The University of Wisconsin
Computer Sciences Department
1210 West Dayton Street
Madison, Wisconsin 53706

OPTIMAL I/O BUFFER SIZE FOR MULTI-PROGRAMMING

by

James Wang Kho and Tad Brian Pinkerton

Technical Report #107

January 1971

OPTIMAL I/O BUFFER SIZE FOR MULTIPROGRAMMING*

by

James Wang Kho and Tad Brian Pinkerton

ABSTRACT

The question of an optimal I/O buffer size for multiprogramming is considered by analyzing a probability model of the operations of multiprogramming. An algorithm is given for finding the optimal solution based on number of jobs in core, length of programs, number of I/O requests, and other parameters. Statistics obtained from the UWCC 1108 system are used in the experimental work. A simulation program is done in SIMSCRIPT and the analysis of the results herein presented.

*The authors wish to express their gratitude to Manly Draper of the University of Wisconsin Computing Center for bringing this problem to their attention and for providing many of the statistics needed in the experimental work.

I. INTRODUCTION

The performance of single thread (uniprogramming) batch systems used on second generation computers suffered from the disparity between high processing rates and low I/O (Input/Output) capability. Current systems employ several techniques for multiprogramming in an attempt to keep the CPU (Central Processor Unit) busy. Although main memory sizes have increased greatly, processor power has increased faster than I/O rates. Thus it is still true today that most large-scale systems cannot store enough complete tasks in main memory to keep the CPU busy.

In order to ready more tasks for the CPU, two types of buffering have been devised: Buffering of data required by executing programs, and buffering of programs themselves. Hardware assistance in data buffering has taken the form of sophisticated channel and I/O processor organizations which offer simultaneity between CPU and I/O activities and between multiple I/O activities. Hardware and software developments in program buffering have led to the virtual memory concept [3,4,14], which allows execution of a fragment of a program while some part of it resides only in secondary storage. The units of a program which are passed back and forth in the memory hierarchy are either arbitrary fixed-size blocks (pages) or larger units (segments) of variable length which correspond to functional units of subprograms and data [16].

A great deal of work has been done on program buffering recently, and efficiency improvements have resulted from advances in file system organization [6,18,19], scheduling of I/O operations [1], and system resource allocation [2,15]. As program buffering increases, so does the need for data buffering. SPOOLing (Simultaneous Peripheral Operations On Line) techniques which were developed for second generation systems require careful integration with multiprogramming systems.

The choice of optimal data buffer size has been an important question for many years. It is closely related to the choice of optimal program buffer (page) size, but there are some differences:

(1) whereas the I/O channel hardware for data buffering assistance exists in some form on all current machines, dynamic address translation hardware necessary for program buffering is present only on a small fraction of today's machines;

(2) the two kinds of buffering are typically controlled by different levels of system programs, and may use different secondary storage devices;

(3) the individual programmer is often allowed to exercise a measure of control over his data buffering, whereas program buffering is controlled by the system.

This paper deals with the selection of an optimal data buffer size. Due to the tremendous difference between I/O transfer rates and CPU processor rate, the use of an optimal data buffer size could greatly increase CPU utilization. The work reported here partly consists of the description and analysis of a probability model of the operations of multiprogramming. This model is used to investigate the multiprogramming of the UNIVAC 1108 system [17] now used at the University of Wisconsin Computing Center. An evaluation plan is described in [5] for measuring the performance ot that system. Statistics for various days' runs are analyzed and in each case, the effect of an optimal data buffer size is demonstrated. A simulation model of the system is also done using SIMSCRIPT [11] with similar results indicated.

II. THE PROBLEM AREA

In the subsystem relevant to this study, the two main functions performed are (1) execution of programs, including the executive routines, and (2) transfer of data between auxiliary memory and the main core. A configuration of this subsystem is shown in Figure 1. This consists of a single central processor, a main storage area, and a peripheral subsystem of I/O and secondary storage devices.

Jobs enter the main memory via I/O devices and reside there while being processed by the CPU. With only a part of the total data file contents of a given job being input, the operation is buffered and parts are processed one at a time. Each job thus undergoes alternating operations of the two functions mentioned: an I/O period, then a compute period, followed by another I/O period, and so on until the job is completed.

Assuming then that there are a number of job segments residing simultaneously in core, each of these segments will be in one of the four states: (1) being processed by the CPU, (2) waiting for CPU processing, (3) engaging in I/O activity, or (4) waiting for I/O facilities. The CPU is busy when it is processing a job. The CPU is inactive when all job segments in core are either undergoing I/O activities or waiting for I/O facilities.

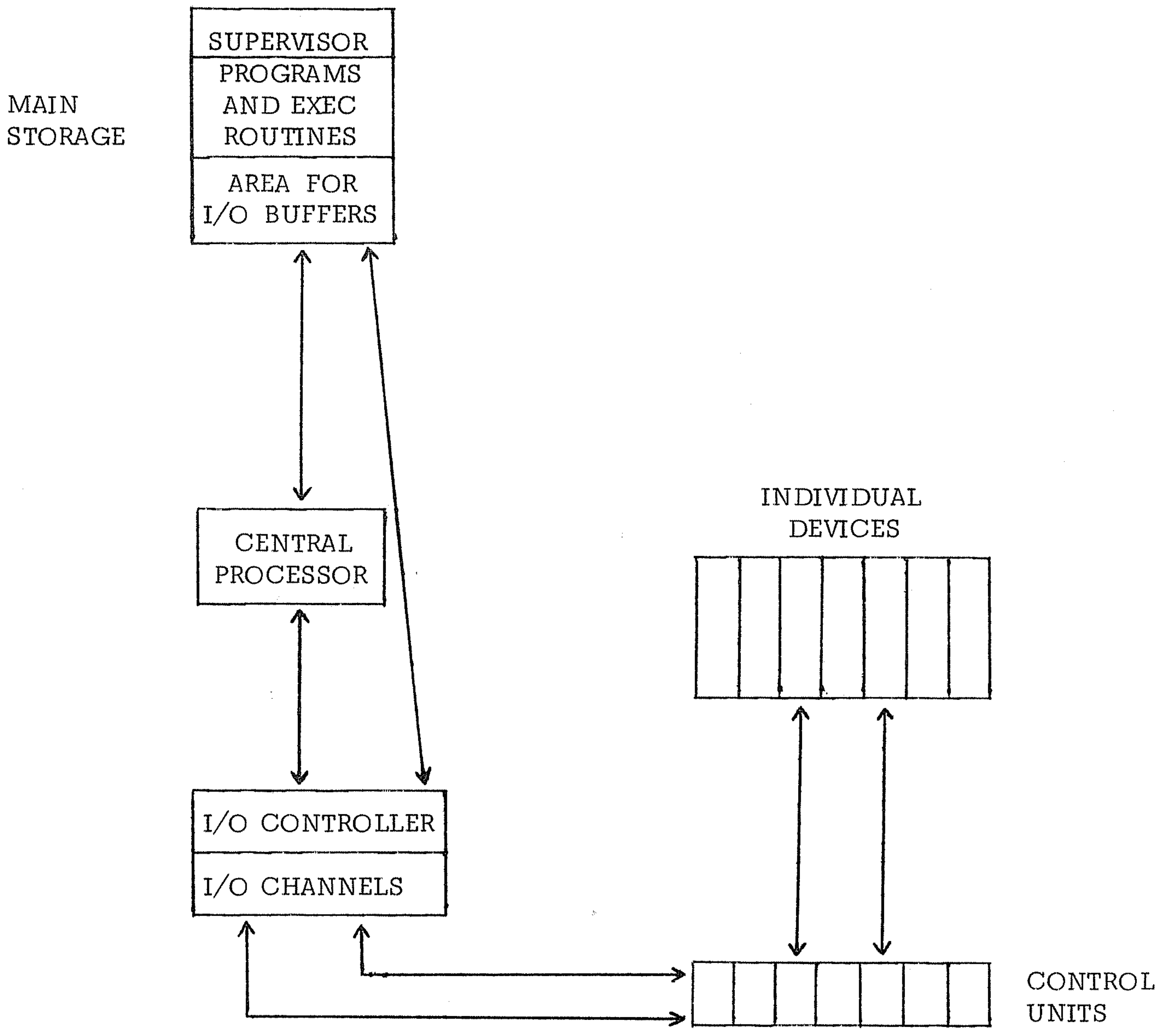


FIGURE 1. A Computer Subsystem

The performance criterion used in this study is the index of CPU productivity as defined in [7]:

$$\text{CPU productivity} = \frac{\text{Expected CPU busy period}}{\text{Expected CPU busy period} + \text{expected CPU idle period}} \quad (1)$$

This is the long-run fraction of time the CPU is busy. The success of the multiprogramming system is measured by the value of this index. In Figure 2, the expected CPU busy and idle periods are shown for four job segments residing in core simultaneously; a basic unit of w words is used as the buffer size.

It is clear from (1) and Figure 2 that CPU productivity is improved if for each job segment, the expected compute period may be increased at a rate faster than that of the expected I/O period. The I/O completion time distribution is a function dependent on the buffer size being used for I/O activities and the number of job segments in core simultaneously. This distribution is derived and analyzed in Section III.1 for direct access storage devices. The expected compute period generally increases or decreases at the same rate as the input buffer size used. With some exceptions*, an

*For example, user programs accessing libraries and their own private files. These I/O operations use separate buffers and would not be affected by changing input buffer sizes.

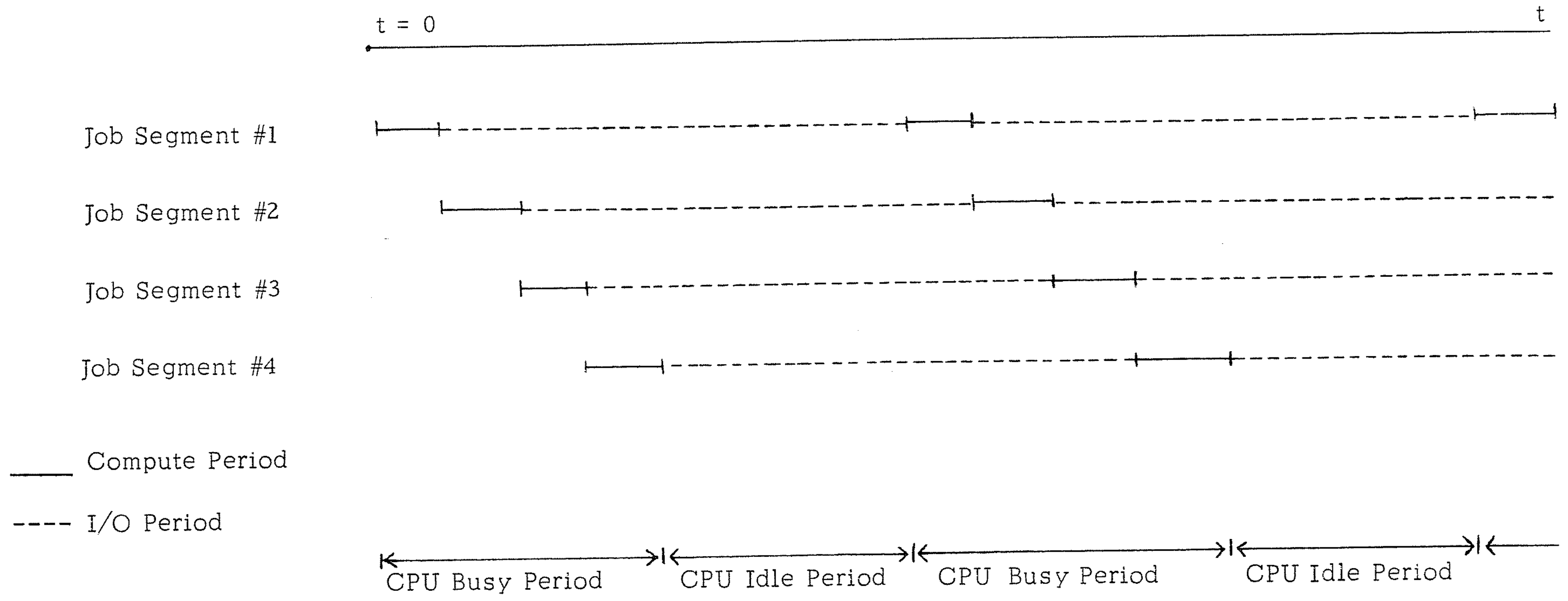


FIGURE 2

Expected CPU Busy and Idle Periods for 4 Job Segments in Core with Page Size = w Words increase in the input buffer size, say doubling it, will in effect double the compute period by delaying the demand for the next logical input buffer. A decrease in the input buffer size will hasten the demand for the next logical input buffer.

In Figure 3, a possible effect of increasing data input buffer size, say to 2w words, on the expected CPU busy and idle periods is shown for the four job segments in Figure 2.

It must be noted that an increase in input buffer size causes a corresponding decrease in the number of job segments which can reside in main storage simultaneously. This will significantly affect the expected CPU busy time and hence the index of CPU productivity. This constraint plays heavily in the optimization problem analyzed in the next section.

SPOOLing techniques have been provided most large-scale multiprogramming systems. Under this scheme, I/O data from the card reader and to the line printer are put on disk or drum prior to transfer to or from core during execution. The reasons for using this procedure are: (1) a better utilization is achieved for the devices concerned; (2) priority queueing is made possible; and more directly related to the model here, (3) I/O delays during execution are reduced considerably because faster devices are used with larger buffer sizes.

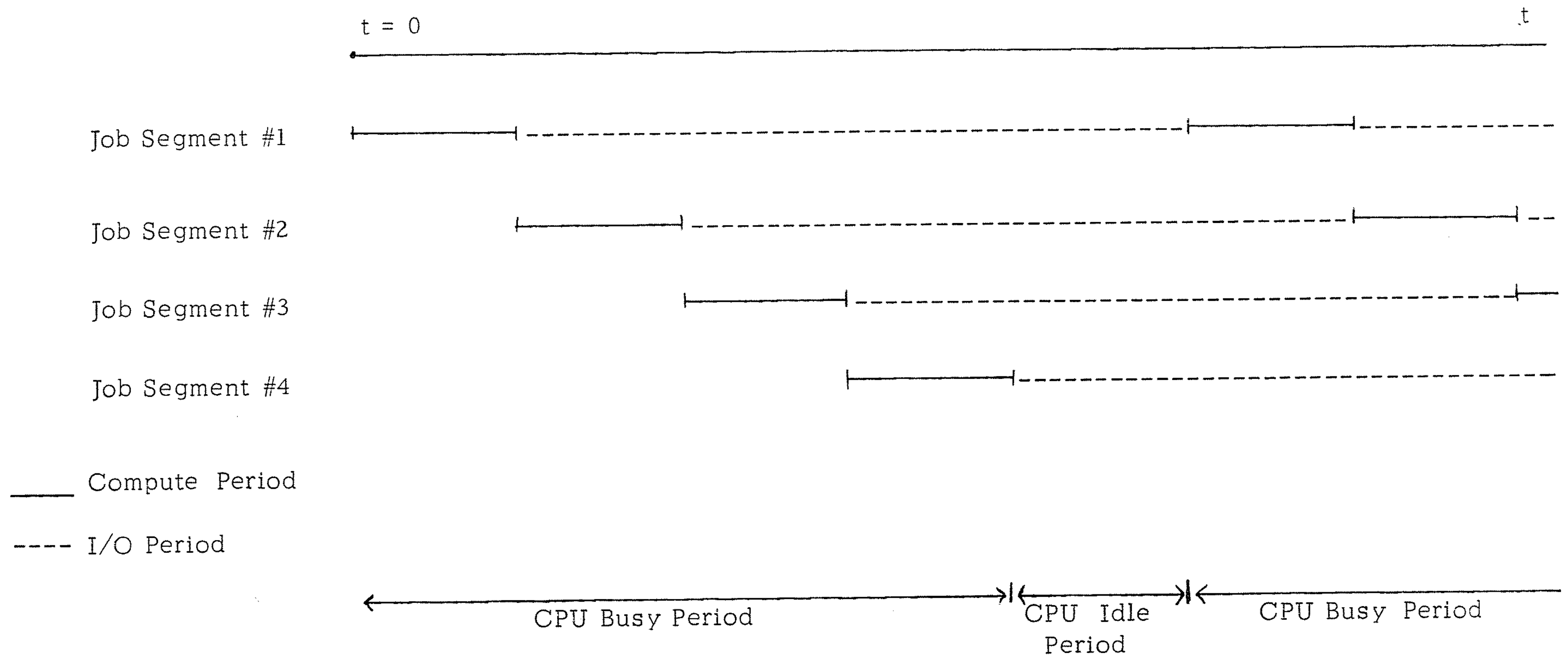


FIGURE 3.

Expected CPU Busy and Idle Periods for 4 Job Segments in Core with Page Size = 2w Words

Figures 4 and 5 show the effect SPOOLing might have on the same four job segments illustrated in Figure 2 and 3. While the CPU busy period remains constant, the reduction in I/O completion time shortens the CPU idle period thus increasing the index of CPU productivity.

It is important to observe that SPOOLing causes a significant increase in the total number of I/O operations. Input data from the card reader must first be brought into core before their transfer to disk or drum. Similarly, output data residing in disk or drum must be brought into core again prior to transmission to the line printer. However, the advantages of SPOOLing noted above usually outweigh the burden of these additional I/O operations.

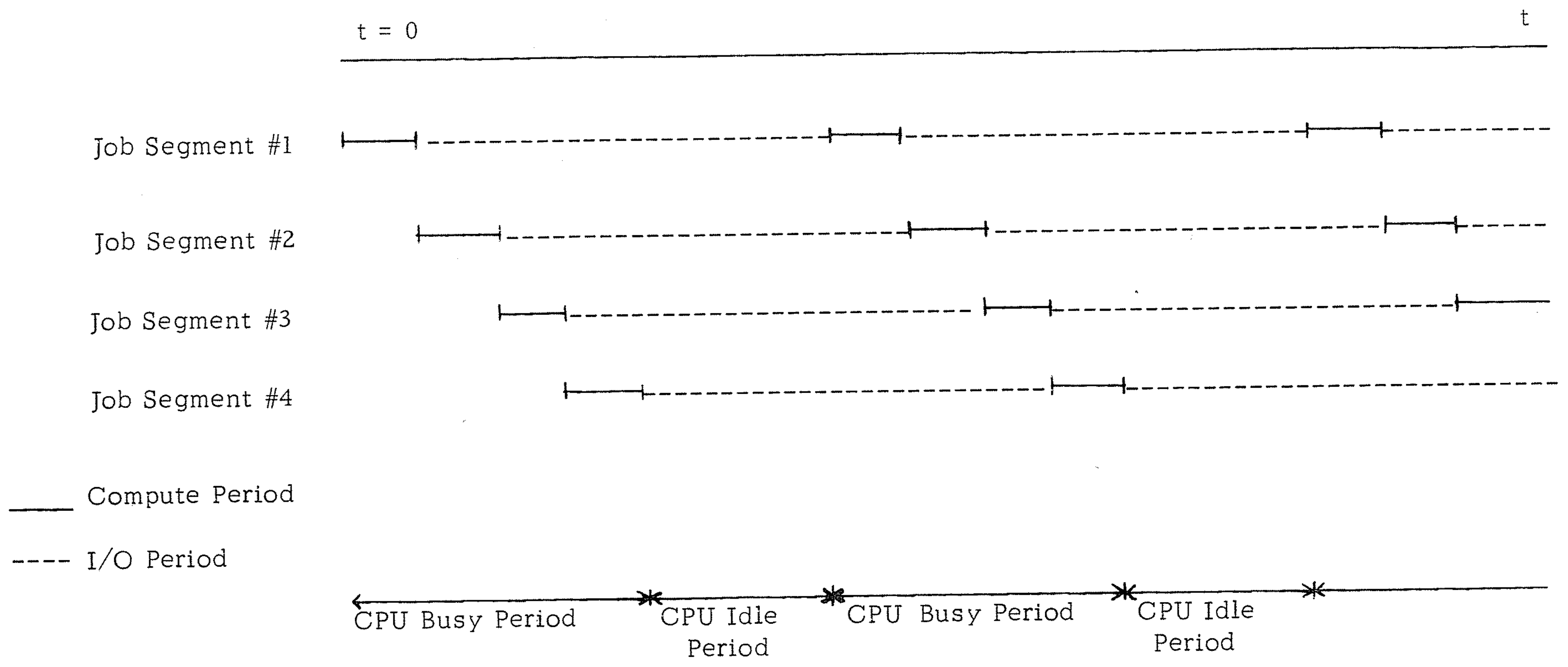


FIGURE 4

Expected CPU Busy and Idle Periods for 4 Job Segments in Core with Page Size = w Words with SPOOLing

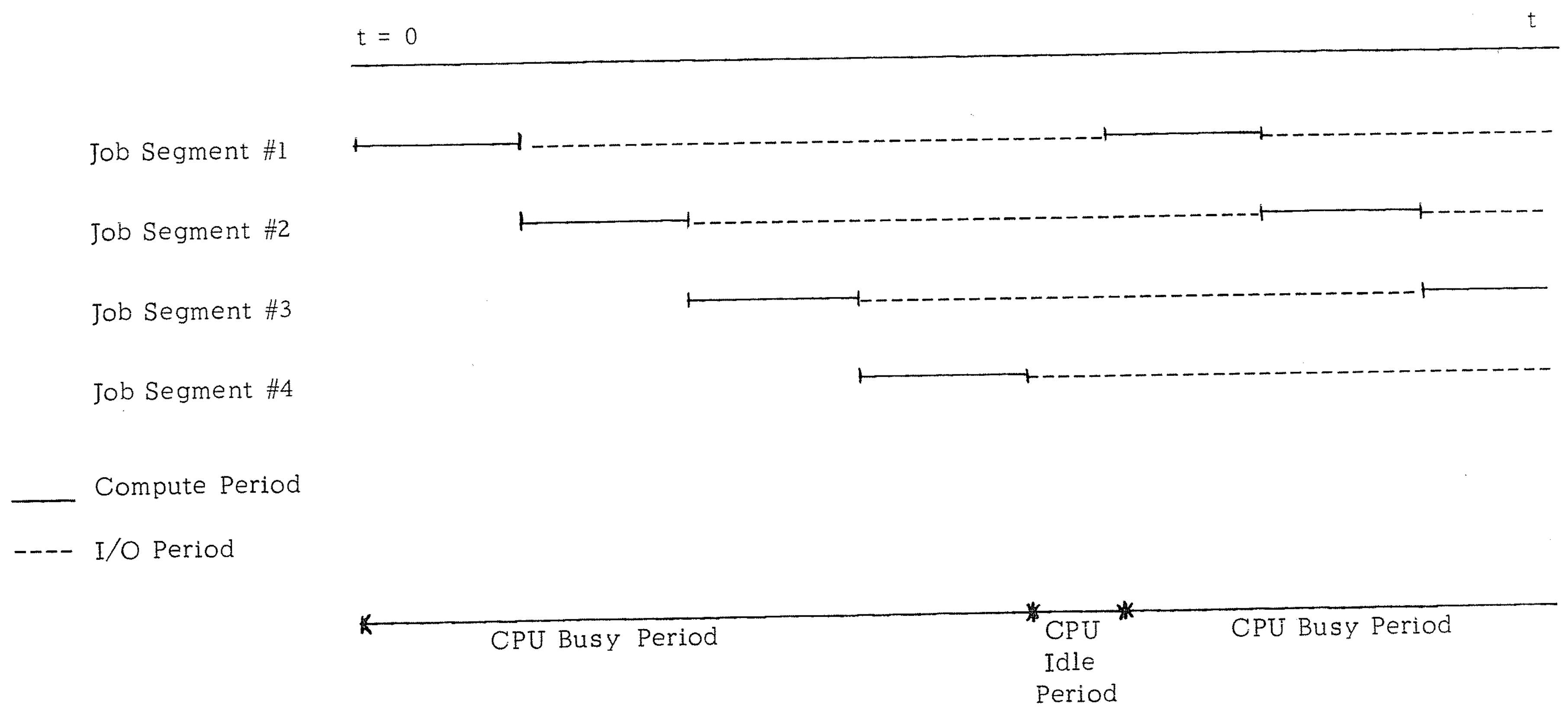


FIGURE 5

Expected CPU Busy and Idle Periods for 4 Job Segments in Core with Page Size = 2w Words with SPOOLing.

III. THE MATHEMATICAL MODEL

An important factor in the mathematical model is I/O completion time. This consists of the total time required to complete an I/O operation from the instant a request is made. By far the biggest obstacle to efficient I/O operation is that file memory devices--being mechanical in nature--are not totally random access. This means that each request may experience a variable positioning delay before the actual information transfer can begin. I/O completion time has the following components:

(1) wait in queue; measured from the moment a request enters the queue of a specified mass storage unit to the moment it is chosen for service.

(2) seek time; the mechanical delay due to the positioning of a read/write head to the addressed track. This delay is a function of the length of the move.

(3) rotational delay; time taken in accessing the sector addressed.

(4) transfer time; the time required to read or write one buffer of information from or to file memory.

1. <u>I/O Completion Time Distribution Function</u>

Standard distributions such as the negative exponential distribution are generally used to represent completion time distributions in queueing models; however, this is often unrealistic. Completion time distribution functions for various I/O devices have been derived in a number of papers. In particular, Pinkerton [14] has given a description and analysis of paging with the IBM 2301 Drum in a general-purpose time-sharing system. The analysis and experimental work on I/O in this study are based mainly on the UNIVAC FASTRAND II movable head drum. This differs from [14] mainly in that a head positioning time (seek time) must be included in most I/O completion times.

FASTRAND II units employ 64 read/write heads moving laterally over 192 recording <u>tracks</u>. Each track may be considered to be situated on the surface of a <u>cylinder</u>. The subsystem positions all of the heads in a drum unit with one movement of its positioning mechanism in an average time of 57 milliseconds. The maximum head positioning time is 86 milliseconds over all the tracks, and the minimum time is 30 milliseconds. All data on a FASTRAND subsystem is recorded in 28-word groups known as <u>sectors</u>.

In the analysis that follows, the assumptions are made that (1) all I/O transfers consist of one buffer of information and are processed sequentially through a single channel, (2) I/O transfers are distributed uniformly over all s drum sectors, (3) transfer time of the information (28 words) lying within one sector is taken as one time unit, (4) a linear relation exists between seek time and length of the seek, (5) seek time is not a function of individual cylinders, (6) a FIFO queue discipline (first-in, first out) is used with the drum sector queues, and (7) unless otherwise specified, the capacity of one buffer of information is the part of a track lying within one sector. By (4) and (5), seek time may be expressed as

$$\text{seek time} = c\,|i-j|$$

for cylinder i to cylinder j where c is a constant. The results derived in [14] are modified for this purpose. Rotational delays are implicitly considered in both formulations.

LEMMA 1. Given that (i) maximum head positioning time is p time units, (ii) the sector associated with a request is known, and (iii) k requests precede the arriving one on the sector queue, then the conditional distribution for completion time is

$$\begin{aligned} F(k,t) &= \text{Prob}[\tau \le t\,|\,k] \\ &= \text{(i)}\quad 0 \text{ if } t < ks + 1 \\ &\quad \text{(ii)}\quad 1 \text{ if } t > (k+1)(s+p)+1 \\ &\quad \text{(iii)}\quad (t - ks - 1)/(s + (k+1)p) \text{ otherwise} \end{aligned} \qquad (2)$$

Proof: Each drum revolution takes s time units. The minimum time required to service $k + 1$ requests is k drum revolutions with zero head positioning delay plus one time unit. The maximum time is $k + 1$ drum revolutions with maximum head positioning delay for each operation plus one time unit. The completion time of the $(k + 1)$th request is treated as if it were distributed uniformly between the minimum and maximum values. (See Section III.4 (ii), p. 32 for more discussion on this.)

LEMMA 2. The probability that a given sector queue contains exactly k entries, given that n transfer requests are enqueued in the entire system, is

$$G(n,k) = C(n,k)(s - 1)^{n - k}/s^{n} \tag{3}$$

where $C(n,k)$ denotes the binomial coefficient.

Proof: The k entries in the sector queue can be chosen from n in $C(n,k)$ different ways, and the remaining $n - k$ requests can be found on the other $s - 1$ queues in $(s - 1)^{n - k}$ ways. The total number of distinct configurations is of course s^{n}.

Since the given sector queues must contain k entries for some value of k between 0 and n, we have also that

$$\sum_{k=0}^{n} G(n,k) = 1 \tag{4}$$

THEOREM 1. Given n prior requests to the drum I/O channel, the unconditional distribution function for the completion time is

$$H(n,t) = \text{Prob}\,[\tau \le t] = \sum_{k=0}^{n} G(n,k)\,F(k,t) \tag{5}$$

<u>Proof</u>: An arriving request is assigned at random to a sector queue in which k requests are already enqueued with probability G(n,k) and n requests are already waiting in the whole system. Its completion time in this case is F(k,t) where k ranges from 0 to n.

THEOREM 2. The mean value M(n) of H(n,t) is given by

$$(s + p + 2n + 2)/2 + np/2s \tag{6}$$

<u>Proof</u>: By definition, $M(n) = \int_0^\infty t \;\; dH(n,t)$.

Applying (4) and integrating,

$$M(n) = \int_0^\infty td \sum_{k=0}^{n} G(n,k)\,F(k,t)$$

$$= \sum_{k=0}^{n} G(n,k) \int_0^{(n+1)(s+p)+1} t \;\; dF(k,t)$$

$$= \sum_{k=0}^{n} G(n,k) \left\{ \int_0^{ks+1} 0 \, dt + \int_{ks+1}^{(k+1)(s+p)+1} t/(s + (k+1)p) \, dt + \int_{(k+1)(s+p)+1}^{\infty} 0 \, dt \right\}$$

$$= \sum_{k=0}^{n} G(n,k) \ \{2(ks + 1) + (k + 1)p + s\}/2$$

$$= \sum_{k=0}^{n} G(n,k)\{k(2s + p) + p + s + 2\}/2$$

$$= (s + p/2) \sum_{k=0}^{n} kG(n,k) + p/2 + s/2 + 1$$

$$= (1 + p/2s)\, n \sum_{k=0}^{n-1} G(n-1,k) + p/2 + s/2 + 1$$

$$= n + np/2s + p/2 + s/2 + 1$$

$$= (s + p + 2n + 2)/2 + np/2s$$

COROLLARY 1. The second moment of the drum completion time distribution is

$$s^2/3 + s + 2sn + n^2 + n + 1 + p\phi \tag{7}$$

where $\phi = 1/3s^2 \, (pn^2 + 3sn^2 - np + 3snp + 8ns^2 + 2s^3 + s^2p + 3s^2)$ (8)

COROLLARY 2. The variance of the drum completion time distribution is

$$(s^2 + 12(s - 1)n + p\psi)/12 \tag{9}$$

where $\psi = 1/s^2 \, (pn^2 - 4np + 6snp + 17ns^2 + 2s^3 + s^2p - 12sn)$ (10)

In order to represent a configuration of several independent buffering mechanisms, the function $H(n,t)$ is simply modified. For example, with two drums on separate channels, and assuming that requests for buffers are equally distributed between the two, then

LEMMA 3. The probability $G^2(n,k)$ that a given sector queue contains exactly k entries, given that n transfer requests are enqueued in the entire system, is

$$G^2(n,k) = C(n,k)(2s-1)^{n-k}/(2s)^n \tag{11}$$

where $C(n,k)$ denotes the binomial coefficient.

<u>Proof</u>: As in Lemma 2.

THEOREM 3. Given n prior requests to the pair of drum I/O channels, the unconditional distribution function $H^2(n,t)$ for the completion time is

$$\begin{aligned} H^2(n,t) &= \text{Prob}\,[\tau \le t] \\ &= \sum_{k=0}^{n} G^2(n,k)\, F(k,t) \end{aligned} \tag{12}$$

<u>Proof</u>: As in Theorem 1.

A simple modification is also possible when buffer sizes are changed in multiples of the sector capacity. In this case, if the new size is a multiple j of the base size, then the transfer time for one buffer is j time units. However, for multi-sector operations, the effective transfer rate is reduced as a result of the following discontinuities in the actual transfer of data:

(1) the interval between reading or writing the last data bit in one sector and the first data bit in the next sector on the track;

(2) the interval between reading or writing the last data bit on the last sector of a track and the first data bit on the first sector of the track under the next head of the same drum.

In the UNIVAC FASTRAND II drums, (1) is approximately 106 microseconds while (2) is between 100 and 150 microseconds. These intervals are very small in comparison to the other quantities of I/O completion time. An additional assumption is made below that these time intervals do not exist.

LEMMA 4. Given the hypothesis of Lemma 1 and that buffer size is j times that of the sector capacity, then the conditional distribution function for the completion time is

$$
\begin{aligned}
F(k,t) &= \text{Prob}\,[\tau \le t \mid k] \\
&= \text{(i)} \quad 0 \text{ if } t < ks + j \\
&\ \text{(ii)} \quad 1 \text{ if } t > (k+1)(s+p) + j \\
&\ \text{(iii)} \quad (t - ks - j)/(s + (k+1)p) \text{ otherwise}
\end{aligned}
\tag{13}
$$

Proof: As in Lemma 1.

THEOREM 4. The mean value $M(n)$ of the modified unconditional distribution function $H(n,t) = \sum_{k=0}^{n} G(n,k)\ F(k,t)$ where $F(k,t)$ is defined in (13) is

$$(s + p + 2n + 2j)/2 + np/2s \tag{14}$$

Proof: Use (4), (5), and the definition of $M(n)$ and integrate as in Theorem 2.

COROLLARY 3. The second moment of the modified drum completion time distribution is

$$\begin{aligned} &s^2/3 + sj + 2sn + n^2 - n + 2nj + j^2 + p^2n^2/3s^2 + pn^2/s \\ &\quad - np^2/3s^2 + np^2/s + 8pn/3 + 2sp/3 + p^2/3 + pj \\ &\quad - np/s + npj/s \end{aligned} \tag{15}$$

COROLLARY 4. The variance of the modified drum completion time distribution is

$$\begin{aligned} &s^2/12 + sn - n + p^2n^2/12s^2 - np^2/3s^2 + np^2/2s + 17np/12 \\ &\quad + sp/6 + p^2/12 - np/s \end{aligned} \tag{16}$$

2. Formulation to the Problem

Objective Function. As stated earlier, our performance criterion is to maximize CPU productivity as defined in (1), p. 6.

Let t_1 and t_2 be the expected compute time and I/O time that each job segment undergoes alternately assuming that buffer size equals one sector capacity, w words. Then the index of CPU productivity γ is

$$\gamma = \frac{\text{Expected CPU busy period}}{\text{Expected CPU busy period + expected CPU idle period}}$$

$$= \frac{m\,t_1}{t_1 + t_2} \tag{17}$$

where m is the mean number of job segments residing in core simultaneously.

When the buffer size is augmented by a multiple j (to j x w words), the expected compute time becomes jt_1^* while the expected I/O time $t_2(j)$ is given by (14) in Theorem 4. Then the objective function is

$$\max\ \gamma = \frac{mjt_1}{jt_1 + t_2(j)} \tag{18}$$

Maximizing (18) naturally carries the constraints

$$\gamma \le 1$$
$$\text{and } m,\ j = \text{positive integers} \tag{19}$$

*This is not completely true as noted in Section II, p. 6 . A modification of this model is given in Section III.5 (ii), p. 35.

Equivalently, (18) and (19) may be rewritten as

$$\min \ \delta = jt_1 + t_2(j) - mjt_1 \tag{20}$$

with constraints

$$\begin{aligned} &\delta \geq 0 \\ &\text{and } m,\ j = \text{positive integers} \end{aligned} \tag{21}$$

where δ denotes the expected CPU idle time.

Additional Constraints. Consider the area of main storage used for the buffer and for storing the m job segments. Let this have a capacity of C words. Let L be the average length of the job segments. Then

$$C = mL + mjw + r \tag{22}$$

where the surplus variable r satisfies

$$0 \leq r < L + jw \tag{23}$$

This expresses the goal of storing as many tasks in the main memory as possible to keep the CPU busy.

Although buffers may be assigned dynamically as needed from a buffer pool (see Section III.5(iv)), this scheme is not used in the model: instead, each job segment is given its own private buffer. This formulation makes it possible to analyze more effectively the correlation between memory utilization and CPU productivity.

Moreover, the job segments at hand are often all engaged in I/O activities at the same time. Requiring a buffer pool to be sufficiently large to eliminate the probability of overflow would result in an equivalent formulation for the purpose here. In practice, the buffer pool used in EXEC 8 on the UNIVAC 1108 provides buffers for many other uses beyond basic I/O buffering. In other cases where buffer pools are used, the assumption is made that not all jobs will require I/O simultaneously.

The Problem. To summarize, the problem may be stated mathematically as

$$\min \begin{cases} \delta = jt_1 + t_2 - mjt_1 \\ 0 \ \text{if} \ \delta < 0 \end{cases} \tag{24}$$

subject to

$$\begin{array}{ll} \text{(i)} & C = mL + mjw + r \\ \text{(ii)} & r < L + jw \\ \text{(iii)} & t_2 = (s + p + 2(m - 1) + 2j)/2 + p(m - 1)/2s \\ \text{(iv)} & m, j \geq 1 \ \text{integers} \\ \text{(v)} & r \geq 0 \quad \text{integer} \end{array} \tag{25}$$

3. Solution Procedure

The optimization problem has the following characteristics:

(1) Quadratic constraints and objective function, thus a nonlinear

programming problem. (2) The variables are integers, thus an integer programming problem. It may be seen that the classical methods of nonlinear programming [8] and integer programming [9] do not apply here. While nonlinear programming methods do not restrict variables to be integer, standard integer programming techniques only deal with optimization problems where the objective function and constraints are linear. Integer programming algorithms with parabolic constraints have also been developed but are difficult to apply. A procedure is presented in this section that solves this optimization problem.

LEMMA 5. The objective function (24) is a concave function.

<u>Proof</u>:

The Hessian matrix of (24) is $\begin{pmatrix} 0 & -t_1 \\ -t_1 & 0 \end{pmatrix}$ which is negative definite for all $m, j \geq 1$ since

$$(m, j)\begin{pmatrix} 0 & -t_1 \\ -t_1 & 0 \end{pmatrix}\begin{pmatrix} m \\ j \end{pmatrix} = (-jt_1, -mt_1)\begin{pmatrix} m \\ j \end{pmatrix}$$

$$= -2mjt_1 < 0$$

Hence, (24) is strictly concave. (See Section 3-10 of [8].)

LEMMA 6. The constraints (25) are convex functions.

Proof:

The Hessian matrix of (i) is $\begin{pmatrix} 0 & w \\ w & 0 \end{pmatrix}$ which is positive definite for all m, j $\geq$ 1 since

$$(m, j) \begin{pmatrix} 0 & w \\ w & 0 \end{pmatrix} \begin{pmatrix} m \\ j \end{pmatrix} = (jw, mw) \begin{pmatrix} m \\ j \end{pmatrix}$$

$$= 2mjw > 0$$

Hence, (i) is strictly convex. (ii) - (v) are linear functions, and thus convex.

THEOREM 5. The feasible solution set of (25) is a closed convex set.

Proof: Follows from Lemma 6. (See Section 2-6 of [8].)

THEOREM 6. The optimal solution of (24) - (25) exists and is an extreme point of the feasible solution set.

Proof: The feasible solution set is closed and bounded from below. Also, the global minimum of the concave objective function is finite (zero). Thus the optimal solution is taken at one or more extreme points of the set (Page 93 of [8]).

The procedure described below find the optimal solution to the minimization problem. It is intended to store the maximum number of tasks, based on core space available, in the main memory while minimizing CPU idle time. In the model, this quantity is determined by the buffer size used. Hence starting with a feasible solution--the minimal buffer size--the objective function is evaluated. This value denotes CPU idle time and equals zero if full utilization is achieved. The buffer size is changed in multiples of the sector capacity. Since the core space in question is large in comparison to buffer and program sizes, increasing buffer size by one sector will either reduce the maximum number of tasks in core by only one or leave it unchanged. In both cases, the change is desirable when CPU idle time is reduced. On the other hand, an increase in objective function value could mean that only a local minimum is found. An analysis of the relation between the objective function and the constraints produces conditions which when satisfied implied global optimality. In the algorithm, local minimum points are found in sequence and tested for this optimality.

ALGORITHM 1.

Step 0. Initialize j at 1 and let

$$m = \left[\frac{C}{jw + L}\right] \tag{26}$$

where [y] means integer part of y.

Step 1. Evaluate δ; stop if $\delta \leq 0$, i.e., CPU idle time equals zero.

Step 2. Let $j =: j + 1$*, $m = [\frac{C}{jw + L}]$. If m remains unchanged, go to Step 3; otherwise proceed to Step 5.

Step 3. If $t_1 > 1/(m - 1)$ (27)

return to Step 1; otherwise, go to Step 4.

Step 4. Let $r = C - m((j - 1)w + L)$ (28)

$q = [r/mw]$ (29)

If $q(1 - (m - 1)t_1) - (p/2s + (m - (j - 1) - 2 - q)t_1) < 0$ (30)

go to Step 1; otherwise, stop.

Step 5. If $p/2s + ((m+1) - (j - 1) - 2)\, t_1 > 0$ (31)

return to Step 1; otherwise, go to Step 6.

Step 6. Let $r = C - m(jw + L)$ (32)

$q = [r/mw]$

If $(-p/2s - ((m+1) - (j-1) - 2)t_1 + q(1 - (m - 1)t_1) < 0$ (33)

go to Step 1; otherwise, stop.

The proof of the algorithm follows:

LEMMA 7. Given $m_1 = [\frac{C}{jw + L}]$ where [y] means the integer

*This means replace j by its original value plus one.

part of y,

and $m_2 = [\frac{C}{(j + q)w + L}] = m_1$,

$m_3 = [\frac{C}{(j + q + 1)w + L}] = m_1 - 1$

then $q = [r/m_1 w]$ where $r = C - m_1 (jw + L)$. (34)

Proof: $m_1 = m_2$ implies that

$$[\frac{C}{jw + L}] = [\frac{C}{(j + q)w + L}] = m$$

or $C = m(jw + L) + r_0$ where $0 \leq r_0 < jw + L$

and $C = m((j + q)w + L) + r_q$ where $0 \leq r_q < (j + q)w + L$

thus $r_0 \geq r_0 - r_q$

$= C - m(jw + L) - C + m((j + q)w + L)$

$= mwq$

the maximum value of q where $r_0 \geq mwq$ is (34).

THEOREM 7. Algorithm 1 finds the optimal solution of (24) - (25).

Proof: Due to (i) and (ii) of (25), m is strictly determined for any value of j by (26). Since $C >> w$, then increasing j by 1 will either not change the value of m or cause it to decrease by 1.

Case (i). $j =: j + 1$, m unchanged: The objective function δ is affected by an amount of

$$1 - (m - 1)t_1 \tag{35}$$

which is negative if (27) holds. Thus, it would be beneficial to make the changes.

Case (ii). $j =: j + 1$, $m =: m - 1$: The objective function δ is affected by an amount of

$$\begin{aligned} 1 - (1 + p/2s) - ((m - 2)(j + 1) - (m - 1)j)t_1 \\ = -p/2s - (m - j - 2)t_1 \end{aligned} \tag{36}$$

If (36) is negative, the changes should then be made.

Suppose (27) does not hold for Case (i) and j may be increased by at most q without affecting m, then letting $j =: j + q + 1$ will increase δ by

$$q(1 - (m - 1)t_1 > 0 \tag{37}$$

due to (35) and decrease it by

$$\begin{aligned} p/2s + ((m - 2)(j + q + 1) - (m - 1)(j + q))t_1 \\ = p/2s + (m - j - 2 - q)t_1 \end{aligned} \tag{38}$$

Thus the changes would be beneficial and should be made only if the total effect is negative. By Lemma 7, q is found to be (29).

Suppose (31) does not hold for Case (ii) and j may be increased by at most q more without affecting m again, then letting $j =: j + q + 1$ will increase δ by the amount in (36) and decrease it by $q(((m - 1) - 1) t_1 - 1)$. The changes should be made only if the total effect decreases δ. Again by Lemma 7, q is determined by (29).

When at any time, (27) and (31) both do not hold, then the algorithm stops and the optimal solution is found. It is noted that (35) and (36) are increasing functions as j increases and m decreases. Thus once either (27) or (31) does not hold, they will remain that way.

4. Limitations

(i) In the probability model, the control variables (i.e. buffer size and number of programs in core) are functions of various parameters such as program size and CPU time. When the value of one or more of the parameters is changed, it will in general be true that the optimal values of the control variables will also change. The algorithm here and standard techniques for solving such problems provide an optimal solution only for one specified set of values of the parameters. Using these methods, one cannot obtain explicit expressions for a set of optimal values of the control variables as functions of all the parameters. A detailed sensitivity analysis to determine how the optimal values of the control variables change as various parameters are changed (singly or in different combinations) could require considerable computational effort.

As it turns out, some of the parameters appearing in the problem must be treated as deterministic ones. Consequently, the standard procedure is to optimize the expected value of the objective function. This procedure belongs to a class of optimization techniques known as stochastic programming [8]. Such methods are avoided here due to the difficulties involved in their solutions. Instead, a simulation model is used for sensitivity analysis with results presented and discussed in Section V.

(ii) The FASTRAND II mass storage unit being a movable head storage device has similar behavior to a movable arm disk. Typically, the average seek time of such device is much greater than its average rotational delay. Hence in general, I/O requests are sorted first by cylinder (or set of cylinders under heads in the same position) and then by sector. However, the difference between the average seek time and the average rotational delay for FASTRAND is very small. Furthermore, since the sector queues are usually empty or not very long under EXEC 8, it is adequate to sort the I/O requests by sector in this case.

Each FASTRAND unit has 64 read/write heads moving laterally over 192 recording tracks, or 3 cylinders per head. The corresponding probabilities of a seek of 0, 1, or 2 tracks are $\frac{1}{3}$, $\frac{4}{9}$, and $\frac{2}{9}$ respectively, by no means a uniform distribution. (See Figure 6). Neither is the seek time uniformly distributed between 0 and the maximum value of 86 milliseconds (See Figure 7). But the shapes of these two distributions counteract each other so in this case, a uniform approximation is satisfactory.

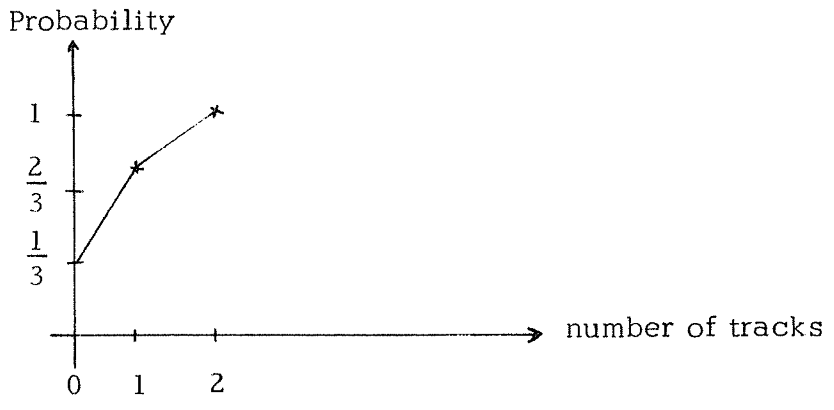


FIGURE 6

Cumulative Distribution Function for the Number of Tracks in a Seek

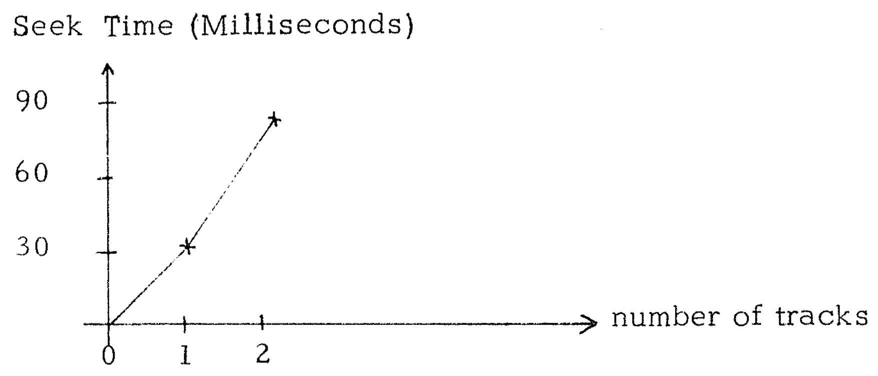


FIGURE 7

Distribution of Seek Time vs Number of Tracks in a Seek

5. Generalizations

(i) When several independent I/O mechanisms are used, e.g. two drums on separate channels, the model may be modified easily. Theorem 3 gives the completion time distribution function for multiple drums when requests for buffers are equally distributed among the drums. For uneven distribution of I/O requests, the objective function may be rewritten as

$$= (1 - m)jt_1 + \sum_{i=2}^{k} \phi_i t_i \qquad (39)$$

where each t_i gives the expected completion time of a particular I/O mechanism and ϕ_i its corresponding probability of being utilized. ϕ_i is easily represented by

$$\phi_i = \frac{\text{expected number of I/O requests for mechanism } i}{\text{expected total number of I/O requests}} \qquad (40)$$

(ii) An improvement on the model is possible and desirable when the number of I/O operations unaffected by a change in the input buffer size becomes great. Let γ be the fraction of I/O operations belonging to this class. Then the index of CPU productivity γ becomes

$$\gamma = \frac{(1 - f)\, mjt_1 + fmt_1}{(1 - f)\, jt_1 + ft_1 + t_2(j)} \qquad (41)$$

Equivalently, the problem becomes one of minimizing the expected CPU idle time δ where

$$\begin{aligned} \delta &= (1 - f)jt_1 + ft_1 + t_2(j) - (1 - f)mjt_1 - fmt_1 \\ &= jt_1 + t_2(j) - mjt_1 + f(m - 1)(j-1)t_1 \end{aligned} \qquad (42)$$

To show that an optimal solution still exists for the modified problem as in Theorem 6, it is necessary only to show that the objective function remains concave.

LEMMA 8. The objective function (42) is a concave function.

Proof: The Hessian matrix of (42) is $\begin{pmatrix} 0 & -t_1 + ft_1 \\ -t_1 + ft_1 & 0 \end{pmatrix}$ which is negative definite for all $m, j \geq 1$ since

$$(m,j)\begin{pmatrix}0 & -(1-f)t_1\\ -(1-f)t_1 & 0\end{pmatrix}\begin{pmatrix}m\\ j\end{pmatrix} = (-j(1-f)t_1,\ -m(1-f)t_1)\binom{m}{j}$$

$$= -2mj(1-f)t_1 < 0$$

Hence, (42) is strictly concave.

THEOREM 8. The optimal solution of (42) and (25) exists and is an extreme point of the feasible solution set.

Proof: As in Theorem 6.

(iii) It is assumed in the model that buffer size is increased only by a multiple j of a standard capacity, i.e. that part of a track lying within one sector. This restriction can be easily removed by setting the sector size w to one. j then becomes the variable that determines the number of words in a buffer.

(iv) Within the UNIVAC 1108 Exec 8 operating system is EXPOOL, a core resident element that contains a buffer pool and two routines to maintain this pool. All system tables, queues, and control words are also located in EXPOOL. In the allocation of buffer areas, a scheme must be adopted for allocating and releasing blocks of memory, maintaining a list of available blocks, and extending the buffer pool when it nears depletion. Basic schemes for dynamic allocation along with algorithms for implementation have been well defined in the computer science literature [10, 12].

A study is undertaken in [13] on two basic algorithms for the allocation of variable size buffers. This study shows under what operating environments one method might compare favorably with another. Buffer pool memory maps resulting from simulations of the two schemes are given to indicate the degree of memory fragmentation caused. In some cases, the problem may become so severe that although there is sufficient space to satisfy the buffer requests, the space is fragmented so there is insufficient contiguous space.

The variability of the size of buffer requests is the main cause of fragmentation. In addition, blocks of memory are released in increments; hence, a page of $j \times w$ words may not be released all at once but rather w words at a time for j times. In Figure 8, a buffer of 8 sectors is allocated to a certain I/O request at time t_0. Although the I/O operation is completed only at time t_2, each of the sectors is released upon completion of the read/write on that sector. An estimate of buffer space available and size of buffer pool necessary to prevent overflow must take all these factors into consideration.

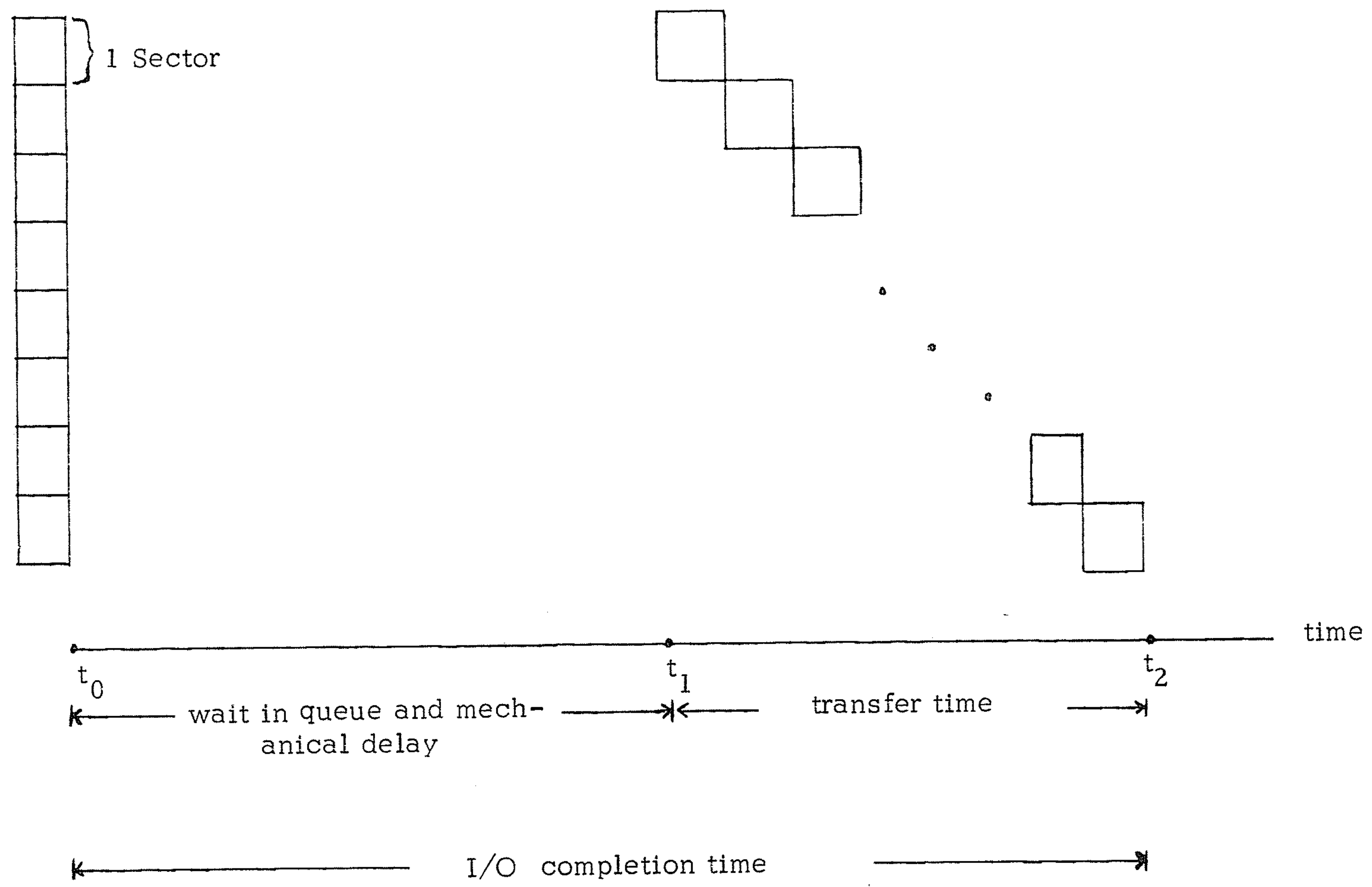


FIGURE 8.

The Incremental Release of Parts of a Buffer

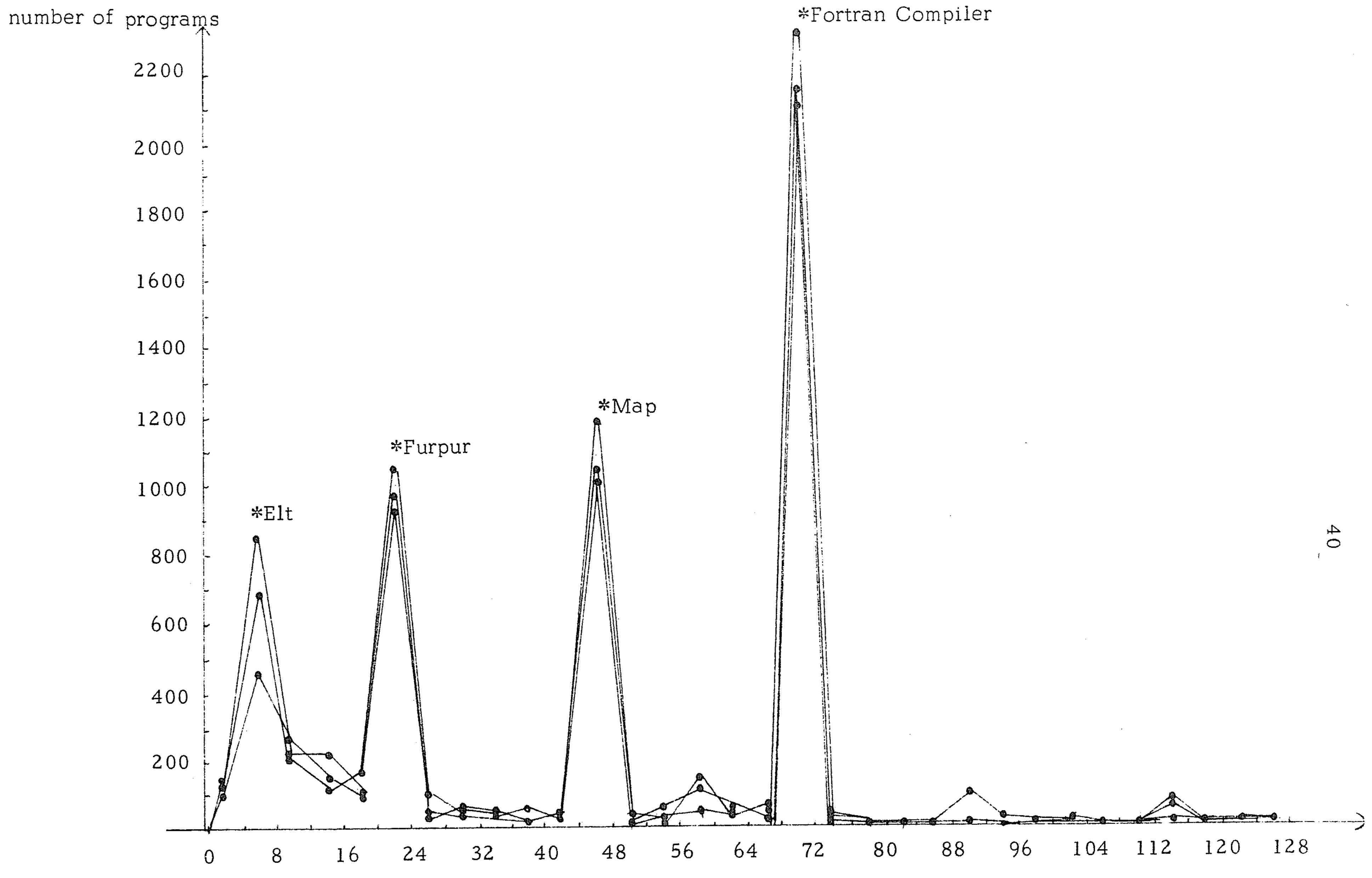
IV. EXPERIMENTAL WORK

1. Data

The statistics of jobs run at the UWCC in five randomly chosen days are compiled and used for analysis here. These data are also used for the simulation model in Section V. Core utilization by user programs vary from 1 block of 512 words to 128 blocks. This distribution of program sizes is by no means even. (See Figure 9.) In fact, local peaks are detectable and are due largely to the popularly used programs such as the FORTRAN compiler.

Figure 10 shows the distributions of the number of jobs versus compute time given in seconds for the five sample days' runs. The compute times are expressed exponentially in powers of 10.

Values of other parameters are also gathered. These include number of jobs, mean number of programs per job, mean compute time, mean I/O time, number of I/O requests, average request time, number of queue requests, and average queue time. The number of I/O requests is further broken down into the requests for each I/O mechanisms which include the FH Drums, FASTRAND, and tapes. (See Table 1 and 2.) Furthermore, a summary of core utilization from a sample day's run is given in Table 3.



*dominating executive routines that cause peaks in the distributions.

FIGURE 9

Program Size Distributions (in 512-Word Blocks)

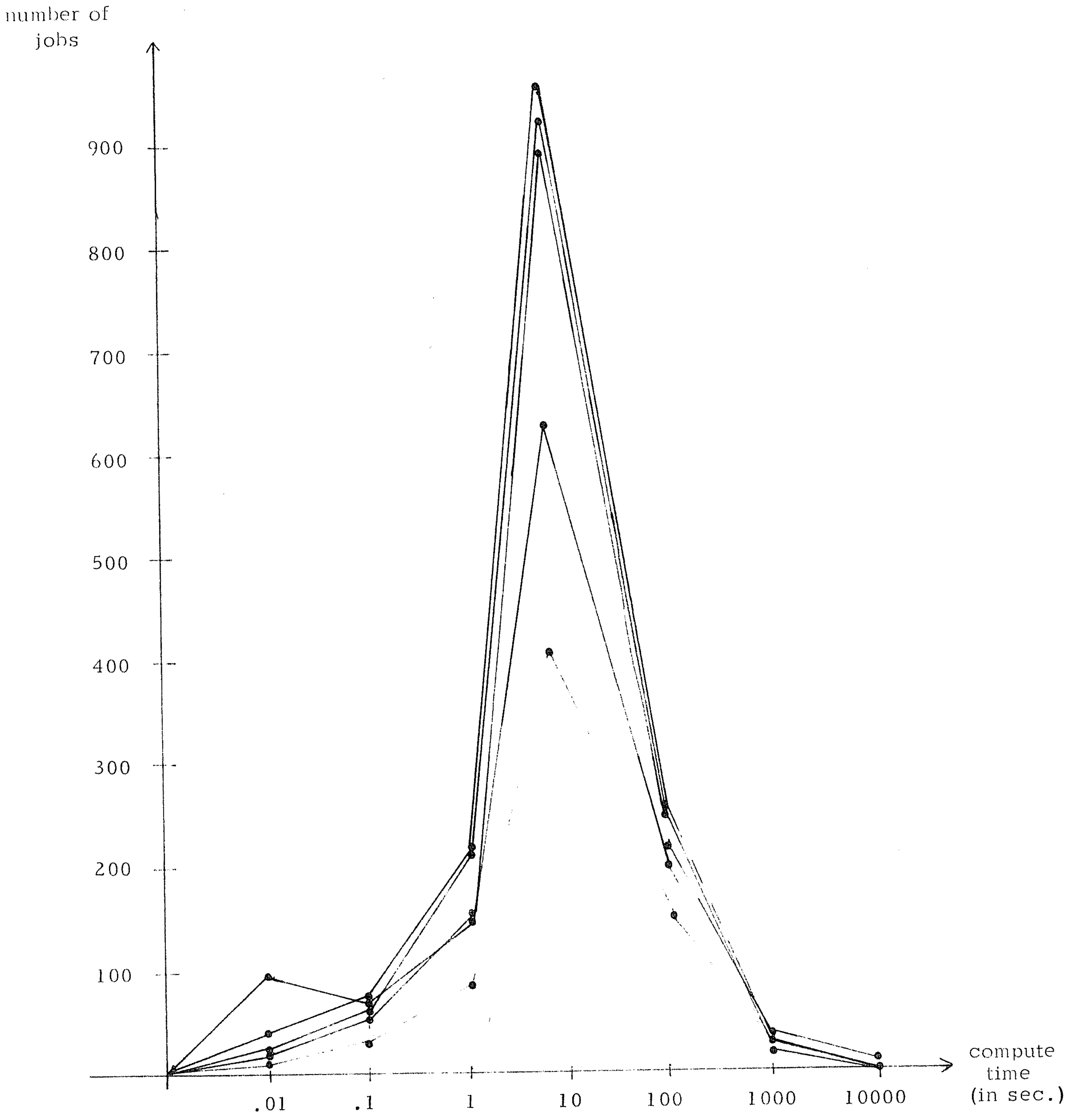


FIGURE 10

Compute Time Distributions (in Seconds) Using an Exponential Scale

TABLE 1. Production Environment for a Sample Day's Run

System Overhead	Throughput
64.0 Percent	105.9 Jobs/Hr.

Number of Jobs	Mean Compute Time	Mean I/O Time
503 Jobs	12.05 Sec.	66.97 Sec.

	Billed Time	Exec 8 Time	Dead-locked Time	Idle Time	Total Time	Channel Active Time
Seconds:	6060.28	4856.08	5907.32	271.07	17094.75	33685.76
Of Total:	35.58	28.48	34.68	1.68		

TABLE 2. Summary of Statistics from a Sample Day's Run

Channel of Unit	Active Time (Seconds)	Number of I/O Requests	Average Request Time (Seconds)	Queue Time (Seconds)	Number of Queue Requests	Average Queue Time (Seconds)	Idle Loop (Seconds)
FH Drums	1817.1	144090	.013				335.8
FH Drums	7312.8	472393	.015				1786.7
FASTRAND 1				4863.3	45044	.108	2517.9
FASTRAND 2	8077.4	109710	.074	6089.6	43424	.140	2086.8
8C Tapes	8029.6	301816	.027	19119.9	158822	.120	3433.7
FH-432 1				1044.3	89004	.012	
FH-432 2				102.3	14380	.007	
FH-432 3				126.8	9129	.014	
FH-432 4				1849.5	16606	.111	
FH-1782				.0	0	.000	
Totals	33685.8	1136904	.030	30072.8	247290	.12210160.9	

TABLE 3. Summary of Core Utilization from a Sample Day's Run

Core Size IS 256 K

Core Size	Time Seconds	Number Requests	Percent Utilization
40K- 48K	.0	0	.0 1
48K- 56K	.0	0	.0 1
56K- 64K	56.9	1231	.3 1*
64K- 72K	258.5	1633	1.3 1***
72K- 80K	81.5	1886	.4 1*
80K- 88K	212.5	3624	1.0 1**
88K- 96K	141.3	2496	.7 1**
96K-104K	166.9	4954	.8 1**
104K-112K	441.4	5108	2.2 1*****
112K-120K	136.9	3393	.7 1*
120K-128K	209.1	2670	1.0 1**
128K-136K	199.6	3336	1.0 1**
136K-144K	172.6	5025	.8 1**
144K-152K	480.3	8785	2.3 1*****
152K-160K	257.4	8523	1.3 1***
160K-168K	406.8	12879	2.0 1****
168K-176K	510.9	17486	2.5 1*****
176K-184K	933.9	25890	4.6 1**********
184K-192K	1147.5	34287	5.6 1************
192K-200K	1758.1	47561	8.6 1*******************
200K-208K	2647.9	65754	12.9 1****************************
208K-216K	2793.9	72519	13.6 1******************************
216K-224K	2684.4	68021	13.1 1*****************************
224K-232K	2358.6	60036	11.5 1*************************
232K-240K	1717.9	40547	8.4 1******************
240K-248K	694.5	15112	3.4 1*******
248K-256K	39.1	689	.2 1*
Totals	20508.6	513445	100.0

As mentioned earlier, SPOOLing is used in the multiprogramming system under study. This is accomplished on the UNIVAC 1108 system by the so-called symbiont routines. However, the additional I/O operations due to SPOOLing are not included in the scope of the study here since they would not be affected by changing data buffer size. The procedure used for gathering data does not exclude these I/O transfers and they must be deducted using an estimate based on number of cards read in and lines printed. Included in the data then are total number of runs, cards in and out per run, and lines out per run. An estimate of 50 characters per card and 80 characters per line is used. These figures are then converted to number of cards or lines per buffer.

2. Experimental Results

In addition to using the data for the simulation model in Section V, these figures are used as parameters in the mathematical model. The algorithm for finding the optimal buffer size is programmed in FORTRAN V.

In Figure 11, distributions of CPU productivity against buffer size are given. Although full utilization is achieved, this is not to be expected in the real situation because of overhead factors which are difficult to include in the mathematical model; such as additional I/O operations due to program swapping and system I/O, other uses of the buffer area, etc. The parts of the graphs that exceed the full utilization line could be represented by utilization of a second CPU. These are intentionally drawn to illustrate where the true optimum points lie. For all practical purposes, a particular buffer size where CPU use is fully achieved is no better or worse than another with the same utilization. However, if the overhead factors mentioned above had been included in the mathematical model, it is expected that the full utilization point will not be reached in the distributions. In that case, it will be necessary to see how one buffer size compare with the other.

The graphs show the distribution up to a buffer size of 30 sectors. CPU productivity at buffer sizes larger than this fluctuates back and forth without reaching again the real optimum points indicated. As should be expected, given a <u>fixed number of programs</u> in core, peaks of the distributions occur at points where the largest possible buffer size is used. A sudden drop in CPU productivity occurs when the buffer size is further increased, causing the number of programs in core to be reduced by one. Thus, many local maxima occur at various points in the distribution. The algorithm finds the true optimum.

From the five sample days' runs, optimal buffer sizes are found to be the following multiples of sectors: 12, 18, 15, 22, 10, or a mean of about 15. This is slightly less than twice the 224-word buffer used currently on the UNIVAC 1108 at the UWCC. The optimal buffer size for each day's run is determined based on parameters such as average program length and number of I/O requests which change dynamically for each day. A sensitivity analysis would better show how the optimal buffer size changes as various parameters are changed. This is done in the simulation model in the next section.

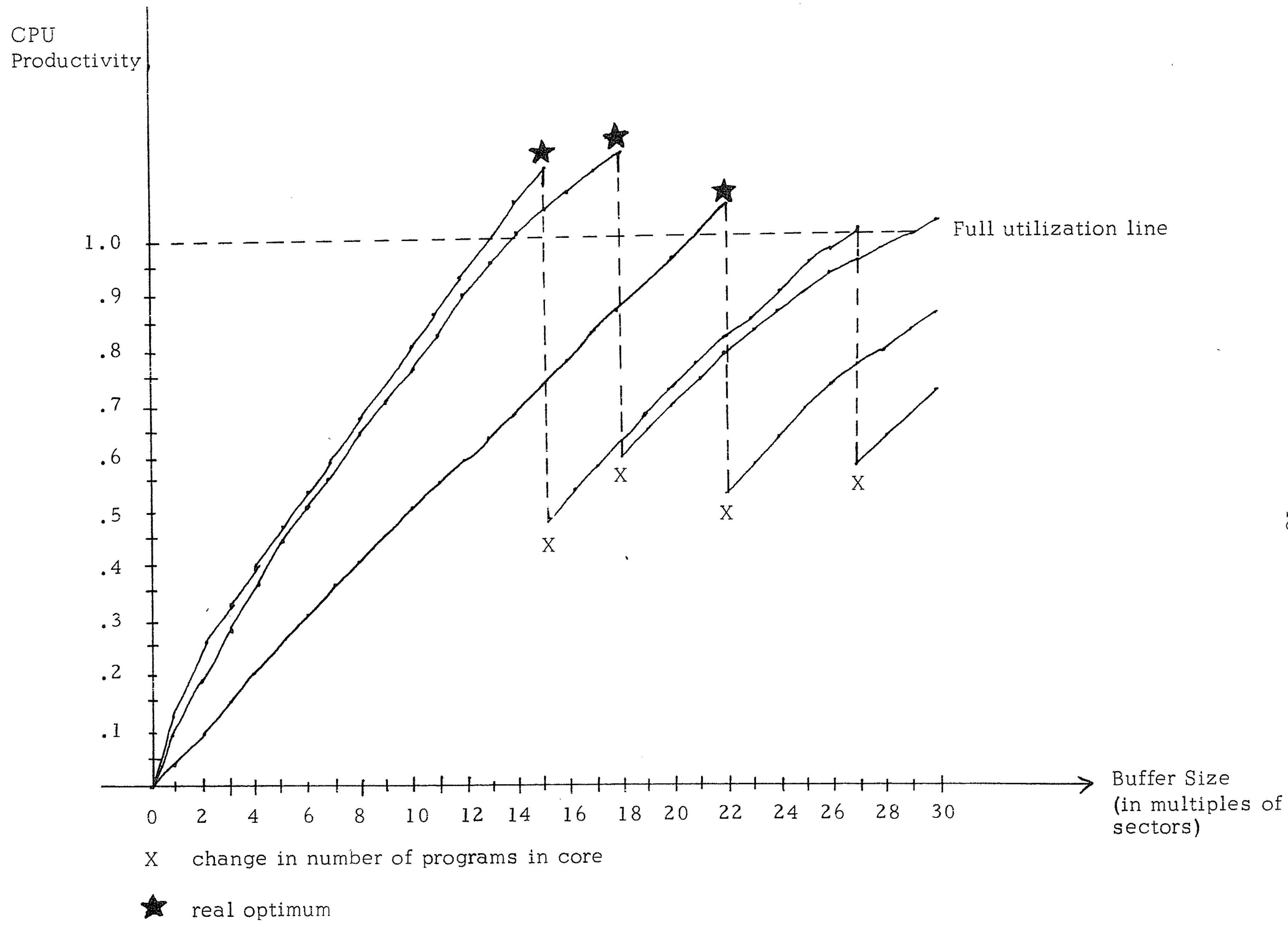


FIGURE 11

Distributionsof CPU Productivity vs Buffer Size

V. THE SIMULATION MODEL

This section discusses the implementation of the simulation model in the SIMSCRIPT language [11] and the data and results obtained from the simulations.

1. Implementation

In SIMSCRIPT, the status of the simulated system is defined in terms of what are called entities, attributes of entities, and sets of entities. Any type of unit to be independently identified in the simulation is called an entity. Each entity is in turn described by enumerating its particular attributes, and interrelationship among individual entities is done by grouping them into sets. The procedure for specifying status consists of defining each different type of entity, attribute, and set on the SIMSCRIPT Definition Form. The definition form for the simulated system here is given in Figure 12.

Briefly, the simulation model has the following features. The system consists of two main "first-in, first-out" queues--one for the CPU and another for I/O. At any instant, each task in core is in one of the two queues; it is either (1) executing or enqueued for execution, or (2) waiting for I/O facilities or completion of an I/O operation.

PROGRAMMER JAMES W. KHØ

PROBLEM

DATE

SIMSCRIPT DEFINITION FORM

01	TEMPORARY SYSTEM VARIABLES: Temporary and Event Notice Entities: T.N	Name	Record Size: Master	Satellite	Attributes: T.N	Name	Record	Word	Packing	Signed	Mode I.F	PERMANENT SYSTEM VARIABLES: Array Number	Name	Number of Subscripts or E	Packing	Signed	Mode I.F	Constant	Random: U.S S.L	SETS: Name	Number of Subscripts	LIFO	FIFO	Ranked	Attribute Used in Ranking	FUNCTIONS: Name	Mode I.F	Previously Compiled
+	T	JØB	8		T	LENTH		1	/		I	1	GRØUP	E	/					QUE	1		*			XIØT	F	
+					T	NIØ		2	/		I	2	FQUE	1	/		I											
+					T	PT		3	/		F	3	LQUE	1	/		I											
+					T	SQUE		4	/		I	4	RLN		/		I		US									
+					T	PTLFT		5	/	*	F	5	RPT		/		F		UL									
+					T	IØT		6	/		F	6	RNIØ		/		I		UL									
+					T	NIØL		7	/		I	7	RIØT		/		F		UL									
+									/			8	ASIC	0	/		I											
+	N	IØRT	2						/			9	BSIZE	0	/		I											
+									/			10	CPRØD	0	/		F											
+	N	PCRT	2						/			11	JBFIN	0	/		I											
+									/			12	IDLTM	0	/		F											
+	N	JBEND	4		N	JBND		3	/		I	13	BUSTM	0	/		F											
+									/			14	SIDL	0	/		F											
+	N	PCND	2						/			15	TNIØ	0	/		I											
+									/			16	MNIØ	0	/		I											
+	N	IØND	2						/			17	TCPT	0	/		F											
+									/			18	MCPT	0	/		F											
+									/			19	TIØT	0	/		F											
+									/			20	MIØT	0	/		F											
+									/			21	UTIL	0	/		F											
+									/			22	PTIME	0	/		F											
+									/			23	ITIME	0	/		F											
+									/			24	TLNTH	0	/		I											
+									/			25	MLNTH	0	/		I											

* NOT AVAILABLE TO EVENT NOTICES

	Prefix to set name		LIFO	FIFO	Ranked	
PREDECESSOR IN SET	P	SET NAME			✓	REQUIRED ATTRIBUTES FOR MEMBER ENTITIES
SUCCESSOR IN SET	S	SET NAME	✓	✓	✓	REQUIRED ATTRIBUTES FOR MEMBER ENTITIES
FIRST IN SET	F	SET NAME	✓	✓	✓	REQUIRED ATTRIBUTES FOR OWNER ENTITIES
LAST IN SET	L	SET NAME		✓	✓	REQUIRED ATTRIBUTES FOR OWNER ENTITIES

PREFIX TO SET NAME

Fig. 12 – SIMSCRIPT Definition Form

An overloaded operation is assumed in the sense that there is always a task ready to enter the system. A record of the tasks is kept during the entire duration of simulated time, beginning with the scheduling of the first I/O request. Once a job is brought into core, it undergoes a series of I/O and compute periods alternately, remaining in main memory until its completion.

Workload Description

As in a few other simulation models that measure the performance of existing systems, real data from the system itself was used. This is invaluable not only in the presentation of subtle correlations and interrelationships among the different characteristics of system operations, but because the simulated results can be compared more directly with the performance of the real system under the same conditions.

Each program to be processed is a temporary entity called a JOB with several attributes such as size of program, number of I/O requests, and processing time. The distribution of program size is obtained from real data of several days' runs and is given as a step function of 512-word blocks. Other parameters are similarly obtained and represented, and values of these attributes are generated randomly from their distributions.

Hardware Description

In the model, direct representation of physical devices occurs in the consideration of the drum rotational delay and positioning of read/write heads. A model parameter specifies the core storage. The loading of jobs into core and allocation of their buffer areas are also modelled.

System Parameters

While a change in the distribution of the workload description may alter the system performance, a number of parameters may be changed when desired to learn more about the behavior of the system. Examples of such parameters existing in the model are the amount of real memory available to a task, the size of the buffer area, and the limits on the length of time a task may continuously use the CPU. These values are specified so system attributes in the SIMSCRIPT program.

Operating System

Several events and subroutines are represented in the model for the operating system. A SIMSCRIPT timing routine permits the occurrence of both endogenous events (those caused by previous events within the simulation) and exogenous events (those introduced from outside the simulation). These include

(1) the interruption of a task currently using the CPU,

(2) the completion of an I/O activity,

(3) the bringing of new tasks into core to compete for the use of the CPU and I/O facilities, and

(4) the completion of a task.

Statistics Gathering

For the purpose of measuring the system performance and summarizing its behavior, statistics are gathered at intervals of the simulated time. These include CPU utilization in the time interval, accumulated CPU productivity, CPU idle and busy time. Other information such as mean program sizes, mean compute time, mean I/O time, number of I/O requests, and number of jobs finished are also gathered.

Although the length of the time intervals where data are to be collected may be easily changed, care must be taken in deciding when the performance of the system is to be measured. To allow transient effects to disappear, the initial period of simulation must be ignored. In SIMSCRIPT, the output of these data is easily generated by calling a report routine.

2. Simulation Results

In order to validate the theoretical results, a number of simulation runs were made using real workload data. Initially, runs were made using only buffer sizes of 8 and 12 sectors to investigate (1) the length of the initialization interval necessary for stabilization of the system, and (2) the length of the run segments at the end of which, statistics are gathered.

Over the relatively short real-time period for each run (60 minutes) which the data represents, there are significant differences both in the system performance and the workload variations. These are shown in Figures 13 to 17, which are discussed below.

Since parameters of the workload are described by distributions and their values determined by random look-up procedures, simulation runs were repeated for the same basic model using a different seed for generating random numbers.

In order to see how the system performs with different buffer sizes as various parameters (or their distributions) are changed, simulation runs are also made where only one parameter at a time is varied from the basic model. This is done with the distributions for CPU time and the number of I/O requests of individual programs.

For ease of making comparisons between runs where parameters are varied, the workload characteristics are shown for each run. A core size of 190K was used in all but Figure 17 where this value was doubled.

Figure 13 shows the basic model where the workload characteristics and system performance are given. Simulation results are obtained for buffer sizes of 8, 10, 12, and 16 sectors. These values are close to the optimum. Statistics are gathered at every interval of 5 minutes. The results of the first 10 minutes of simulated time (initialization interval) may be ignored. Of the four buffer sizes used, that of 12 sectors or 336 words gives the best system performance.

The result of the simulation run using a different random seed is given in Figure 14. The run is made only for the two buffer sizes that yield the best system performance, 10 and 12 sectors. Again, using buffer size of 12 sectors gives better CPU utilization, 0.83, compared to 0.78 for buffer size of 10 sectors.

In Figures 15 and 16, the results are presented for simulation runs where only one parameter at a time is varied from the basic model. In Figure 15, a different compute time distribution (taken from a different day's data on the 1108) is used causing the mean compute time to increase from 11.65 seconds per job to 14.72 seconds. The effect on system performance is as expected: a better CPU utilization. The distribution for the number of I/O requests is changed in Figure 16. The mean number of I/O requests increases from 2271 to 2710, causing CPU utilization to drop as expected.

Finally (see Figure 17), a simulation run is made for the basic model except that core size is doubled. This causes almost full utilization of the CPU. This should be expected since, on the average, twice the usual number of programs are permitted to reside in core and compete for CPU use.

Workload Characteristics	
Real-time period	60 minutes
Mean no. of I/O requests	2271
Mean I/O time	72.43 seconds
Mean compute time	11.65 seconds
Average size of program	20288 words

Index of CPU Productivity

Buffer size / Simulated Time (mins)	8 sectors	10 sectors	12 sectors	16 sectors
5	0.69	0.69	0.61	0.58
10	0.74	0.79	0.73	0.51
15	0.83	0.81	0.88	0 73
20	0.76	0.74	0.84	0.66
25	0.80	0.83	0.79	0.72
30	0.76	0.77	0.86	0.79
35	0.81	0.75	0.91	0.80
40	0.72	0.81	0.83	0.64
45	0.79	0.86	0.92	0.81
50	0.84	0.79	0.82	0.63
55	0.71	0.83	0.84	0.77
60	0.82	0.78	0.79	0.71
Average	0.77	0.80	0.82	0.66

FIGURE 13

Basic Simulation Data

Workload Characteristics	
Real-time period	60 minutes
Mean no. of I/O requests	2463
Mean I/O time	65.89 seconds
Mean compute time	12.15 seconds
Average size of program	30167 words

Index of CPU Productivity		
Buffer size / Simulated Time (mins)	10 sectors	12 sectors
5	0.63	0.58
10	0.77	0.70
15	0.82	0.84
20	0.74	0.76
25	0.81	0.89
30	0.77	0.93
35	0.73	0.88
40	0.83	0.89
45	0.80	0.94
50	0.76	0.79
55	0.85	0.82
60	0.79	0.90
Average	0.78	0.83

FIGURE 14

Simulation Data Using a Different Random Seed

Workload Characteristics	
Real-time period	60 minutes
Mean no. of I/O requests	2271
Mean I/O time	72.43 seconds
Mean compute time	14.72 seconds
Average size of program	20288 words

Index of CPU Productivity		
Buffer size / Simulated Time (mins)	10 sectors	12 sectors
5	0.71	0.64
10	0.79	0.76
15	0.86	0.88
20	0.81	0.94
25	0.87	0.90
30	0.78	0.93
35	0.83	0.85
40	0.77	0.92
45	0.84	0.87
50	0.88	0.94
55	0.75	0.83
60	0.90	0.90
Average	0.81	0.86

FIGURE 15

Simulation Data Using a Different Compute Time Distribution

Workload Characteristics	
Real-time period	60 minutes
Mean no. of I/O requests	2710
Mean I/O time	72.43 seconds
Mean compute time	11.65 seconds
Average size of program	20288 words

Index of CPU Productivity

Buffer size / Simulated Time (mins)	10 sectors	12 sectors
5	0.64	0.58
10	0.72	0.76
15	0.80	0.83
20	0.74	0.79
25	0.70	0.88
30	0.77	0.77
35	0.83	0.77
40	0.79	0.84
45	0.86	0.80
50	0.80	0.87
55	0.78	0.76
60	0.74	0.74
Average	0.76	0.77

Figure 16

Simulation Data Using a Different Distribution for the Number of I/O Requests

Workload Characteristics	
Real-time period	60 minutes
Mean no. of I/O requests	2271
Mean I/O time	72.43 seconds
Mean compute time	11.65 seconds
Average size of program	20288 words

Index of CPU Productivity

Buffer size / Simulated Time (mins)	10 sectors	12 sectors
5	0.93	0.89
10	0.96	0.92
15	1.00	0.98
20	1.00	1.00
25	1.00	1.00
30	1.00	1.00
35	1.00	0.98
40	1.00	1.00
45	0.97	1.00
50	1.00	1.00
55	1.00	1.00
60	0.96	1.00
Average	0.98	0.97

FIGURE 17

Simulation Data Using Double Core Size

VI. CONCLUSION

This study has concentrated on demonstrating the correlation of I/O buffer size used in a multiprogramming system to productivity of the CPU. The results of the work done shows that CPU utilization is sensitive to the size of the data buffer and system performance may be greatly improved by an optimal choice of this value. In the mathematical model, an algorithm is formulated for finding the optimal I/O buffer size given the workload and system parameters.

Analysis and numerical results are presented specifically for the UNIVAC 1108 with FASTRAND II storage units. Using real data from five days' runs at the UW Computing Center, results from both the mathematical and simulation model agree closely with the opinion of a system analyst at the UWCC. This points to the need to consider an increase in the data buffer size currently used in the multiprogramming of the UNIVAC 1108 system under a similar workload.

The analysis here is based on the use of a single buffer for each program in core. It may be useful to do a similar study for systems using two data buffers for each program where one buffer may be filled while the other is being processed (double buffering). Another technique involves the use of floating buffering where in addition to individual buffers, a number of floating buffers may be allocated to any of the programs in core. As systems grow larger and even more parallel processing is done, the analysis of these buffering techniques and the optimal selection of their buffer size will prove increasingly beneficial.

BIBLIOGRAPHY

1. Denning, P. J. "Effects of Scheduling on File Memory Operations," Proceedings of the Spring Joint Computer Conference, April, 1967.

2. Denning, P. J. "The Working Set Model for Program Behavior," Proceedings of the ACM Symposium on Operating System Principles, Gratlinburg, Tennessee, October, 1967.

3. Denning, P. J. "Virtual Memory," Computing Surveys, September, 1970.

4. Dennis, J. B. "Segmentation and the Design of Multiprogrammed Computer Systems," Journal of the ACM, October, 1965.

5. Draper, M. D., and Milton, R. C. UNIVAC 1108 Evaluation Plan, Technical Report 3, University of Wisconsin Computing Center, March, 1970.

6. Fife, D. W., and Smith, J. L. "Transmission Capacity of Disk Storage Systems with Concurrent Arm Positioning," IEEE Transactions on Computers EC-14, August, 1965.

7. Gaver, D. P., Jr. "Probability Models for Multiprogramming Computer Systems," Journal of the ACM, July, 1967.

8. Hadley, G. Nonlinear and Dynamic Programming, Addison-Wesley, 1964.

9. Hu, T. C., Integer Programming and Network Flows, Addison Wesley, 1969.

10. Knuth, D. E., The Art of Computer Programming, Vol. 1, Fundamental Algorithms, Addison-Wesley, 1968.

11. Markowitz, H. M., Hausner, B., and Karr, H. W., SIMSCRIPT, A Simulation Programming Language, The RAND Corporation, 1963.

12. Martin, J., Design of Real-Time Computer Systems, Princeton-Hall, 1967.

13. Minker, J., Crooke, S., and Yeh, J., Analysis of Data Processing Systems, Technical Report 69-99, University of Maryland Computer Science Center, December 1969.

14. Pinkerton, T. B., Program Behavior and Control in Virtual Storage Computer Systems, Technical Report 4, University of Michigan, April, 1968.

15. Randell, B., and Kuehner, C. J., "Dynamic Storage Allocation Systems," Proceedings of the First ACM Symposium on Operating System Principles, October, 1967.

16. Smith, J. L., "Multiprogramming under a Page on Demand Strategy," Communications of the ACM, October, 1967.

17. Sperry Rand Corporation UNIVAC 1108 Multi-processor System, 1968.

18. Weingarten, A., "The Analytical Design of Real-Time Disk Systems," Proceedings of IFIPS Conference, Booklet D, August, 1968.

19. Weingarten, A., "The Eschenbach Drum Scheme," Communications of the ACM, July, 1966.