs was used in all experiments: this number was chosen because it resulted in a total running time in the range from 5 to 10 minutes. The stream was calibrated by adjusting the random number generator seed and the degree of multiprogramming: a degree of 2 was found to provide a good match with the measured CP utilization, and was adopted for all the runs. Comparisons of the measured mean I/O rate and mean I/O request rate with the corresponding rates generated by our workload model convinced us that this model adequately represented a reasonably heavy real workload. Further details are given in [3]. Reproducibility tests demonstrated the synthetic model to be much more stable than any of the other workload models used in the same installation: in 6 runs performed under similar conditions, the total elapsed times ranged from 432 to 468 seconds, with a mean of 446 seconds and a standard deviation of 13.4 seconds, i.e., about 3% of the means. Other synthetic workloads had shown 30% or more variation from run to run.

Once a representative, flexible and reproducible workload model was available to allow experiments to be performed in a controlled environment, the predictions of the model described in Sections 4 and 5 could be compared with the actual system's performance under our synthetic job stream. However, before being used, the model had to be calibrated. The main calibration parameter was the number D of disk drives.

The disk traffic of the synthetic jobs was designed to correspond to the observed usage during the production system trace, which was approximately in the ratio 5:3. We, therefore, ran the model with $D = 1.6$ as well as with $D = 2$, to see if using $D = 1.6$ would more accurately predict the results.

Note that the model is based on the assumption that all D disks are equally utilized. Letting $D = 1.6$ thus results in a computation based on 1.6 equally utilized disks, whatever that may mean; and not on two disks, one transferring .6 times as much as the other. So the use of $D = 1.6$ is only an approximation.

In fact, the differences between model predictions using $D = 2$ and $D = 1.6$ were very small; however, the agreement with the experimental results was better with $D = 1.6$ than with $D = 2$. We, therefore, used $D = 1.6$ in all subsequent predictions of synthetic job performance.

The model was used to predict values for the real time (R) and number of CIO calls (X) for each of several sets of parameters of the synthetic jobs: with 8 and 16 sector buffers, with and without CP reactivation, and with many values of C. The error in the prediction of the number of CIO calls (X) was not large, being at worst about 20%, and averaging about 15%. As shown in Figure 6, the error in R was, however, spectacularly small: in the worst case, it was about 3.5%, and in all other cases less than 2%.

A large number of predictions were then made using the model. Many combinations of system calling algorithms (i.e., the amount of information C in the buffer at which CIO is called) with various buffer sizes, with and without CP reactivation, and with various degrees of multiprogramming, were examined in these experiments. Most of the results are presented in [3]. A sample set of predictions is given in Figure 7. The main parameters used to initialize the model (see Figure 5) in that case were: $A = 62$ ms, $C_j = 260$ s, $M = 3$, $D = 1.6$, $F = 4$ and NSECT = 40,300. The curves in Figure 7 show that, as is the case for $M = 2$ (see Figure 6), when $M = 3$ the throughput rate is quite sensitive to the size of the buffer if there is no reactivation of the CP. If the CP is reactivated, the impact of buffer size becomes minor, and R approaches the total user CP time C_j. The effects of CP reactivation are especially felt for small buffer sizes, and become smaller as this size increases. The system calling algorithm is never very important, and decreases in importance with large buffer sizes.

During our study, a reduction in the permanent disk file usage caused the average seek time, and hence the access time A, to decrease.

The values computed for R based on the newly observed access times were almost always more accurate than predictions based on the previously used value of 62 milliseconds. This result is pleasing, since an accurate model must agree more closely with experiment if more accurate data is used.

We also noted that agreement between experiment and theory is much better in cases with lower real time R. The reason is evident: in such cases, the system is close to being CP bound (the sum $C_s + C_j$ is typically close to 300 seconds for all of our experiments and predictions). Thus, almost all of the I/O processing is overlapped with CP execution, and errors in the calculations of I/O times are of little consequence. But in the cases resulting in high values of R, CP utilization is much smaller, and errors made in the numerous approximations involved in predicting I/O times and overlaps are significant.

We might point out that in a model of system performance to be used for system tuning, the "good" cases from the point of view of the system analyst (the cases in which R is small) are of more interest and importance than the "bad" cases (those in which R is large). It is a virtue of our model that it predicts the good cases accurately; these are the ones which are most important in operating system adjustment and tuning.

References

(1) H.D. Schwetman and J.C. Browne, "An experimental study of computer system performance," Proc. ACM National Conference 1972, 693-703.

(2) D. Ferrari, "Workload characterization and selection in computer performance measurement," Computer 5, 4 (July-August 1972), 18-24.

(3) D.S. Lindsay, "A study in operating system performance measurement and modeling," Ph.D. dissertation, University of California, Berkeley, June 1975.

Appendix - Computation of C_s

The method for estimating the system CP time C_s depends on the details of the operating system. In the case of CALIDOSCOPE, we conjecture that C_s varies approximately linearly with the number of CIO calls X, since each call requires some system CP processing to read the code for the appropriate PP program from ECS (Extended Core Storage). While the computation which we present below is thus designed specifically for CALIDOSCOPE, we expect a linear assumption to be reasonable also for other systems.

In any case, the system CP time will probably be small compared to C_j, the CP time consumed by the user jobs. In fact, C_s was usually between 15% and 20% of C_j in our experiments. Therefore, a rough approximation to C_s is generally sufficient for an adequate prediction of R.

A linear first-order regression performed on our experimental data resulted in the equation

$$C_s = (33.6 + X \cdot .007) \text{ seconds} , \qquad \text{(A.1)}$$

with an RMS error of approximately 5%. Note that equation (A.1) does not depend on whether the CP reactivate feature is present or not. Also, the coefficient of X is in perfect agreement with the result of our measurements, which gave us a processor delay of 7 milliseconds per CIO call.

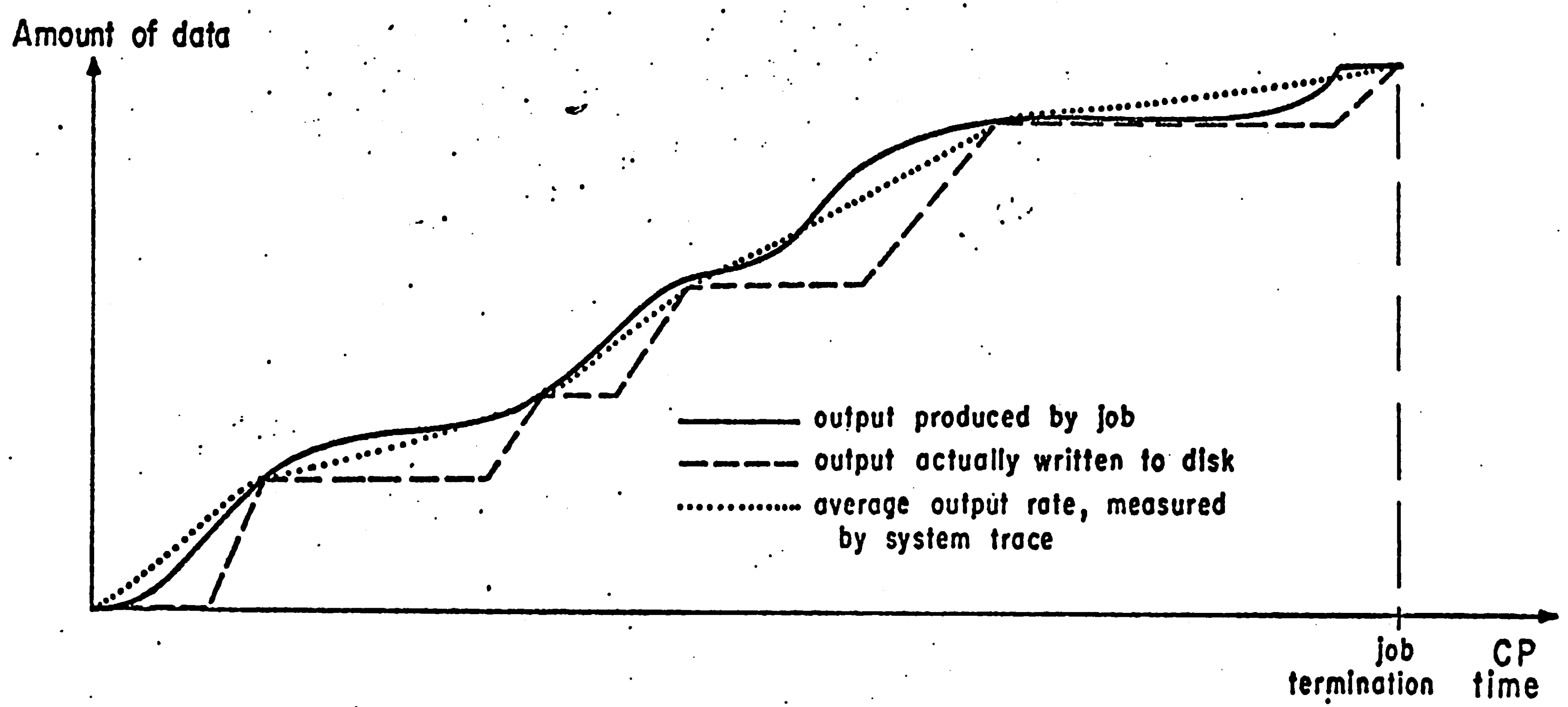


Figure 1. A job producing output.

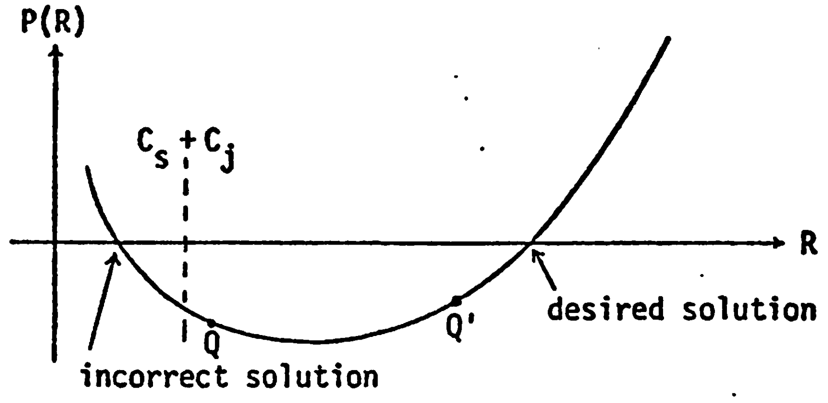


Figure 2. Possible appearance of P(R).

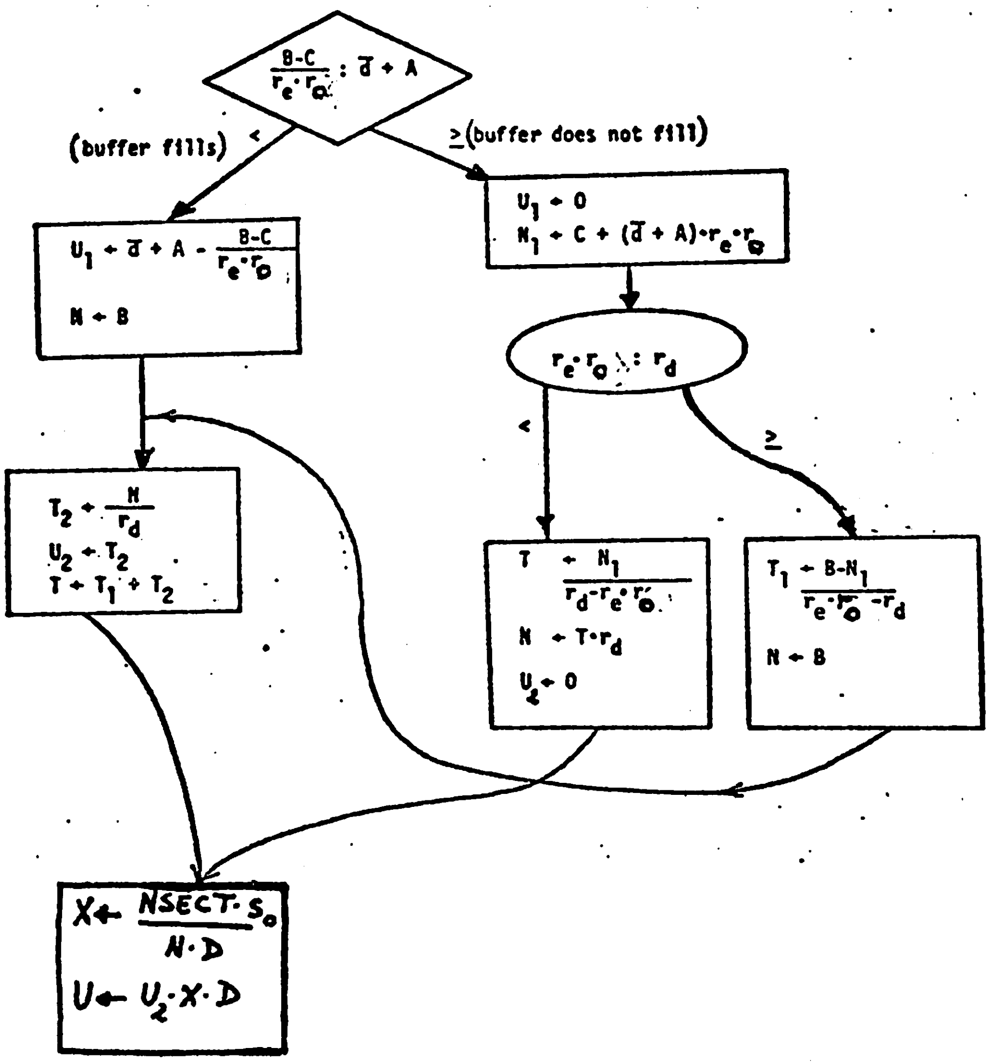
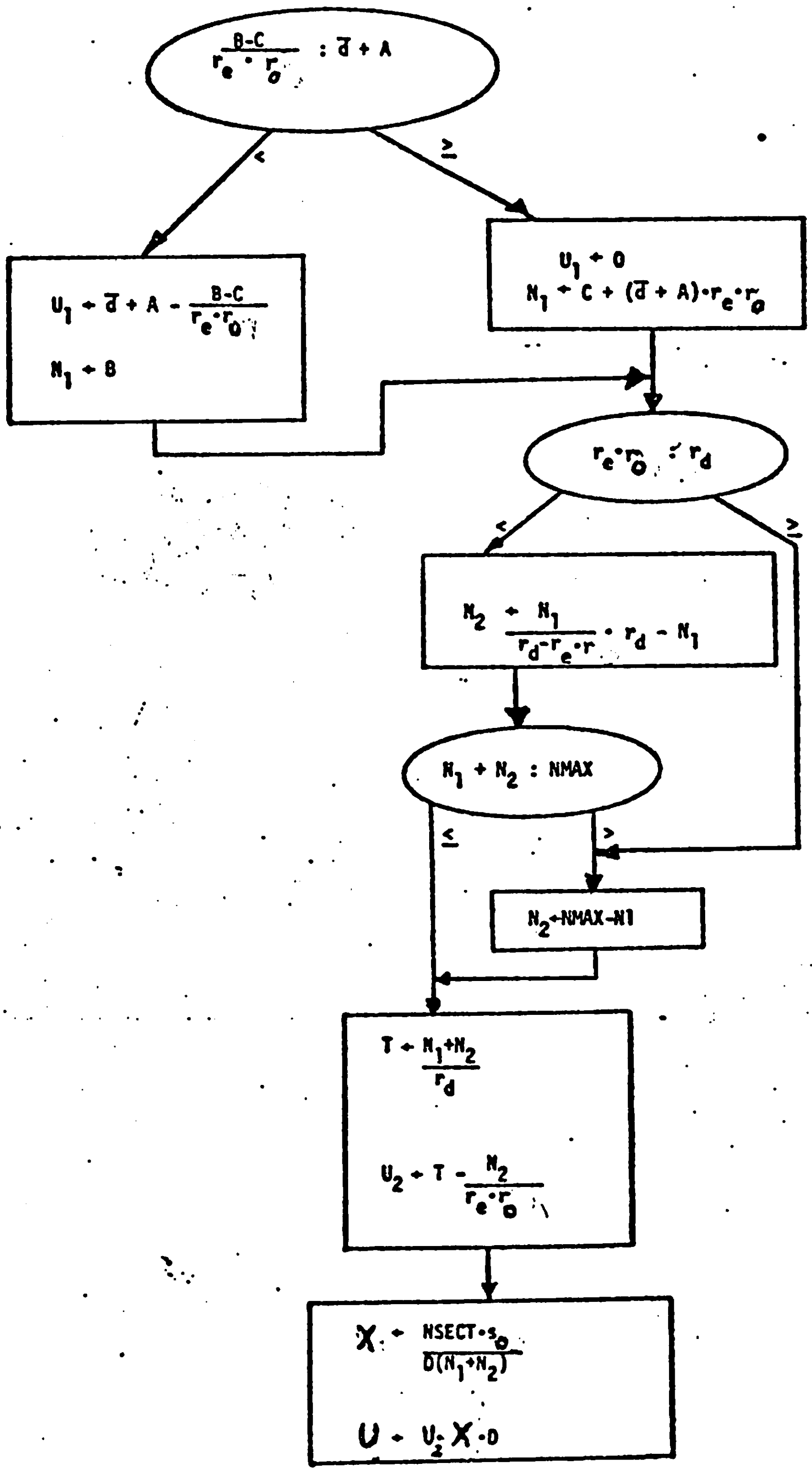

$\frac{B-C}{r_e \cdot r_o} : \bar{d} + A$
(buffer fills) <
≥ (buffer does not fill)
$U_1 \leftarrow \bar{d} + A - \frac{B-C}{r_e \cdot r_o}$
$N \leftarrow B$
$U_1 \leftarrow 0$
$N_1 \leftarrow C + (\bar{d} + A) \cdot r_e \cdot r_o$
$r_e \cdot r_o : r_d$
<
≥
$T_2 \leftarrow \frac{N}{r_d}$
$U_2 \leftarrow T_2$
$T \leftarrow T_1 + T_2$
$T \leftarrow \frac{N_1}{r_d - r_e \cdot r_o}$
$N \leftarrow T \cdot r_d$
$U_2 \leftarrow 0$
$T_1 \leftarrow \frac{B-N_1}{r_e \cdot r_o - r_d}$
$N \leftarrow B$
$X \leftarrow \frac{NSECT \cdot S_o}{N \cdot D}$
$U \leftarrow U_2 \cdot X \cdot D$

Figure 3. Computation of unavailable CP time (U) and number of disk transfers (X) without the reactivate CP feature. Subscript 1 refers to quantities accumulated before transfer begins, subscript 2 to quantities accumulated during transfer.



$\frac{B-C}{r_e \cdot r_o} : \bar{d} + A$
<
≥
$U_1 \leftarrow \bar{d} + A - \frac{B-C}{r_e \cdot r_o}$
$N_1 \leftarrow B$
$U_1 \leftarrow 0$
$N_1 \leftarrow C + (\bar{d} + A) \cdot r_e \cdot r_o$
$r_e \cdot r_o : r_d$
<
≥
$N_2 \leftarrow \frac{N_1}{r_d - r_e \cdot r} \cdot r_d - N_1$
$N_1 + N_2 : NMAX$
≤
>
$N_2 \leftarrow NMAX - N1$
$T \leftarrow \frac{N_1 + N_2}{r_d}$
$U_2 \leftarrow T - \frac{N_2}{r_e \cdot r_o}$
$X \leftarrow \frac{NSECT \cdot s_o}{D(N_1 + N_2)}$
$U \leftarrow U_2 X \cdot D$

Figure 4. Computation of unavailable CP time (U) and number of disk transfers (X) with the reactivate CP feature. Subscript 1 refers to quantities accruing before transfer begins; subscript 2 to quantities accruing during transfer.

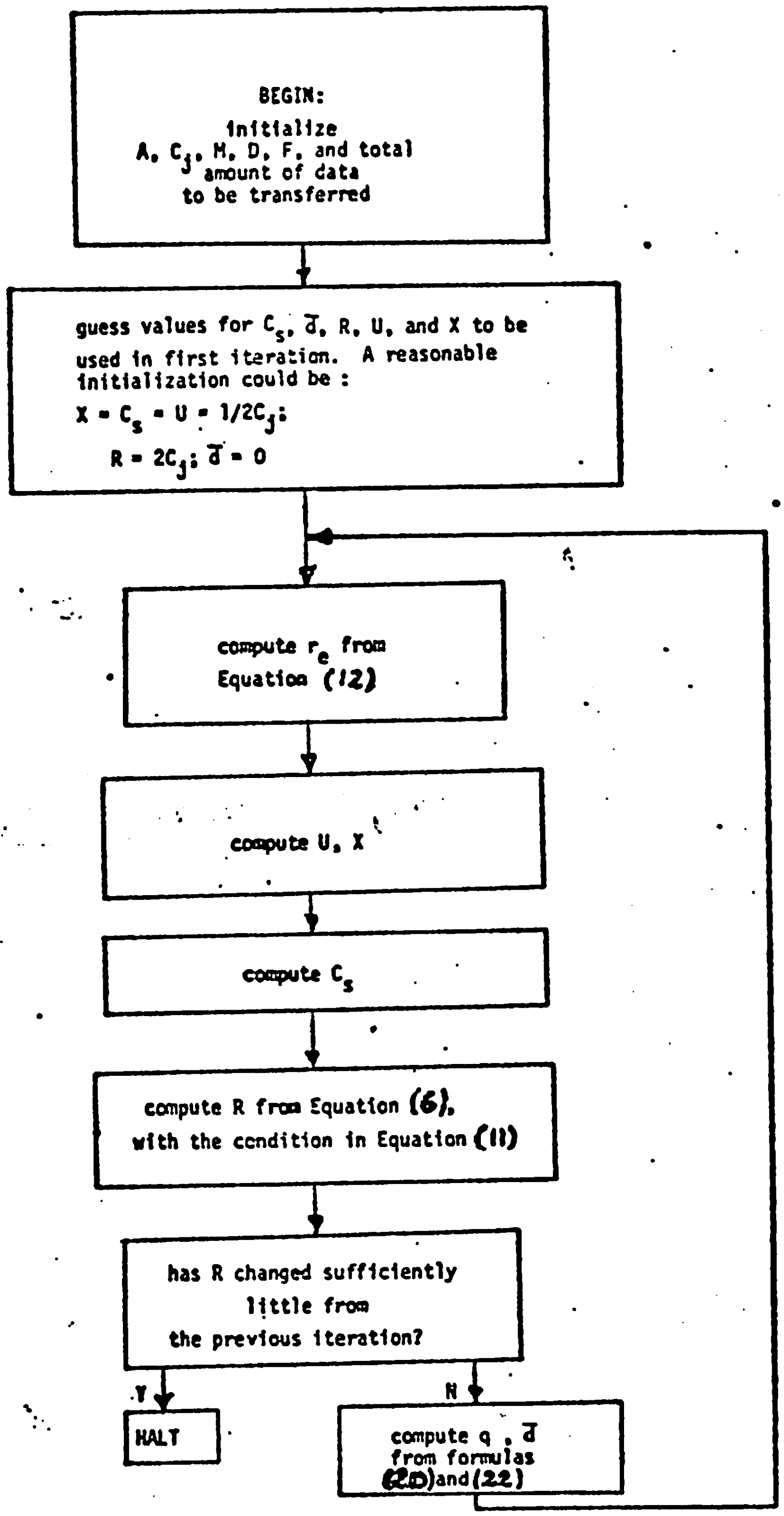

BEGIN:
initialize
A, C_j, M, D, F, and total amount of data to be transferred
guess values for C_s, $\bar{d}$, R, U, and X to be used in first iteration. A reasonable initialization could be :
$X = C_s = U = 1/2C_j$;
$R = 2C_j$; $\bar{d} = 0$
compute r_e from Equation (12)
compute U, X
compute C_s
compute R from Equation (6), with the condition in Equation (11)
has R changed sufficiently little from the previous iteration?
Y
N
HALT
compute q, $\bar{d}$ from formulas (20) and (22)

Figure 5. Flow-chart of the iterative procedure for the computation of R.

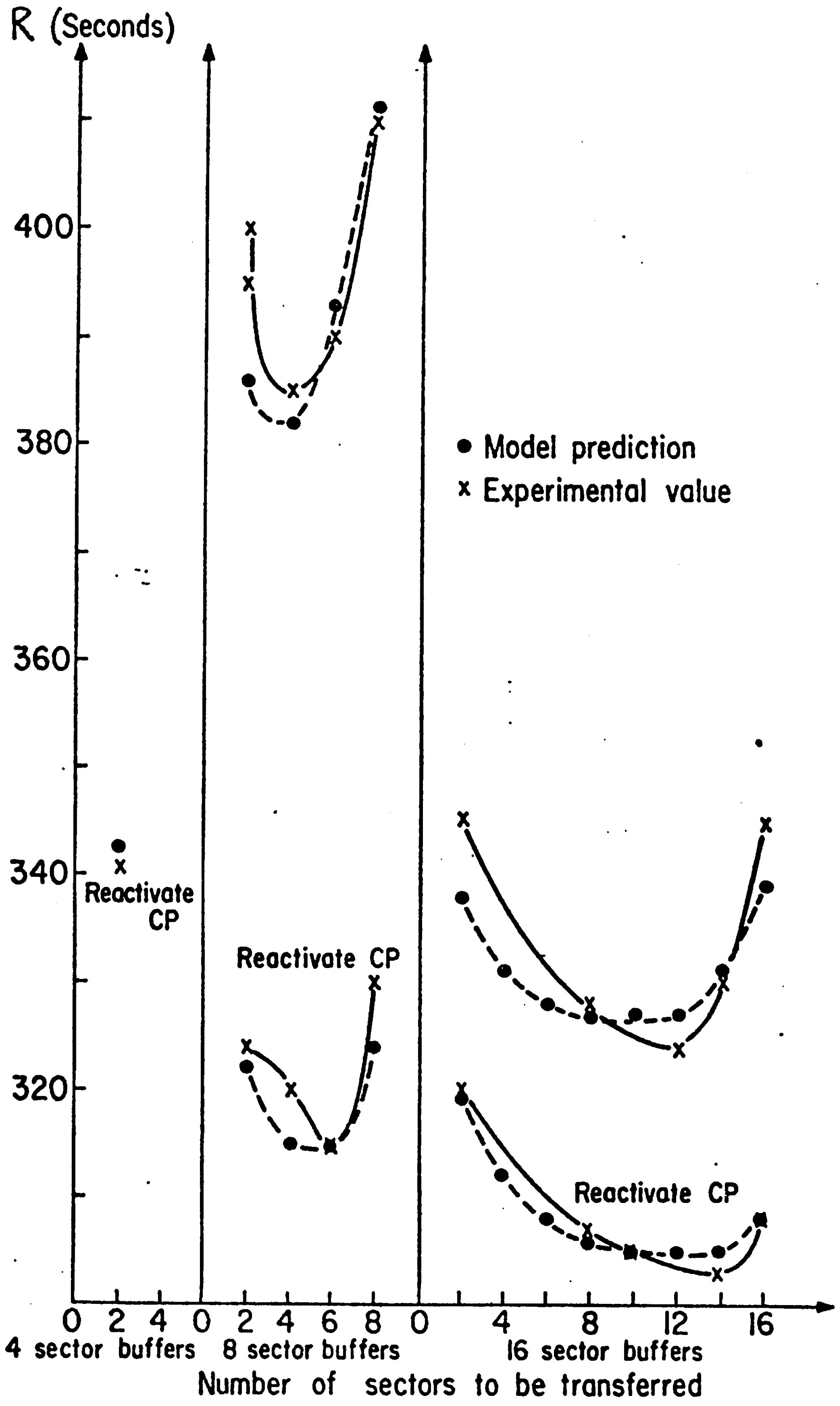


Figure 6. Comparison of analytic and experimental results.

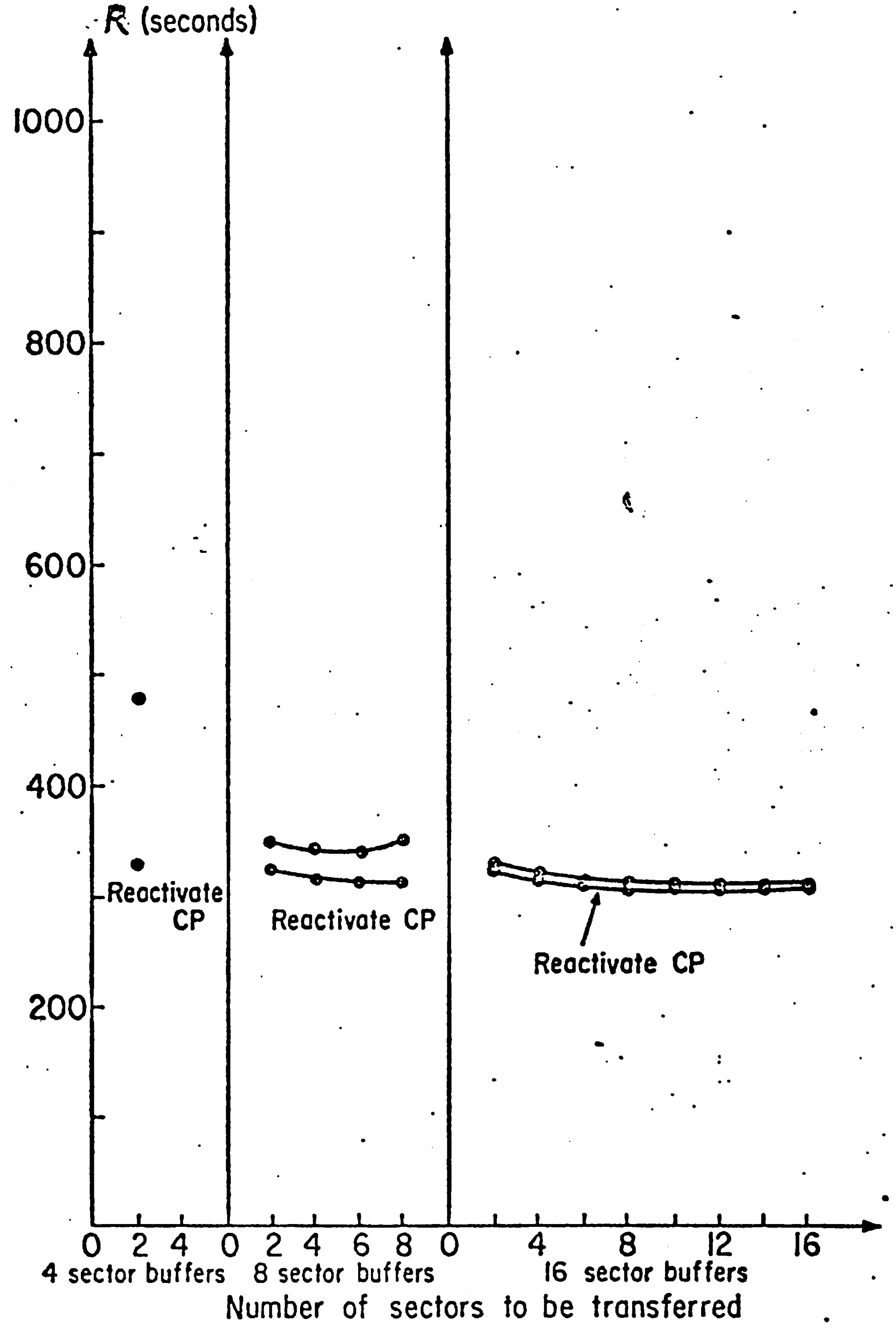


Figure .7. Model predictions with 3 control-points.