A STUDY OF INFORMATION FLOW IN MULTIPLE-COMPUTER AND MULTIPLE-CONSOLE DATA PROCESSING SYSTEMS

K. B. Irani
A. W. Naylor
J. D. Foley
et al

The University of Michigan

FOREWORD

This final technical report was prepared by K. B. Irani, A. W. Naylor, J. D. Foley, J. W. Boyse, D. M. Coleman, D. R. Doll, V. L. Wallace, and A. M. Woolf of the University of Michigan, Systems Engineering Laboratory, Ann Arbor, Michigan 48104. This is annual report No. 3 under Contract AF 30(602)-3953, Project 5581, Task 02. The Rome Air Development Center Project Engineer was Lawrence W. Odell, EMIRA.

The distribution of this report is restricted by the U.S. Mutual Security Acts.

This report has been reviewed and is approved.

[illegible]

Approved: LAWRENCE W. ODELL
Project Engineer

[illegible]

Approved: A. E. STOLL, Colonel, USAF
Chief, Intel & Info Processing Division

FOR THE COMMANDER: [illegible]
IRVING J. GABELMAN
Chief, Advanced Studies Group

ABSTRACT

This report documents the achievements from January 1968 to March 1969 of continuing research into the application of Queueing Theory and Markov Decision Processes to the design and investigation of multiple-computer, multiple-user systems. A summary of the theoretical investigation conducted, the major conclusions reached, and some typical applications are included.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued

LIST OF TABLES

EVALUATION

This final report documents the last fourteen months of progress on a three year effort. Earlier progress was documented in RADC TR-67-345 dated July 1967 and RADC TR-68-57 dated March 1968. The work was performed under contract AF30(602)-3953 between the Rome Air Development Center, Griffiss AFB NY and The University of Michigan, Systems Engineering Laboratory, Ann Arbor, Michigan. The contract formed a portion of overall efforts to develop advanced data processing techniques and concepts under Air Force Project Number 5581.

This effort was undertaken to explore and develop better techniques for analyzing the performance, control, and structuring of complex data processing systems. Emphasis has been placed on developing and testing improved theories and modeling tools permiting rapid and accurate analysis of system designs. These developments have then been applied to various system design aspects in order to verify the accuracy of the models and to predict system performance. Of particular significance in this area has been the refinement and extension of the earlier Recursive Queue Analyzer (RQA) program (a computational tool based on Queueing and Markovian Modeling theories). This program has recently been shown to provide acceptable analytical solutions at rates significantly faster than standard "simulation" techniques. Additional work has been completed toward extending the program's capability for processing extremely complex models which are beyond the range of present analysis techniques. (See also RADC TR 69-134).

Major progress has also been made toward the formulation of basic design guidelines and principles which can be used by designers of large-scale, multiple-user computer systems. These principles are being developed through the analysis of models of various system aspects in order to determine optimum or near-optimum configuration and performance algorithms. Initial results in this area have been published in RADC TR-69-132. This work shows significant promise for improving the efficiency and effectiveness of future large-scale system design efforts and makes steps toward moving system's design from the realm of "art" to a refined "science". Of equal value is the promise of more cost-effective initial system implementations by applying these analytical techniques to proposed designs prior to actual development or production commitments. By these means, many alternate approaches and design trade-offs can be evaluated much more rapidly and cheaper than with present "built-it and try-it" or "full-scale simulation" techniques.

Additional effort to continue research in these areas is being supported by RADC under Contract F30602-69-C-0214.

Lawrence W. Odell

LAWRENCE W. ODELL
Project Engineer
Reconnaissance Engineering Section (EMIRA)
Rome Air Development Center, Griffiss AFB NY

1. INTRODUCTION

1.1 CONTRACT OBJECTIVE

The objective of this contract is to develop analytical techniques to assist system planners, designers, and evaluators in their determinations of the gross structure, control rules, and performance characteristics of multiple-computer, multiple-user systems, and to apply the techniques to large computer systems shared on a real-time basis by a large number of users, through consoles.

1.2 CONTRACT REQUIREMENTS

a. Continued exploration of Queueing Theory and Markov Decision Theory to improve their capability in determining scheduling and performance parameters for general on-line computer system problems.

b. Development of improved computer programs for carrying out analysis and optimization of system models, based on procedures resulting from the study of Markov Decision Theory and Queueing Theory.

c. Utilization of the computer programs and theoretical investigations to develop new conclusions and rules which can be used by persons performing initial design of real-time computer systems having a large number of user consoles. Such rules will allow system designers to more rapidly choose the type of hardware/

software system needed to fit a particular organization or problem.

d. Utilization of the computer programs to complete a queueing theory model of actual or proposed systems which will be used to project system performance and operating characteristics. The Markov Decision Technique will be applied to determine optimum scheduling and priority structures for such systems, and determine any system operating "bottlenecks" which are anticipated.

e. Final analysis of The University of Michigan Computing Center data, collected under contract AF 30(602)-2661, and use of the final conclusions of that analysis to analyze data from the other systems (when it is available). These data and conclusions will be used to provide parameters and validity-checks to other system models.

f. Statistical data will be gathered, where possible, from a number of running systems to gain a better understanding of user demand and processing structures and to confirm theoretical conclusions. The data from idem "d" above will also be used for this purpose.

g. A search for other theoretical approaches to the analysis, control and design of multiple computer systems will be pursued.

1.3 PROGRESS TOWARD CONTRACT OBJECTIVES

Progress has continued in both the theoretical and empirical areas. Of particular interest is the completion of three long-range projects concerning systems analysis and optimization. The first of these projects has successfully laid the theoretical basis for extending the concepts of numerical queueing analysis to infinite state Markov chains. This work is reported briefly here in Section 2.2, and in detail in a separate report [3]. The second project has resulted in a telecommunications network design procedure which is less restrictive and yields more nearly optimum results than previous methods. This work is described in Section 3.1 and in a report [1]. The third project concerns the optimum design of computer driven display systems. This is described briefly in Section 3.2 and in detail in a report [2].

Other shorter-term projects have been completed: Chapter 2 reports on conversion of our numerical queueing analysis program to a new computer, and the analysis of several proposed information storage and retrieval systems. Chapter 4 reports on data collected from The University of Michigan's time-sharing system. The data is analyzed, and some useful conclusions are drawn.

In addition to the completed long-range projects, there is one which is not yet completed. This concerns the optimal design of large memory systems, and is discussed in Section 3.3.

Finally, Appendix A presents a soon to be published paper which discusses multiprogramming in page-on-demand systems.

REFERENCES, CHAPTER 1

1. Doll, Dixon R., "The Efficient Utilization of Resources in Centralized Computer-Communication Network Design," SEL Technical Report No. 36, The University of Michigan, Systems Engineering Laboratory, April 1969.

2. Foley, James D., "Optimum Design of Computer Driven Display Systems," SEL Technical Report No. 34, Systems Engineering Laboratory, The University of Michigan, March 1969.

3. Wallace, Victor L., "The Solution of Quasi Birth and Death Processes Arising from Multiple Access Computer Systems," SEL Technical Report No. 35, Systems Engineering Laboratory, The University of Michigan, March 1969.

2. NUMERICAL QUEUEING THEORY AND THE ANALYSIS OF SYSTEM MODELS

Our past work has led to the development of numerical techniques for the analysis of complex queueing systems. The programs implementing these techniques have been converted to the University's new IBM 360/67 computer, as is described in Section 2.1. Unfortunately, these techniques are not adequate to study certain types of large computer systems. Section 2.2 reports on the extension of numerical analysis techniques to more (but not all) large computer systems.

The remaining three sections report on the analysis of several proposed systems and system models.

2.1 RECURSIVE QUEUE ANALYZER

The Recursive Queue Analyzer, or RQA [4], has been reprogrammed to run on the University's IBM 360/67 computer. Because of the greater computational capabilities of this new computer (as contrasted to the old IBM 7090), the new RQA can do more in less time than the old version. The most important consideration here is that the 360 version of RQA can handle system models having up to 32,000 states, as opposed to 5,000 states in the old version. Interestingly enough, the limit of 32,000 states can be easily increased even further!

2.2 THE NUMERICAL SOLUTION OF INFINITE MARKOV CHAINS

2.2.1 Introduction

Markovian models have found considerable application to the art of computer system design for two significant reasons. First, they have been applied because there is a sizable theory upon which to draw, and many powerful theoretical results which are available. Second, computer systems have come to a stage of evolution where they are predominant "high-traffic" systems and, as a consequence, the modeling and estimation of statistical performance has become a crucial aspect of their design.

However, there are limitations to the models of this type which are capable of analysis. The models which must be solved are often too complex for classical queueing theory, and often have too many states for modern numerical techniques [4]. The precision required over a wide range of parameter variations often rules out simulation as too costly.

The study of multiple-access computer systems and the so-called "computer utilities" are particularly difficult in this regard. Furthermore, as these systems grow larger and more complex (through servicing more user circuits, the introduction of multiprocessing, the use of satellite computers and data-concentrators, etc.) this situation can be expected to become worse. Complexity and largeness of state spaces will be the rule. Yet these are the very systems for which analysis of Markov chain models can be of greatest use to the system architect. Thus, new techniques are urgently needed.

A typical example of such a model will serve to make this need more concrete. Consider a multiple-access computer system with two thousand common-carrier trunk lines leading to a bank of fast processors operating on a large, multiprogrammed, paged main memory. Suppose that the memory is demand paged with a large bulk-core "virtual" memory. Suppose, finally, that we wish to study the relationship between the expected number of queued unserviced requests for computation and, say, the size of main core memory.

A queueing model which could be used for this study is illustrated diagrammatically in Figure 2-1, where circles represent queues and squares represent service-holding. This model anticipates that congestion in the channel to the bulk-core will be a major problem (see Appendix A), and so it details the paging operations in a manner similar to the model of Appendix A. However, arriving tasks in this case are queued in a "wait queue" which can, potentially, assume very large values. (Typically, for a system of this size it could contain an <u>average</u> of several hundred requests.) Because of this, the state space is very large and has been modeled as infinite, a consequence of assuming that the wait queue can assume any nonnegative value. A detailed description of this model is not vital to this discussion. The important thing is the form of the state space and the transition intensity matrix from which the equilibrium probabilities and expectations are to be calculated.

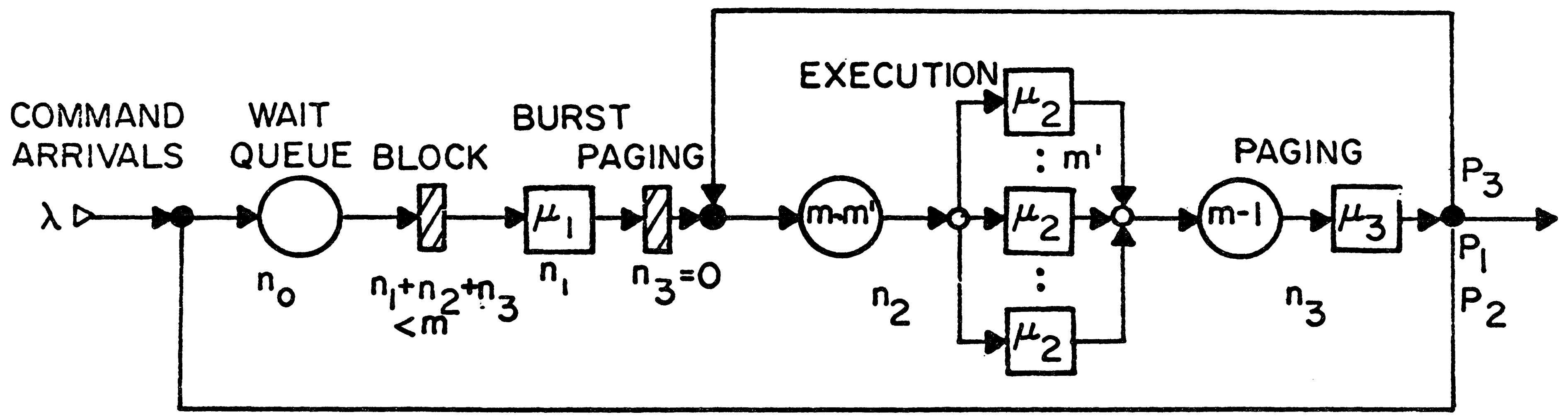


Figure 2-1

Illustrating a Model of a Multiple Access System

To make things as simple as possible for this illustration, we let the

Table 2.1

States of Figure 1-1, when m=2, and m'=1

	n_0	n_1	n_2	n_3		n_0	n_1	n_2	n_3		n_0	n_1	n_2	n_3
1.	0	0	0	2	10.	1	0	0	2	16.	2	0	0	2
2.	0	0	1	1	11.	1	0	1	1	17.	2	0	1	1
3.	0	0	2	0	12.	1	0	2	0	18.	2	0	2	0
4.	0	1	0	0	13.	1	1	0	0	19.	2	1	0	0
5.	0	1	0	1	14.	1	1	0	1	20.	2	1	0	1
6.	0	1	1	0	15.	1	1	1	0	21.	2	1	1	0
7.	0	0	0	0						22.	etc.			
8.	0	0	0	1						⋮				
9.	0	0	1	0										

Table 2. 2

The Transition Intensity Matrix of the Model of Figure 2-1

$$\left[\begin{array}{ccccccccc|cccccc|cccccc|cccccc|cccccc|c}
x & p_3 & 0 & 0 & p_2 & 0 & 0 & p_1 & 0 & \lambda & 0 & 0 & 0 & 0 & 0 \\
\mu_2 & x & p_3 & 0 & 0 & p_2 & 0 & 0 & p_1 & 0 & \lambda & 0 & 0 & 0 & 0 \\
0 & \mu_2 & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 & 0 \\
0 & 0 & 0 & x & 0 & 0 & 0 & 0 & \mu_1 & 0 & 0 & 0 & \lambda & 0 & 0 \\
0 & 0 & 0 & p_1 & x & p_3 & 0 & 0 & 0 & 0 & 0 & 0 & p_2 & \lambda & 0 \\
0 & 0 & \mu_1 & 0 & \mu_2 & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda \\
0 & 0 & 0 & \lambda & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & p_2 & \lambda & 0 & p_1 & x & p_3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \lambda & 0 & \mu_2 & x & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & p_1 & 0 & 0 & 0 & 0 & x & p_3 & 0 & 0 & p_2 & 0 & \lambda & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & p_1 & 0 & 0 & 0 & \mu_2 & x & p_3 & 0 & 0 & p_2 & 0 & \lambda & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mu_2 & x & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mu_1 & 0 & 0 & 0 & 0 & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_1 & x & p_3 & 0 & 0 & 0 & p_2 & \lambda & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mu_1 & 0 & \mu_2 & x & 0 & 0 & 0 & 0 & 0 & \lambda \\
\hline
 & & & & & & & & & 0 & 0 & 0 & 0 & p_1 & 0 & x & p_3 & 0 & 0 & p_2 & 0 & \lambda & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & & & 0 & 0 & 0 & 0 & 0 & p_1 & \mu_2 & x & p_3 & 0 & 0 & p_2 & 0 & \lambda & 0 & 0 & 0 & 0 \\
 & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mu_2 & x & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 & 0 \\
 & & & & & & & & & 0 & 0 & 0 & 0 & 0 & \mu_1 & 0 & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 \\
 & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_1 & x & p_3 & 0 & 0 & 0 & p_2 & \lambda & 0 \\
 & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mu_1 & 0 & \mu_2 & x & 0 & 0 & 0 & 0 & 0 & \lambda \\
\hline
 & & & & & & & & & & & & & & & 0 & 0 & 0 & 0 & p_1 & 0 & x & p_3 & 0 & 0 & p_2 & 0 & \lambda & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & p_1 & \mu_2 & x & p_3 & 0 & 0 & p_2 & 0 & \lambda & 0 & 0 & 0 & 0 \\
 & & & & & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mu_2 & x & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 & 0 \\
 & & & & & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & \mu_1 & 0 & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 & \lambda & 0 & 0 \\
 & & & & & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_1 & x & p_3 & 0 & 0 & 0 & p_2 & \lambda & 0 \\
 & & & & & & & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mu_1 & 0 & \mu_2 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda \\
\hline
 & & & & & & & & & & & & & & & & & & & & & & & & & & & \text{etc.} & & & & & &
\end{array}\right]$$

2.2.2 The QBD Process

Let m_o and m be positive integers, with

$$m_o \geq m. \tag{2.1}$$

Let $N = \{N(t),\ t \geq 0\}$ be a continuous parameter Markov chain with stationary transition probability functions continuous at t=0, and with state space $\mathcal{N}$, where

$$\mathcal{N} = \{<0,1>, <0,2>, \ldots, <0,m_o>, <1,1>, <1,2>, \ldots, <1,m>, <2,1>, \ldots, <2,m>, \ldots, \ldots\}. \tag{2.2}$$

The process N is called a QBD process if its transition intensities $\{u_{jk};\ j,k \in \mathcal{N}\}$, satisfy the properties:

(a) If $j = <j_1, j_2>$ and $k = <k_1, k_2>$ are two states for which $|j_1 - k_1| \geq 2$, then $u_{jk} = 0$.

(b) If $j = <j_1, j_2>$ and $k = <k_1, k_2>$ are two states for which $|j_1 - k_1| \leq 1$, if $(j_1 + k_1) \geq 3$, and if $j' = <j_1 - 1, j_2>$ and $k' = <k_1 - 1, k_2>$, then $u_{jk} = u_{j'k'}$.

A continuous-parameter Markov chain will also be called a QBD process if a mere one-one mapping of its states results in a QBD process. However, no loss in generality will result from consideration of only state spaces of the form (2.2), represented by the set $\mathcal{N}$.

The state space $\mathcal{N}$ is more graphically described as a two-dimensional integer space whose first dimension is countably infinite, whose second dimension is finite, and which has the configuration of Figure 2-2.

States in the set $\mathcal{N}_0 = \{<0,1>, <1,2>, \ldots, <0, m_0>\}$ will be called boundary states and the set $\mathcal{N}_0$ will be called the boundary of N.

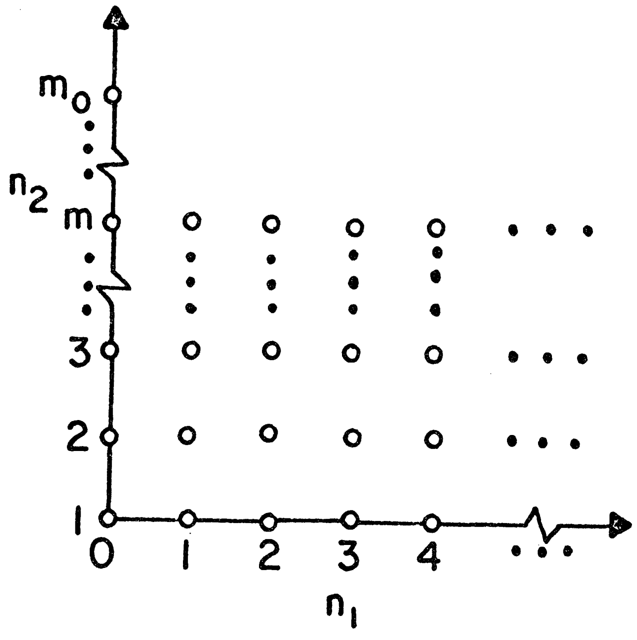


Figure 2-2 State Space

If the lexicographic ordering of the states used in equation (2. 2) is preserved, then the conditions (a) and (b) are equivalent to a specification that the transition intensity matrix $U = \{u_{jk}\ ;\ j,\ k \in \mathcal{N}\}$ is an infinite matrix which can nevertheless be displayed in the partitioned form

$$U = \begin{bmatrix} E & \hat{A} & & & & \\ \hat{C} & B & A & & 0 & \\ & C & B & A & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ 0 & & & & & \end{bmatrix} \qquad (2.\ 3)$$

where the submatrix E is an $m_o \times m_o$ matrix; where A, B, and C are $m \times m$ matrices; and where $\hat{A}$ and $\hat{C}$ are $m_o \times m$ and $m \times m_o$ matrices, respectively, and are further partitioned into the forms

$$\hat{A} = \begin{bmatrix} A \\ 0 \end{bmatrix} \tag{2.4}$$

and

$$\hat{C} = \begin{bmatrix} C & 0 \end{bmatrix} \tag{2.5}$$

Many QBD processes whose transition intensity matrices do not have the appearance of Equation (2. 3) can nevertheless be identified as QBD processes by simply regrouping the states and reidentifying them.

When $m_o = m = 1$, the QBD process is readily recognized to be a simple birth and death process. The "QBD" in the name "QBD process" is an acronym for the somewhat barbaric, though apt descriptor "quasi birth and death", in recognition of this fact. (The word "quasi" is frequently used in matrix theory in describing matrix forms created by replacing scalar elements by submatrices, as in "quasi-diagonal" and "quasi-triangular" matrices.)

In order to calculate an equilibrium distribution of a QBD process N, reasoning reminiscent of that commonly followed in solving birth and death processes can be followed. This reasoning and the calculation technique that results is essentially the same, in outline, as that proposed by Evans [1].

An equilibrium distribution is a sequence $\pi = \{\pi_k, k \in \mathcal{N}\}$ of nonnegative real numbers which satisfies the equation $\pi \mathcal{U} = \underline{0}$ and whose components have unit sum. If we let $\pi = [\phi_0, \phi_1, \phi_2, \ldots]$ be a partitioning of that sequence for which ϕ_0 is an m_o-component row vector, and $\phi_1, \phi_2, \ldots,$ are m-component row-vectors, then the equation $\pi \mathcal{U} = \underline{0}$ is equivalent to the equations

$$\phi_0 E + \phi_1 \hat{C} = \underline{0} \qquad (2.6a)$$

$$\phi_0 \hat{A} + \phi_1 B + \phi_2 C = \underline{0} \qquad (2.6b)$$

$$\phi_{n-1} A + \phi_n B + \phi_{n+1} C = \underline{0} \qquad n=2, 3, \ldots . \qquad (2.6c)$$

Now suppose that there is a solution $[\phi_0, \phi_1, \phi_2, \ldots]$ to Equations (2.6) of the form

$$\phi_1 = \phi_0 \hat{R} \qquad (2.7a)$$

$$\phi_n = \phi_1 R^{n-1} \qquad n=2, 3, \ldots, \qquad (2.7b)$$

where R is an $m \times m$ matrix, where $\hat{R}$ is an $m_o \times m$ matrix partitioned to the form

$$\hat{R} = \begin{bmatrix} R \\ \underset{\sim}{0} \end{bmatrix}, \qquad (2.8)$$

and where ϕ_0 is some row-vector whose components are not all zero. Then it develops that, by substitution of Equations (2.7) in Equation (2.6), the matrix R and the vector ϕ_0 must satisfy the equations

$$\phi_0(E + \hat{R}\hat{C}) = \underline{0} \tag{2.9a}$$

$$\phi_0(A + \hat{R}B + \hat{R}\,RC) = \underline{0} \tag{2.9b}$$

$$\phi_0\hat{R}R^{n-2}(A + RB + R^2C) = \underline{0}\,. \tag{2.9c}$$

Furthermore, since a solution of Equations (2.6) where $\phi_0, \phi_1, \phi_2, \ldots,$ have only nonnegative components is sought, ϕ_0 and R must guarantee that nonnegativity as well. If we can show that such ϕ_0 and R exist, and yield a solution π which is an equilibrium distribution, then the assumption of Equations (2.7) will be vindicated. The Equations (2.9) suggest a sufficient condition for such a solution to be found. This condition is that a matrix R exists having the properties:

(a) All entries of R are nonnegative.

(b) $E + \hat{R}\hat{C}$ is singular, and has a nonnegative, nonzero vector ϕ_0 in its null-space (so that Equation (2.9a) is satisfied).

(c) All entries of $A + RB + R^2C$ are zero.

(This latter property is reminiscent of the classical characteristic roots of a difference equation, while the first two are reminiscent of the application of boundary conditions to difference equation solutions.)

It is our purpose to show sufficient conditions that a matrix R does exist having the properties (a), (b) and (c). Remarkably, those conditions are very general, and include almost all interesting QBD processes. Under those conditions, an equilibrium distribution can be calculated by the four steps

(1) Determine a matrix R having properties (a), (b) and (c).

(2) Determine a vector ϕ_0.

(3) Normalize ϕ_0, if possible, so that π will have unit sum.

(4) Determine as many of the ϕ_n as desired using Equations (2.7a) and (2.7b).

It will be shown also that such a matrix R can be determined by an iterative process in every interesting case, and that ϕ_0 can also be calculated by known techniques.

Thus, the problem of determining an equilibrium distribution of a QBD process (which has an infinite state space) will be shown to reduce to several routine finite problems, each of which does not tax the memory or computing capabilities of a digital computer of moderate-to-large size.

In addition, results are given providing (1) necessary and sufficient conditions on R for an irreducible QBD process to be positive recurrent, (2) sufficient conditions for R and π to be unique, with π equal to the limiting distribution.

2.2.3 Boundary Leading QBD Processes

The sufficient conditions for the existence of the quadratic root R are satisfied by a very general type of QBD process, the "boundary leading" process.

Definition

A QBD process N is boundary leading if every nonboundary state leads to a boundary state.

This condition is a very weak one, and encompasses all irreducible QBD processes as well as a considerable variety of reducible ones.

Transition graphs of a number of simple boundary leading QBD processes are shown in Figure 2-3, demonstrating this variety. These graphs represent every state by a vertex, and every nonzero, non-diagonal transition intensity $u_{jk} (j \neq k)$ by a directed branch from state j to state k. The states are arranged as in Figure 2-2. Two-way paths in Figure 2-3 are shown by heavy lines, and boundary states by square vertices. These graphs are useful in visualizing the communication properties of particular QBD processes, since a state j "leads to" a state k if and only if there is a directed path in the graph which can be followed from j to k. Note that only the graphs (e) and (f) describe an irreducible process. The rest are reducible.

It should be particularly noted that the definition of boundary leading processes makes no mention of paths between boundary states. Thus, branches between square vertices in Figure 2-3 can be removed or added at will without altering the boundary leading character of the examples. This exercise offers still other interesting examples. The most interesting aspect of all the examples, however, is the way that the horizontal repetitiveness, characteristic of QBD processes, forces certain communication properties.

The assumption that N is boundary leading is so general that it automatically includes every QBD process which has a finite number of

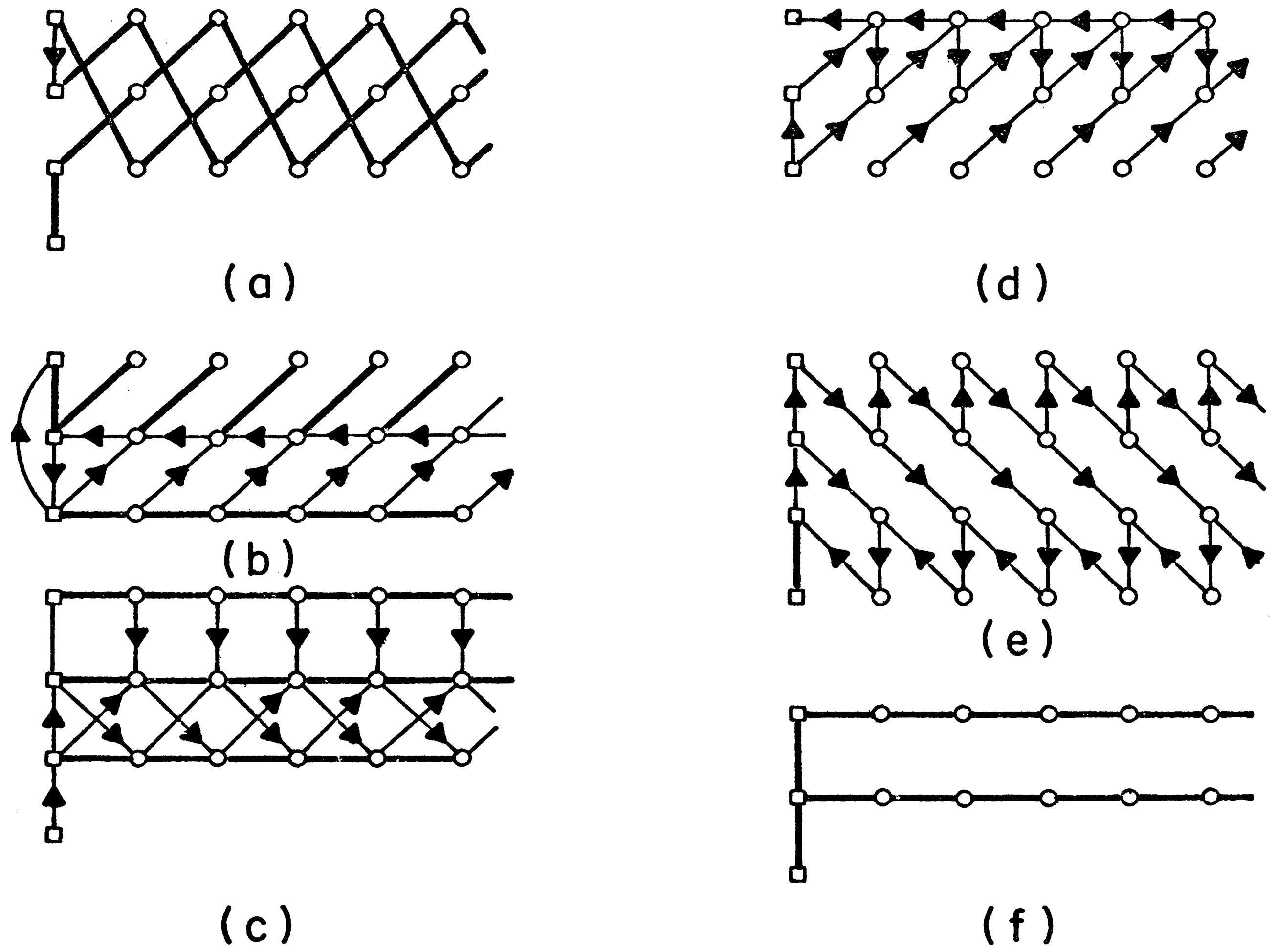


Figure 2-3

Graphs of Boundary Leading QBD-processes communicating classes. The converse is not true, however, since Figure 2-3(d) describes a boundary leading process which (degenerately) has an infinite number of communicating classes. The boundary leading QBD process has been singled out for study here because it represents one of the most general processes for which a quadratic root matrix R exists having the required properties.

2.2.4 Conclusions

It has been shown that, under common and predictable circumstances, the QBD processes lend themselves to expeditious numerical solution, and are therefore a useful class of stochastic models for computer system analysis (or other queueing application). The equilibrium distributions and other related measures are determined by a procedure which involves operations with only finite matrices, and which is analogous to procedures used in solution of the difference equation of simple birth and death processes.

This conclusion has been demonstrated by a comprehensive algebraic theory whose two main theoretical conclusions can be summarized as:

Conclusion 1

If a QBD process is boundary leading, then the matrix quadratic $A + XB + X^2C$ is, indeed, analogous to the characteristic equation for the birth and death process, having a root matrix R which generates the characteristic solution

$$\phi_n = \phi_1 R^{n-1} \qquad n=2,3,\ldots .$$

The m-vector ϕ_1 satisfies a boundary condition

$$\phi_0(E + \hat{R}\hat{C}) = \underline{0}$$

$$\phi_1 = \phi_0\hat{R},$$

and the equilibrium solution is given by

$$\pi = [\phi_0, \phi_1, \phi_2, \ldots]$$

The root matrix is in a known set $\mathcal{R}$. If π is convergent, the equilibrium distribution is merely a scalar multiple of π.

Conclusion 2

If the QBD process is irreducible and positive recurrent, then the root R can be found by a well-defined iteration process

$$S_{n+1} = T(S_n)$$

$$R = C(B + S_\infty)^{-1}$$

and the resulting solution π will be convergent. Conversely, if the process is irreducible and a root R is found for which the resulting π is convergent, then the process is positive recurrent.

Much more has been said, of course, but these results are the most significant and general. The theory presented is broad enough so that there are numerous specific questions about QBD processes which can be answered by intelligent use of the theorems, corollaries, and lemmas explicitly supplied. Thus, this theory provides a good base on which to build even greater usefulness.

2.3 RELATIVE PERFORMANCE ANALYSIS OF SELECTED COMPUTERS AND ASSOCIATED BULK STORAGE DEVICES

A very important use of system analysis techniques is the selection of hardware to be included in new or modified computer systems, and also to determine how the hardware is best used. Thus this and the next two sections discuss specific systems which have been analyzed. In this section, we discuss the computer and bulk storage needed in an information retrieval system. In a system of this type, two things are of concern: (1) Data and programs must be moved from peripheral bulk storage devices into and out of the core memory; and (2) while in the core memory, data must be manipulated by the central processing unit (CPU) by executing programs.

We shall use speed of response of the system as the performance criterion. The system performance is related to both the speed of the CPU and the speed at which data can be moved into and out of the core memory. If the CPU is in use 100% of the time and its speed is doubled, then if the CPU can still be kept busy 100% of the time, the system speed of response doubles. On the other hand, suppose the CPU is only in use 50% of the time because there are not enough programs and data in core memory available to keep it busy. In this case doubling the CPU speed will have little effect on system response. It will merely mean that the CPU will now be idle 75% rather than 50% of the time.

Thus two kinds of imbalance may occur in the system. If there is excess CPU capacity, increasing this capacity further will have little (if any) effect on system performance. If data can be transferred from bulk storage to core memory faster than the CPU can process it, then increasing this data transfer rate will have little effect on system performance. Of course, the relative CPU and bulk storage transfer rates required to maintain a balance vary depending on the type of problem the computer system is solving. In many scientific calculations, for example, there is much manipulation of a small quantity of data. Here the speed of the CPU should be high relative to the data transfer rate. In the system we are concerned with, however, the reverse is true. Large quantities of tactical intelligence data must be scanned for parameters of interest. This involves transferring a lot of data into the core memory but involves little computation on each record read in. Thus for a balanced system the relative CPU to data transfer rate can be much lower in this case.

2.3.1 System Specifications

The computer specifications we considered in this analysis are shown in Table 2-3. Note that the Execution Speed is the ratio of word size to core cycle time. The specifications of the bulk storage devices are shown in Table 2-4. Two storage systems are considered. One consists of 200 million bits of drum storage and is called the dual channel drum. From Figure 2-4 it can be seen that transfers can take place from the disk and

Table 2-3

Computer Specifications

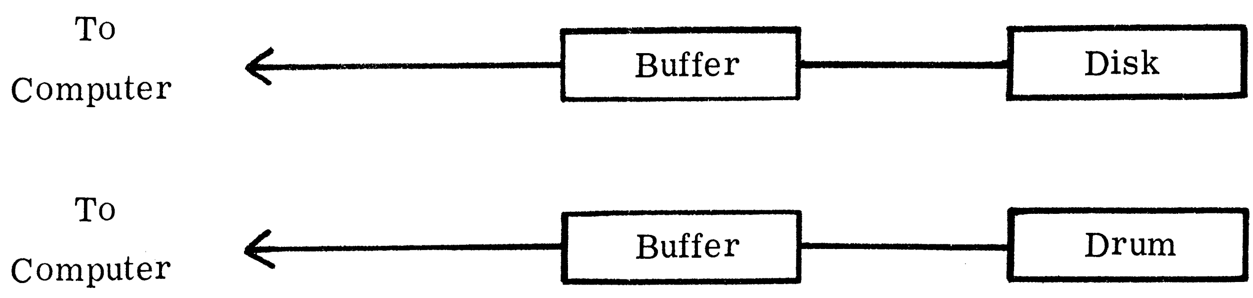
Computer	Beta*	Epsilon*
Word Size	18 bits	30 bits
Core Cycle Time	2 μsec	1.8 μsec
Execution Speed	9 bits/μsec	16.7 bits/μsec
Max Data Transfer Rate	7.20 mega-bits/sec	4.8 mega-bits/sec
	(400, 000 words/sec)	(160, 000 words/sec)
	also	
	(100, 000 words/sec),	
	(55, 500 words/sec)	

*"Beta" and "Epsilon" are designations chosen for ease of reference and to avoid use of any commercial designations.

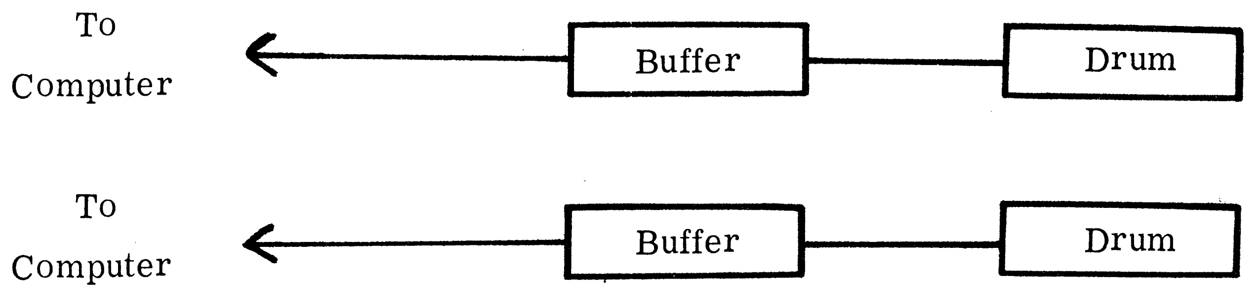
Table 2-4

Bulk Storage Device Specifications

Device	Disk	Drum
Avg. Access Time	184 msec	9 msec
Transfer Rate	.730 mega-bits/sec	2 - 6 mega-bits/sec



(a) Disk-Drum Storage



(b) Dual Channel Drum Storage

Figure 2-4

Storage Configurations

drum storage simultaneously in the disk-drum storage system and from both halves of the dual channel drum storage system simultaneously. It should be noted that the access time is much more significant in determining bulk storage speed than the transfer rate. Average access time is the average time required for the disk or drum to rotate to a point where the desired data are located. Once this point is reached the transfer rate is the rate at which data are read from the disk or drum. Thus for the dual channel 6 mega-bits/sec drum 9 msec average access time is required but only 1. 5 msec are required to read a 512 word, 18 bit record.

2. 3. 2 Results

We have mentioned the concept of balance between the CPU rate and the rate of data transfer. For the proposed use of the computer systems under consideration we estimate conservatively that only rarely will more than five execution cycles be required per data word transferred. This estimate is made because the system is basically a data storage, and retrieval system (file system) in which little manipulation of data by the CPU will be required outside of the executive control program. For example, a target record may be searched on a few key words, but much of the record is text. Well under one execution cycle/word transferred would probably be needed here.

Using the estimate in the preceding paragraph the disk-drum storage system may be rejected immediately for both computers as being highly inefficient. This system is so slow (average access time 184 msec) that approximately 90 execution cycles/word transferred would be available in Beta and about 160 cycles/word transferred in Epsilon. (See triangles in Figure 2-6.) This means that the Beta processor would be idle 95% of the time and the Epsilon processor would be idle 97% of the time. At the other end of the scale consider the six mega-bit/second dual channel drum with these two computers. At its maximum rate (average access time 9 msec), approximately 6.5 execution cycles/word transferred are available in Beta and about 13.5 execution cycles/word transferred are available in Epsilon. Thus the performance of these two computers should be roughly equivalent even when the fastest available bulk storage system is attached to them. Both these systems are much faster and make much more effective use of the CPU capabilities than did the systems using disk drum storage. In fact there will be roughly an order of magnitude improvement in the performance of either computer with the dual channel drum over that same computer with the disk-drum.

With the fast dual channel drum storage (6 mega-bits/sec) the computer system will be fairly well-balanced. Only in this case will the execution speed have any appreciable effect on performance because now the storage system may occasionally have to wait for the performances of the two computers as a function of the percentage of time the storage system is held up by the CPU in Beta. For most programs the percent of time storage must wait for the CPU will probably be 1% or below. In this range the performance of the two computers is essentially the same. Note that for a given storage configuration the relative performance of Epsilon to Beta will never rise above about 1.85 since this is the ratio of their execution speeds.

We also studied the relative performance of the systems as the fraction of time varied for which the storage is idle waiting for the CPU to generate an I/O request. This is plotted in Figures 2-7 and 2-9. It is the ratios between these curves that are important and not absolute values. The system should usually be operating at or below the one percent level because the system is seldom compute bound.

The curves are useful in comparing the relative merits of the various storage systems for different (small) degrees of compute-boundedness.

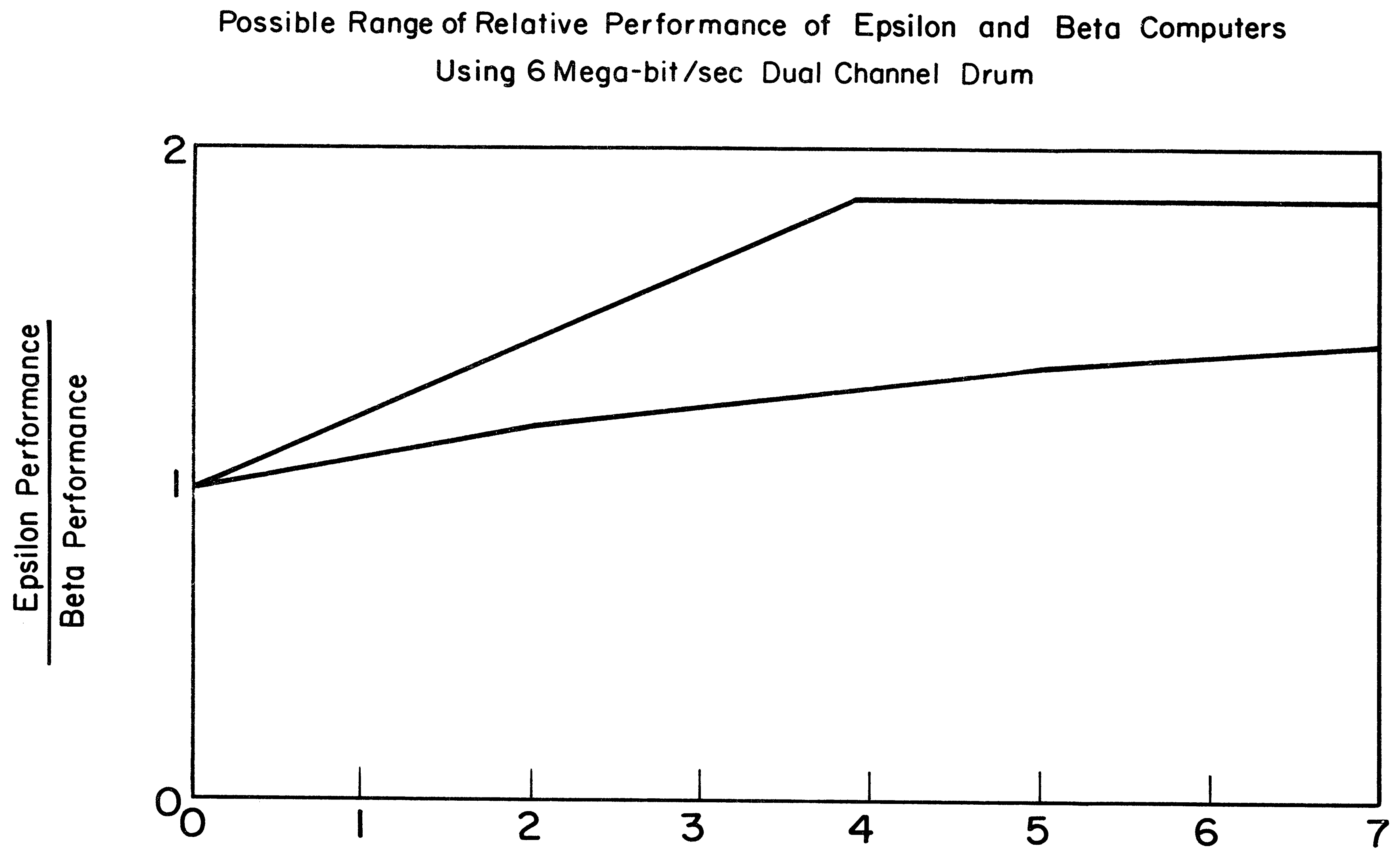


Figure 2-5

Percent of Time Storage Waits for CPU in Beta

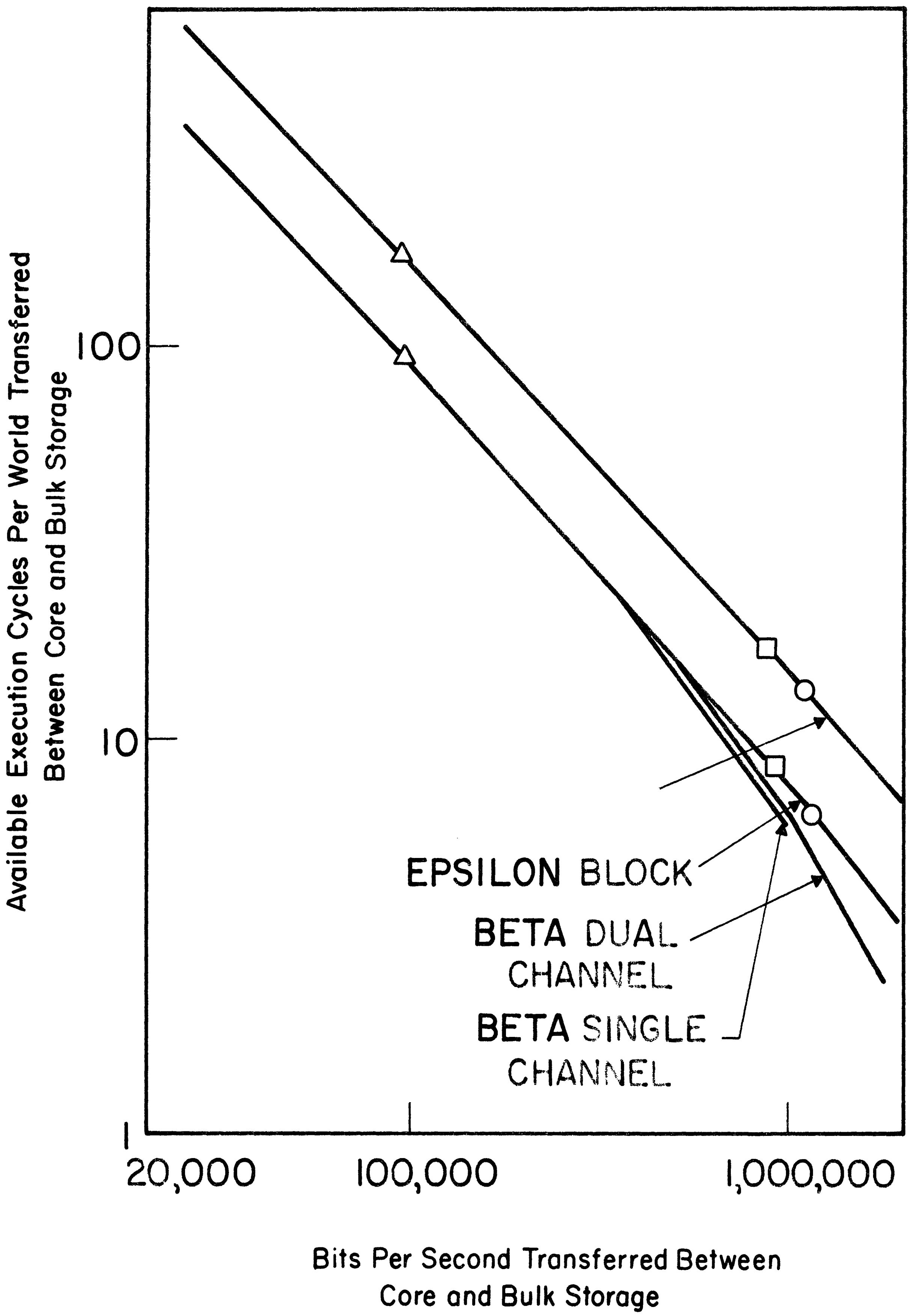


Figure 2-6. Available Execution Cycles

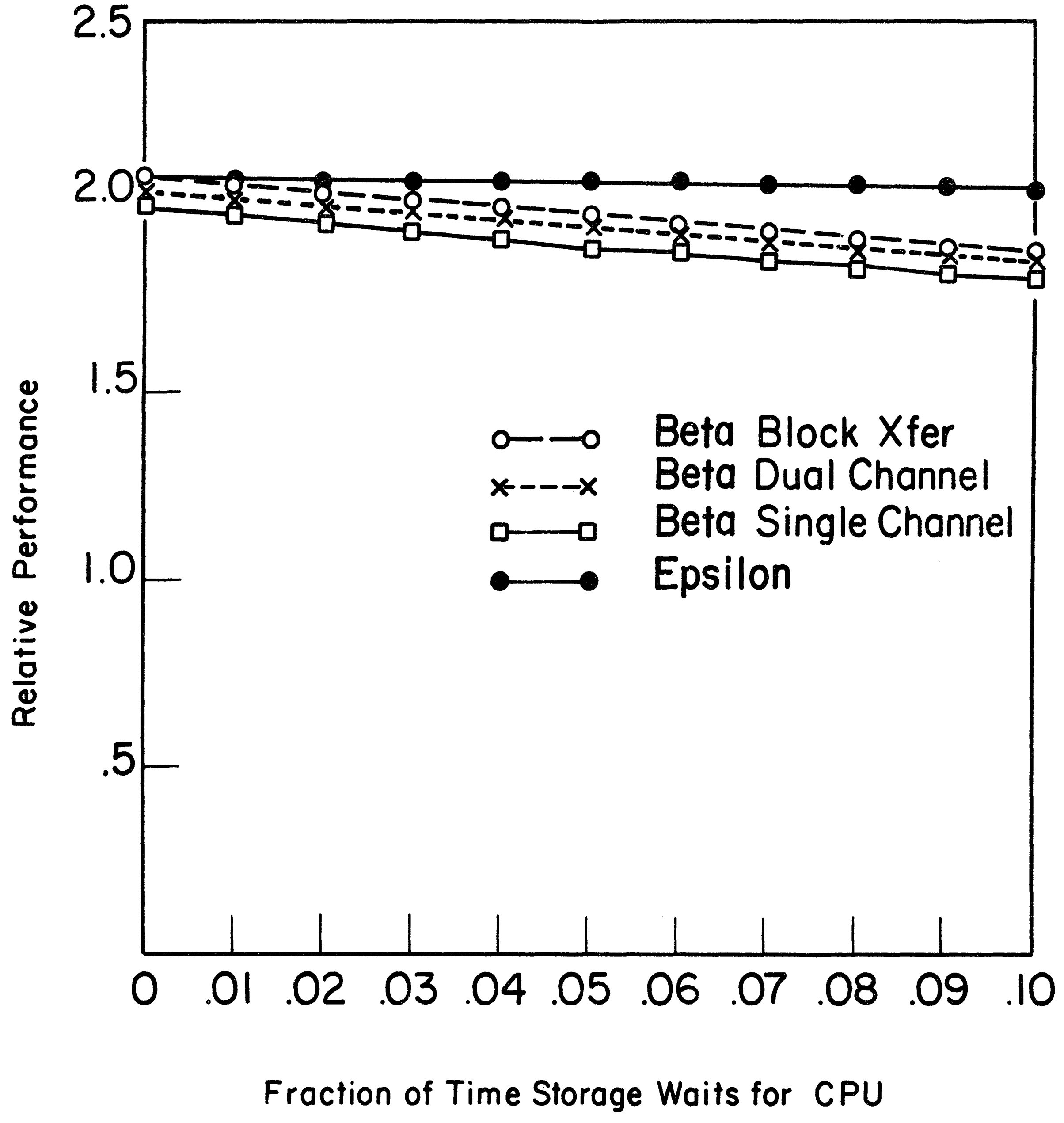


Figure 2-7

Performance Relative to Beta With

Disk Only Storage System: Disk-Drum

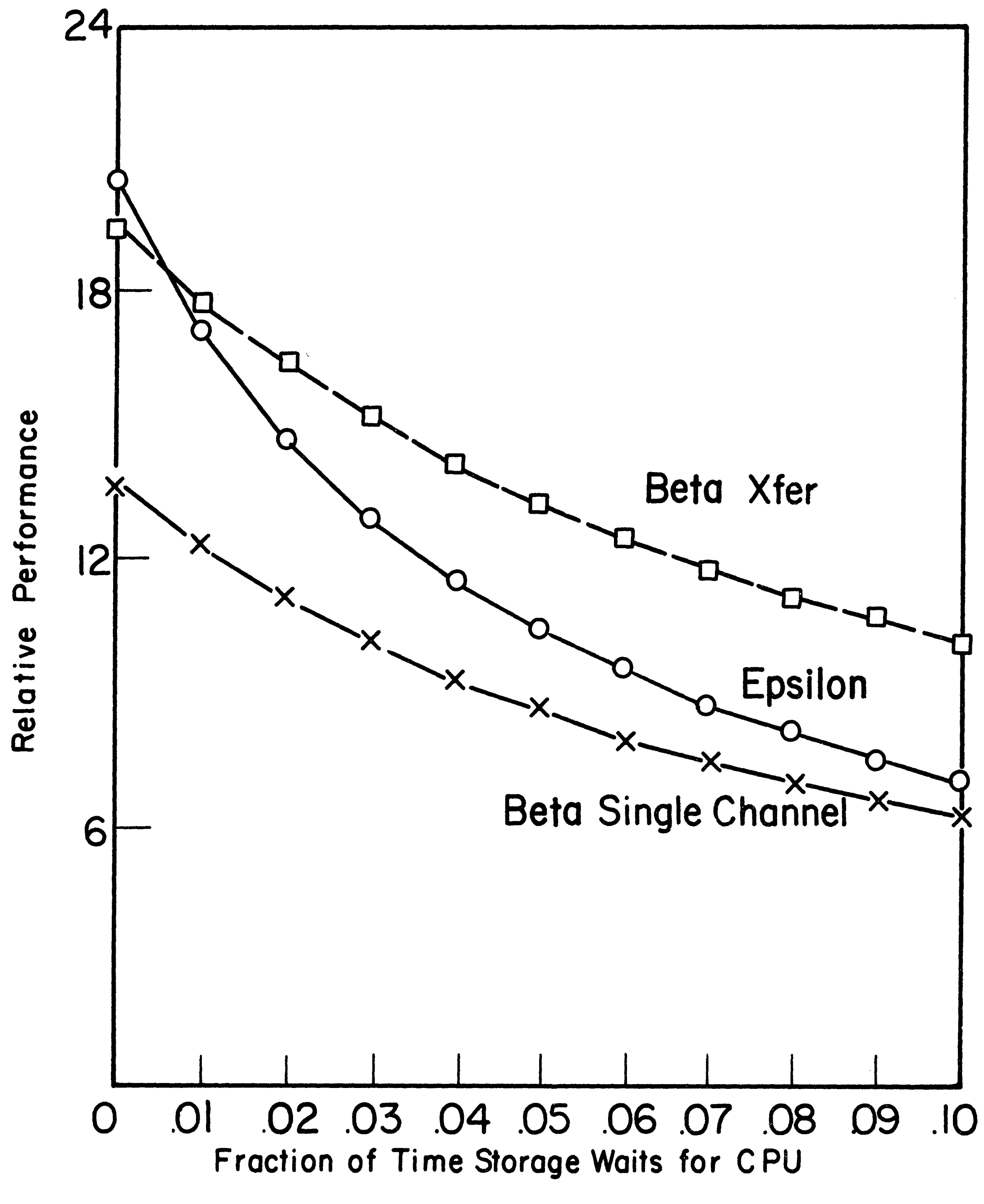


Figure 2-8

Performance Relative to Beta With Disk Only

Storage System: 2 Mbs Dual Channel Drum

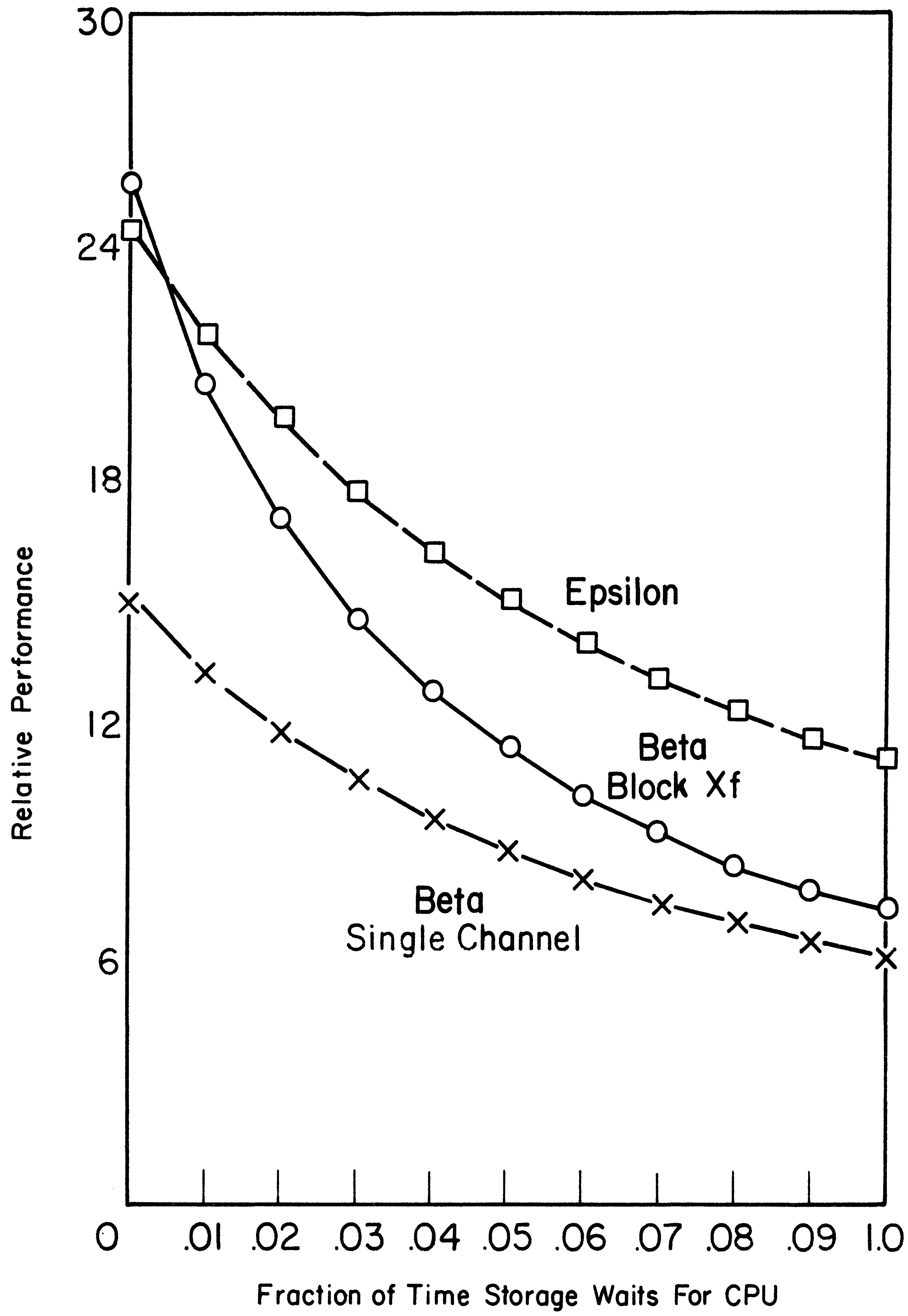


Figure 2-9

Performance Relative to Beta With Disk Only.

Storage System: 6 Mbs Dual Channel Drum

2.4 ANALYSIS OF PROPOSED ON-LINE SYSTEM

A proposed on-line storage and retrieval system has been modeled to study response time and CPU utilization. Analysis of the model has been completed using both minimal queueing analysis and simulation. Figure 2-10 shows the physical configuration of the system being modeled as it is presently envisioned. Figure 2-11 is the general queueing model which was developed for the purpose of analysis by both the Recursive Queue Analyzer (RQA) [4] and the General Purpose Simulation System (GPSS) [2].

The specific descriptions of the respective models with pertinent assumptions made for each are presented in Sections 2.4.2 and 2.4.3.

2.4.1 Description of the Physical System being Modeled

The console-computer system being studied is assumed to have 14 active consoles at which operators generate job requests requiring processing by the main computer. These requests originate at the consoles and are for terminal support functions of the following types:

a) Copy from one terminal to another

b) Column tab

c) Scrolling

d) Page turning

Requests generate jobs which are entered in a queue for execution at the CPU. In both models it has been assumed that the computer (CPU) is

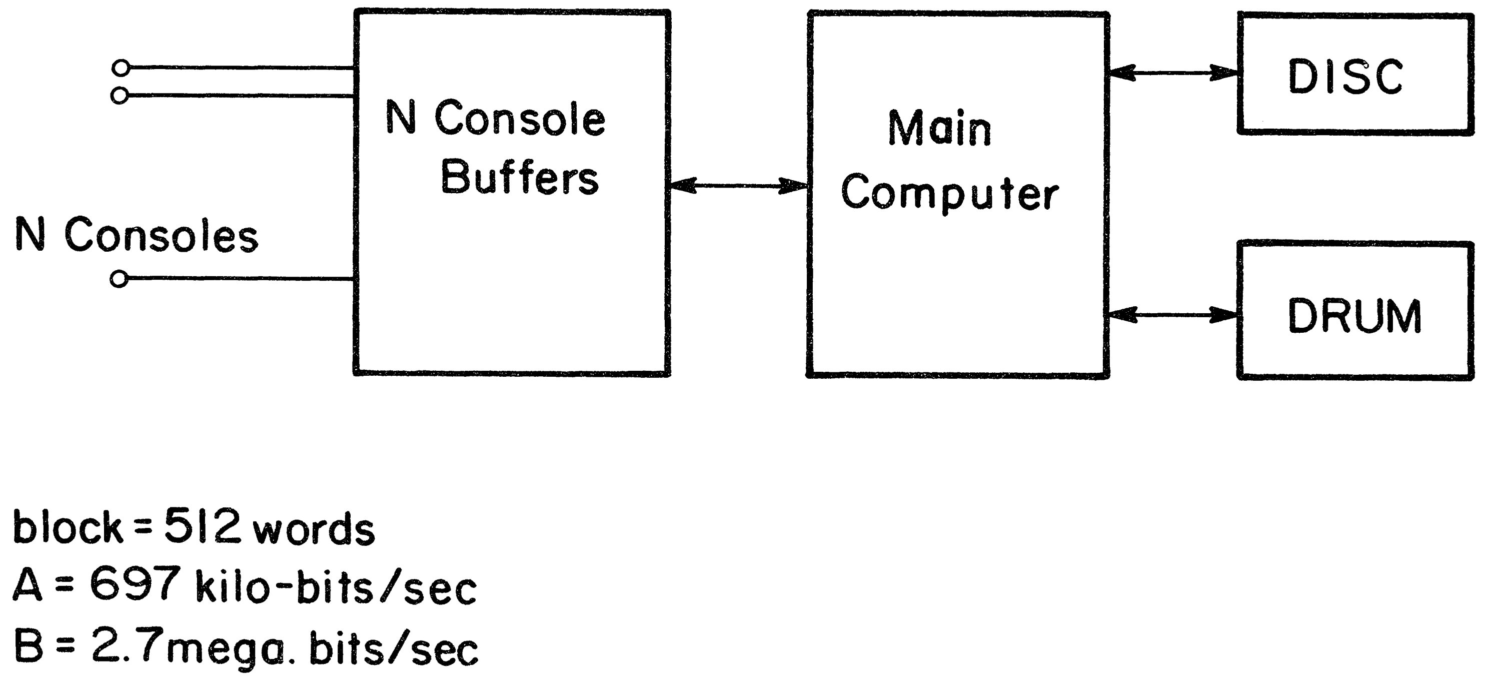


Figure 2-10

Proposed System Diagram

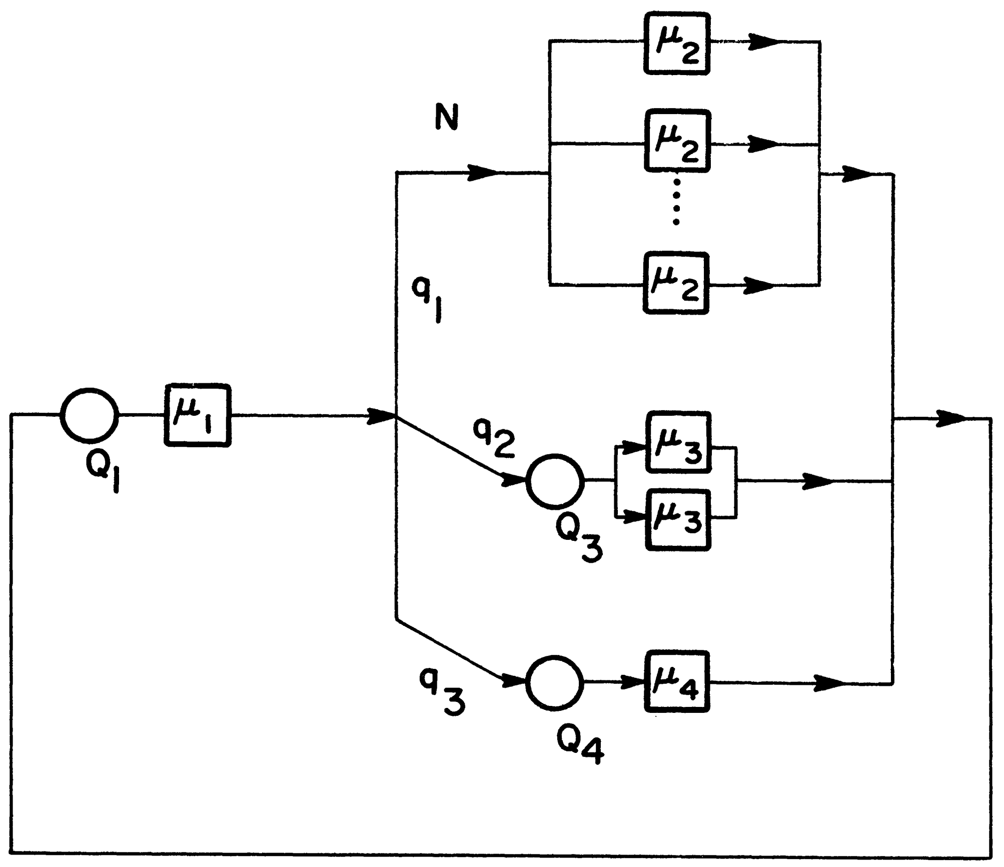


Figure 2-11

Model of Proposed System

dedicated to the four tasks associated with terminal support.

Jobs may also require access to a drum and/or a parallel arm disc unit for input-output activity before being completed. If there is a request for such a data transfer from the disc, jobs queue at the disc data channel awaiting service by the disc which processes the incoming requests on a FCFS basis. There is a second independent data channel for the drum and its operation is considered to be independent of the disc. Again requests for this data channel are allowed to queue and the FCFS processing discipline is used. Basic performance features of a typical drum unit have been used in both models.

2.4.2 Queueing Analysis

In the queueing model it is necessary that the various service time distributions be derived from the negative exponential distribution. For simplicity, the negative exponential distribution itself has been used. The same negative exponential random variable is used for the CPU service time distribution for each of the four job types. In the actual system a deterministic service time (in general different for each type of job) would be the most realistic manner of describing the CPU service mechanism. It is also assumed that upon completion of the CPU execution phase, jobs are either completed with probability q_1 or request some type of input-output activity with probability $q_2 + q_3$. In the actual computer system, certain job types request a deterministic number of drum and disc accesses and in a specific sequence; other job types are completed without any input or output; hence if the type of job departing the CPU is known it can be determined with certainty whether a drum access, disc access or neither will follow.

It has also been assumed that in the queueing model the console interaction time (the time elapsing from the completion of a job at the CPU to the time when the next request emanates from that console) is an exponentially distributed random variable with mean 24 sec. A further assumption is that the service time distributions for the drum and disc are exponentially distributed with means of 12 and 120 millisec, respectively. In the physical system the console interaction time would be more nearly uniformly distributed; certainly the service time distributions for the rotational storage devices would be more nearly uniformly distributed between the minimum and maximum positioning times than exponential.

In summary, exponential service times have been assumed for all service mechanisms in the queueing model; in reality the service distributions range from deterministic at the CPU to uniformly distributed for the drum and disc service times. The actual process which is being treated as Markovian is significantly devoid of the necessary exponential characteristics.

2.4.3 The Simulation Model

The simulation model has been formulated to simulate the behavior of the actual computer system in every element of detail. Each of fourteen facilities corresponding to the consoles initially submits one job (chosen from a mix of job types) to the computer system after a think time interval determined by a sampling from a uniform distributed random variable following system startup. Associated with each job submitted is a parameter which indicates which type of job (Page Turn - Type 1, Scroll - Type 2, Tab - Type 3, Copy - Type 4) has been submitted. Job types 1 and 2 always require 50 ms; job type 3 always requires 75 ms; job type 4 always requires 100 ms of CPU service time. Different types of jobs "flow" through the system facilities in differing sequences, since each type of job requires a unique sequential service from the various system facilities. Table 2-5 indicates the sequential flow through the system for jobs as a function of type.

Upon the termination of one job at the CPU, a new job is chosen from the mix of job types. A free console facility is seized by the job and after a think time interval of 24 sec has elapsed the new job reenters the CPU queue for service.

For example, a type 2 job generated at a console will in sequence request service by the CPU, drum CPU, disc, and CPU facilities before being terminated. A type 4 job does not request any input-output activity and is completed following a single CPU service increment.

The job mix is specified in the simulation model on a probabilistic basis through use of a discrete valued random variable for the job mix distribution. Table 2-4 indicates the job mixes that were used in various

Table 2-5

Flow of Control by Job Type

Type 1	Type 2	Type 3	Type 4
CPU	CPU	CPU	CPU
DRUM	DRUM	DRUM	TERMINATE
CPU	CPU	CPU	
DISC	DISC	TERMINATE	
CPU	CPU		
DRUM	TERMINATE		
CPU			
DISC			
CPU			
DRUM			
CPU			
DISC			
CPU			
TERMINATE			

runs.

When a particular type job does enter the queue for service by the drum, the service time is uniformly distributed between 4 and 20 ms; for the disc the service time is uniformly distributed between 10 and 230 ms.

Response time is defined to be the elapsed time between job submission at the console and the job termination following its final CPU processing operation.

In determining the length of the simulation, runs of 1000, 5000, 10, 000, and 25, 000 jobs were made. The difference in response times for the 10, 000 and 25, 000 job runs was less than two percent for each of two runs made with different job mixes. Since the average IBM 360/67 execution time for one 10, 000 job GPSS run was approximately five minutes, the significantly increased expenditure of 360/67 processing time on the longer 25, 000 job run did not produce a proportionate improvement in the results. Hence 10, 000 jobs were used for all the simulation runs made.

A critically important question arises as to how one determines the set of parameters for the queueing analysis which correspond to those for a particular simulation. Since the queueing analysis parameters T1, T2, T3, T4, q_1, q_2, and q_3 are all expected values, averaged over all four job types, it is obvious that different sets of parameter values in the simulation model might correspond to one particular set of parameter values in the queueing model. In the comparisons which have been made, only the job mix distribution function in the simulation model was changed in order to vary q_1.

Define the following parameters, taken from the GPSS model:

$d_i \triangleq$ the total number of departures from the CPU in one complete sequence of processing operations for a type i job.

$m_i \triangleq$ fraction of the d_i departures which are followed immediately by a drum processing operation.

$c_i \triangleq$ fraction of the d_i departures which are followed immediately by a disc processing operation.

$x_i \triangleq$ mix fraction of job type i (an input to the GPSS model).

	i = 1	i = 2	i = 3	i = 4
d_i	7	3	2	1
m_i	3/7	1/3	1/2	0
c_i	3/7	1/3	0	0

The following expressions are used to compute the parameters q_1, q_2, and q_3 for the RQA model, given the GPSS model's values for c_i, d_i, m_i, and x_i.

$$q_1 = \frac{\sum_{i=1}^{4} x_i}{\sum_{i=1}^{4} x_i d_i} = \frac{1}{\sum_{i=1}^{4} x_i d_i}$$

$$q_2 = \frac{\sum_{i=1}^{4} x_i d_i c_i}{\sum_{i=1}^{4} x_i d_i}$$

$$q_3 = \frac{\sum_{i=1}^{4} x_i d_i m_i}{\sum_{i=1}^{4} x_i d_i}$$

To compute the average CPU service time for an RQA run the following expression was used

$$\text{1) Avg CPU Service Time, } \overline{T1} = \frac{\sum_{i=1}^{4} T_i x_i d_i}{\sum_{i=1}^{4} x_i d_i}$$

where T_i is CPU processing time for a type i job and x_i and d_i were previously defined.

Since the RQA output consisted of steady state probabilities for various states in the model, the following expression relating various queue lengths, service times, etc., was used to compute the system response time.

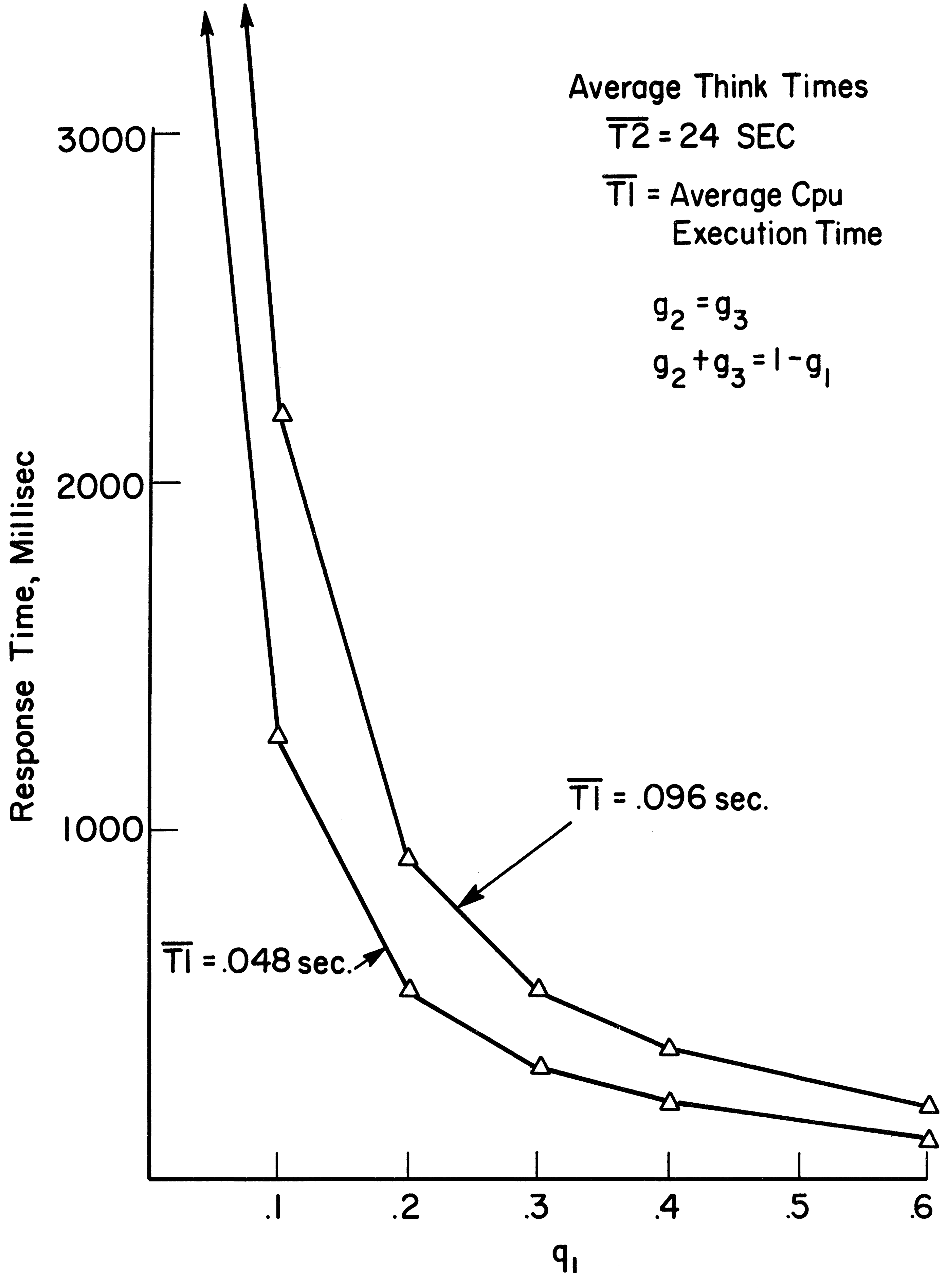


Figure 2-12 RQA Analysis of Proposed System

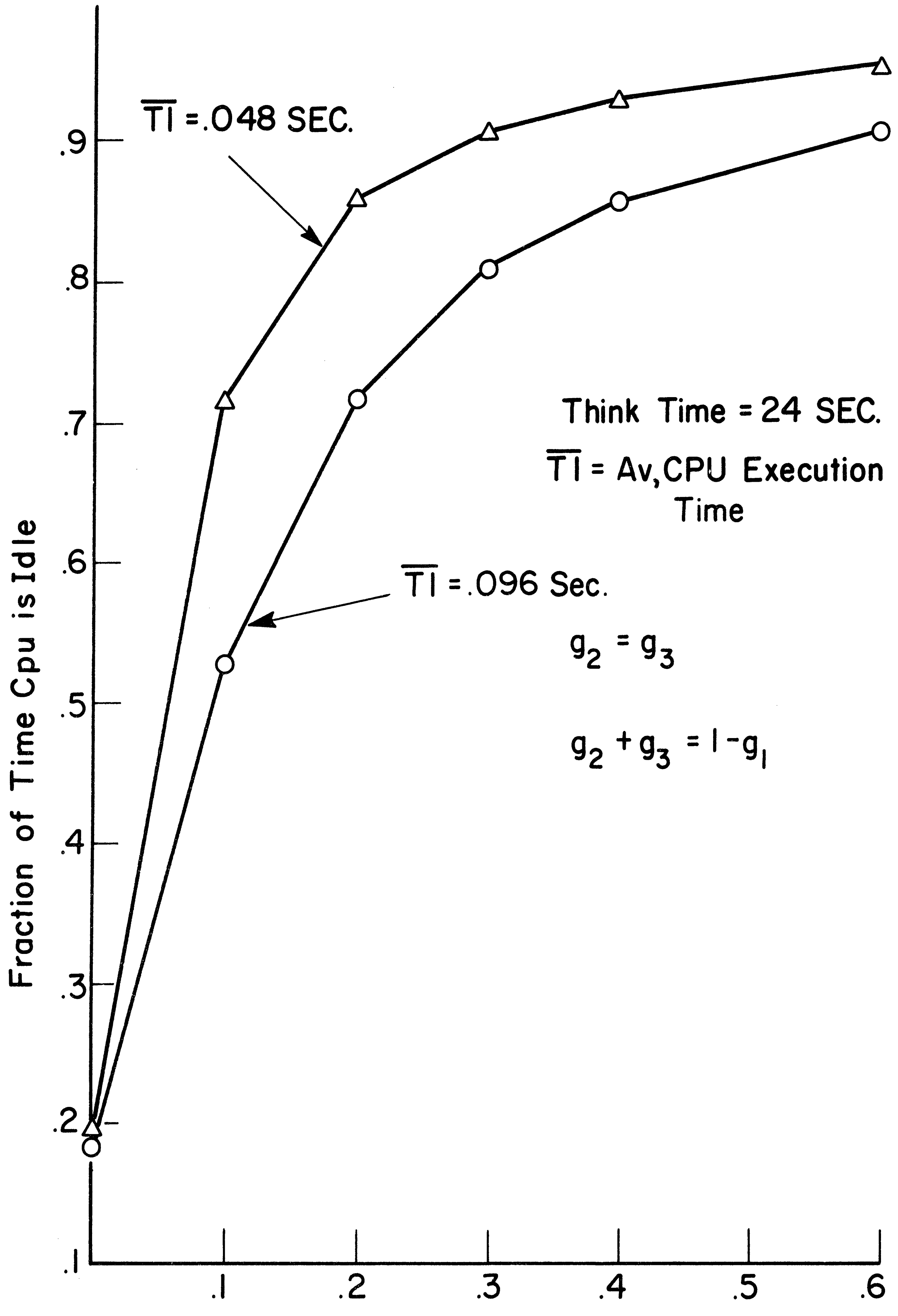


Figure 2-13 RQA Analysis of Proposed System

System Response Time =

$$\frac{1}{q_1}\left[\frac{Q_{CPU} * \overline{T1}}{1 - \Pr\{\text{CPU is idle}\}}\right] + \left[\frac{1}{q_1} - 1\right]\left[\frac{q_3}{q_2+q_3}\right]\left[\frac{\bar{Q}_{DRUM} * \overline{T4}}{1 - \Pr\{\text{Drum is idle}\}}\right]$$

$$+ \left[\frac{1}{q_1} - 1\right]\left[\frac{q_2}{q_2+q_3}\right]\left[\frac{\overline{T3}(PDSC1 + .5(\bar{Q}_{DISC} - PDSC1))}{1 - \Pr\{\text{Disc is idle}\}}\right]$$

where $\bar{Q}_{CPU}$ is the average number of jobs in the queue for CPU service, $\bar{Q}_{DRUM}$ and $\bar{Q}_{DSC}$ are similarly defined, and PDSC1 is the joint probability that one disc mechanism is busy and there are no jobs in the disc queue. All the $\bar{Q}$'s and probabilities were obtained as output from the RQA program.

For example, a type 2 job generated at a console will in sequence request service by the CPU, drum, CPU, disc, and CPU facilities before being terminated. A type 4 job does not request any input-output activity and is completed following a single CPU service increment.

The job mix is specified in the simulation model on a probabilistic basis through use of a discrete valued random variable for the job mix.

2.4.4 Discussion of Queueing Analysis Results

Figures 2-12 and 2-13 depict the results of RQA runs. The following parameters were used in the calculation of the system response time with the model of Figure 2.

N	number of consoles in operation = 14
$T_1 = 1/\mu_1$	mean job execution time
$T_2 = 1/\mu_2$	mean operator response time

$T_3 = 1/\mu_3$	mean block transfer time for disc = 120 ms
$T_4 = 1/\mu_4$	mean block transfer time for drum = 12 ms
q_1	% of jobs departing the CPU which do not require I/O accesses
q_2	% of jobs departing CPU which require a disc access
q_3	% of jobs departing CPU which require a drum access.

As indicated on the plots, it is assumed that q_2 and q_3 are identical for these particular RQA runs of Figures 2-12 and 2-13.

2.4.5 Discussion of Results of Comparison

Figures 2-14 and 2-15 contain plots of response time and CPU utilization as functions of q_1, the probability that a job departing the CPU does not require further input-output activity. The RQA curves for these figures were obtained using the same model as for Figures 2-12 and 2-13 with different parameter values. Table 2-6 indicates the parameter values which were used in making the runs.

The lower bound curve of Figure 2-14 was computed by assuming no queueing delay at each server; hence response time was simply the summation of the average service times of the various servers which a job sequences through in the system between input and termination.

As would be expected the results of GPSS and RQA compare most favorably in the middle and upper ranges of q_1 values. This interval corresponds to the area in which the CPU service time distribution used in the

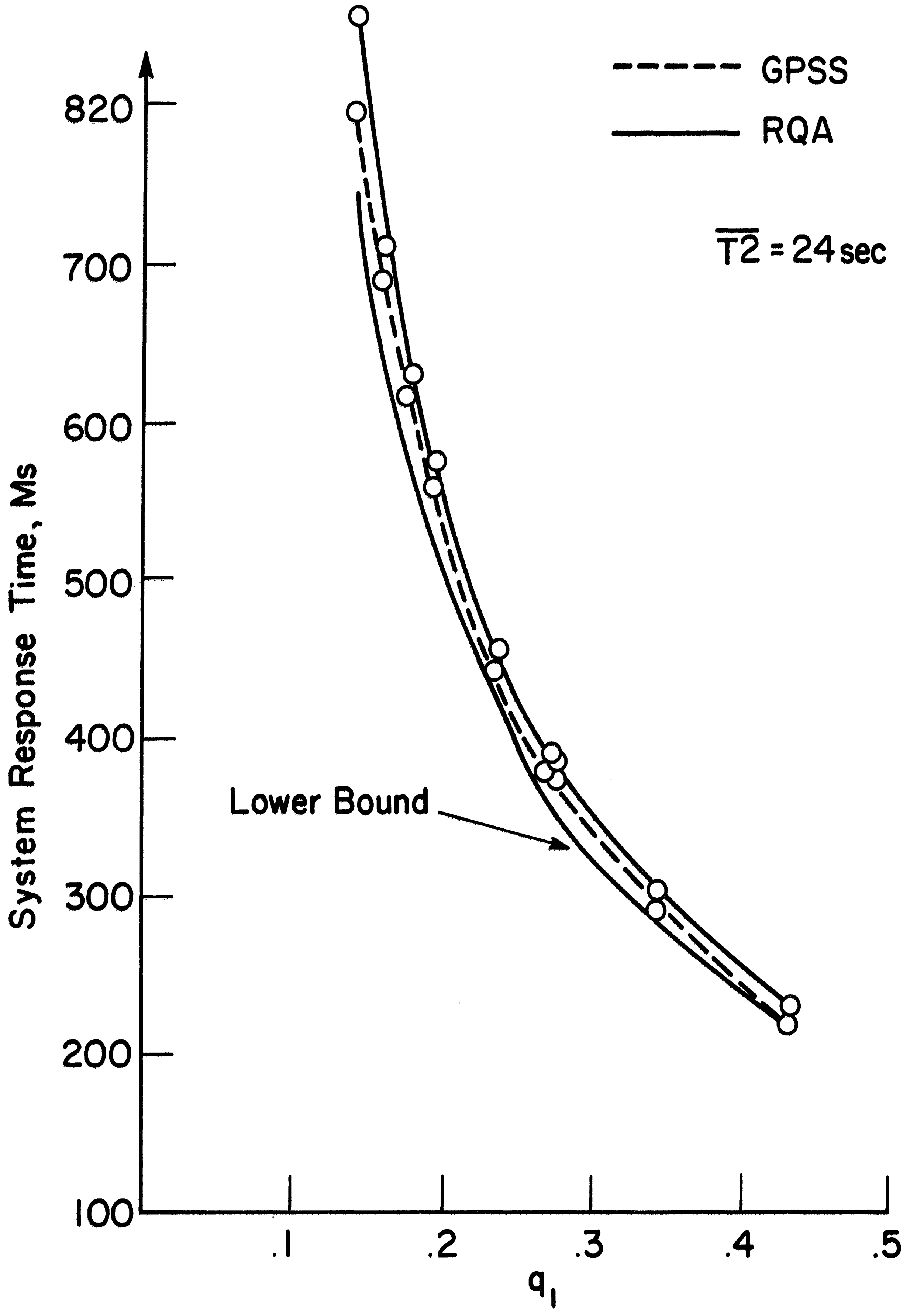


Figure 2-14 Response Time from RQA and GPSS

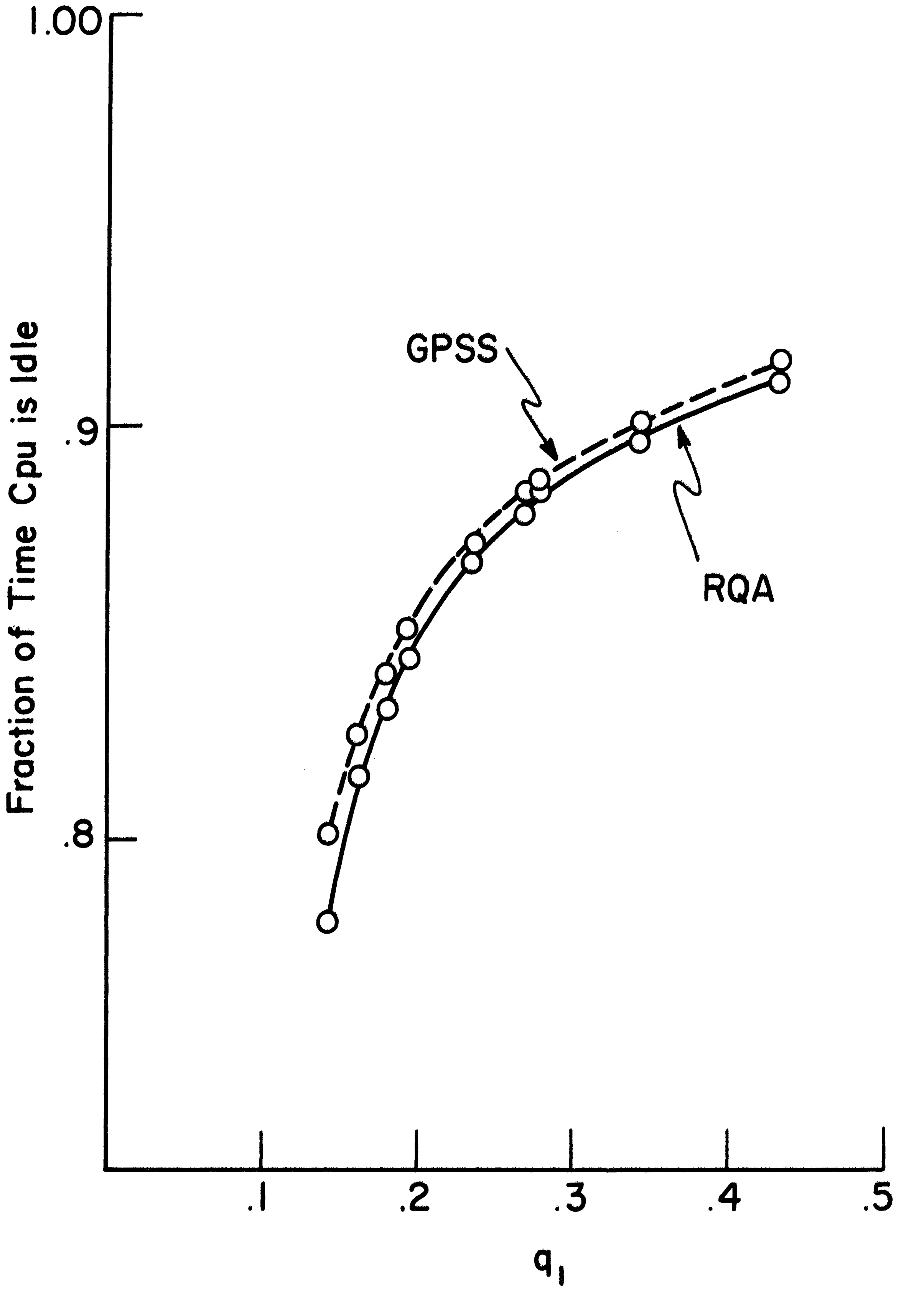


Figure 2-15 CPU Idle Fraction from RQA and GPSS

	← GPSS Job Mix →									
	fraction of type 1, x_1	fraction of type 2, x_2	fraction of type 3, x_3	fraction of type 4, x_4	$\overline{T1}$ ms	q_1	q_2	q_3	lower bound system resp. time, ms	system resp. time, ms
RQA					50	.142857	.42857	.42857		856.47
GPSS	1.00	0.	0.	0.					746	795.311
RQA					50.819	.16393	.40984	.42623		711.735
GPSS	.8	.1	.1	0.					641	691.36
RQA					51.818	.18182	.40	.41818		631.427
GPSS	.7	.1	.1	.1					576	620.319
RQA					51.96	.19608	.39216	.41176		576.299
GPSS	.6	.2	.1	.1					529	562.87
RQA					53.571	.2381	.35714	.40476		457.747
GPSS	.4	.3	.2	.1					425	446.63
RQA					55.405	.27027	.32432	.40541		393.415
GPSS	.3	.3	.3	.1					366	382.728
RQA					55.555	.27778	.3333	.38889		393.265
GPSS	.3	.3	.2	.2					364	375.85
RQA					60.345	.34483	.27586	.37931		303.138
GPSS	.2	.2	.3	.3					284	292.673
RQA					65.2174	.43478	.21739	.34783		233.34
GPSS	.1	.2	.3	.4					220	224.5

Table 2-6

GPSS and RQA Parameter Values

CPU idle time fraction	% error in resp. time using RQA	% error in CPU idle time using RQA
.7809	7.7	2.88
.804		
.816	2.95	1.21
.826		
.832	1.79	.45
.840		
.8442	2.39	.80
.851		
.8674	2.49	.53
.872		
.879	2.79	.57
.884		
.886	4.64	.11
.887		
.8965	3.58	.50
.901		
.91126	3.94	.41
.915		

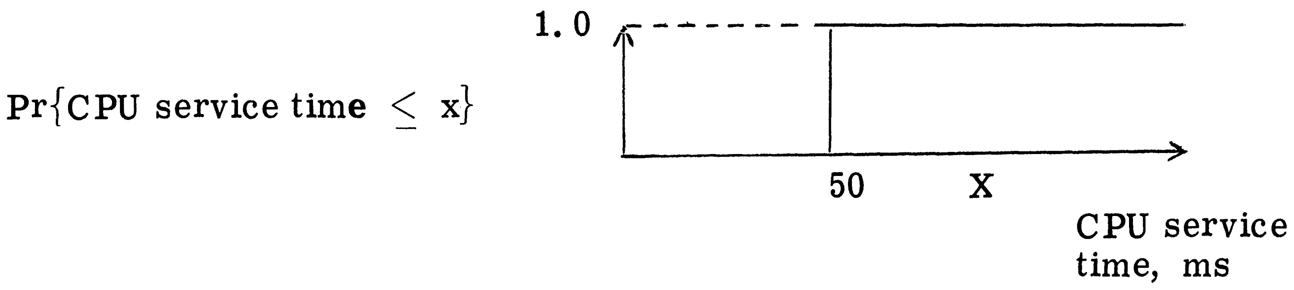
In all runs: T2 = 24000 ms, T3 = 120 ms and T4 = 12 ms
GPSS input parameters: Cols. 1-4
RQA parameters: Cols. 5-8
Output from models: Cols. 10-11
Values in Cols. 9, 12, 13 were determined analytically

Table 2-6 Continued

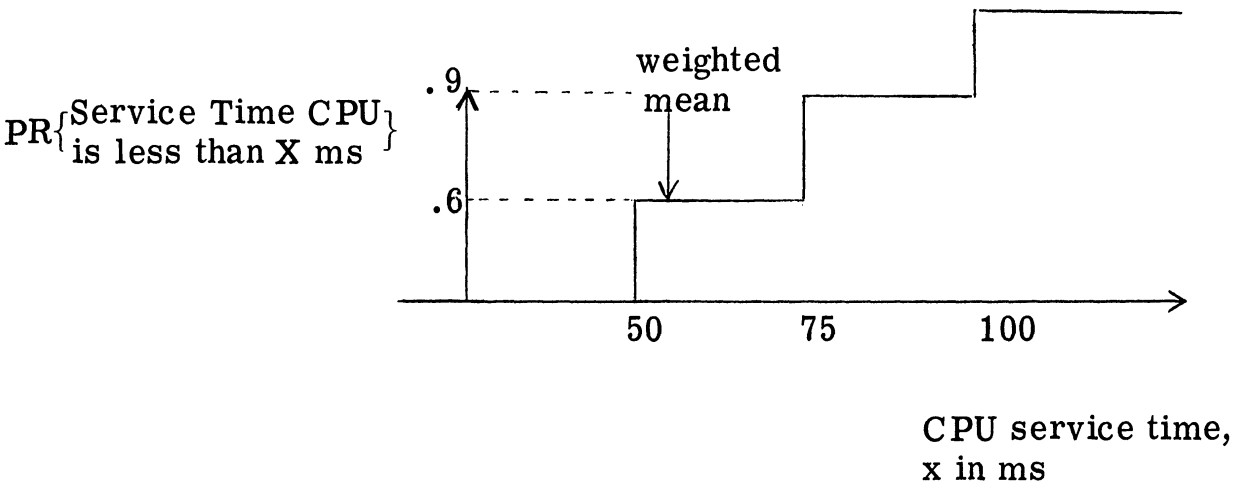
GPSS and RQA Parameter Values

GPSS model most closely resembles the exponential distribution assumed in the RQA model (even though the resemblance is in fact quite poor).

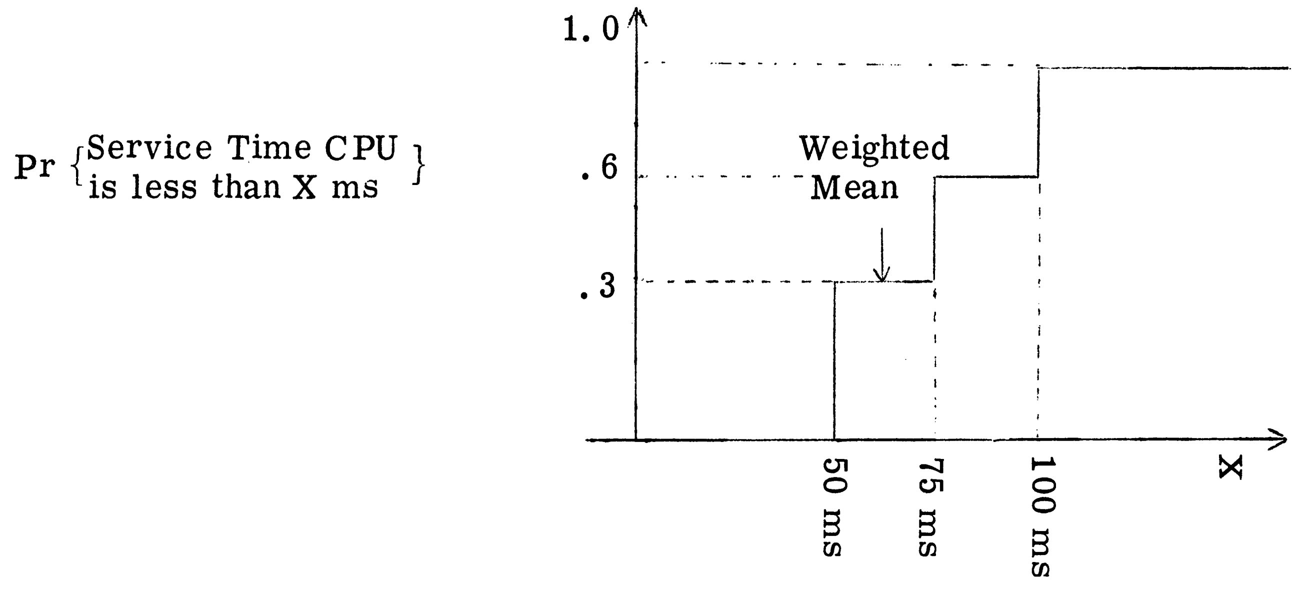
For example the weighted CPU service time distribution is given by



for the job mix (100%, 0%, 0% 0%), which corresponds to $q_1 = .1428$. When the job mix is changed to (30,%, 30,%, 30,%, 10%) corresponding to $q_1 = .27778$ the weighted CPU service time distribution is given by



At the upper end of the q_1 range with a job mix of (10%, 20%, 30%, 40%) corresponding to $q_1 = .43478$ the weighted CPU service time distribution is given by



This in the middle and upper portions of the q_1 range, the weighted CPU service time distribution for the GPSS model most resembles the negative exponential distribution function used in the corresponding RQA model. As would be expected, the percentage of error in using RQA is large where the dichotomy of corresponding service time distributions is relatively large. Even though in the worst case a constant rate server in a GPSS model was assumed to be exponential in the corresponding RQA model, the percentage error in system response time and CPU idle time was less than 8%.

Furthermore, in all cases, as would be expected the response time computed using RQA exceeded that obtained using GPSS. This is so because the variance (normalized to the mean) of the respective service times in GPSS is always less than the normalized variance which arises as a result of the negative exponential service time distributions assumption for the same server in RQA.

2.4.6 Conclusions

The results of this analysis (Figures 2-12 to 2-15) show that system response time as well as the CPU idle time is critically affected by the intensity of I/O demand generated by the consoles. The response time (defined as the average time for a request originating at the consoles to be serviced) is seen to grow monotonically with an increase in the fraction of jobs requiring I/O. This is to be expected since the disc and drum I/O times enter additively into the computation of response time and if on the average more jobs require I/O, then the average response time will increase.

It suffices to say that the q_1 parameter should be made as large as possible, corresponding to minimum I/O. The natural question which arises is: what can be done in the actual system to force changes in the q_1 parameter of the model? Certainly the job mix would be fixed once the operating environment has been established. However it is possible that the disc and drum units can be organized so that multiple accesses can be made without interrupting the CPU. If this is done, then the resulting q_1 value is effectively increased and system operation improved.

2.4.7 Comparison of Analysis Times

Since the Recursive Queue Analyzer program is implemented on the IBM 7090 and GPSS is implemented on the IBM 360/67, it is most meaningful to normalize execution times before comparisons are made.

The computations for all nine points on the RQA curve of Figures 2-14 and 2-15 required approximately 18.67 minutes of CPU time on the 7090. The corresponding GPSS computations required 49.48 minutes of CPU time on the 360/67. Since the 360/67 is approximately three times faster than the 7090, the RQA time normalized to a 360/67 equivalent machine would be reduced to approximately 6.22 minutes.

In conclusion, the GPSS model effectively required 7.95 times more CPU execution time than did the RQA model to compute nine points. The maximum error in system response time introduced by using RQA was 7.7% while the maximum error in CPU idle time introduced by using RQA was 2.88%. It is noteworthy therefore that the Markovian modeling tool, RQA, which is substantially more efficient than GPSS from a computational standpoint, can yield such satisfactory results in modeling a system whose behavior is significantly devoid of the characteristics theoretically necessary for a valid Markovian model.

2.5 ANALYSIS OF A CANDIDATE CRT SUPPORT SYSTEM

This section presents results of an initial analysis of a candidate CRT Support System. The purpose of the analysis is to determine the additional utility gained by implementation of the system on an IBM 360 model 40; the basic capabilities are presently being implemented on an IBM 360 model 30.

The basic on-line capability will provide a CRT user with the ability to retrieve, by specifying a record number, a record from a single file and to display that record on a CRT in one of two formats. The system will be so partitioned as to allow batch processing to proceed in the background. The CRT's will be locally operated, that is, they will be cable connected to the computer. The essential system components considered in determining how well the system will perform the initial functions are:

1) IBM 2311-Disk
2) IBM 2848-Display Control
3) IBM 2260-Displays (8)
4) IBM Multiplexor Channel
5) IBM 2030-Processing Unit.

The multiplexor channels available have data transfer rates well in excess of the devices to which they will be attached, i. e., the disk and display control. Therefore, the rates at which blocks of data are transferred will be determined by the devices themselves. Assuming a block of data consists of eight-80 column lines (640 characters), the average transfer time for the devices will be obtained from the following information. The 2311 disk has an average positioning time of 75 milliseconds; the average latency time is 125 milliseconds; the transfer rate is 156 Kilobytes/second. Therefore for a block of data the total time required is

$$75 + 12.5 + 640 \times \frac{1}{156 \times 10^{3}} = 91.75 \text{ milliseconds.}$$

Normally a request will require six blocks of data from the disk; this increases the total disk time to 113 milliseconds (assuming all blocks of data are contiguously located). Assuming that a full screen display is 12 lines (960 characters), we compute the time for the 2848 as follows: The time required for synchronization of the channel and the 2848, prior to data transfer, averages 8.4 milliseconds. After synchronization, the transfer of data requires approximately .4 ms/character for the characters on a given display line. During the transfer of data from the channel to the 2848, data transfer is halted for a period of 16.7 ms before the start of each display line except the first. During this pause the 2848 services 2260 display station key boards, i.e., it stores the data from a single 2260 keyboard in the associated 2260 buffer or performs the specified control function. Therefore the 2848 time required for transfer of 12 lines each consisting of eighty characters is

$$8.4 + 80 \times .4 \times 12 + 11 \times 16.7 = 576.1 \text{ ms.}$$

Therefore, for the average request, the time required to display the first block of data will be 576 + 113 = 690 milliseconds plus the required processing time (generally less than a second). These times are independent of the 360 model used. With the model 30 there is suspension of the CPU while the data transfer is in progress. The model 40 allows the multiplexor mode transfers and the CPU to operate concurrently once the data transfer has been initiated.

To get an estimate on the response time for the system we made the following assumptions:

1) Average request requires six blocks of data (3840 bytes)
2) Requests handled sequentially
3) Buffer space provided for complete record (all six blocks)
4) Core space not released when data is transferred to display
5) CPU is suspended during data transfer in channel (as is done with model 30).

If response time is defined as the time interval between the system first receiving a target number request and the first screen display associated with the request, the minimum response time is .69 + 1 = 1.69 seconds. This response time is composed of one second processing time associated with a request, plus the disk and 2848 time. In the unlikely event that eight CRT's required service simultaneously, the maximum response time will be 13.6 seconds. On the average, the response time will be 6.8 seconds with half the CRT's in the service queue. If core buffer space must be provided for each of the consoles, and a 65K core model 30 is used, up to 45% of core could be tied up for buffering.

The decision to use a given 360 model should, of course, be based on the ultimate uses of the system; however we can make some qualitative judgments. If an average response time of almost seven seconds is not unreasonable, and if the size of the supervisor and the associated on-line software package will allow up to 45% of core to be tied up for the information retrieval function, then the model 30 is adequate. If this is not the case, then it might be necessary to change to the model 40 since:

1) With the model 40, overlapping of CPU execution with the I/O channel will allow a more efficient operation. If the system is multiprogrammed, response time can be decreased considerably.

2) The added complexity and the size of the software package might make multiprogramming infeasible with the smaller model 30 core.

3) The larger core of the model 40 gives potentially a larger buffer area, thereby giving possibility of fewer disk accesses and hence better response time.

Before we can make more quantitative estimates on system performance, additional information is needed:

1) How much core will be used for buffering?

2) When a record of six blocks is accessed, will only that part of a record which can be displayed be brought into core? Or will all six blocks of a record be stored in core?

3) Will the system be multiprogrammed or will the supervisor queue up request from CRT's and handle on a first come—first served basis?

4) Is core released when information is transferred to display buffer?

5) Under what conditions are data swapped within each core partition?

6) How is core partitioned between batch and on-line jobs?

7) How much of core will be occupied by the supervisor and the resident on-line software package?

REFERENCES, CHAPTER 2

1. Evans, R. V., "Geometric Distributions in Some Two-Dimensional Queueing Systems," Operations Research, 15, 5 (October-November 1967), pp. 830-846.

2. International Business Machines Corporation, General Purpose Simulation System/360 User's Manual, IBM Publication H20-0326, White Plains, New York.

3. Wallace, V., "The Solution of Quasi Birth and Death Processes Arising from Multiple Access Computer Systems," SEL Technical Report No. 35, Systems Engineering Laboratory, The University of Michigan, March, 1969.

4. Wallace, V. and Rosenberg, R., "RQA-1, The Recursive Queue Analyzer," SEL Technical Report No. 2, Systems Engineering Laboratory, The University of Michigan, February, 1966.

3. OPTIMUM DESIGN TECHNIQUES

As computer system designers our goal must be not just to find a system adequate for the job at hand, but to find the best, or optimum, system design. In virtually all cases, the optimum design must meet certain constraints, usually with respect to performance or cost. The three sections in this chapter discuss in turn the optimum design of telecommunications networks, graphical display terminals, and large storage systems.

3.1 OPTIMUM DESIGN OF TELECOMMUNICATIONS NETWORKS

3.1.1 Introduction

In this section we will be concerned with the problems of designing the portions of remote access systems which lie outside the central location.

A central digital computer complex and the communication facilities which connect remotely located terminals to the central complex constitute a teleprocessing system. The most significant analytic distinction between a teleprocessing system and a conventional batched input system lies in the random manner in which requests for service (jobs) arrive at various entry points to the system. In the conventional batch mode, arrival patterns are deterministic in the sense that they are monitored by machine operators and computing center scheduling personnel, while inputs to a teleprocessing system are seldom controllable. Furthermore, a very significant fraction of capital expenditures in a teleprocessing system is devoted to communications facilities outside the central complex; in the batch system almost all capital expenditures are devoted to software and hardware facilities at the central computer complex.

In the design of such teleprocessing systems, various queueing problems arise as consequences of both the random or unscheduled requests for service and the importance of effectively utilizing the expensive communications equipment linking the remote terminals to the central computational facility. This research is accordingly centered around the development of techniques for the analysis and synthesis of the communications network portion of the teleprocessing system.

In the communication network portion of the teleprocessing system, the costs of communication link transmission capacity and data concentrators are frequently of the same order of magnitude as the costs of the central processing facility. Hence there is significant motivation to make efficient utilization of these communications resources in order to maximize the cost effectiveness of system operation.

In general, the important macroscopic lead variables with which the designer of the network must be concerned are:

- Topological structure
- Link capacity value
- Response time or performance of the network
- System reliability
- Control procedures
- Types of data concentrators

- System cost.

The development of the topological structure of the network is probably the single most important portion of the design procedure; it is next to impossible to achieve any degree of cost-effective performance in a network where structure is poorly chosen.

A completely point-to-point network (Figure 3-1) is defined to be a structural configuration in which each remote terminal is connected to the computer over a channel which is not shared with any other remote terminals. A multipoint network, (Figure 3-2), on the other hand, is a configuration in which one or more information channels in the network are shared by two or more remote terminals.

In general, the completely point-to-point network can seldom be justified from an economic standpoint since there is no sharing of communication lines, and the cost of lines in this structure tends to be high in comparison to other structures. Thus a strong motivation exists for the development of techniques which enable the communication channels of a network to be shared, thus reducing the total cost of lines in the system. However, as multipoint configurations are formed, additional costs for concentrating equipment are incurred and must be considered. Furthermore, the average response time increases as the congestion builds up due to the formation of more multipointed branches.

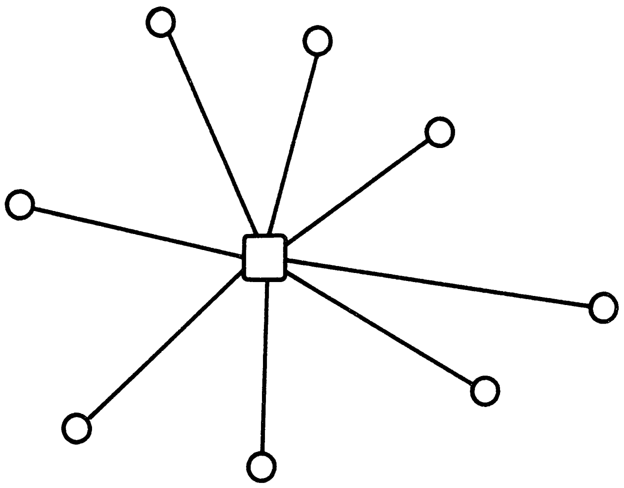


Figure 3-1

Point to Point Configuration

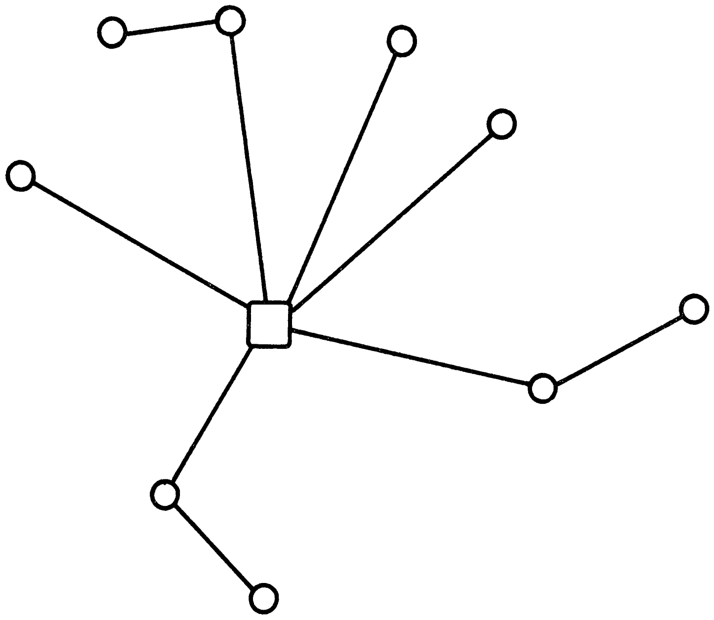


Figure 3-2

Multipoint Configuration

In designing the network, one must be able to assess these tradeoffs with queueing and other types of analytic models which characterize the interrelationships in mathematical terms. As the first step in the desired direction, we consider some relatively simple models for data concentrates in networks having fixed structures.

These models form the basis for the more complex design problems in which topological structure and link capacity values are treated as variables in the design procedure.

3.1.2 Models for Data Concentrators

In this section we present models which are useful in assessing the performance of multiplexor and message switching center (MSC) types of concentrators. The multiplexor model is valid for either frequency or time division schemes; the MSC model holds for any store and forward devices which can buffer message blocks at intermediate nodes of the communication networks.

3.1.2.1 Message Switching Center Model

Consider the single concentrator configuration of Figure 3-3. The queueing model for this network can be depicted as shown in Figure 3-4. where it is assumed that messages arrive at node i according to a Poisson distribution having mean γ_i. The mean lengths of all messages are $\frac{1}{\mu}$ and we let $\{\alpha_1, \ldots, \alpha_n\}$ denote the capacity of the input links to the concentrator. The capacity of the high speed shared link is denoted by α_H. For most

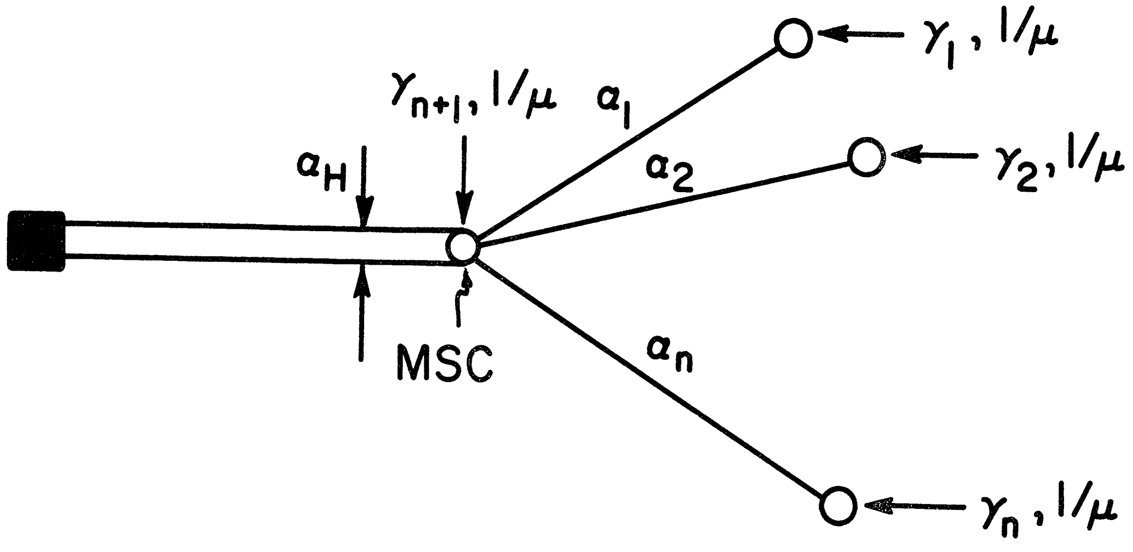


Figure 3-3

Single Stage Message Switching Center

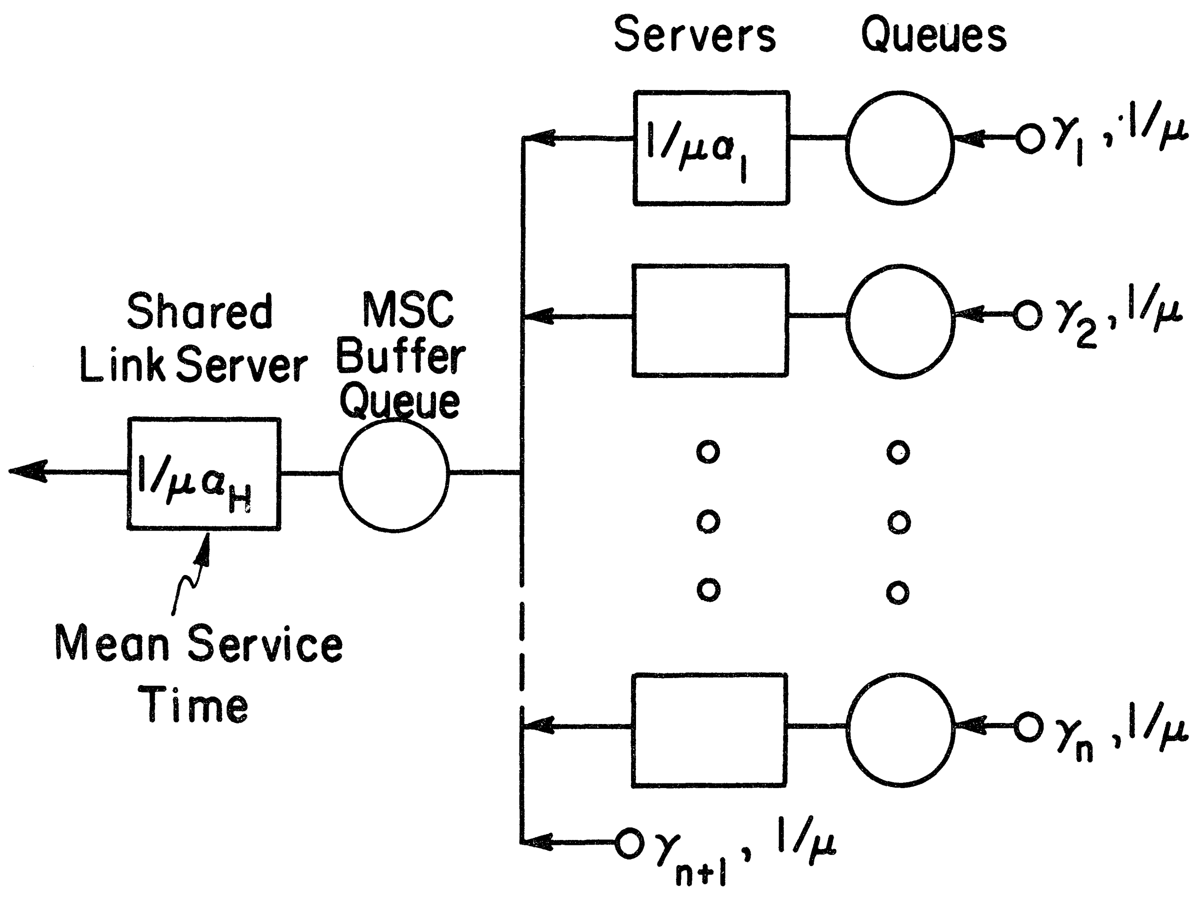


Figure 3-4

Queueing Model for Single Stage

Message Switching Center

reasonable state of the art MSC's it can be shown that the buffer capacity is large enough so that the average message delay in the network can be written as follows, assuming a first in—first out discipline at the queue in the MSC,

$$T_{MSC} = \frac{1}{\sum_{i=1}^{n+1} \gamma_i} \left[\sum_{i=1}^{n} \left(\frac{1}{\frac{\alpha_i}{\gamma_i / \mu} - 1} \right) + \frac{1}{\frac{\alpha_H}{\lambda_{n+1} / \mu} - 1} \right]$$

where λ_{n+1} is the sum of the mean arrival rates of messages from all remote terminals in the configuration.

3.1.2.2 Multiplexor Model (MX)

In the network of Figure 3-3, if a MX concentrator is used instead, the equivalent structural model can be shown as depicted in Figure 3-5. Passages refer to the fractions of the total link transmission capacity which are assigned to the individual remote terminals in the sharing group. It is possible to express the average delay for a single multiplexor network as follows:

$$T_{MX} = \frac{1}{\sum_{i=1}^{n+1} \gamma_i} \left[\sum_{j=1}^{n+1} \left(\frac{1}{\frac{C_j}{\gamma_j / \mu} - 1} \right) \right]$$

where C_j is the capacity of the j-th passage on the shared link and it is assumed that each passage capacity is less than the capacity of the

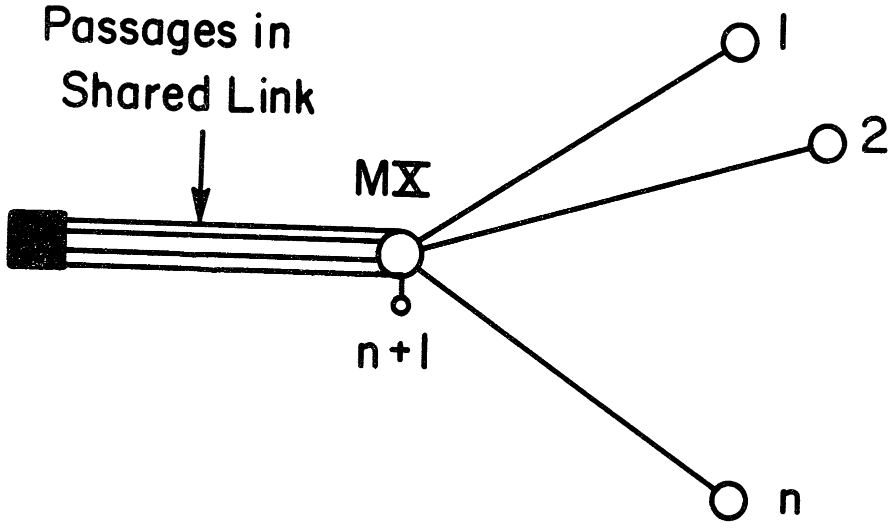


Figure 3-5

Equivalent Structural Model

for Multiplexor Concentrator

corresponding input link.

The problem of determining the values of the C_j capacity assignments which minimize T_{MX} for an arbitrary centralized network has been investigated. An optimization procedure has been developed for minimizing the average message delay in any centralized network where the link capacity values are known and where one or more MX concentrators are used.

These models were used to conduct some comparisons between MX and MSC concentration sehemes. It is well known that MSC's are more costly than MX's. However, there is no basis for determining which type of concentrator should be used in a given situation, unless one uses models like those just presented to quantify the differences in performance between multiplexors and MSC's.

Figures 3-6 and 3-7 depict the results of two studies which were made to compare the performances of a MX and a MSC at the concentrator node in the network of Figure 3-3 when n=9. As the next phase in adding reality (and hence complexity) to the design model the problem of efficiently allocating the link capacity resources in a network has been considered.

3.1.3 Efficient Allocation of Link Capacity

We now consider the problem of how to assign capacity values for the links of a structure so as to get the minimum line cost and satisfy an average response time constraint. The problem is combinatorial in nature and hence the size of the optimization space can get very large for networks having more than 15 or 20 remote terminal nodes. Computational short

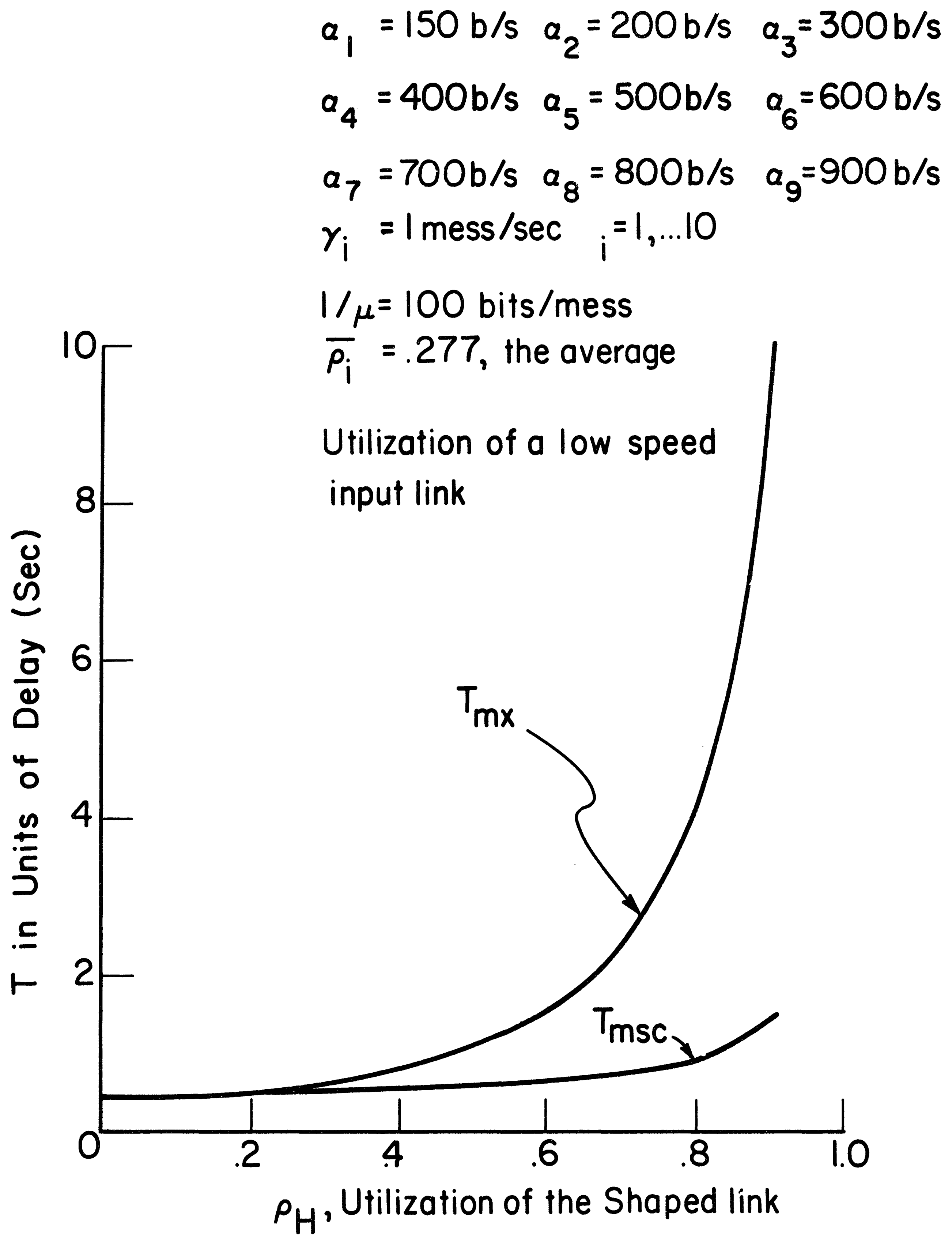


Figure 3-6 Comparison of MSC and MX Concentrators

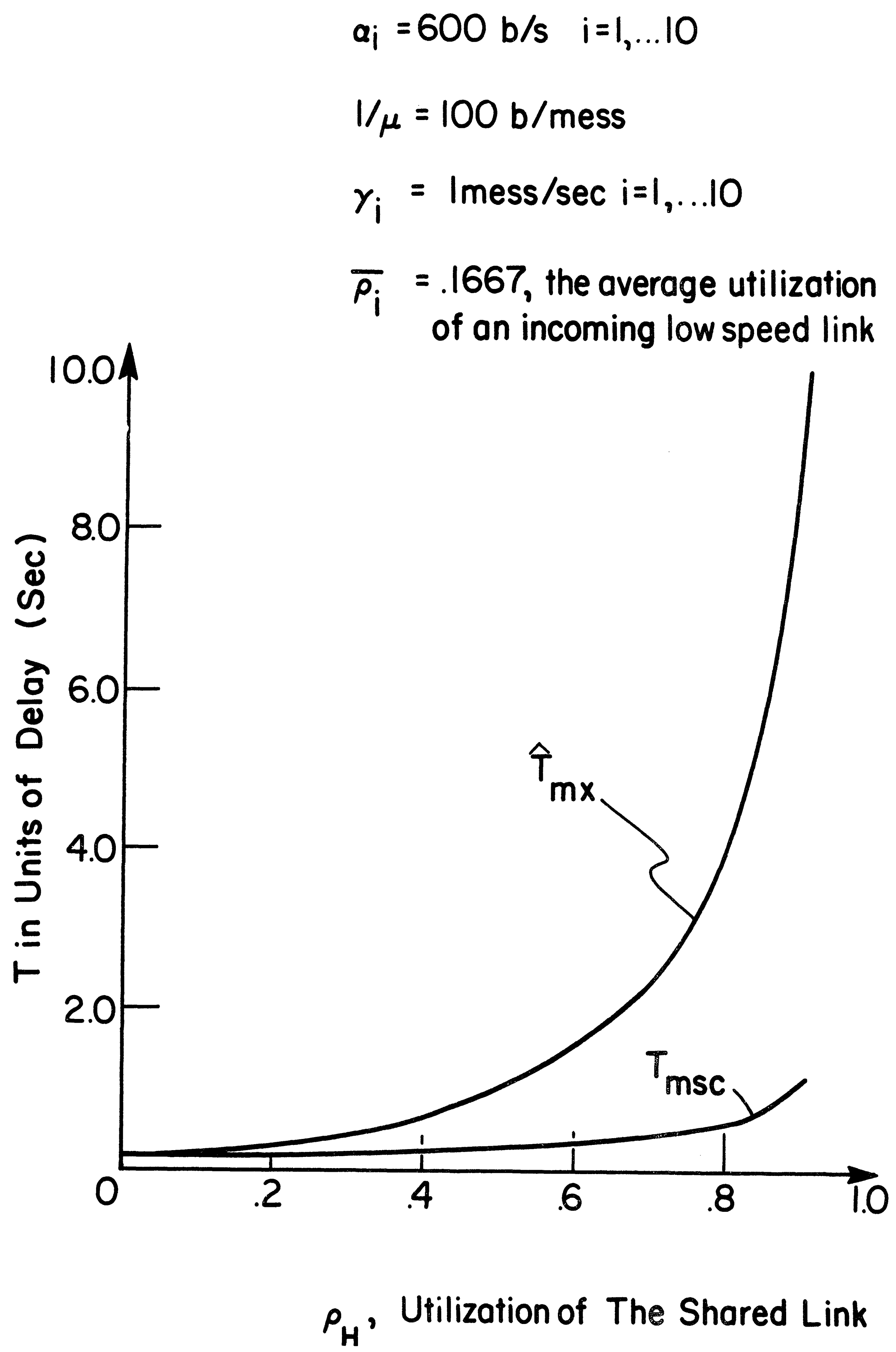


Figure 3-7 Comparison of MSC and MX Concentrators

cuts have been developed which enable one to optimize otherwise untractable networks. Also, an approximate solution which is computationally very easy to determine, even for networks of moderate size, has been obtained. These approximation procedures consistently produced solutions to the capacity assignment problem which were within reasonable limits of the true optimum and did so in an efficient way.

3.1.4 Network Design when Structure is a Variable

All of the optimization models developed for the analysis of data concentrators and the assignment of link capacity obviated shortcomings of existing network design procedures such as the minimum spanning tree algorithm [11] and one due to Esau and Williams [7] of IBM. These procedures do not treat link capacity as a variable and also do not consider the average response time property which was introduced in Section 3.1.2.

In this research, we have directed our efforts toward the solution of the network synthesis problem: Minimize total telecommunication network system cost (lines and concentrators) subject to the constraint that the average response time of the network must not exceed T_{max}, a given maximum acceptable value. The important control variables are network topology, link capacity values, and the types of data concentrators.

A procedure has been developed for solving this problem which, although it generally produces suboptimal solutions, represents a significant improvement over existing network design algorithms. The improvements accrue as a consequence of the greater level of reality and generality which are embedded in the design model.

Some of these major improvements are now summarized:

- It is capable of working with nonlinear discrete valued cost-capacity functions.
- It treats the type of data concentrator as a design variable.
- The performance (average response time) of any potential configuration is readily assessable.
- Since the problem statement involves a given constraint on average response time, it is possible to predict cost-performance relationships for different levels of response time, before the network is actually constructed.

We now summarize the fundamental approach which has been taken in solving the above formulation of the design problem.

A multi-dimensional design parameter space is constructed in which topological structure, link capacity value and types of data concentrator are varied. Topological structure is varied by systematically decreasing the number of central links one at a time, reducing link costs at each step until no further reductions can be made. Then link capacity values are assigned for the structures of these topological sequences in a manner which creates efficient utilization of the system resources. The type of data concentrator is varied by using MSC's in those situations where they produce a large enough performance improvement to justify their larger cost.

No configurations are ever formed by the topological variations which have average response times in excess of a certain given value.

A number of examples which illustrate the usage of the proposed design procedures and the results of their application to realistic problems are discussed at length in the final report on the communication network synthesis effort [6].

3.2 DESIGN OF OPTIMUM DISPLAY SYSTEMS

3.2.1 Introduction

The subject of the study reported here is the systems design of highly-interactive graphical display terminals for time-shared computer systems. A more detailed report of the work is available in a reference [8]. The overall goal of the study is to develop insight into how the choice of subsystems for a display system can affect the system's performance, and to develop methods of finding the combination of subsystems which will be optimum for any well-defined display application, where optimum is defined as minimizing a display system's response time subject to a cost constraint.

Viewed in a slightly different way, display system design can be thought of as presenting a problem in resource allocation. The resource is a fixed number of dollars, which are allocated to the purchase of display subsystems in a manner which minimizes the total system's response time.

Response time is important in any highly-interactive remote access computer system, and is even more important when considering the graphics terminals which often form part of such systems, because fully capitalizing on the potential interaction rates achievable with a graphics terminal demands good response time. Dollars are significant because display system hardware is still quite expensive, and improper allocation of the dollars can produce a display system whose response time is orders of magnitude worse than what can be achieved with the optimum allocation.

Figure 3-8 shows the type of system of interest to us. The four subsystems of particular interest are the data link, the remote computer-display control, the display terminal's core storage, as well as the display terminal's bulk storage. In specific cases the terminal's bulk store may simply be a cable between adjoining rooms. In most cases, the remote computer-display control and its core storage will be part of a display system, with the core serving both as a refresh buffer for the display, and as program and data storage for the remote computer. This computer (usually small and inexpensive) is used to take the burden of much of the display processing from the main computer. The exact nature of the relationships among the display control, remote computer, and core storage is discussed in a reference [12]. The main computer is included in the system to provide inexpensive bulk storage and, above all, because of the extensive computations demanded by most display applications, such as network analysis [4, 5 , 14, 17], drafting and numerical control [13], integrated circuit layout [9,15], and many others.

When designing a display system, there are large numbers of

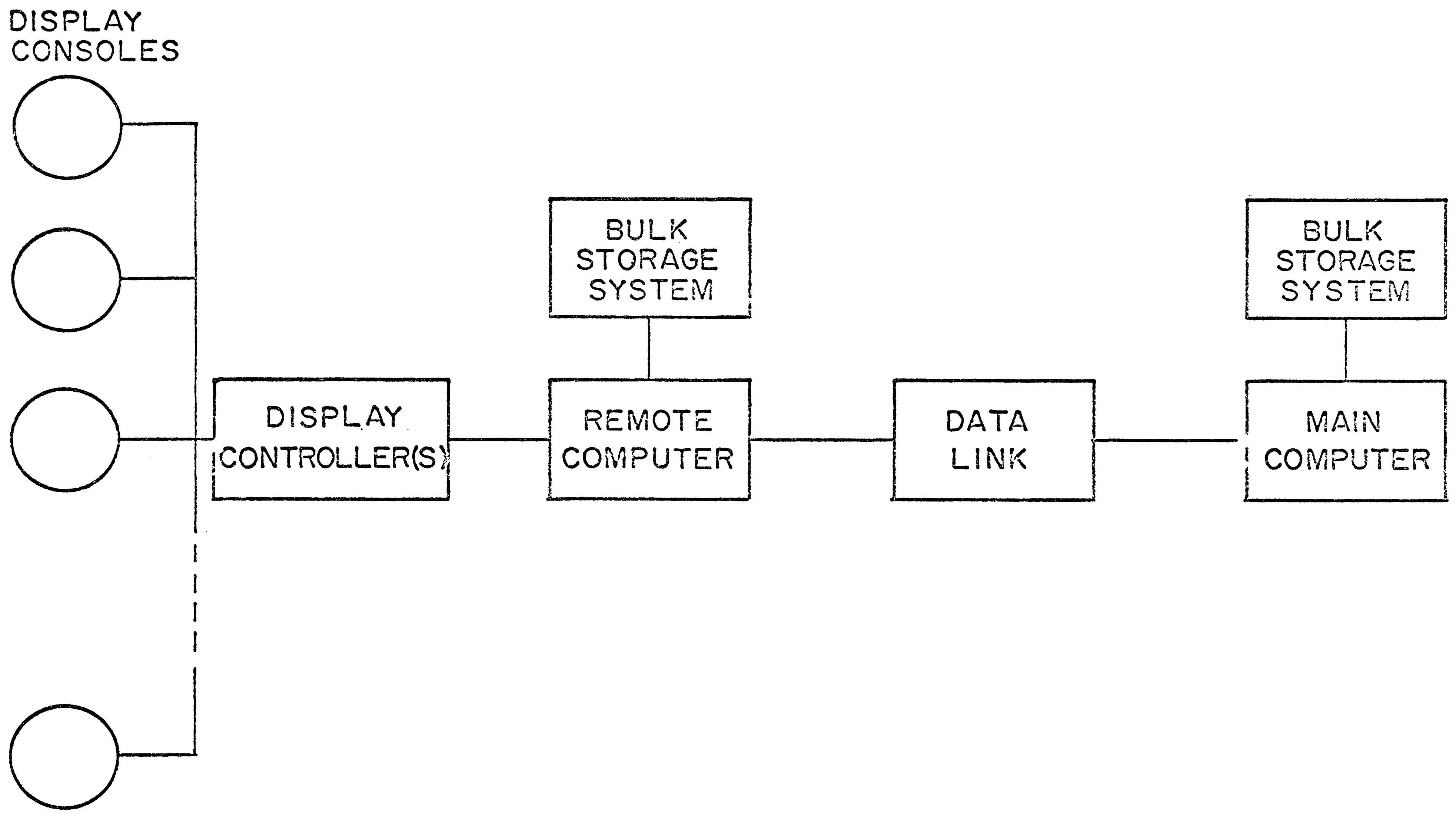


Figure 3-8

Display System

hardware-hardware and hardware-software tradeoffs or alternatives which can be exploited. By hardware-hardware tradeoffs is meant the possibility of increasing the capability of one display subsystem in exchange for decreased capability of another display subsystem, while keeping response time constant. The purpose of juggling hardware in this manner is of course to minimize cost. As a specific example, if we wish to maintain a specified response time at the display console and wish to decrease the data link speed, it is necessary to increase the "power" of the remote computer/display control, or the amount of core storage, or the amount of remote bulk storage, in order to compensate for the extra data transmision time. The converse also applies. Increasing remote computer power cuts computation time, while increasing core storage decreases bulk storage accesses, either at the terminal or at the main computer, and increasing remote bulk storage cuts down on data link usage.

In some cases, however, it may not be possible to completely compensate for a lower transmission rate. This will depend both on the decrease in transmission rate and on the relative usage of the four system components. Specifically, a small decrease in transmission rate for an infrequently used data link is far easier to accommodate than a large decrease in a heavily used link.

Similar statements can be made with respect to each of the other system resources: a decrease in any one can be compensated for by increases in one or more of the other resources, within certain limits.

With hardware-software tradeoffs, we are referring only to the remote computer-display control. Certain display-oriented functions can be implemented either by software with the remote computer or by hardware in the display control. Several possibilities exist here.

When a position indicating device, such as a RAND table [3] is used at the display console, it is often necessary to correlate a position with an entity currently being displayed on the CRT. This can be done with software, or with display control hardware which continually compares the current CRT beam position with the indicating devices' position [10]. The first method can consume much remote computer time, but costs nothing: the second method takes neither computer time nor display control time, but does take money.

If, on the other hand, a light pen-type entity indicating device is employed, its position will frequently need to be known: this is the familiar light pen tracking problem. Once again, the work can be done with either special purpose hardware built into the display control, or with a program running on the remote computer. A current hardware implementation takes about 10% of the display control's time, decreasing by a like amount the quantity of flicker free material which can be displayed [16]. Software implementations of various pen tracking algorithms do not affect the display, but do require remote computer time to execute.

One of the most demanding display functions is the rotation of a three dimensional object, which requires a matrix multiplication operation of six scalar multiplications and four scalar additions for each point and line of the display. Implemented in software, this can be very slow, and can limit the smoothness and rate of dynamic rotation. A first step toward improvement is adding hardware multiplication to the remote computer. A second step is implementation, in the display control, of the actual matrix multiplication. There are two current manifestations of this second possibility. One uses binary rate multipliers, followed by digital addition [16]. The second uses analog multipliers and analog addition [1].

Hardware facilities for display subroutining allow one display list to be used many times in the course of drawing a picture, and therefore avoids needless duplication in core of display instructions. This is all a direct parallel to subroutining for computer programs.

Similar display control hardware-remote computer software tradeoffs exist with respect to problems of dashed lines, blinking lines, transfer of control, recursive subroutining, displaying lines, and displaying alphanumerics.

With this multitude of tradeoffs between the various display system components, an important question arises: for a given display system application, and a given dollar cost, what combination of display subsystems will produce the best possible service for display users? The best hardware will produce the fastest, or minimum, average response time experienced by the display system's users.

What is needed for use by display system designers is a rigorous objective method for evaluating the effects upon system cost and response time of the various tradeoffs discussed qualitatively in the previous section. By now the qualitative tradeoffs, which are in fact rather obvious, are well understood. The tradeoffs need to be quantified for the sake of intelligent systems design, because the consequences of using poor systems design are the overloading of some subsystems, under utilization of other subsystems, and decreased productivity for the system's user. The work reported here has been conducted with this goal always foremost.

3.2.2 Display System Model

In order that display systems be studied in a rigorous manner, particularly to find optimum display systems, a mathematical model or abstraction of how a display system operates is needed. To be useful, the model must reflect the varying capabilities of the display subsystems; the remote computer-display control, the data link, and the remote terminal's core and bulk storage. The model must also be sensitive to the varying computational, storage, and data transmission requirements of the many different applications which might be implemented with a display system. Furthermore, any explicit or implicit assumptions imbedded in the model must be tenable. Finally, the model, when appropriately analyzed, must yield some measure of system performance; specifically, the system's response time will be the desired performance measure.

A model which satisfies these requirements has been developed. It possesses the properties that as the capabilities of the display subsystems increase, the system's response time decreases and its cost increases. These two complementary properties will greatly facilitate the finding of optimum display systems. The model also includes a mechanism for dividing display processing between the main and terminal computers.

3.2.3 Analysis Techniques

The manner in which the display system can be analyzed to find response time is of significance. When the display terminal serves only one display console, no queueing occurs in the system. An analytic expression is then available to calculate response time. However, when queueing occurs, either simulation or Markovian (queueing) analysis must be used.

A problem arises here. Markov analysis requires that the model possess certain properties; there is unfortunately no assurance that the display model satisfies the requirements. Simulation, on the other hand, places virtually no restrictions on the model. The problem is that in terms of computer time, simulation costs from 5 to 15 times more than Markov analysis.

Because of this problem a study was made to compare the results of simulation and Markov analysis as applied to the model, even when the model did not meet the requirements for Markov analysis. The results showed less than a seven percent difference between the two analysis methods for the specific parameter set used with the model. While this result is definitely not generalized to other models or to greatly different model parameters, it does justify the use of Markov analysis for the work reported here. This means that the Recursive Queue Analyzer (RQA) programs developed by the Systems Engineering Laboratory[18] can be used to analyze the model.

3.2.4 Optimization

An optimization procedure has been devised to find the display system (set of four subsystems) which minimizes response time subject to a cost constraint. The optimization accepts as inputs a paramaterized description of the display system's application, and descriptions of up to sixteen choices for each of the four display subsystems.

The most important feature of the optimization is that it minimizes the number of times RQA is used: even though RQA is less expensive to use than simulation, it is still not cheap. Typically in examining about a thousand possible display systems, RQA will be used but two or three times during the optimization.

3.2.5 Evaluation of Computing Power

A critical necessity in the optimization is the creation of a suitable data base of hardware subsystems from which an optimal display system can be chosen. This can rather easily be done for all subsystems except the remote computer-display control subsystem for which not just computing power, but also display capability must be determined. Display capability must be known because not every display control can display the same amount of information. Only those able to display more than some minimum amount of information can even be considered for inclusion in a display system; this minimum amount of information is application dependent. Having eliminated unacceptable display controls, the remaining display controls—remote computers must be rated according to their computational abilities.

A convenient way to measure the display capability of various display controls is with standard test patterns typical of various applications. The method used by Adams Associates in The Computer Display Review [2] is typical of this approach.

The second part of the problem, measuring the computing power of remote computer-display controls, is a bit more difficult. A list of display oriented macro-level operations was drawn up; such that any and all display work could be performed by programs made up of sequences of these macros. Different pieces of hardware are evaluated by finding the execution time of each macro. These times will vary as the hardware's capabilities change, and will decrease or increase as more or less of the hardware—software tradeoffs are manifested in hardware. By taking a weighted sum of the execution times (where the weights are application dependent), a particular remote computer-display control is evaluated for a particular application.

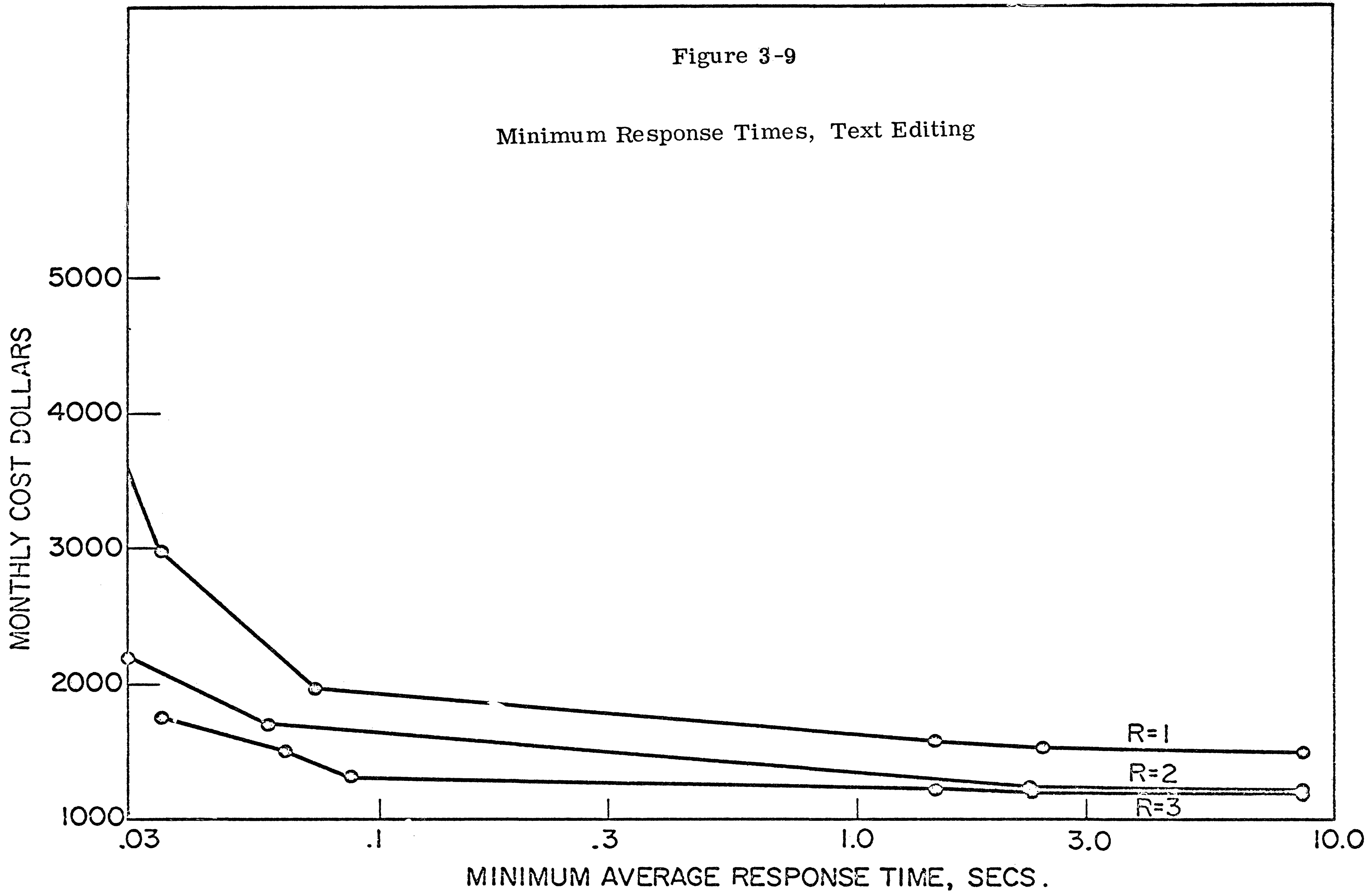
3.2.6 Results

Four display system applications were selected for close study. They are text editing, general two-dimensional drawing, general three-dimensional drawing, and general network analysis. These four applications were chosen for their differences; that is, they each use the facilities of a display system in different ways. For each application all the parameters needed by the model and optimization were estimated (not measured). Optimizations were performed for each of the four applications for systems with one, two, and three display consoles, and for different levels of capital investment. This resulted in a large number of display system designs. The per console cost and average response times of these systems are graphed in Figures 3-9 to 3-12 . R refers to the number of display consoles in a display system. There are no display systems below and left of the curves: there are many display systems above and right of the curves, but they are nonoptimum. They cost more and give poorer response times than systems on the curves. If a display system is built, it should be chosen from any one of the curves, so that it is optimum. Choosing which one of several systems to build can be based on a cost criteria, a response time criteria, or on some combination of cost and response time which gives a cost-effectiveness measure, such as interactions per second per dollar.

It is useful to notice the changing slopes of the graphs. For long response times, a small additional investment yields very good returns in

Figure 3-9

Minimum Response Times, Text Editing



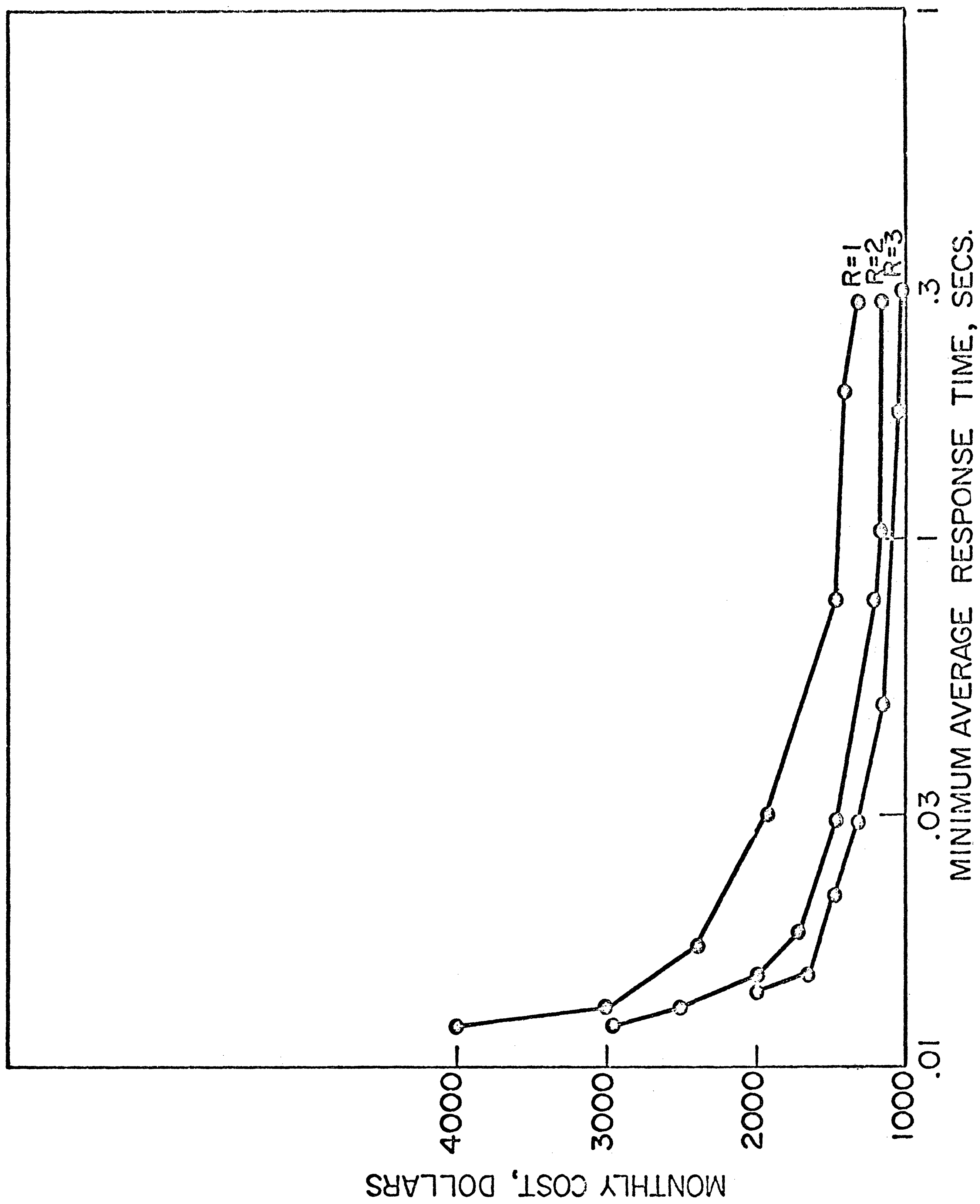


Figure 3-10

Minimum Response Times, Two-Dimensional Drawing

Figure 3-11

Minimum Response Times, Three-Dimensional Drawing

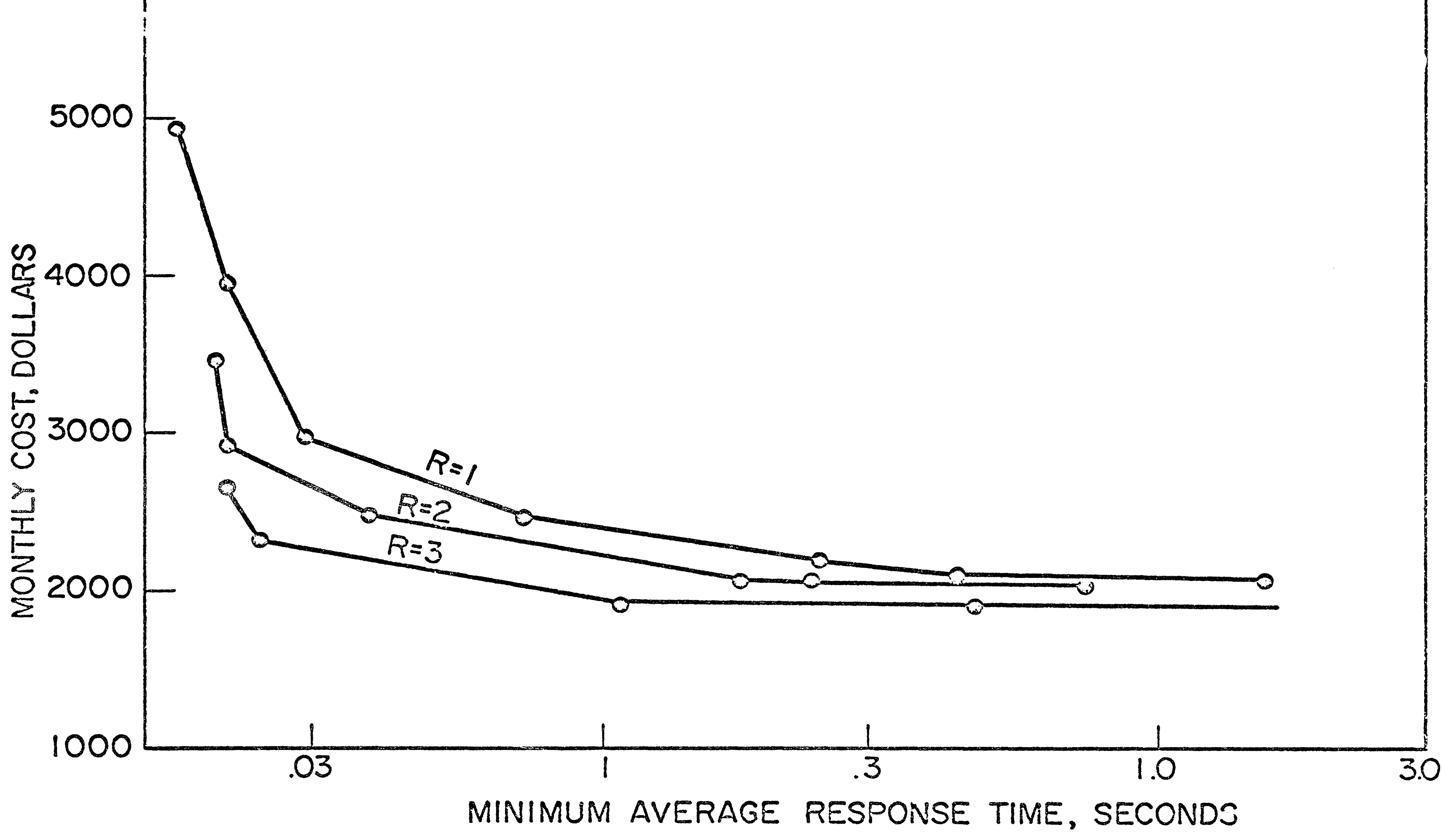
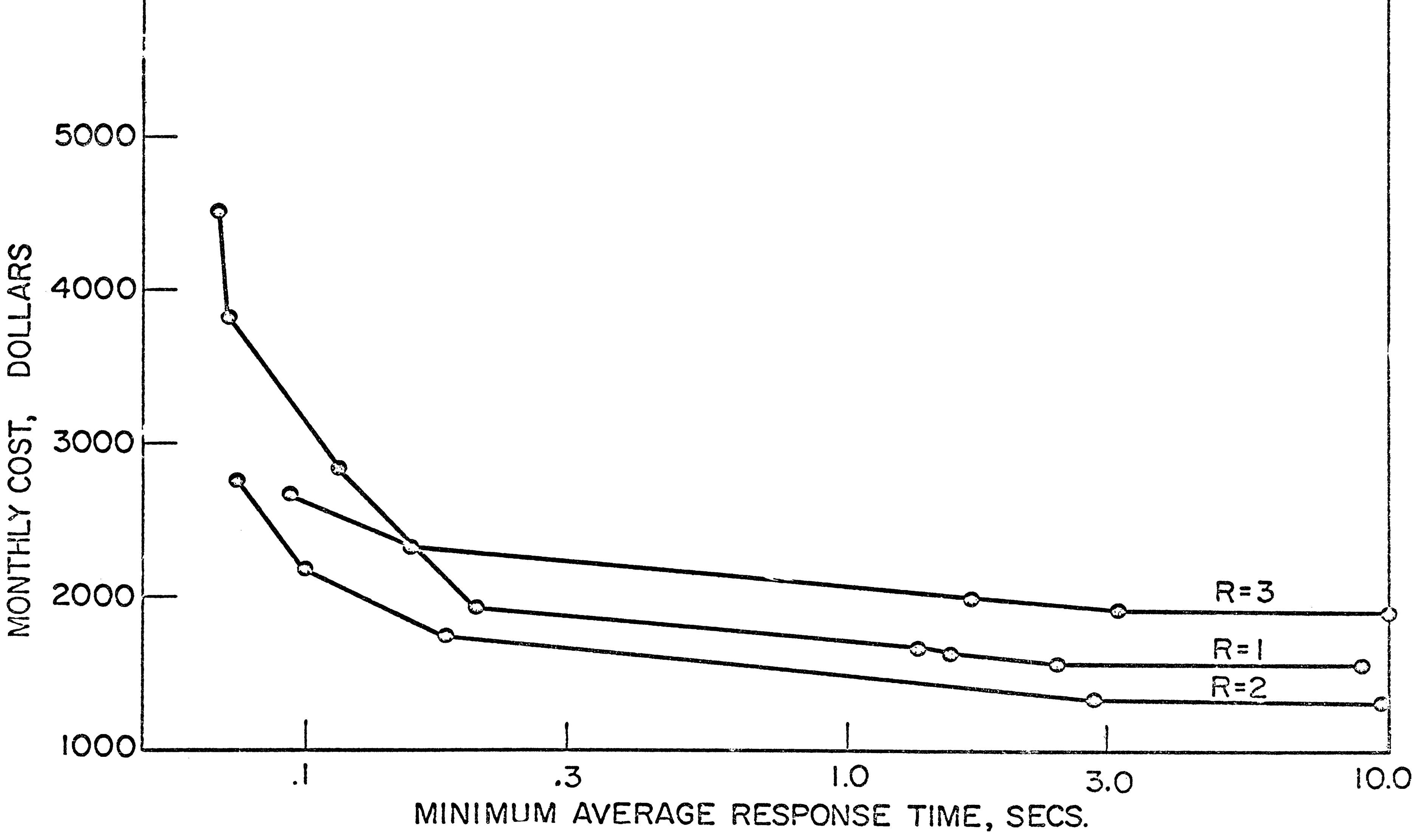


Figure 3-12

Minimum Response Times, Network Analysis



terms of decreased response times; the returns diminish as response time continues. By studying this behavior and by correlating it with the hardware used in each of the display systems, a set of general design guidelines has been developed. The read as follows:

> "A satisfactory inexpensive display system uses a voice grade data link, no bulk storage, little or no core storage beyond the min imum needed, and the least expensive remote computer-display control. For little additional expenditure, the addition of a significant amount of bulk storage provides better response time. Inexpensive increases in the remote computer-display control's capabilities are also helpful. Further response time decreases are achieved with broad band data link speeds and more bulk storage. Additional response time improvements are obtained (at high cost) first by improving the remote computer-display control and then by using more core storage." [8]

To justify the necessity for these guidelines, Figure 3-13 shows just how severe the penalties of using a nonoptimal design can be. The figure plots response time versus cost for the network analysis application with three users. Both optimum and worst case response times for various values of cost are plotted. The worst case response time is the maximum response time of those display systems which cost the same as an optimum system. The graph's interpretation is that, for instance, is that if $2340 per month is to be spent on a display system, the response time can range from .155 seconds (Point A) to 3.92 seconds (Point B). These times differ by a factor of 25. Also, the graph can be interpreted to show that if a response time of .1 seconds is needed, the display system's monthly cost can vary from about $2600 (Point C) to about $3400 (Point D). This represents an unnecessary expenditure of up to $800!

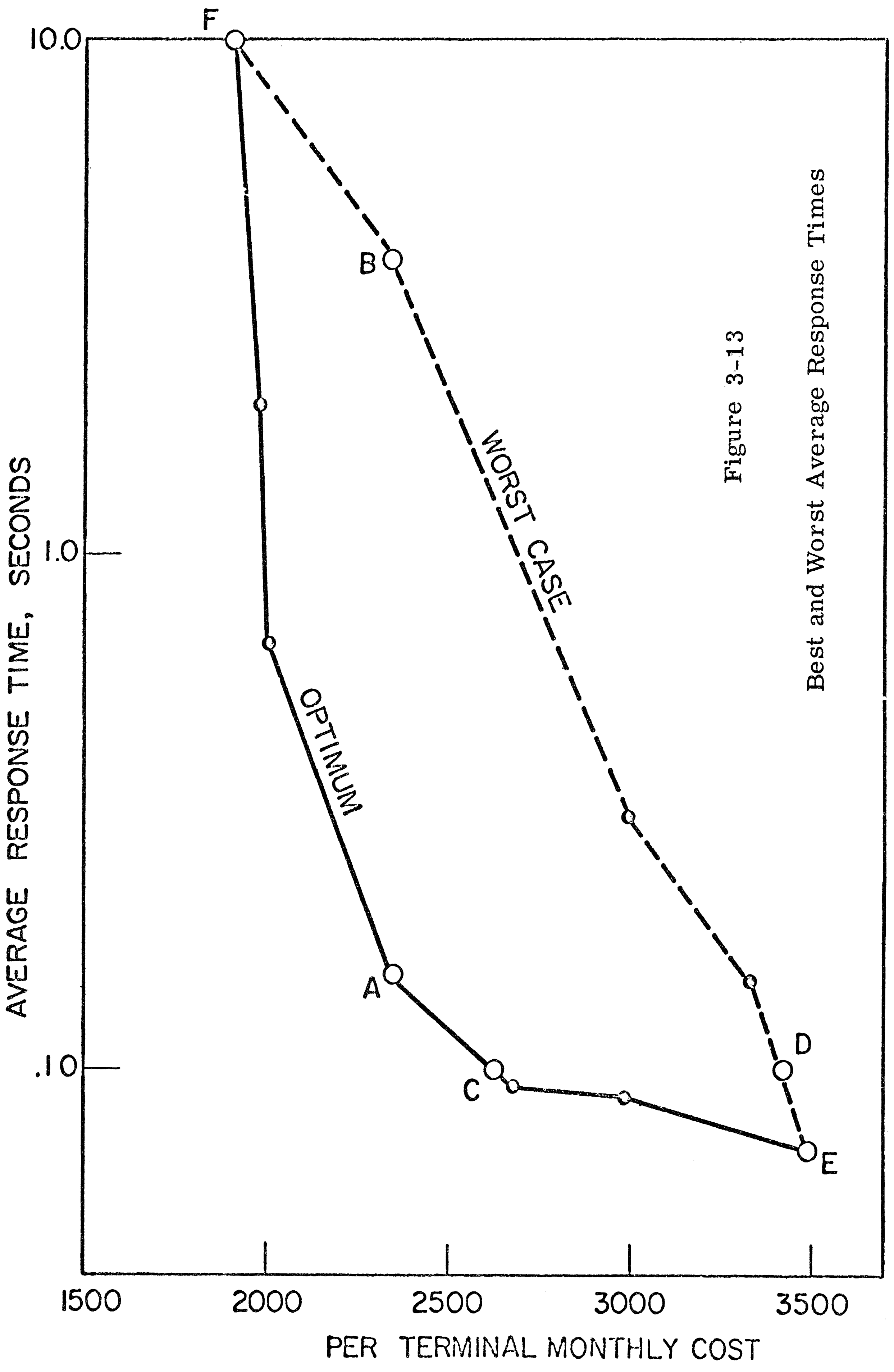


Figure 3-13

Best and Worst Average Response Times

In conclusion, the differences between optimum and worst case display systems are significant, with respect to both cost and response time. Therefore, display system designers must have at their disposal means of making intelligent design decisions, because the consequences of making bad decisions are too serious.

3.3 DESIGN OF OPTIMUM BULK MEMORY SYSTEMS

3.3.1 Introduction

The purpose of this work is to provide usable techniques for designing large scale computer storage systems. The approach being taken has been to develop techniques for the optimal design of such systems. Our concern in this research is assembling existing components rather than developing new ones.

In the normal process of developing optimal design techniques it is common to encounter obstacles which require that simplifications or constraints be made in the mathematical models being used. The final result is a precise, well formulated technique which will design optimal memory systems under a set of restricting conditions and assumptions.

There are several uses for such an optimal technique. First, one may find many situations where the memory design problem to be solved does not conflict severely with the restrictions imposed by the technique so that an exact optimal solution may be found. Second, when the problem to be solved does not conform to the restrictions of the techniques, often much can be learned by solving related problems which do conform to the restrictions. Third, the rigorous optimal technique provides a base for future work as well as the possibility of generating general theorems relating to memory structures.

Having stated the general goals and direction of this work, we now turn our attention to the specific goals of the optimal memory technique which has been developed to date. We will also discuss the specific restrictions which limit the present techniques as well as what direction has been and will be taken in removing or reducing some of these restrictions.

3.3.2 Design Method and Restrictions

Progress has been made along two branches. The first is the development of improved mathematical techniques. The second is the implementation of those techniques on the computer. As would be expected the development of a mathematical techniques precedes its implementation by a considerable period of time.

At the beginning of this past year, the mathematical limitations of the method which had been developed required that the size of the system, i.e. the total number of storage bits, be stated. The method also required a description of how storage will be used. This took the form of the expected value of how many times a given unit of storage (a unit of storage may be a bit, word, page, etc.) is accessed, divided by the number of times that the entire system is accessed. The method also required a statement of the cost and specifications of all devices which might be used to construct the storage system. From this information a set of configurations could be generated which represented the optimal storage systems having average access times which vary over a predetermined range.

During the past year all the capabilities of the earlier methods were realized in software. Figure 3-14 shows part of the results obtained from a single run of the current software. In this case the total system capacity is fixed at four million bits and the devices used to build the system were fictitious. This computer generated graph shows the optimal storage systems generated, with the system cost in dollars plotted versus the average access time of the system. Each asterisk in this figure represents a com plete storage system composed of from a few to many devices with varying characteristics. These systems were selected as optimal by taking into account the various factors discussed above for a specified case.

There are several interesting facts to be discussed concerning this plot. Because of the discrete nature of the components used the resulting optimal systems are discrete. The large gaps between some optimal systems is caused by large gaps in device performance. The average access time of a typical core memory is about 10, 000 less than that of a typical drum. There are 34 systems plotted and there are only 34 optimal systems in the range considered. This implies that between system A and B of Figure 3-14 no system exists with a cost less than that of system A.

Notice that it is simple to build systems above and to the right of the plot. That is, a system with a capacity of four million bits, a .1 sec. average access time, and a cost of ten million dollars is easy to design.

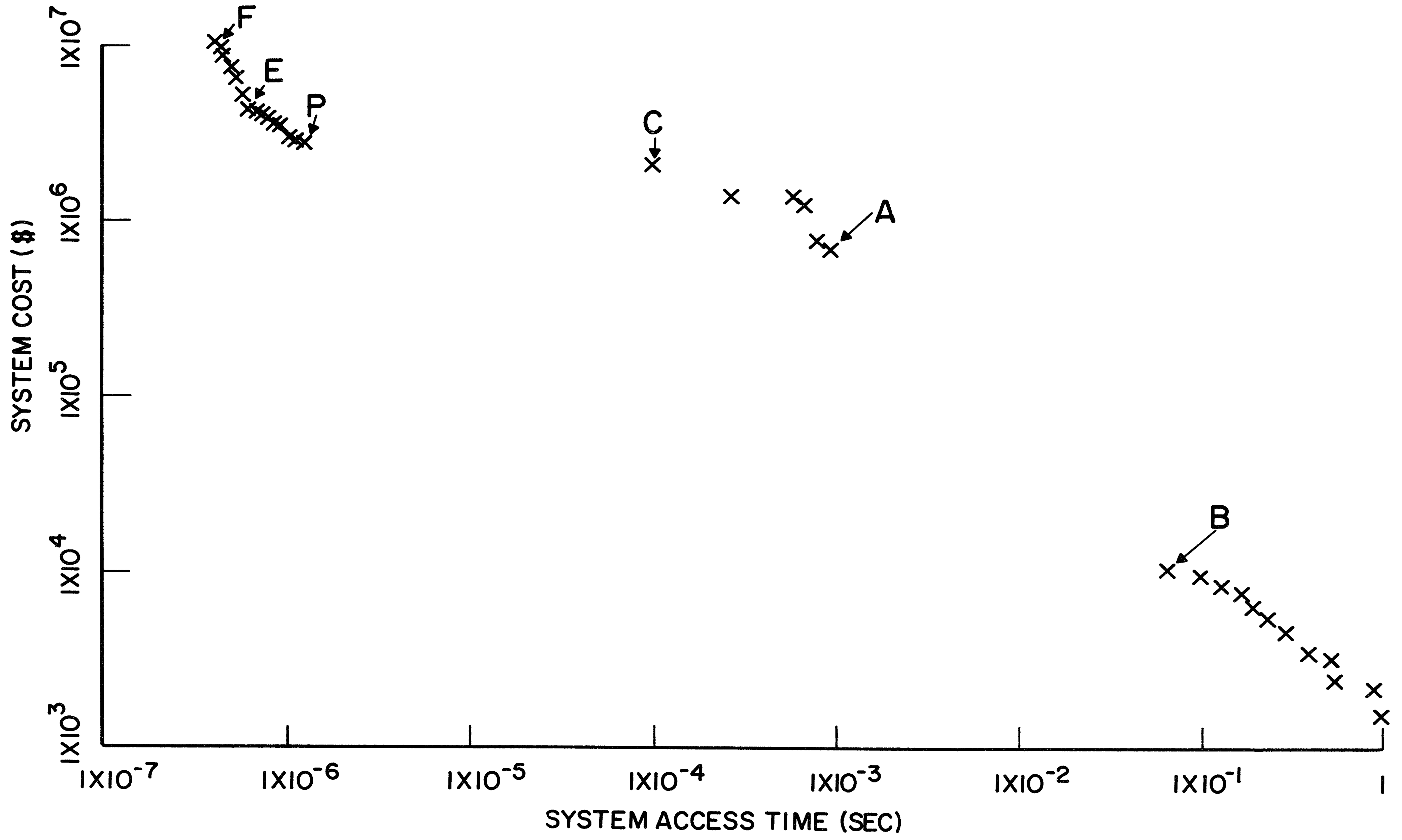


Figure 3-14 Storage System Cost and Access Time

Conversely, it is impossible to design systems below and to the left of the plot of optimal systems shown.

The general shape of this curve is also important and can be utilized in the design considerations of the overall computing system. Notice that between systems C and D there is very little increase in cost and a corresponding large increase in performance. Between systems E and F there is a relatively small performance gain corresponding to a large cost increase.

During the past year the storage system optimization procedure has progressed, becoming extremely general. Currently in principal there is no limit to the detail which may be introduced in the description of how the system is to be used. Correspondingly, there is no limit to the detail with which prospective components may be described. There is also a wide range of possible relationships between components selected and system performance which may be used. In order to realize this new flexibility in the software implementation of this procedure, great care must be exercised to isolate the basic optimization procedure, which does not change, from the software which deals with the details of a specific problem. This partitioning of software (into subroutines) will have two effects.

1. Maintain the new flexibility of the technique in the software implementation
2. Result in a more logical and manageable software organization, which will be easier to maintain.

During the past several months numerous changes have been made to the internal structure of the programs in order to effect this partitioning of the software. These changes have not changed the operation of the implemented algorithms in any way but have increased the flexibility and maintainability of the package.

Other changes are being made in the software. These changes are of a different nature and involve the basic process of optimization. Among these are several small modifications which provide increased performance and improved error checking. One large modification is under way which will result in a major change in performance. Currently the dynamic programming process must take rather coarse increments in capacity during its operation. This results in an error which is almost undeterminable. The major change now being effected will allow the increment size to be reduced by a factor of about 10^{-6}.

This major change will be the result of approximately four relatively small changes and one major addition. The small changes may be made without changing the basic operation of the system and serve a preparation for the major addition.

The modifications discussed above have been in progress and will remain in progress for some time to come. It is expected that they all will be completed by the end of 1969.

Both the old and the new approaches produce the same result. That is, the least expensive storage system having an average access time equal to or less than some fixed time, and the storage system which gives the smallest average access time for a cost equal to or less than some fixed cost. In either case, the storage system will be completely specified. The most important change is increased flexibility and increased accuracy.

One of the primary restructions remaining concerns the problems of describing how storage will be used. This concerns the difficulties involved in predicting the usage of a future system. There is a straightforward problem of not being able to closely predict the future. A second and far more complex problem, however, arises from the fact that the way in which a storage system is used is often a function of its design. Therefore we may face the problem of designing a storage system for an existing computing system whose storage usage characteristics are not changing with time, yet the introduction of a new storage system will cause those characteristics to change. One may be able to predict the changes in usage which will occur due to the introduction of a new storage system, but this will not be simple. For many situations we may be able to reason that any changes in usage which take place due to the introduction of a new storage system, which is optimal for present usage, will result in further improvement of the overall system economy.

An interesting situation arises when a computing system is performing some well defined service for which there are several different implementation methods. Very often the specific method used is dictated by the storage system design. It would be possible under these condi tions to compare the methods when each is using an optimal storage structure. In this way we may select an optimal method and an optimal storage system. An example of this type of problem is found in large scale information retrieval problems.

A second major restriction is imbedded in the definition of average access time which is being used. In the older technique it was assumed that, regardless of the number of storage devices, there was no overlap in its usage; that is, each storage request is handled sequentially. The new techniques allow much greater flexibility in this area.

Another limitation which the older method encountered is accuracy versus computer time. This problem has been all but eliminated in theory and only awaits implementation.

Accompanying the effort to improve the basic theory and its associated software has been an effort to bring this technique to bear on real design problems. As a result a data base of components has been structured and will be added to in the coming months. Also, considerable effort has been expended in making the results produced by the computer as useful as possible to the user. This is being done by presenting the results in graphical form.

During the coming year a complete report on the most advanced technique will be written.

REFERENCES, CHAPTER 3

1. Adage, Inc., System Reference Manual—Adage Graphics Terminal, Boston, 1968.

2. Adams Associates, The Computer Display Review, Bedford, Massachusetts, 1968.

3. David, M. R. and T. O. Ellis, The RAND Tablet; A Man-Machine Graphical Communication Device, Proc. FJCC, Spartan Books, Washington, D.C., 1964, pp. 325-331.

4. Dawson, D. F. et al., "Computer-Aided Design of Electronic Circuits—A User's Viewpoint," Proc. IEEE, 55, 11 (November 1967), pp. 1946-1954.

5. Dertouzos, Michael L., "An Introduction to On-Line Circuit Design," Proc. IEEE, 55, 11(November 1967), pp. 1961-1971.

6. Doll, Dixon R., "The Efficient Utilization of Resources in Centralized Computer-Communication Network Design," SEL Technical Report No. 36, The University of Michigan, Systems Engineering Laboratory, April 1969.

7. Esau, L. R. and Williams, K. C., "On Teleprocessing System Design, Part II, A Method for Approximating the Optimal Network," IBM Systems Journal, 5, 3(1966).

8. Foley, James D., "Optimal Design of Computer Driven Display Systems," SEL Technical Report No. 34, Systems Engineering Laboratory, The University of Michigan, March 1969.

9. Koford, J. S., et al., Using a Graphic Data-Processing System to Design Artwork for Manufacturing Hybrid Integrated Circuits, Proc. FJCC, Spartan Books, Washington, D.C., 1966, pp. 229-246 246.

10. Konkle, K. H., "An Analog Comparator as a Pseudo-Light Pen for Computer Displays," IEEE Trans. on Electr. Comp., 17, 1(January 1968), pp. 54-55.

11. Kruskal, J. B., Jr., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," Proc. Am. Math. Soc., 7:48-50(1956).

12. Lewin, Morton H., "An Introduction to Computer Graphic Terminals," Proc. IEEE, 55, 9(September 1967), pp. 1544-1552.

13. Prince, David M., "Man-Computer Graphics for Computer-Aided Design," Proc. IEEE, 54, 12(December 1966), pp. 1698-1708.

14. So, Hing C., "OCLA: An On-Line Circuit Analysis System," Proc. IEEE, 55, 11(November 1967), pp. 1954-1961.

15. Spitalny, Arnold and Goldberg, M. J., "On-Line Graphics Applied to Layout Design of Integrated Circuits," Proc. IEEE, 55, 11(November 1967), pp. 1982-1988.

16. Stotz, Robert, "Man Machine Console Facilities for Computer-Aided Design," Proc. SJCC, Spartan Books, Washington, D.C., 1963.

17. Temes, Gabor C. and Calahan, D. A., "Computer-Aided Network Optimization— The State-of-the-Art," Proc. IEEE, 55, 11(November 1967), pp. 1832-1863.

18. Wallace, Victor L. and Rosenberg, R. S., RQA-1, The Recursive Queue Analyzer, Systems Engineering Laboratory Technical Report no. 2, The University of Michigan, Ann Arbor, Michigan, 1966.

4. MEASUREMENTS

It is always desirable to understand how the various components of a computer system are utilized. Such knowledge can be helpful when used to analyze and modify existing systems, or to synthesize new systems. Existing systems often need modification to decrease response time or to improve the utilization of system resources. New systems need to be designed to give high levels of performance from the start of their operation.

The following sections describe two different types of data collected from the Michigan Terminal System (MTS), which is a time-sharing system running on an IBM 360/67.

4.1 GENERAL MTS DATA

This section describes the general data collected from MTS [4] using a data acquisition system [3] built into the MTS operating system. The information has been taken from a more detailed report on Virtual Storage Computer Systems [2].

4.1.1 General Description

The data acquired for this study was taken during the period October 15, 1967, to March 31, 1968, in normal operating periods of the MTS system. Data collection periods ranged from 15 minutes to 7 hours duration, depending on the volume of data being generated and the nature of the jobs being observed. The periods were selected insofar as possible to mirror the typical prevailing demand on the system: unusual circumstances of light load (such as just after system startup or malfunction) and heavy load (such as the hours preceding a student problem due date) were avoided.

Essentially three types of jobs are run in the MTS system:

1) Normal remote terminal, interactive use of MTS with Model 33/35 Teletypewriters, IBM 1050 and 2741 Communications Terminals. (During the data acquisition period the number of such tasks which could be supported concurrently by the system grew from about 4 to 40).

2) Non-interactive use of MTS via batch mode, using an IBM 2540 card reader/punch and 1403 line printer as input/output devices, while providing full use of the command language and other systems features.

3) Peripheral support programs for an IBM 7090 batch-processing system which produce input tapes from punched card, and print and punch output tapes.

During normal daily operation of the MTS system, from 9 a. m. to midnight, the then current maximum number of communications lines for remote terminals was enabled, one or two batch streams were processed, and up to two additional line printers and reader/punches were used for peripheral support jobs. Generally, less than half of the remote terminals were active at one time, one batch stream was rather consistently busy, and an average of two to three of the SPOOLing[1] operations were in progress. Data are not included here for the SPOOLing jobs, since they exhibit a very regular behavior: each I/O wait for tape or unit record device requires 50-200 msec., and is separated from the next such event by 2 to 3 msec. of processing. These peripheral support programs create a maximum of about 15% of the CPU load, and normally only 5 - 10%.

Data given here for the use of the CPU and I/O devices are separated for batch and conversational tasks. The I/O delays are also shown separately for different kinds of devices. A great deal of information is implicit in the data being collected which is not shown here, and additional kinds of data can be collected to answer specific questions.

At least three distinct points of view could be taken to govern the organization and collection of computing system data: workload, system, or user. For the purposes of this investigation we have regarded their relative importance in the given order. For example, in displaying CPU data, we are more interested in the CPU intervals requested by a task than by the service actually supplied by the system. Again, when considering I/O delays, we are concerned about the length of time the system must wait before it can resume processing a task rather than the time an individual must wait to receive his answer. Thus in our case some output delays appear to take almost no time because the lines are buffered and the computation can proceed while the line is still being typed.

----

[1]Simultaneous Peripheral Operations On-Line

4.1.2 Specific Characteristics

Some facts about the MTS data chosen for display in this section are given in Figure 4-1. The number of jobs is the approximate number of University of Michigan Multi-Programming System (UMMPS) jobs using MTS that were observed with the data collection facility. The number of tasks is the approximate number of individuals who used these jobs during the collection interval. The total number of unit record devices, communication lines, disks, etc., that were referenced by these jobs is also given. The identification numbers 0-5 of these different sets of data will be used to label graphs in the succeeding figures.

Several general observations can be made about the environments in which the data was taken. MTS # 4 was obtained after only a few weeks of experience with paging in MTS. At that time the system had a communications line capacity of about 10: at most 10 conversational users could run in addition to the batch and SPOOLing operations. Furthermore, use averaged considerably less than the capacity. The data sets # 2 and # 3 include only data for the batch streams active at the time. Because they use unit record devices instead of remote terminals, batch jobs are processed much more quickly and create a heavier system load than conversational jobs. More I/O operations generally occur in batch, since people take advantage of its lower cost and speed for input and output functions.

The data # 1 was taken with 10-20 conversational users, which approximated an average load at that stage of MTS development. With less than

General Characteristics of the MTS Data

ID	MTS # 0	MTS # 1	MTS # 2*	MTS # 3*	MTS # 4
Date	3-20-68	3-18-28	3-18-28	1-15-68	11-29-67
Time	15:27	15:36	11:41	14:51	10:23
Duration	20 min.	76 min.	150 min.	79 min.	266 min.
# Items	542246	1355274	1123503	466223	705890
# Jobs	40	40	2	1	9
# Tasks	50	150	75	16	84
Devices	57	65	29	20	29

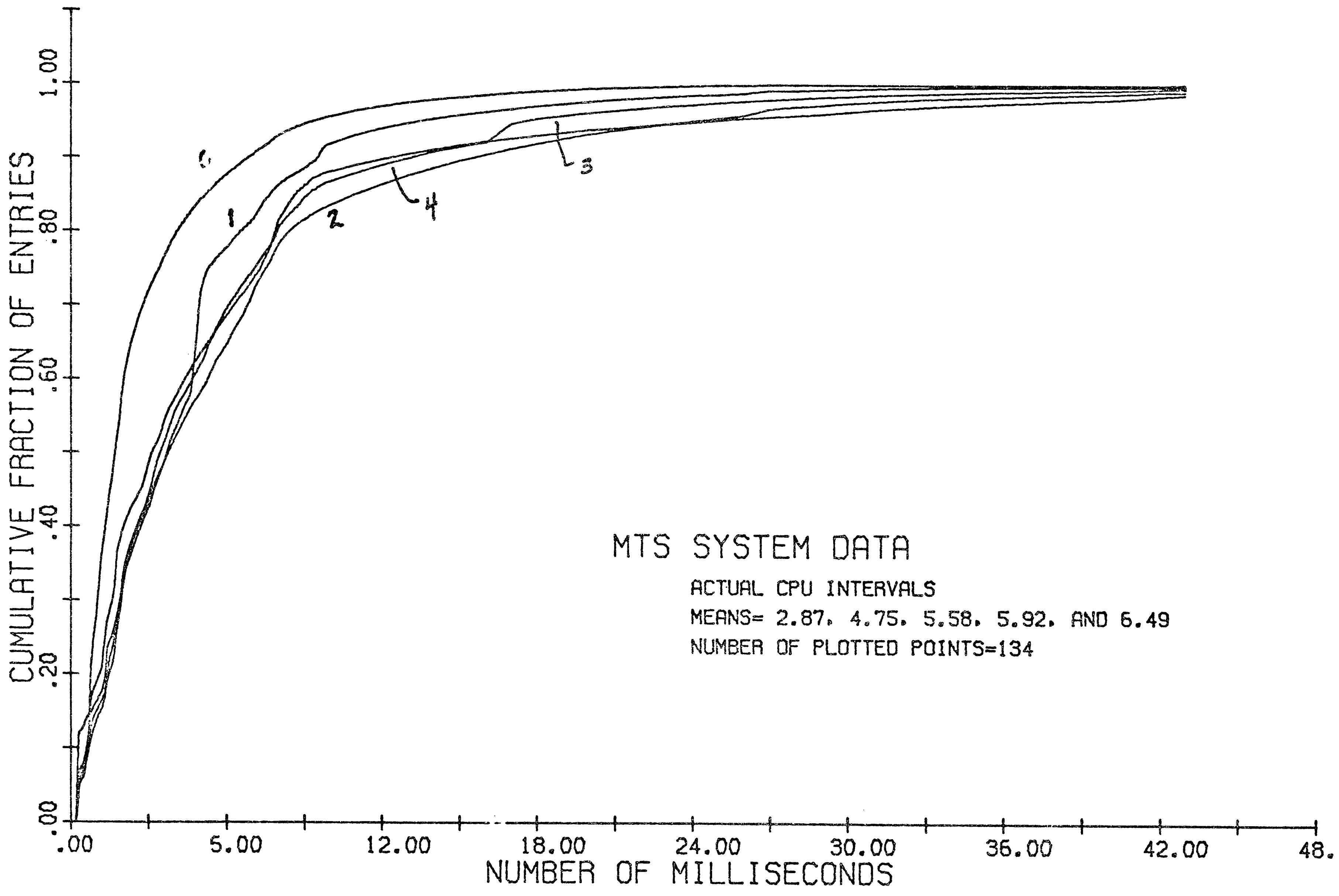
* Denotes batch job data.

Figure 4-1

MTS Data

about 16-20 users, the paging drum frequently had periods of inactivity: most command chains for I/O operations were half-empty, and there was often no channel program in progress at all. In order to observe tasks under a heavier system load, data set = 0 was taken while a special background job (called PAGE-IT) was running to increase drum activity. The PAGE-IT job acquires a large number of virtual storage pages and references them cyclically in rapid succession. When run with a moderate load of normal taks, PAGE-IT keeps drum channel programs running more or less continuously and forces other tasks' pages out of main storage at a faster than normal rate.

Figure 4-2 shows the distribution of actual CPU intervals obtained by tasks during the data collection periods. Any interruption in service to process another task ends the CPU intervals appearing in this distribution. One to four percent of these intervals lie in the neighborhood of about 300 microseconds, or about the minimum time required by UMMPS to service an interrupt which requires little or no processing. The smoothest curve, and the one with the smallest mean, is that of data set # 0. The reason for this is that the system was the most occupied with ordinary tasks at that time, and PAGE-IT was running to force additional page-wait interruptions for the normal tasks. Since # 0 was the latest of the given data to be taken, it also reflects some improvements made to the system which make it operate more efficiently under periods of heavy load.



MTS SYSTEM DATA

ACTUAL CPU INTERVALS

MEANS= 2.87, 4.75, 5.58, 5.92, AND 6.49

NUMBER OF PLOTTED POINTS=134

Figure 4-2

MTS CPU Intervals

Actual CPU intervals depend on the frequency of interrupts, hence it is not surprising that the data set (# 2) with the longest mean value was collected over a noon hour. Except for that case, the mean length of a CPU interval decreases steadily with time—average system load grew significantly over the period of study. Data set # 1, whose curve shows a large number of intervals of length about 4 msec., was taken during the testing of a new I/O routine when an error occurred, and over 28, 000 I/O operations were executed in rapid succession, generating an equal number of short CPU intervals.

In Figure 4-3 we have distributions of CPU intervals requested by individual tasks: here the interruptions in service to process other tasks are removed from the data, so that the given times represent CPU intervals terminated only by I/O and paging operations for the task using the CPU. The curves for # 3 and # 4 also accumulate time across paging delays for the given task—they show the distributions of CPU time requested between I/O operations. We note here that both of the latter curves exhibit a gap in observed values near the origin: there are a few nearly immediate I/O operations, but then almost none of duration 800 to over 2000 microseconds. The difference between the curves for # 3 and # 4 is probably due to the fact that the former data is for batch jobs, which generally do more I/O operations.

Turning to the curves for # 0 and # 2 in Figure 4-3, which represent CPU intervals terminated by either page or I/O wait for the given task, we see again that a heavy paging load (# 0) significantly reduces the mean length

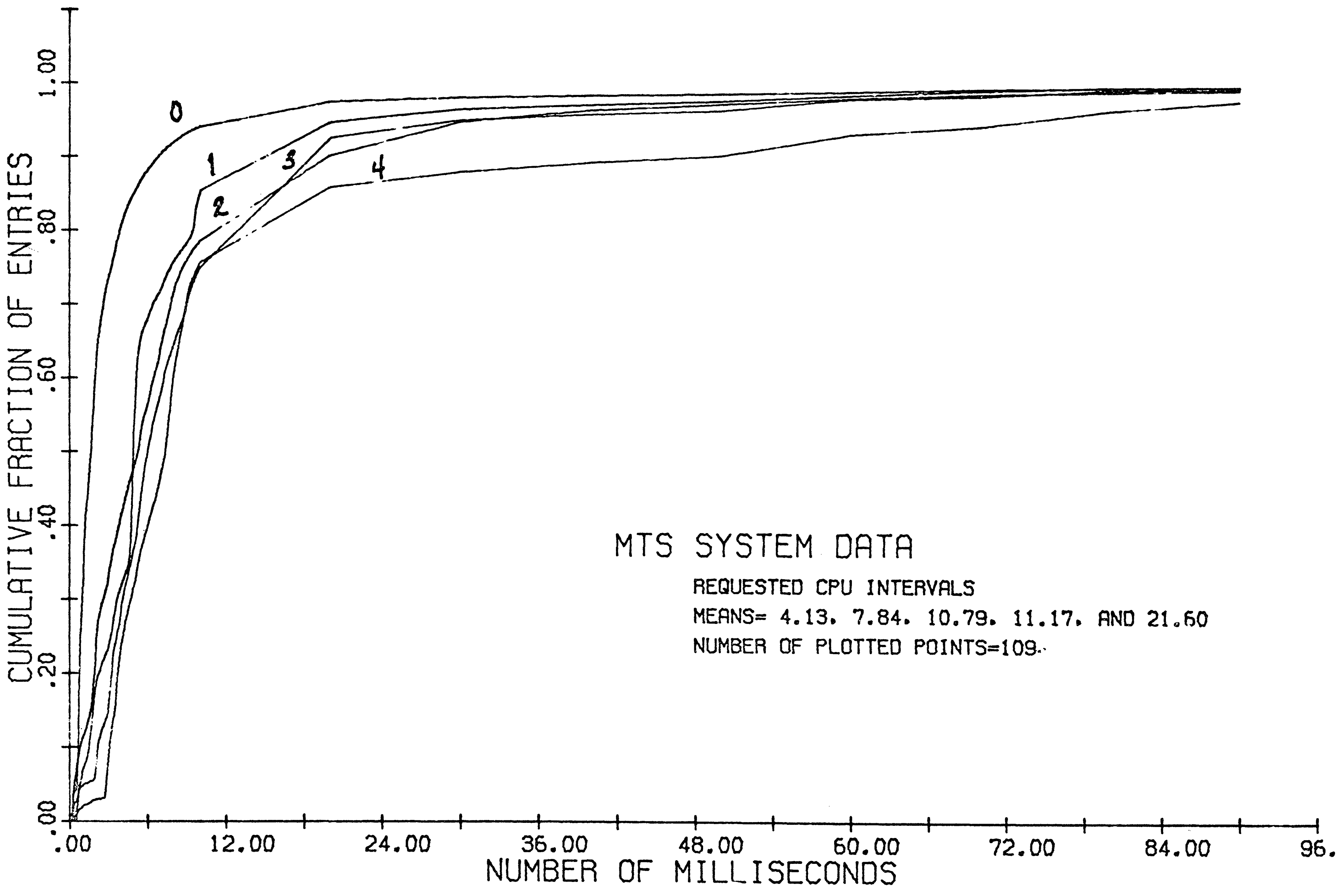


Figure 4-3

Requested CPU Intervals

of a CPU interval. There is too much variation in the origin and makeup of batch jobs to draw hard and fast conclusions from differences between the curves for # 1 and # 2, which represent conversational and batch jobs run on the same day. Batch runs include a number of distinctly different tasks: small student problems limited to batch mode, use of faster devices for listing and card reading jobs, and long computational tasks. The difference between the two curves is probably more influenced by the fact that the batch data was taken earlier in the day, when a lighter overall load contributed an average of fewer page-wait interruptions. The density functions for requested CPU intervals exhibit a number of local maxima in the range 0-6 msec., which are present in almost every case and more noticeable with less paging load. A careful analysis could probably associate these peaks with one or more frequently-used system functions.

Figure 4-4 displays the distributions of page-wait delays experienced with the MTS paging drum processor. The mean values of these distributions strictly increase with increasing system load, which is the reverse order from which the data sets are numbered. Although the given data all represent the time required to obtain a requested page, each curve is a composite of several different kinds of distributions. Once an unavailable page is referenced one of five actions may be taken:

1) It is a new page for which space can be allocated immediately in core (1-5 msec.).

2) It is a new page for which core space can be allocated only after

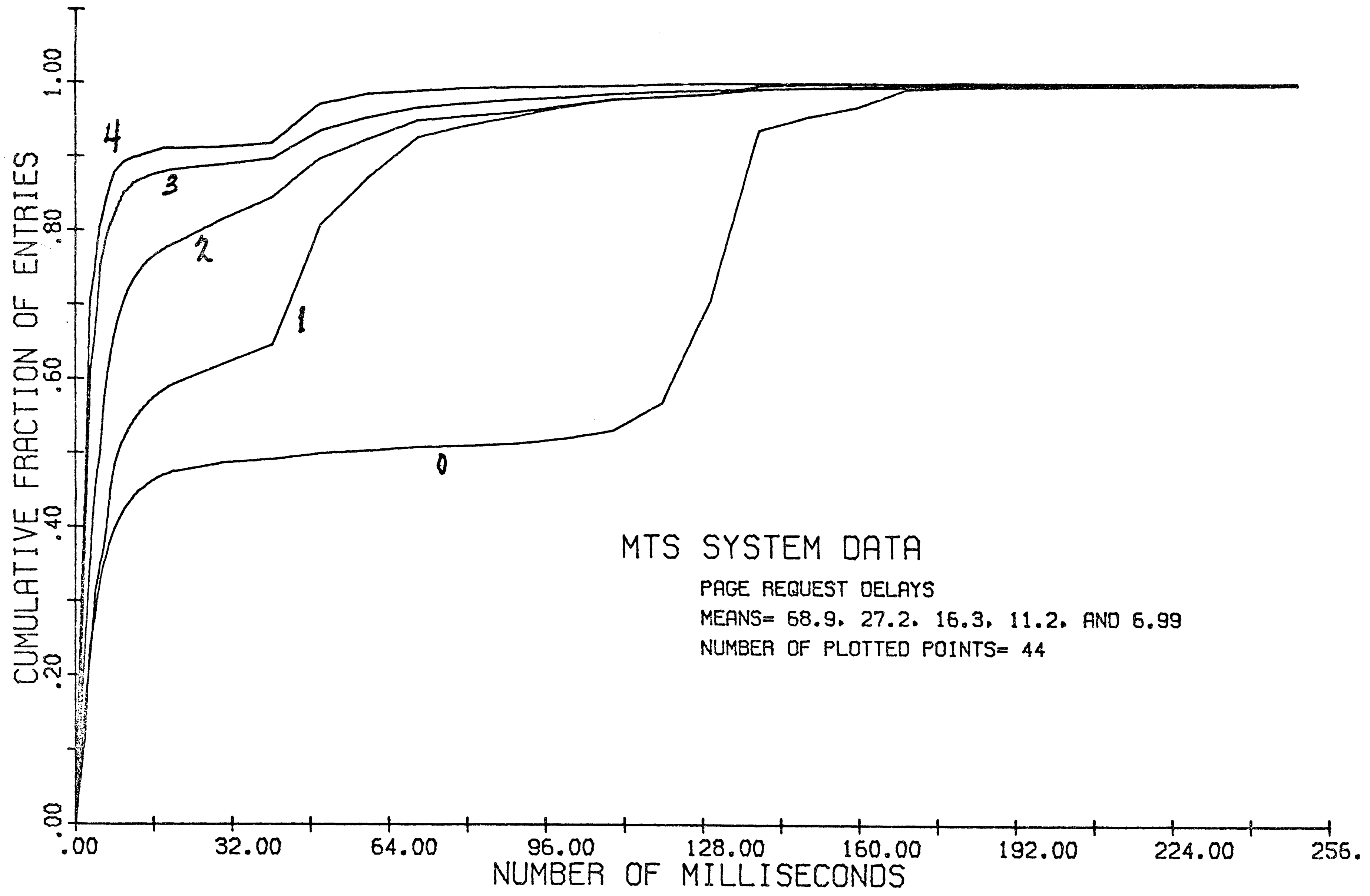


Figure 4-4

Page Request Delays

another page has been pushed out (a written page may be posted at any time)

3) An existing page must be read from the drum (at least 36 msec.)

4) An existing page must be read from the drum but must wait for a write to provide core space.

5) The page may still exist in core even though a write operation is currently in progress, which may be cancelled to make it available immediately.

If a drum operation is required (cases 2 to 4 above) then a further distinction is possible: whether or not the drum is currently under the control of a channel program to which the new request(s) can be chained. If so, a distribution of actual completion time can be found. [2] If the drum is not currently transmitting an additional average delay of half a physical revolution is experienced, since a new channel program is always started at the drum index point. In the latter case, however, it is unlikely that more than a single revolution will be necessary to reach and transmit the desired page.

One final fact is of importance in understanding the distributions of Figure 4-4: MTS page transfers were posted (at the time) exactly once at the end of each logical revolution of the drum. Thus every actual read or write operation is known to be completed only after some multiple of the logical revolution time (35 msec.), plus any delay in synchronizing with the construction of channel programs.

All the data in Figure 4-4 is dominated by the characteristics of start-stop rather than continuous drum operation except # 0, where the drum was forced to run more or less continuously. In that case a great many read operations took three logical revolutions or more. Apparently most available core pages were "reclaimed" or used for newly created pages, so that read requests for drum-resident pages often had to wait for drum writes as well as queueing and read delays. In any case the performance of the drum under the conditions of # 0 leaves a great deal to be desired.

The overall distribution of I/O wait times given in Figure 4-5 is poorly shown due to a gap in the plotted points in the range from 0. 2 to 2. 5 seconds. Most delays fall into three ranges according to the type of device:

a) Terminal and other I/O to buffered devices which appears to take almost no time at all.

b) Disk and unit record I/O, most of which lies in the range from 35 to 70 msec.

c) Terminal output with buffer full, and unbuffered input, which usually takes over half a second.

The curve for MTS # 1 is missing from Figure 4-5 because of the large number of identical I/O waits occurring in the test mentioned earlier. Batch and conversational jobs have distinct distributions—in the former case the longest normal I/O operation is a maximum disk seek, which requires about half a second.

The ready distributions of Figure 4-6 show how the MTS system

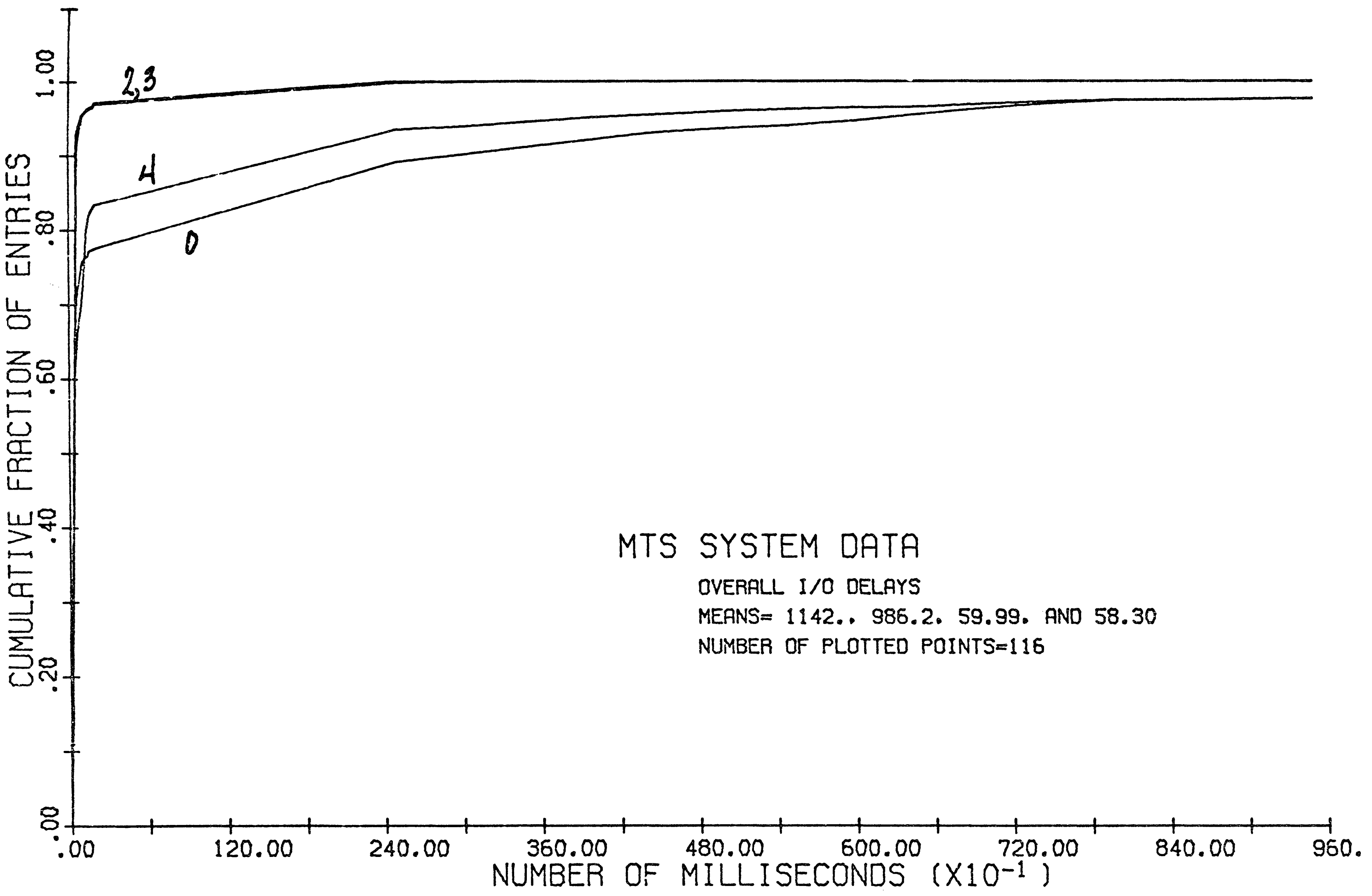


Figure 4-5

Overall I/O Wait Distribution

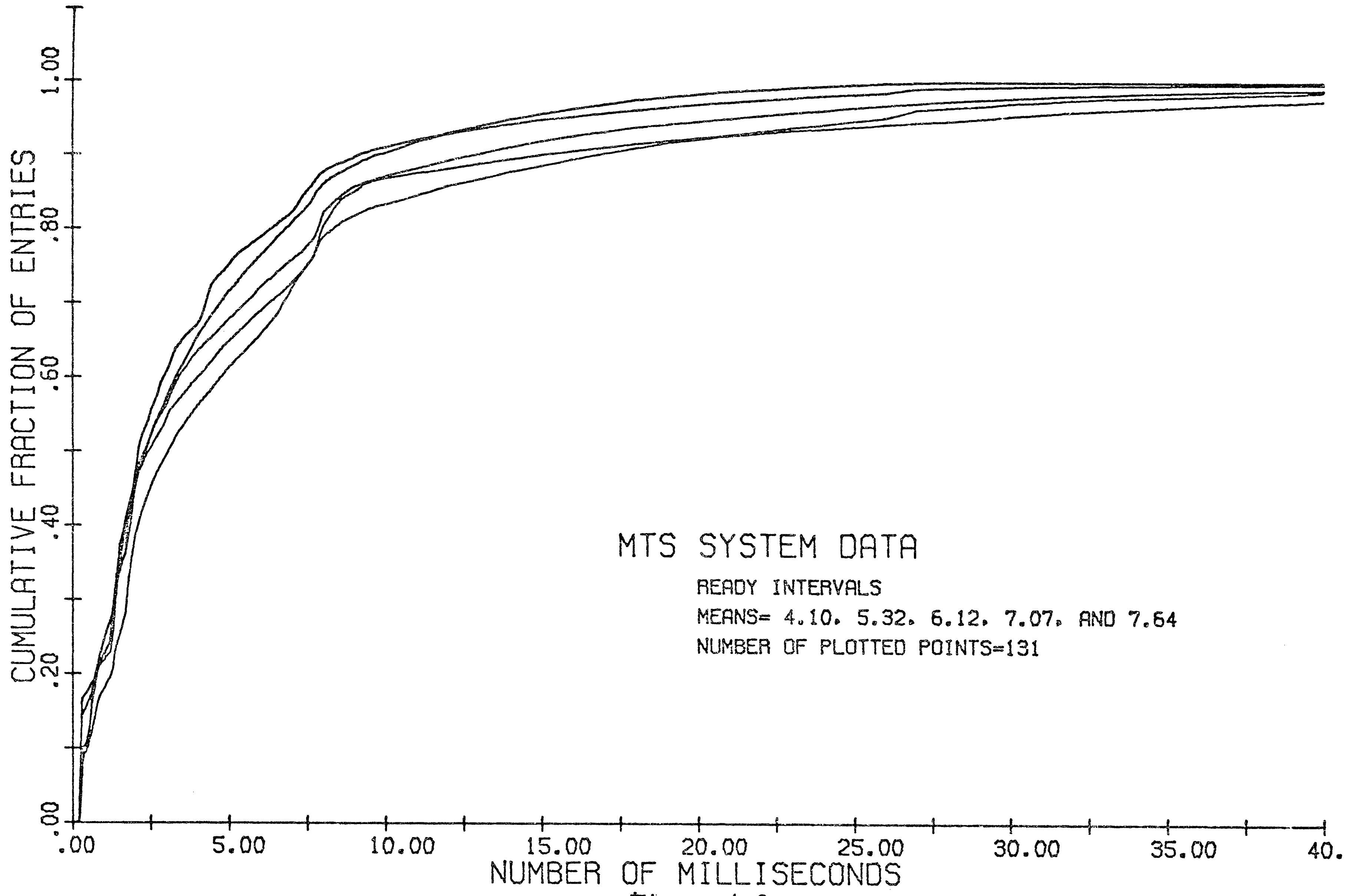


Figure 4-6

MTS Ready Intervals

responds to requests for service. This data is rather consistent. The fact that # 0 has the shortest mean is again due primarily to the fact that the heavy paging load forces many more transitions between tasks entering page-wait.

The remaining figures in this section give disk and terminal characteristics. The disk observed during this period was the IBM 2314 Disk Storage Unit, which provides a bank of eight separate packs on a single control unit and channel. The disk is used in MTS for line file storage and utility files for compilations and assemblies. In Figure 4-7 we see the actual lengths of disk I/O operations. The principal components are

a) Control unit wait (transmission of data or commands from anothere disk pack on the unit)

b) Seek time to move the read/write heads to the proper cylinder (which is often not required)

c) Rotational delay to reach the front of the appropriate record on the disk track

d) Transfer time for actual reading or writing of the record.

The distribution with the longest mean is that for the data taken under the heaviest load. Under these conditions we expect control unit wait to be a significant factor, since at the time up to six of the packs could be in competition for the use of the single control unit. Another very large factor is the frequency with which seeks are necessary: a single task will often require several records from the same cylinder in short succession, hence

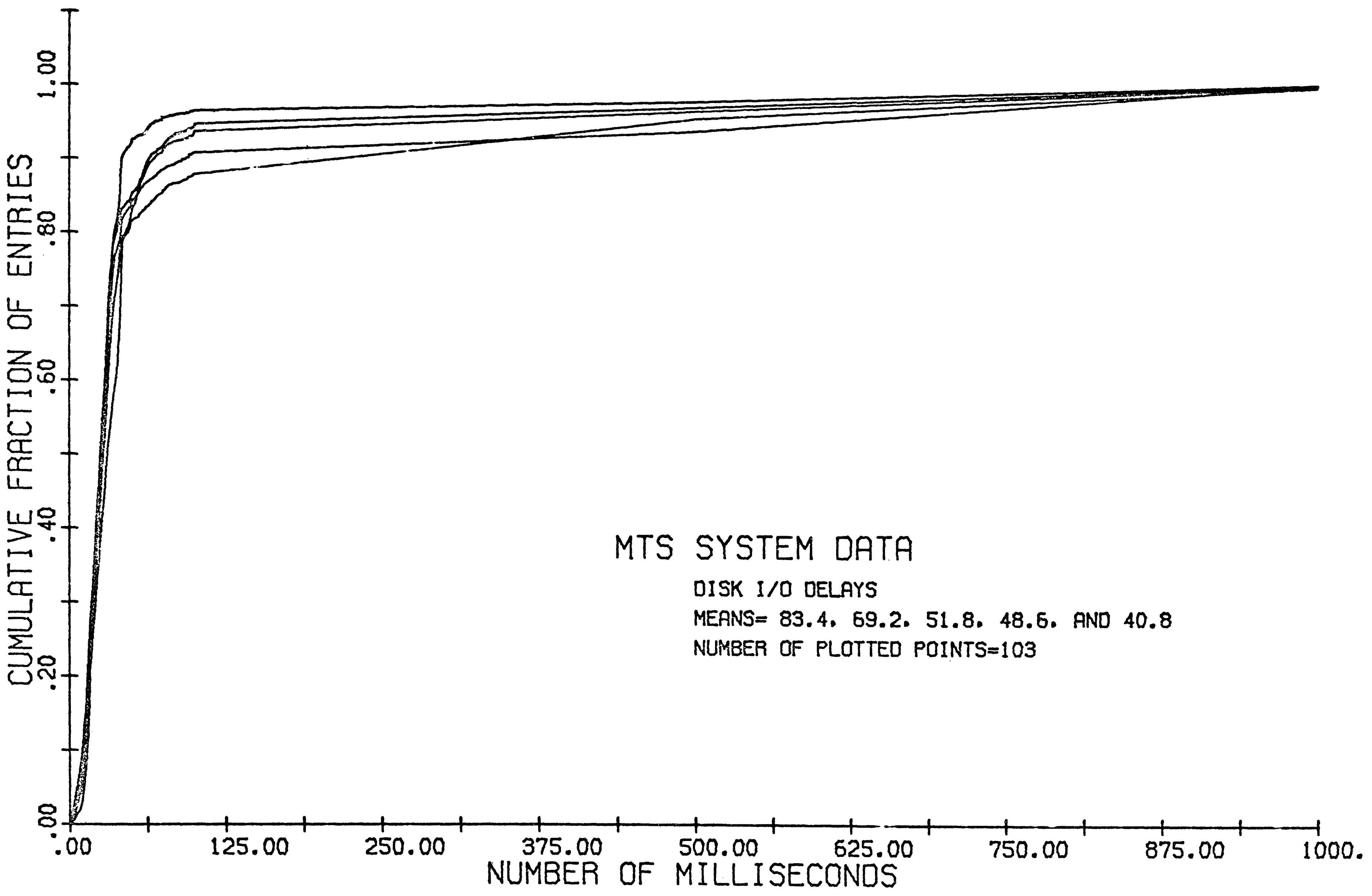


Figure 4-7

Disk I/O Relays

few seeks are necessary unless the load is such that a task can obtain only one record at a time before another task causes a seek to a new cylinder.

The fact that the earliest data has the second longest mean value is due in part to the fact that less efficient file routines were in use at the time, which required more long search operations that tie up the control unit. Another cause is the fact that only four disk packs were available. Similar remarks can be made for Figure 4-8, which shows the distributions of times that tasks had to wait in queue for the use of a specific disk pack. The times in Figure 4-8 are somewhat inflated because the end of a wait for a pack is non-interrupting, and hence it is not discovered to be over until the task reaches the head of the CPU queue.

Finally, we display in Figure 4-9 a few distributions of terminal I/O times. Two distributions each are given for specific Teletype lines from the three sets of conversational data. Although this data exhibits considerably more variation than the synchronous intervals, we can see the effects of the following characteristics of terminal I/O operation: over half the times are for output lines, which clearly predominate. Many of these lines can be placed immediately in the one-line output buffer, so that most output times are less than the time required to actually print a line at the remote device. Shorter times for the input lines of MTS # 0 data suggest that as system response time moves away from practically instantaneous, an individual can make use of the time to formulate his next command, which he then gives more quickly.

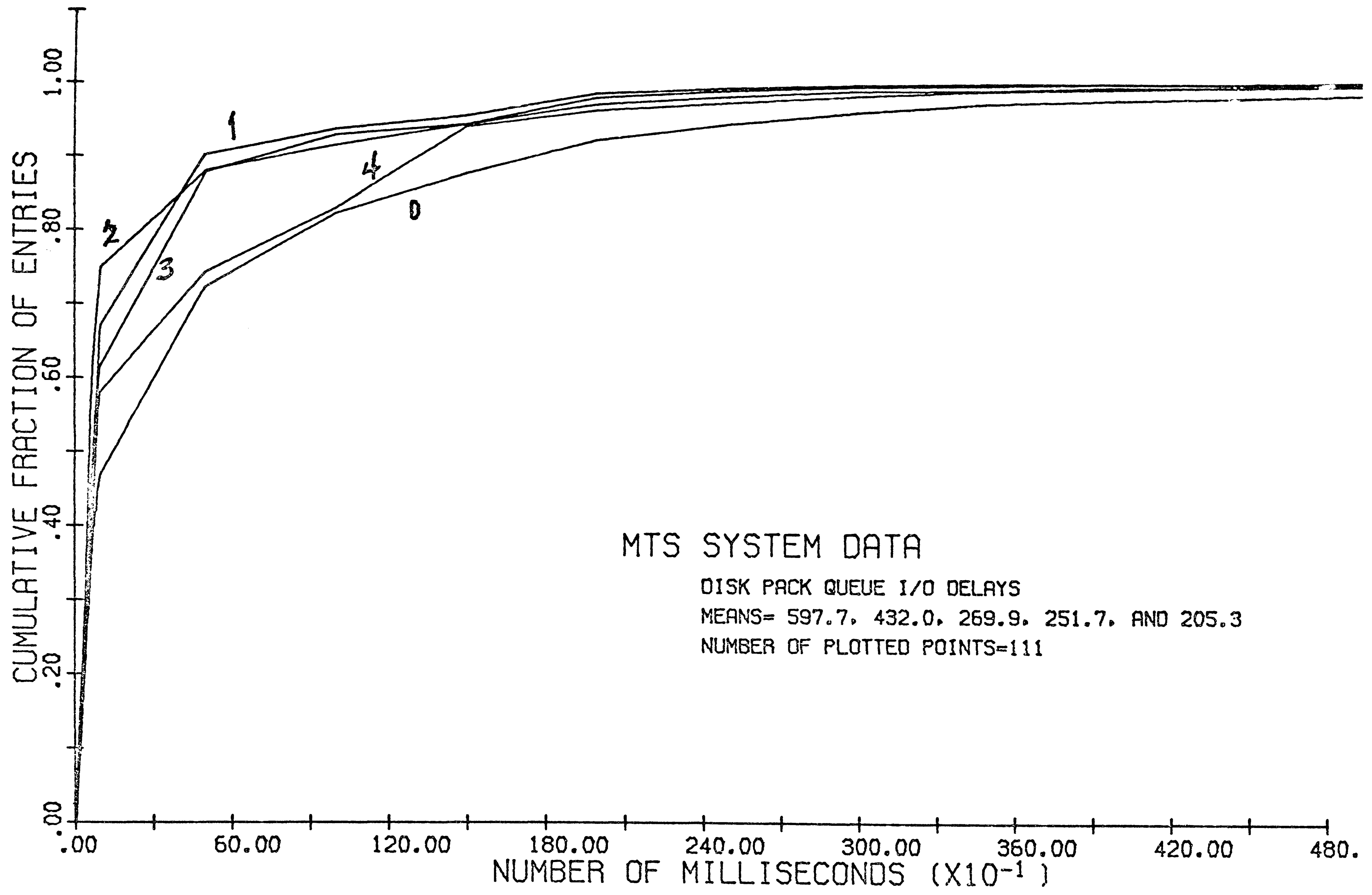


Figure 4-8

Disk Pack Queueing Delays

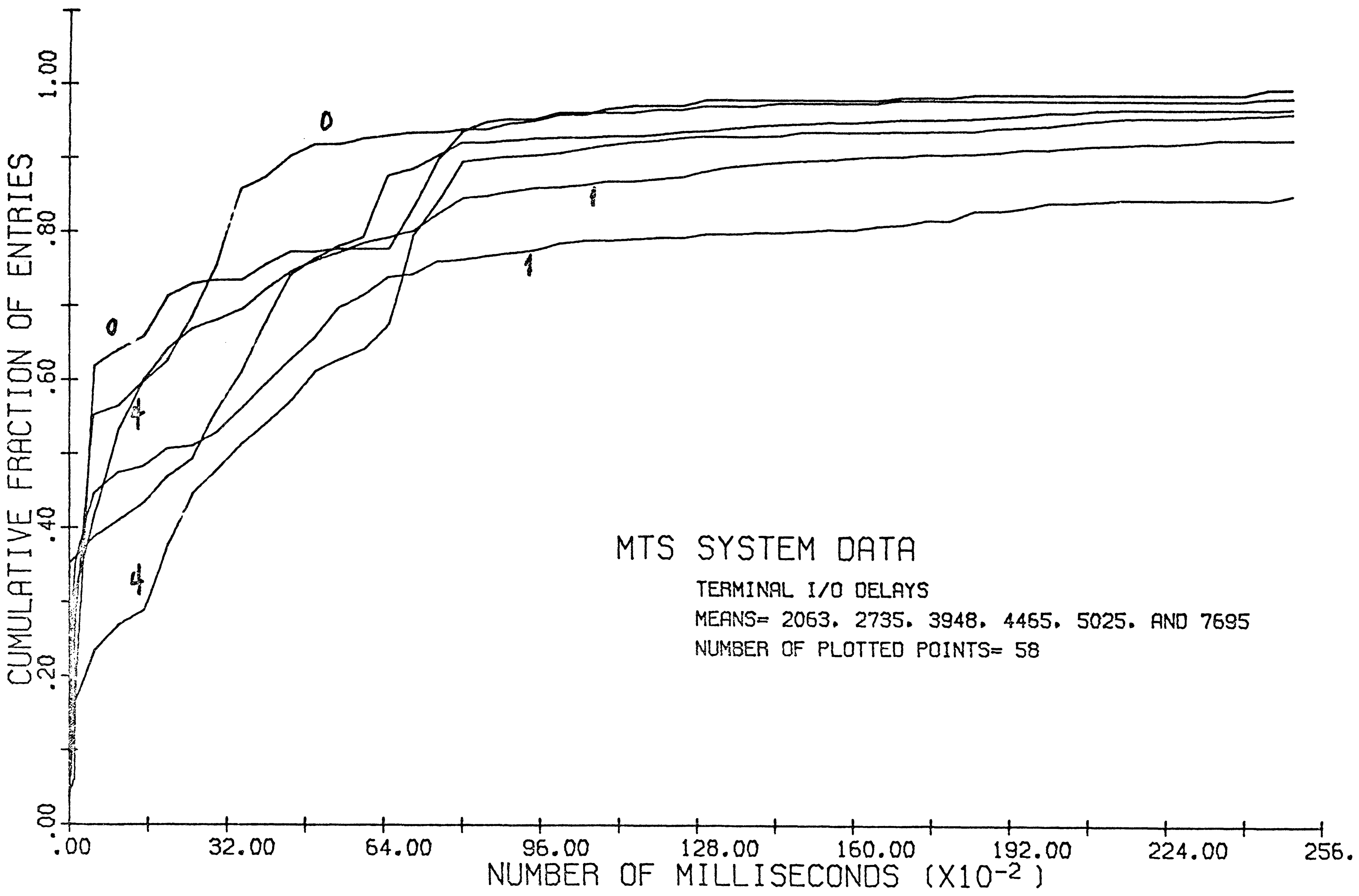


Figure 4-9

Terminal I/O Waits

4.1.3 GPSS/360 Simulation Model

A simulation model of MTS was written to study the effects of hardware changes on system performance. Input to the simulation is the data discussed in the preceding section.

4.1.3.1 GPSS/360

The General Purpose Simulation System/360 is a discrete, digital simulation program developed by the IBM Corporation [1] which runs in the standard IBM Operating System/360. Simulation model statements, after simple processing by an assembly program, are interpreted during execution by the GPSS/360 program.

GPSS/360 is the outgrowth of a series of general purpose system simulators for the IBM 7000-series computing machines, the most recent version of which is titled GPSS-III. GPSS/360 relaxes many GPSS-III restrictions, introduces new entities and block types, allows much more control over storage allocation, and provides a limited graphic output feature.

The GPSS language is particularly suited for modeling in such commercial applications as job shop scheduling and manufacturing network flow. However, it is also useful for computer system simulation at the fairly gross level of detail which is used here. Its primary disadvantages are that it is comparatively difficult to learn and relatively slow in execution. Its generality and flexibility were considered to outweigh those disadvantages for this particular application.

4.1.3.2 Organization of the Model

The simulation model described in this paper consists of about five essentially different parts:

1) hardware descriptions
2) operating system algorithms
3) workload characteristics
4) variable system parameters
5) statistics gathering.

The first of these is the simplest to implement in a simulation language, and requires a negligible fraction of the set of model statements. Characterizing the data in the system (in this case requests for computing services) and specifying the parameters by which the system is "tuned" to improve its performance require about the same amount of effort, and together account for only a fifth or so of the total description. By far the most design and analysis work and the resulting model statements are required to describe the algorithms used by the operating system for handling devices and service requests, and to decide what information about the model operation is relevant and measurable, and how it should be collected and presented. Each of these two aspects of this particular model required over a third of the total simulation effort.

The following sections will discuss each of the basic aspects of the model in terms of its content, overall structure, and the GPSS statements essential to this implementation.

4.1.3.3 Model Data: the Workload Description

An approximate description of each MTS interaction is given by the parameters of a single transaction (XACT). The size of the executing program, the time intervals required for execution and synchronous I/O waits, and the frequency of requirements for additional virtual memory pages during execution are all described by XACT parameters.

The master XACT representing each MTS interaction is passed through the job scheduling part of the model. Transactions are also used elsewhere to stand for a single memory page as it passes through the drum I/O channel, an entry on the queue for use of the CPU, and to implement the data transfer during the drum rotations.

Virtual memory page XACTS required by a task are members of a GPSS assembly set, and a specified number of them must be ASSEMBLED before a task is ready for execution. These XACTS are SPLIT from others as needed, and TERMINATED when they are no longer required to trigger additional events by their flow through the model.

4.1.3.4 Hardware Descriptions

The only parts of this model which directly represent physical devices occur in the choice of the time unit as the drum sector rotational delay, and in the model parameter which specifies the capacity of core storage. Hardware/software combinations such as the operation of the drum as an extension of main memory and the supervisor algorithm which drives it are also modeled.

As each page transfer request arrives at the drum processor, it is LINKed to one of the GPSS "user chains," which are used to represent drum sector queues. Once every time unit a special "clock" XACT passes through an UNLINK block which attempts to remove a queue entry from the sector currently at the read/write heads. Any XACT so UNLINKed is next ADVANCED for one time unit to represent the transfer time, and then either sent to be ASSEMBLEd for CPU service (if a page-in request) or simply TERMINATEd to remove it from the model (if a page-out request).

4.1.3.5 Operating System Algorithms

Additional algorithms represented in the model for the operating system itself include

a) the choice of when to interrupt the task currently using the CPU

b) the choice of when to allow a new task to bring its pages to real memory and compete for the use of the CPU

c) the choice of when to page-out a task not ready to use the CPU.

Decisions are made with these algorithms at specific "control points" in the model by routing task XACTS based on the value of task characteristics (XACT parameters), system load factors (QUEUE lengths and STORAGE usage), and decision-aiding parameters (SAVEVALUEs). The control points occur when the executing task requests a virtual memory page which is not in real memory, when a task has used a certain amount of CPU time, when real memory usage drops enough to allow room for the pages of another task, when a task begins an I/O operation, and when an executing task is interrupted.

4.1.3.6 System Parameters

Decision-making algorithms for operating systems are normally designed to use a number of parameters—numerical values which can be changed as frequently as desired to improve performance as more is learned about how the system behaves, as the hardware configuration changes, and as the workload develops different patterns of demand for system resources. Examples of such parameters existing in this model are

a) the maximum amount of real memory available to a single task at one time

b) the maximum length of time a task may continuously use the CPU

c) the maximum number of real memory page requests allowed to enqueue in the system when core is full

d) the number of real memory pages required to be free before a new task is allowed to compete for the CPU.

These values are stored in GPSS SAVEVALUE locations. They are initialized for each run, and can be changed during the simulated operation if necessary.

4.1.3.7 Simulation Results

This section gives the data obtained from GPSS/360 simulations made with the model described in the previous sections. A number of simulation runs were made using a pair of JOBTAPEs of task transactions from the MTS data. These tapes were obtained by abstracting MTS data sets # 0 and # 1 during the analysis of that data for the presentation in Section 4.1.2. Each simulation run begins at the front of one of these two tapes and uses as many transactions as necessary in order to simulate system operation for a specified amount of elapsed time.

For initial runs, both JOBTAPEs were run with a pair of "extremal" choices of the model parameters:

a) a Basic configuration of hardware and programming techniques, using:

   80 core pages

   a 9-sector drum

   16 time unit time-slice

   read before write drum queue discipline

   drum writes to shortest queue

   FIFO CPU queue discipline

   posting of pages once per revolution

b) an Advanced configuration, which differs from the Basic one in the following choices of parameter values:

   144 core pages

   large core storage instead of drum

   24 time unit time-slice

   priority in drum queues to large tasks

immediate posting of pages.

The remaining runs were made using parameter choices differing from Basic in only one or two parameter values, and generally with values chosen from the Advanced model substituted for the Basic values.

Each run begins with an initialization period, after which two or three sets of data are taken for intervals of 30 simulated seconds. Because the initialization intervals were taken to be rather long, their data (which is not included here) agrees quite closely with the values observed for the regular intervals. The Basic and Advanced models were each run for three intervals, and the remaining models for two. In several cases, however, runs were terminated after a somewhat shorter last interval because of a problem with controlling the input rate of task data from the JOBTAPEs. Thus the data displayed in the figures of this section is normalized by the length of the run segment.

Figures 4-10 and 4-11 show the results of running the Basic model with the MTS # 1 and MTS # 0 JOBTAPEs, respectively. The same data for the Advanced model is given in Figures 4-12 and 4-13. Figure 4-14 lists the values obtained by using the Basic model except for the larger core size taken for the Advanced model. Similarly, Figure 4-15 gives the data for the Basic configuration altered only by immediate page posting. Subsequent figures show other variations. Only one run (described in Figure 4-18) was made with values deliberately chosen outside the techniques used in the Basic and Advanced models. In this case several poorer techniques

MTS Data # 1

PERFORMANCE...	I	II	III
• CPU Utilization	.456	.436	.591
Average queue contents	.601	.508	.767
Maximum queue contents	9	7	6
Percent zero entries	53.6	55.3	40.7
• Paging mechanism utilization	.499	.685	.672
Average queue contents	12.5	22.1	19.4
Maximum queue contents	69	69	65
Mean completion time	25.1	32.0	28.9
Tasks Completed/Second	10.5	12.8	16.4
Mean No. of Active Tasks	8.8	7.1	9.2
WORKLCAD	**I**	**II**	**III**
Task No. Range	250- 563	564- 949	950- 1319
No. of CPU Intervals/Second	52.4	43.2	63.8
Mean CPU Service Time	2.24	2.60	2.38
Percent Initial Pages	86.4	89.7	82.3
No. of Pages Written/Second	64.2	88.3	86.2
Mean No. of I/O Operations	5.49	3.38	5.20
Average I/O Time	42.3	41.3	41.6

Figure 4-10

Basic Simulation Data

BASIC SIMULATION MODEL

MTS Data # 0

PERFORMANCE...	I	II	III
• CPU Utilization	.253	.311	.189
Average queue contents	.304	.261	.136
Maximum queue contents	8	6	5
Percent zero entries	37.6	51.0	55.3
• Paging mechanism utilization	.439	.401	.263
Average queue contents	11.0	5.56	3.48
Maximum queue contents	71	47	48
Mean completion time	24.6	13.8	13.2
Tasks Completed/Second	11.0	7.8	4.9
Mean No. of Active Tasks	7.5	8.7	6.0

WORKLOAD	I	II	III
Task No. Range	252-581	582-815	816-901
No. of CPU Intervals/Second	47.7	62.8	36.5
Mean CPU Service Time	1.37	1.28	1.34
Percent Initial Pages	51.3	34.2	34.9
No. of Pages Written/Second	56.5	51.7	33.4
Mean No. of I/O Operations	3.73	5.55	3.24
Average I/O Time	102.4	68.4	82.5

Figure 4-11

Basic Simulation Data

MTS Data # 1

PERFORMANCE...	I	II	III
• CPU Utilization	.717	.621	.856
Average queue contents	2.19	2.14	5.08
Maximum queue contents	16	20	16
Percent zero entries	28.9	34.9	15.2
• Paging mechanism utilization	.804	.779	.721
Average queue contents	32.9	36.8	21.1
Maximum queue contents	≥100	≥100	95
Mean completion time	40.2	47.2	29.3
Tasks Completed/Second	13.7	13.8	16.2
Mean No. of Active Tasks	14.5	12.9	18.3
WORKLOAD	**I**	**II**	**III**
Task No. Range	251-660	661-1076	1077-1483
No. of CPU Intervals/Second	81.5	64.8	100.6
Mean CPU Service Time	2.26	2.47	2.19
Percent Initial Pages	78.2	76.1	74.6
No. of Pages Written/Second	104.5	99.2	92.3
Mean No. of I/O Operations	6.69	3.98	7.60
Average I/O Time	38.1	37.5	32.0

Figure 4-12

Advanced Simulation Data

MTS Data # 0

PERFORMANCE...	I	II	III
• CPU Utilization	.642	.630	.555
Average queue contents	1.55	1.79	1.74
Maximum queue contents	16	17	17
Percent zero entries	36.3	37.7	40.2
• Paging mechanism utilization	.900	.881	.957
Average queue contents	25.4	29.4	45.2
Maximum queue contents	96	≥100	≥100
Mean completion time	28.0	33.2	47.2
Tasks Completed/Second	21.1	16.9	20.5
Mean No. of Active Tasks	18.5	15.7	17.9

WORKLOAD	I	II	III
Task No. Range	251-883	884-1391	1392-2007
No. of CPU Intervals/Second	114.7	100.5	88.7
Mean CPU Service Time	1.44	1.61	1.61
Percent Initial Pages	46.6	48.4	60.3
No. of Pages Written/Second	116.2	112.5	123.1
Mean No. of I/O Operations	7.32	3.69	2.48
Average I/O Time	59.2	37.0	32.9

Figure 4-13

Advanced Simulation Data

MODIFIED BASIC SIMULATION MODEL

(Using 144 core pages)
MTS Data # 1

PERFORMANCE...	I	II
• CPU Utilization	.601	.707
Average queue contents	1.78	4.08
Maximum queue contents	15	17
Percent zero entries	38.0	21.8
• Paging mechanism utilization	.854	.856
Average queue contents	49.3	56.2
Maximum queue contents	≥100	≥100
Mean completion time	56.8	65.5
Tasks Completed/Second	17.6	20.0
Mean No. of Active Tasks	12.8	13.3
WORKLOAD	**I**	**II**
Task No. Range	250-776	777-1378
No. of CPU Intervals/Second	68.1	72.2
Mean CPU Service Time	2.27	2.52
Percent Initial Pages	91.0	89.2
No. of Pages Written/Second	110.3	110.2
Mean No. of I/O Operations	6.78	4.79
Average I/O Time	42.9	30.5

Figure 4-14

Modified Simulation Data

MODIFIED BASIC SIMULATION MODEL

(Using immediate page posting)
MTS Data # 0

PERFORMANCE...	I	II
• CPU Utilization	.484	.310
Average queue contents	.250	.099
Maximum queue contents	6	5
Percent zero entries	73.9	82.8
• Paging mechanism utilization	.705	.387
Average queue contents	16.6	5.96
Maximum queue contents	69	55
Mean completion time	23.5	15.4
Tasks Completed/Second	14.0	6.1
Mean No. of Active Tasks	7.7	7.6
WORKLOAD	**I**	**II**
Task No. Range	201-620	621-805
No. of CPU Intervals/Second	98.3	60.1
Mean CPU Service Time	1.44	1.33
Percent Initial Pages	44.3	28.9
No. of Pages Written/Second	90.7	50.0
Mean No. of I/O Operations	4.41	5.05
Average I/O Time	51.9	69.3

Figure 4-15

Modified Simulation Data

MODIFIED BASIC SIMULATION MODEL

(Using LCS for paging mechanism)
MTS Data # 0

PERFORMANCE...	I	II
• CPU Utilization	.362	.202
Average queue contents	.562	.332
Maximum queue contents	10	10
Percent zero entries	36.2	39.8
• Paging mechanism utilization	.598	.369
Average queue contents	10.8	7.15
Maximum queue contents	59	62
Mean completion time	18.0	19.3
Tasks Completed/Second	16.9	9.8
Mean No. of Active Tasks	12.6	7.3
WORKLCAD	**I**	**II**
Task No. Range	251-758	759-1052
No. of CPU Intervals/Second	72.7	37.6
Mean CPU Service Time	1.28	1.39
Percent Initial Pages	53.5	61.6
No. of Pages Written/Second	77.0	47.5
Mean No. of I/O Operations	7.42	3.24
Average I/O Time	94.5	85.2

Figure 4-16

Modified Simulation Data

MODIFIED BASIC SIMULATION MODEL

(Using preemptive CPU queue discipline)
MTS Data # 1

PERFORMANCE...	I	II
• CPU Utilization	.425	.407
Average queue contents	.339	.347
Maximum queue contents	5	8
Percent zero entries	53.0	53.3
• Paging mechanism utilization	.513	.709
Average queue contents	13.5	22.9
Maximum queue contents	68	74
Mean completion time	26.4	32.1
Tasks Completed/Second	10.8	13.0
Mean No. of Active Tasks	8.7	7.0
WORKLOAD	**I**	**II**
Task No. Range	250-573	574-963
No. of CPU Intervals/Second	54.4	42.8
Mean CPU Service Time	2.01	2.45
Percent Initial Pages	85.4	89.3
No. of Pages Written/Second	66.2	91.4
Mean No. of I/O Operations	5.28	3.30
Average I/O Time	39.8	41.9

Figure 4-17

Modified Simulation Data

MODIFIED BASIC SIMULATION MODEL

(Using random drum writes and FIFO queues)
MTS Data # 1

PERFORMANCE...	I	II
• CPU Utilization	.410	.291
Average queue contents	.455	.154
Maximum queue contents	9	5
Percent zero entries	59.7	65.9
• Paging mechanism utilization	.493	.515
Average queue contents	13.2	15.2
Maximum queue contents	62	61
Mean completion time	26.7	29.4
Tasks Completed/Second	9.7	8.9
Mean No. of Active Tasks	9.1	7.3
WORKLOAD	**I**	**II**
Task No. Range	250-541	542-808
No. of CPU Intervals/Second	52.3	37.8
Mean CPU Service Time	2.02	1.98
Percent Initial Pages	82.9	80.7
No. of Pages Written/Second	63.3	66.5
Mean No. of I/O Operations	4.88	3.24
Average I/O Time	39.3	51.4

Figure 4-18

Modified Simulation Data

MODIFIED BASIC SIMULATION MODEL

(Using queueing options and longer time slice)
MTS Data # 0

PERFORMANCE...	I	II
• CPU Utilization	.324	.238
Average queue contents	.533	.180
Maximum queue contents	10	5
Percent zero entries	33.9	42.2
• Paging mechanism utilization	.478	.296
Average queue contents	9.56	3.72
Maximum queue contents	67	39
Mean completion time	20.0	12.6
Tasks Completed/Second	11.2	5.9
Mean No. of Active Tasks	7.7	7.8

WORKLOAD	I	II
Task No. Range	251-587	588-765
No. of CPU Intervals/Second	55.1	47.9
Mean CPU Service Time	1.51	1.28
Percent Initial Pages	47.6	31.3
No. of Pages Written/Second	61.7	38.1
Mean No. of I/O Operations	3.68	4.98
Average I/O Time	80.1	79.8

Figure 4-19
Modified Simulation Data

MODIFIED BASIC SIMULATION MODEL

(Using 4-sector drum and 20% unchanged pages)
MTS Data # 0

PERFORMANCE...	I	II
• CPU Utilization	.426	.350
Average queue contents	.475	.323
Maximum queue contents	6	7
Percent zero entries	30.6	36.9
• Paging mechanism utilization	.574	.458
Average queue contents	11.2	6.08
Maximum queue contents	70	57
Mean completion time	19.5	13.3
Tasks Completed/Second	13.9	9.7
Mean No. of Active Tasks	9.2	8.2
WORKLOAD	**I**	**II**
Task No. Range	253-668	669-959
No. of CPU Intervals/Second	81.4	68.2
Mean CPU Service Time	1.35	1.32
Percent Initial Pages	63.4	62.2
No. of Pages Written/Second	64.6	52.7
Mean No. of I/O Operations	5.32	5.05
Average I/O Time	45.4	67.4

Figure 4-20

Modified Simulation Data

were used to gauge the sensitivity of paging drum processing to such changes.

Figure 4-20, which was run with a 4-sector drum in the (otherwise) Basic configuration, also provides for a fraction of unchanged pages: 20% of the write requests were not executed in this case, assuming that the pages were unchanged since their last trip to the drum.

4.2 MTS DISPLAY CONSOLE DATA

In this section data taken on different display applications will be presented. The purpose of this data is to compare the computational requirements of various display (and non-display) applications. The data has been collected for jobs running on the University of Michigan's 360/67, using the Michigan Terminal System.

Imbedded within MTS's executive system is an efficient data collection system. Basically, a data item is recorded (on tape) whenever an event pertaining to specified jobs occurs. Examples of the events are 1) the start of a CPU processing interval, 2) the end of a CPU processing interval, 3) page read in or page read out start and end, 4) acquiring or releasing virtual memory pages, and 5) I/O to terminals, printers, or tapes.

A program has been written to analyze the data for any job and produce a series of probability distributions and summary data for the following quantities.

1) User think time. This begins when the computer system is ready to accept new input from the user, and ends when the input is completed with an end-of-line indication.

2) CPU time used during the think period.

3) Computer system response time. This begins at the completion of an input line, and ends when all output has been finished and the computer system is again ready to accept input.

4) Processing interval lengths during response periods. During a processing interval a job has exclusive use of the CPU, except for supervisor functions.

5) Number of processing intervals during a response period.

6) Number of characters in input lines.

7) Number of characters in output lines.

When the analysis program was originally conceived and implemented, some of its results were intended to be used as part of the application specification required by a display system model (section 3.2). A serious deficiency in the data collection facility became evident and frustrated this aim. The problem is that input-output information is gathered only at the MTS level. The display service routines for the IBM 2250 display console do not use the MTS I/O routines once a graphics application program has been loaded and started from the console until it has been terminated. Therefore to the data collection facility, running a graphics program appears as one long response time, despite the many user interactions generated with the light pen and function buttons. Because this was the case, gathering much in the way of useful statistics for display applications became impossible. All of use that can be garnished from the display applications statistics is CPU utilization during response periods, and also averaged over think and response periods. While this information will be shown to be useful, it is not what was anticipated. This information is in Table 4-1.

The first application referred to in the table, Michigan's Own Math ematical System (MOMS) is used to manipulate and plot mathematical functions. All interaction is via the light pen. The second application, text editing, uses the light pen and keyboard to modify text displayed on the console.

The 2250 display console could also be used in MTS as a teletype, with a screen instead of printer and paper. Table 4-2 shows pertinent data from this mode of operation. The first three applications consist of general program preparation, correction, and debugging. The last application, running the system program *TASKS, gives a listing of jobs active in MTS.

The data collection facility was also used to monitor all MTS users for about one hour. The statistics gathered from a random sampling of the monitored teletype terminals are given in Table 4-3.

Finally, a small amount of data was collected from a remote display terminal using MTS. The system consists of a DEC 339 with

Table 4-1

Data Gathered for IBM 2250 Display Console Used for Graphics

Application	Average Use of CPU by Program	Use of CPU by Program During Response Time	Elapsed Time, Seconds	CPU Time, Seconds	Average Processing Interval, Microseconds
Michigan's Own Mathematical System	4. 10%	4. 25%	1342	54. 8	7521
Michigan's Own Mathematical System	4. 90%	5. 30%	412	20. 0	6030
Michigan's Own Mathematical System	4. 95%	5. 50%	1184	58. 6	5795
Text Editing	2. 30%	2. 40%	5080	116. 0	5116
Text Editing	2. 80%	3. 30%	2380	66. 0	5038

Table 4-2

Data Gathered for IBM 2250 Display Console Used as a Teletype

Application	Average Use of CPU by Program	Use of CPU by Program During Response Time	Elapsed Time, Seconds	CPU Time, Seconds	Average Think Time, Seconds	Average Response Time, Seconds	Average Processing Interval Length, Microseconds
Program Preparation and Testing	1. 10%	4. 0%	1280	14. 0	42. 0	13. 70	6518
Program Preparation and Testing	0. 75%	9. 8%	2360	17. 7	20. 9	0. 98	4780
Program Preparation and Testing	4. 55%	12. 5%	2650	121. 0	22. 2	10. 00	7634
Executing *TASKS	0. 37%	11. 4%	247	0. 9	47. 0	1. 05	5092

Table 4-3

Data Gathered for Random Teletype Users

Average Use of CPU by Program	Average Use of CPU by Program During Response Time	Elapsed Time, Seconds	CPU Time, Seconds	Average Think Time, Seconds	Average Response Time, Seconds	Average Processing Interval Length, Microseconds
3.50%	4.8%	1525	53.0	0.85	1.84	6103
1.80%	3.1%	578	10.2	25.00	31.90	3316
1.20%	2.4%	297	3.7	16.10	15.80	5275
0.15%	4.4%	4760	7.0	194.00	4.60	5837
2.40%	4.5%	512	12.3	14.60	15.50	3998
4.50%	9.8%	779	34.7	42.60	35.20	5782
0.70%	1.2%	207	1.5	12.10	39.80	3947
1.10%	1.5%	83	0.9	14.00	28.60	4610
2.10%	2.9%	356	7.6	25.30	63.60	3462
1.10%	1.8%	1580	17.5	24.10	36.70	4454
0.20%	1.0%	3610	7.1	35.90	5.70	3438
0.90%	3.3%	2930	25.2	44.60	14.10	4807

16,384 words of core storage, connected to the main computer via a 2000 bits per second data link. The application is queueing network analysis, in which the user draws on the display a queueing network, and then can see various probability distributions pertaining to the network. All graphics work is done by the display terminal: the main computer merely solves the network for the required results. The data is in Table 4-4. Only a very simple queueing network was analyzed. It can be expected that for more complicated and realistic models, the average CPU utilization of 7.3% would increase. Note that in this case "Think Time" does not refer to the user, but to the remote display terminal. Thus an average of 2.5 seconds after receiving a reply from the main computer, the remote computer sends a new request for service to the main computer.

Table 4-5 compares and summarizes some of the more important statistics from the four modes of using MTS reported on in Tables 4-1 through 4-4. Several points should be noted. First, using a display in lieu of a teletype reduces response time by a factor of 4, with think time remaining nearly constant. This is a consequence of the fast rate of output attainable with the display console. The input rate (and consequently think time) is unaffected because a keyboard is used in both cases. Second, the faster output rate results in higher CPU use by the terminal, because more computation is done in less time. This means that a display terminal user gets more done in unit time than does a

Table 4-4

Data Gathered for Remote Display Terminal

Average Use of CPU by Displav Terminal	7. 30%
Average Use of CPU by Display Terminal During Response Time	13. 30%
Elapsed Time, Seconds	800. 00
CPU Time, Seconas	58. 50
Average Display Terminal "Think Time," Seconds	2. 50
Average Response Time, Seconds	2. 96
Average Processing Inter-val Length, Microseconds	6007. 00

Table 4-5

Comparison of Statistics

Application Class	Average Use of CPU	Average User Think Time	Average Response Time
Teletype	1.05%	22.8	24.40
2250 Display Used as Teletype	2.36%	23.0	6.40
2250 Display Used for Graphics	3.03%	*	*
Remote Display Used for Graphics	7.30%	*	2.96

* Not Applicable

```
MEAN                     SIGMA                   SAMPLES

   2557643.0               6649902.0               145     THINK TIME
     10393.4                  3563.5               145     TOTAL CPU TIME USED DURING THINK PERIODS
   2962184.0              10045788.0               144     RESPONSE TIME
    393670.6               1468383.0               144     CPU TIMES DURING RESPONSE PERIODS
      6007.1                  9990.4              9437     CPU INTERVALS DURING RESPONSE PERIODS
        65.5                   258.0               144     NUMBER OF CPU INTERVALS DURING RESPONSE PERIODS
         5.8                     1.8               145     INPUT MESSAGE LENGTHS
        19.3                    29.7               144     OUTPUT MESSAGE LENGTHS
```

TIMES ARE IN MICROSECONDS

MESSAGE LENGTHS ARE IN CHARACTERS

Figure 4-21

Summary Data for Remote Display Terminal

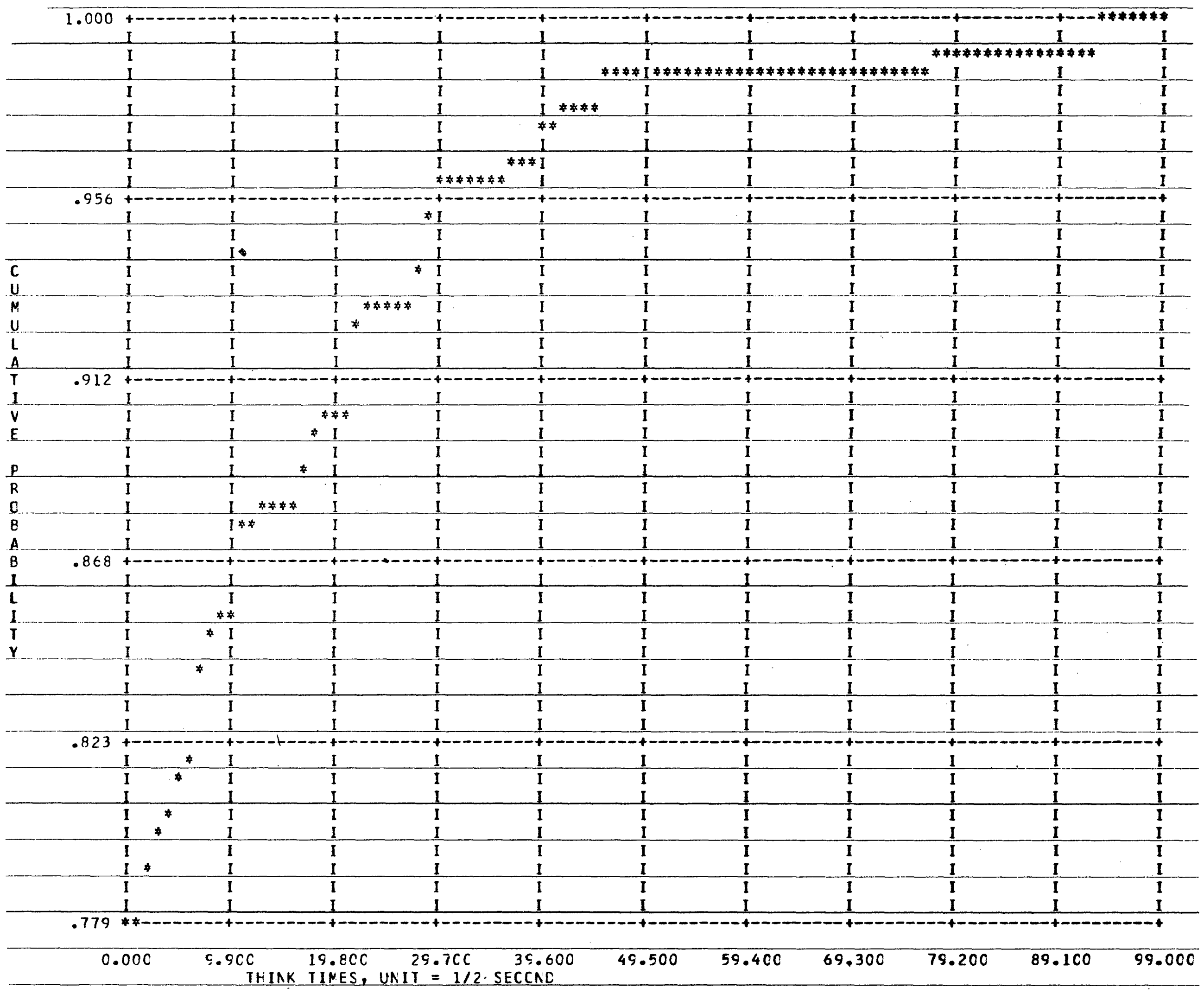


Figure 4-22

Think Time Distribution

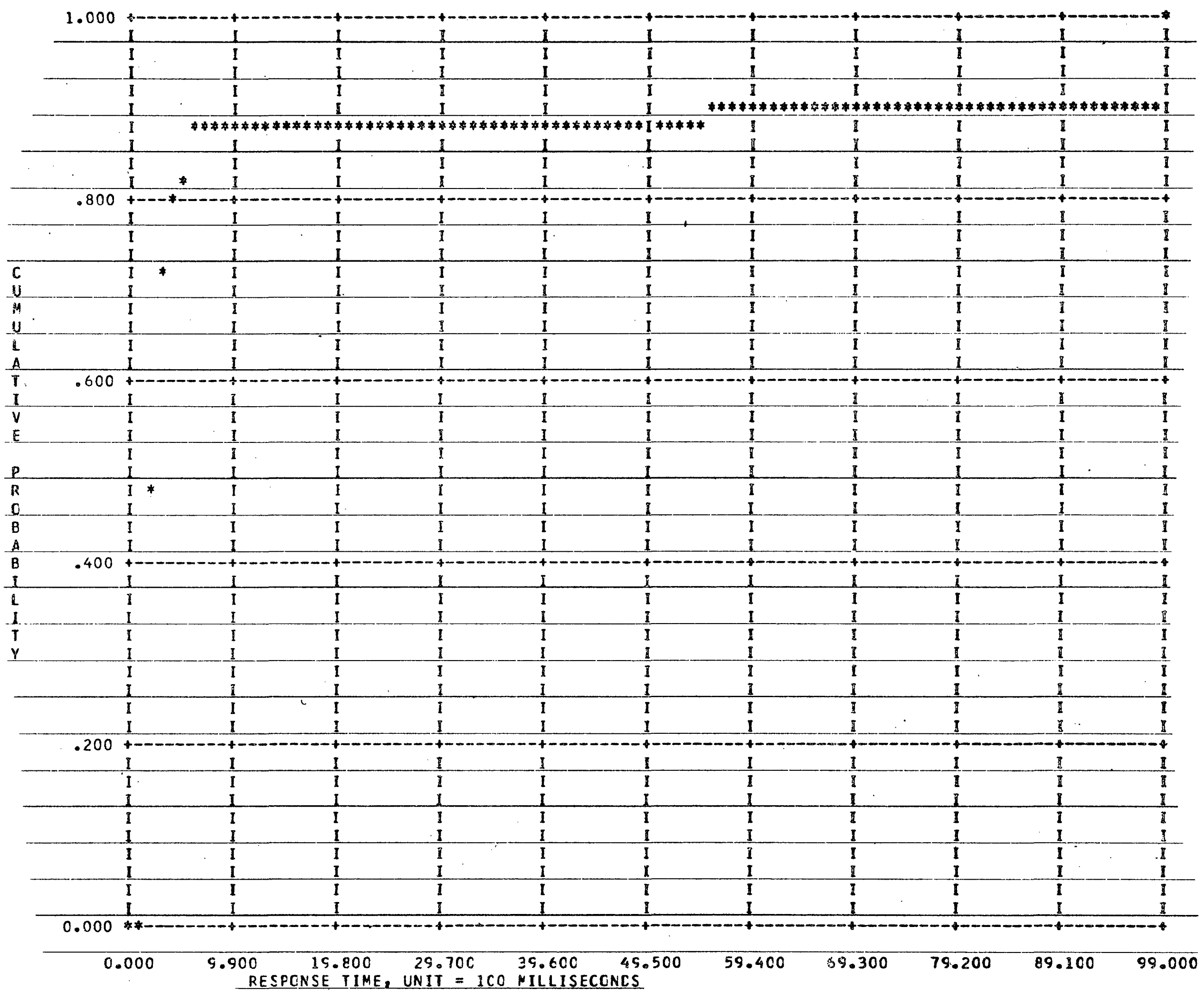


Figure 4-23

Response Time Distribution

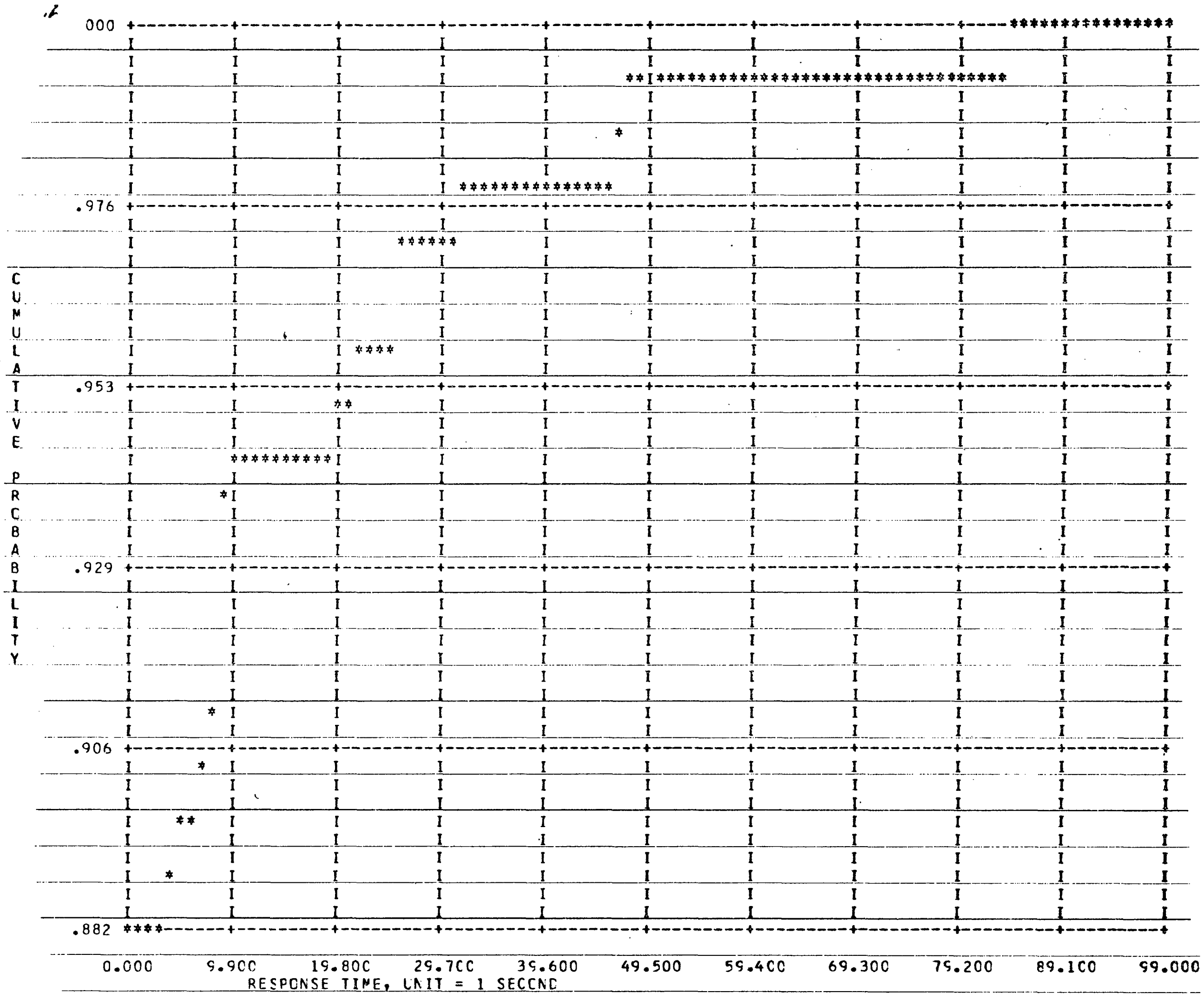


Figure 4-24

Response Time Distribution

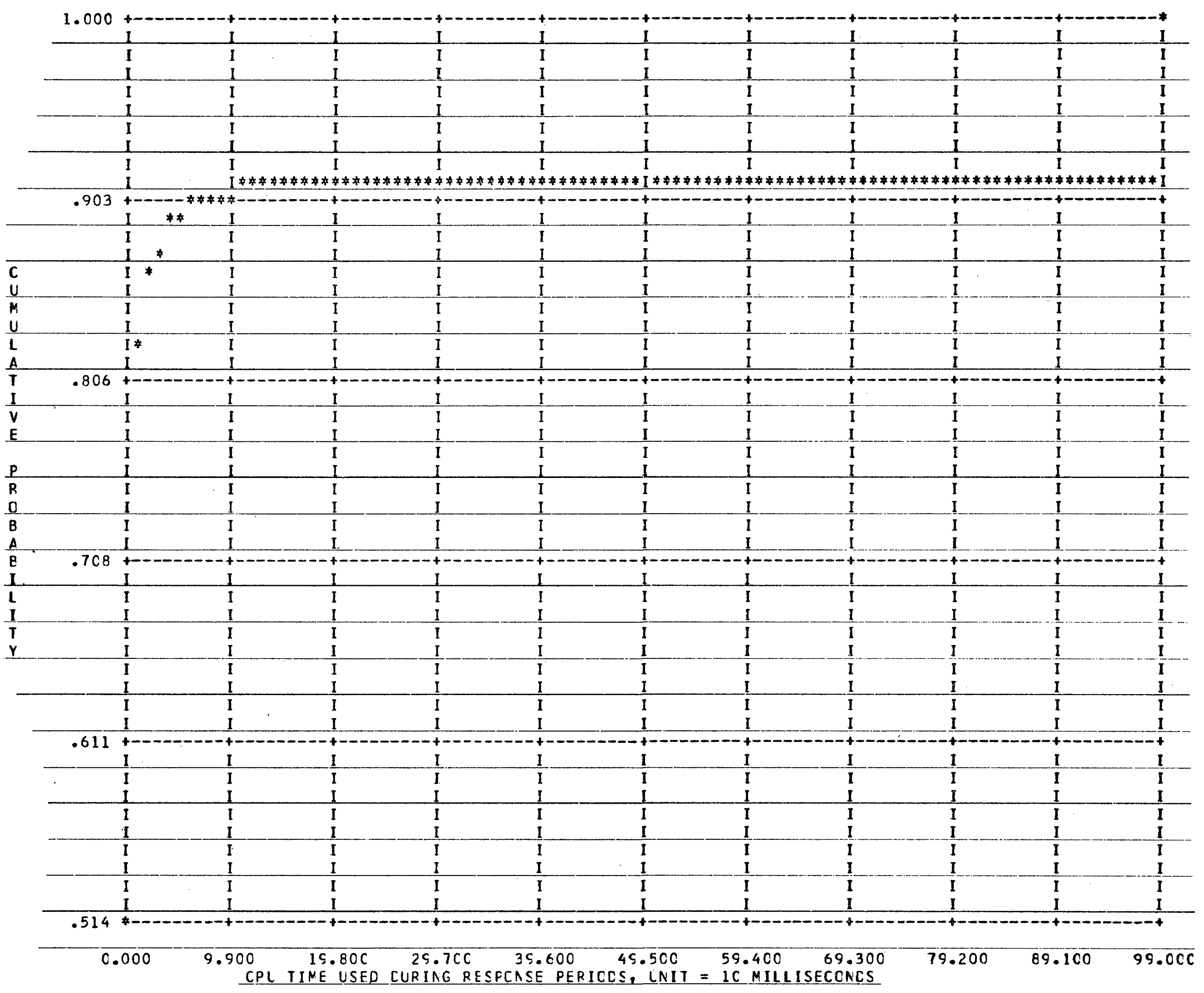


Figure 4-25

Distribution of Total CPU Time per Response Period

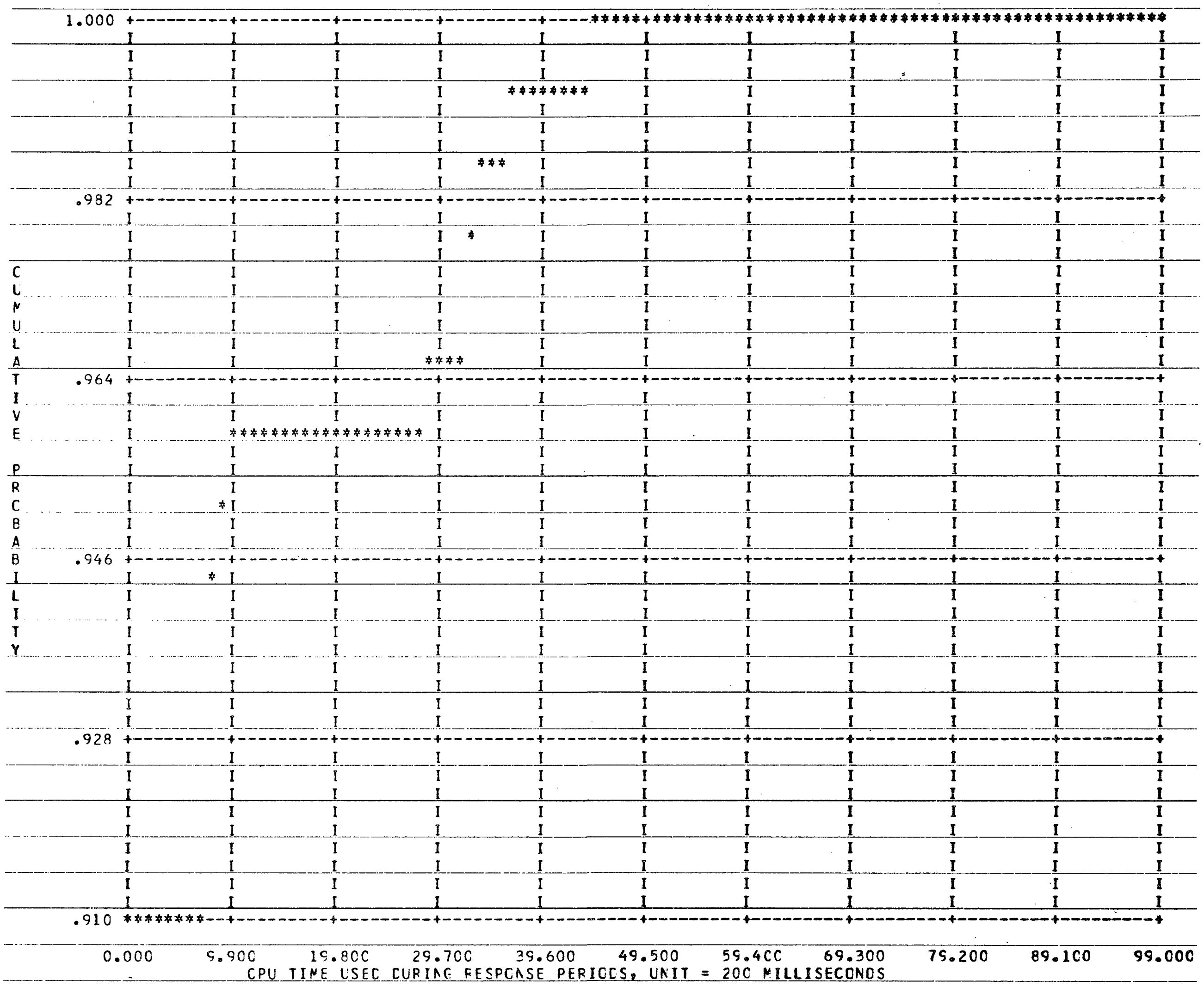


Figure 4-26

Distribution of Total CPU Time per Response Period

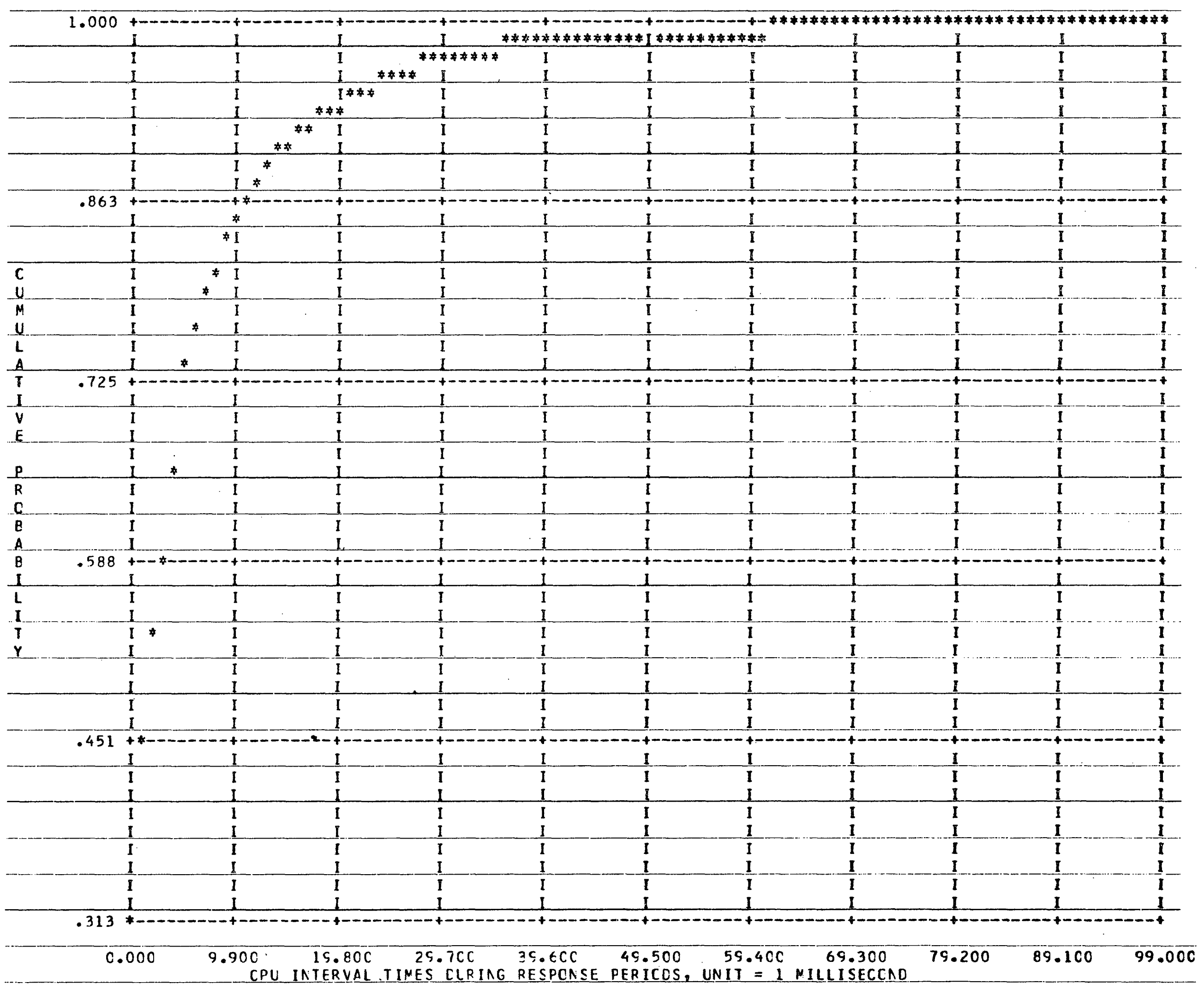


Figure 4-27

Distribution of CPU Processing Interval Times

During Response Periods

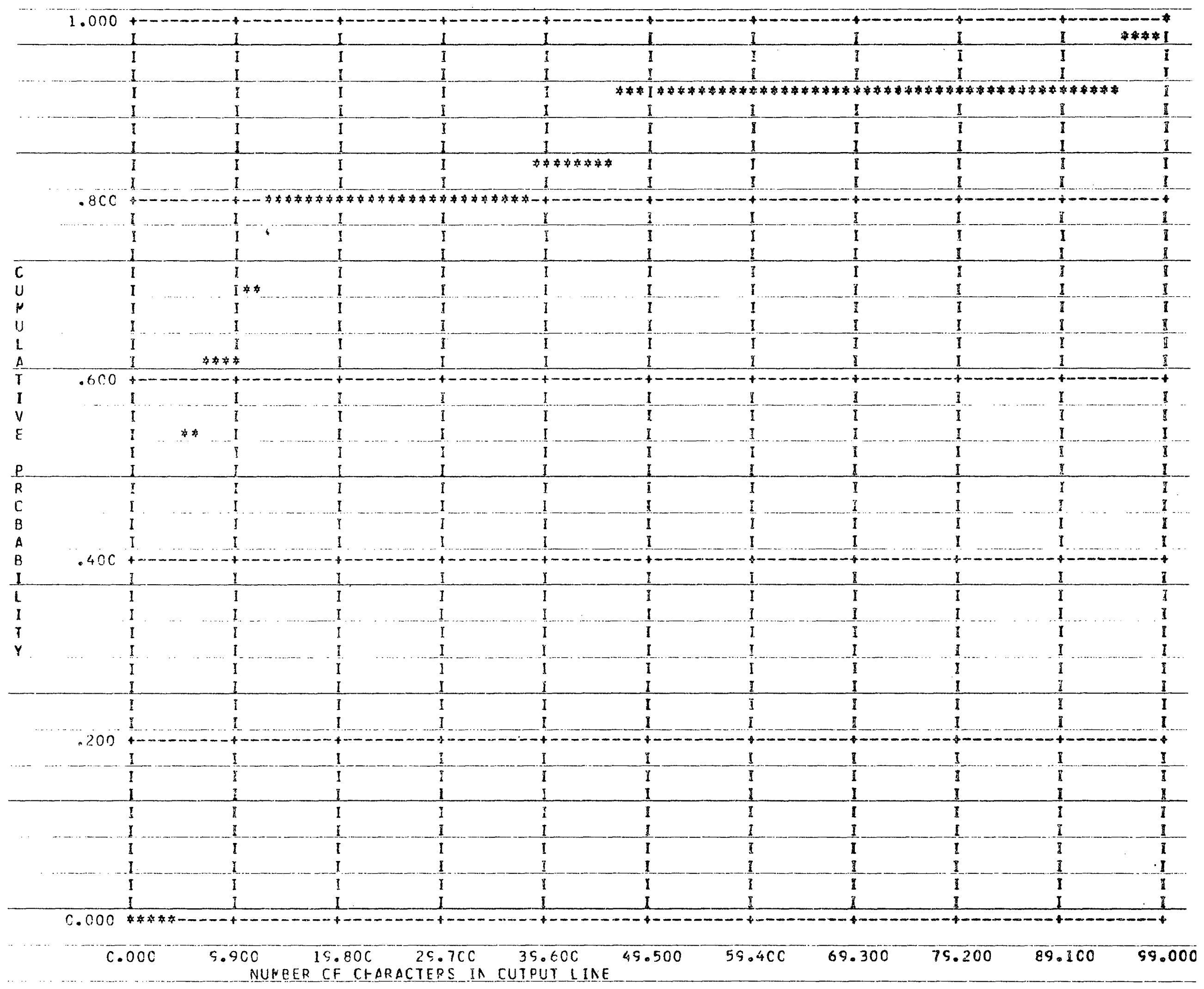


Figure 4-28

Distribution of Output Line Lengths

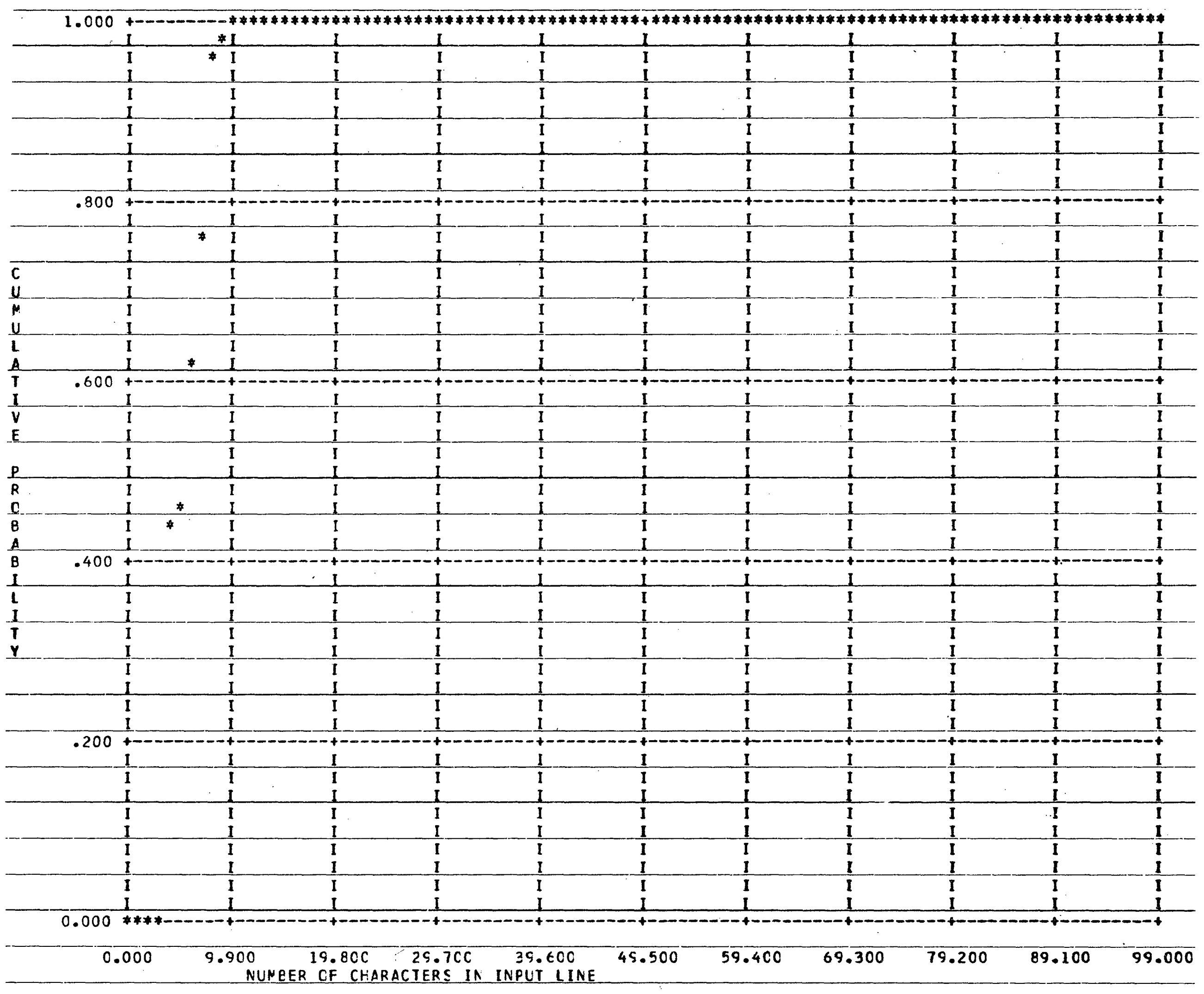


Figure 4-29

Distribution of Number of CPU Intervals per Response Period teletype terminal user, and the display user should therefore have a shorter connect time. Third, the 2250 display used for graphics work takes more CPU processing than when it is used as a teletype. Last, the remote display terminal uses more CPU processing than any other application class. However this final point, being based on just one data set, must at best be regarded as tentative. Also, the 360/67 hardware configuration in use when the remote display data was taken differed from the hardware in use when the other data was taken.

All this data, then, gives very positive confirmation to the idea that displays in place of teletypes use the CPU more and increase response time, and that graphics-oriented work requires more CPU time than does teletype-oriented work.

Figures 4-21 through 4-29 show some of the probability distributions and the summary data found from the remote display terminal statistics. They are meant to be indicative only of the type of information available: because the data represents only one application, no conclusions should be made from it.

REFERENCES, CHAPTER 4

1. International Business Machines Corporation, General Purpose Simulation System/360 User's Manual, IBM Publication No. H20-0326, White Plains, New York.

2. Pinkerton, Tad B., "Program Behavior and Control in Virtual Storage Computer Systems," Concomp Technical Report 4, The University of Michigan, April 1968.

3. Pinkerton, Tad B., "The MTS Data Collection Facility," Concomp Memorandum 18, The University of Michigan, June 1968.

4. University of Michigan Computing Center, *Michigan Terminal System*, (2nd Ed.), Ann Arbor, 1967.

APPENDIX A

The Degree of Multiprogramming in Page-On-Demand Systems

by

V. Wallace and D. Mason

published in
<u>Communications of the ACM</u>
Volume <u>11</u>/Number 6/June, 1969

1. Introduction

The relationship between the number of programs permitted simultaneous assignment of core, drum traffic rates, and central processing unit (CPU) utilization is a central relationship to a page-on-demand, multiprogrammed, time-shared computer system [1]. A clear understanding of this relationship is vital to systems planners (software or hardware) if adequate performance is to be obtained. In this apaper, a simple stochastic model will be described which offers a base for that understanding. This paper also serves to further demonstrate the feasibility of applying numerical analysis of queueing models to the development of general insight into key computer organization questions.

Considerable investment has been made in building systems to operate under the page-on-demand strategy, and difficulty has been encountered in obtaining satisfactory performance. Several excellent studies, both analytical [2, 3] and simulational [4, 7, 10], have been applied in attempts to explain and predict their performance. One of the chief underlying conclusions of each of these studies is that the number of tasks simultaneously in core memory must be severely limited if efficient operation is to be achieved. However, because of the detail attempted in these studies, a clear, general picture of the fundamental trade-offs has not emerged. The model to be presented offers such a picture by ruthless simplification, particularly in those places where reliable data is sparse and the use of "guesstimates" must still prevail. The simplicity can then be exploited by exploring a much broader range of environmental parameters than would otherwise be possible. The insights resulting can better educate our intuition when reviewing more refined results and more refined environmental statistics as they become available.

2. A Model

To that end we shall consider a system (Figure 1) which possesses a single CPU, a single secondary store for page swapping (which will be called a drum, for concreteness), and a single data channel serving that secondary store. Overhead (execution of system software) and non-drum input-output (I/O) operations will be ignored. Furthermore, it will be assumed that the supervisor is so designed that the number of users simultaneously having core assigned (which will be called the "degree of multiprogramming") will have a maximum value N, and further requests for execution when N users' programs are already in this active condition will simply be queued until one of the N active programs terminates or is terminated by the supervisor.

We wish to determine, among other things, the rate R at which jobs having a known statistical character can be completed when requests for execution continuously exceed the rate of service. On the other hand, our only concern is with the behavior of user programs during the interval when they actually have core memory assigned to them. Thus, a "job" in our context will be that set of operations which starts when a page of core is first put at the disposal of a user, and ends the next time that the user's program is made ineligible to hold core memory. It is during that interval, which will

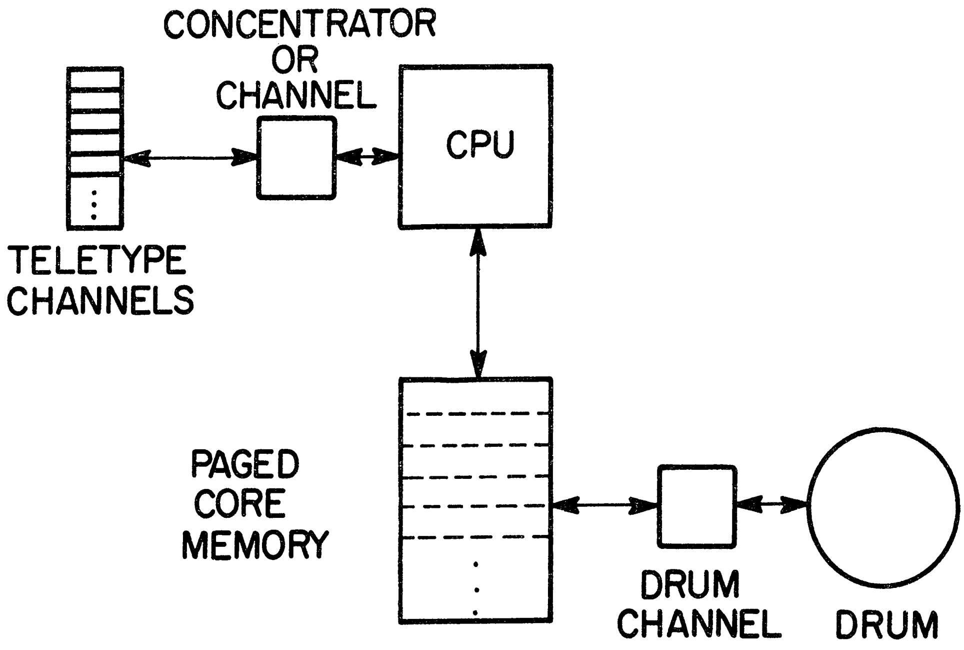


Figure 1

The Simplified System

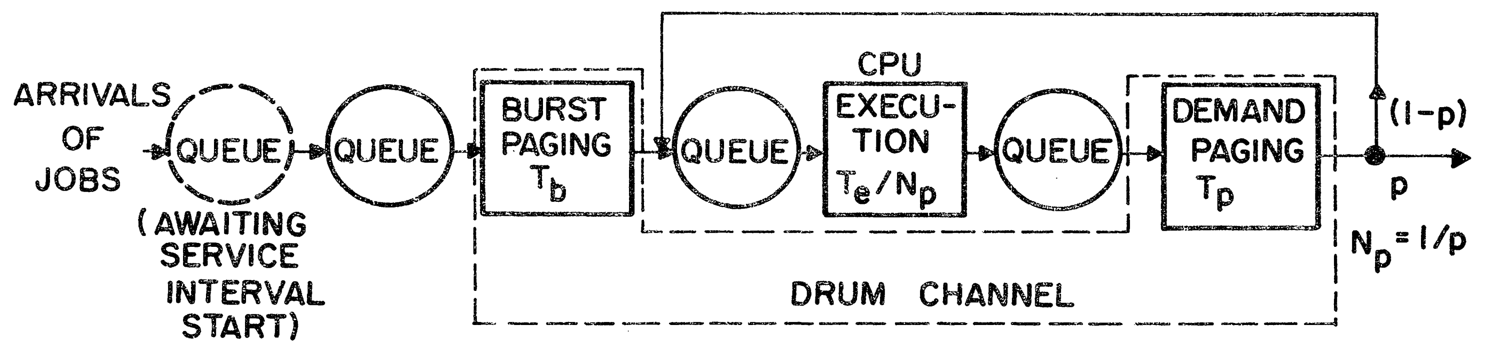


Figure 2

Schematic of the Model

be called a "service interval", that execution of the program will take place—in smaller intervals interspersed with the page transfers demanded from the drum. During any service interval, the control of the CPU will be shifted among the N jobs in core as they receive their demanded pages.

Finally, we make the following assumptions about the page demands:

1) Upon initiation of a "job", there is a burst of page demands during which a negligible amount of execution is achieved for that particular job.

2) The cumulative drum-channel service time for this burst of demands is an independent, exponentially distributed random variable having mean T_b.

3) Once the burst is over, a job executes for an independent, exponentially distributed interval of time before a page demand occurs.

4) The drum-channel service time for the latter request is an independent, exponentially distributed interval with mean T_p.

5) The execution-page demand cycle is repeated a random number of times, with the number of pages demanded after the initial burst (but before termination of the job) being a geometrically distributed random variable having mean N_p.

6) Jobs which become eligible for either execution or page transfer join an appropriate queue whenever another job is already undergoing execution or page transfer. Jobs in the

midst of their initial burst of page demands will be interrupted by the non-burst demands. When the burst demands cannot proceed, the jobs waiting for such such service also form a queue, separately from the execution and non-burst page demand queues. (See Figure 2.)

It is readily shown that the above assumptions assure that the total execution time taken by each job is also an independent, exponentially distributed random variable having a mean which is N_p times the mean execution interval described in assumption (3) above. The mean total execution time will be designated T_e, so that the mean execution interval will be T_e/N_p.

The assumption that the supervisor will enforce an upper limit (N) on the number of programs allowed to have core assigned results from the conviction, supported by the results below and the cited prior art, that such a policy <u>must</u> be enforced (under heavy use of the system) if the page demands are not to swamp the drum channel, with serious consequences to CPU utilization and to the rate of job completions. Indeed, it is a central idea of this paper that for a given set of environmental statistics (T_e, T_b, N_p, T_p), there will be a finite "best" value for the variable N.

The assumed "burst" behavior reflects, qualitatively, behavior described by Fine, Jackson, and McIssac [5] and by Coffman and Varian [8] in experimental studies. Although their data were taken from a simulated paged system, and although they measured paging characteristics for programs not specifically wirtten for a paged system, their results do give an idea of the nature of the page-on-demand environment. Besides showing that the rate of page demands is remarkably greater when the number of pages assigned to a job is small (as would be the case for recently started jobs), they also argue forcefully that the area of primary concern should be the high level of page requests when any program has significantly less core than its total storage requirements.

This "burst" assumption also corresponds to the assumption of the existence of a "working set" of user pages, used by Denning [9] to model paging behavior. The "burst" represents acquisition of the initial working set. Obviously, that set will consist of an average of T_b/T_p pages, and total memory will need to be at least N T_b/T_p pages if the burst is not to be prolonged by considerable reshuffling of pages. The assumption of exponential distributions for drum channel service time gains some credence from the realization that several factors (access, latency, transfer, and posting delays) make up its total, and their durations are in general independent of one another. Latency will favor shorter intervals because the write operations will use the first sector available. Organization of queue disciplines can also affect distributions. Nevertheless, need for simplicity is still the major justifiction.

Unfortunately, the samples of the statistical measurements by Fine, et al., are too small, and the results insufficiently differentiated, to give an accurate notion of the distribution function for the execution intervals conditional on the degree of multiprogramming. Smith [ ] argues for the hyperexponential approximation. However, the principal reason for the exponential assumptions used here is to avoid complicating the model, especially in view of the sketchiness of present data.

It seems reasonable, also, to assume that continuing jobs would be given priority in the use of the drum-channel over jobs undergoing their initial burst of paging. Prudence suggests that once you have committed a major portion of core to a user, you should do everything possible to expedite completion before committing yourself to new arrivals.

While this model does not explicitly show the process of task or job creation, "thinking" delays at terminals, priority assignments in queues, page removal strategies, time-slicing, and many other facets of time-sharing operation, it is nevertheless a fairly realistic representation for the purpose at hand. Each of these factors has an effect on the parameters T_e, T_b, N_p, T_p, through their influence on the distributions of the job properties— execution intervals, number of pages demanded, etc. While there may be considerable dependence among the statistical properties of the jobs of a particular user at a terminal, these jobs will begin their service intervals at widely spaced times, mixed in order with the jobs of many other users, making the assumed independence a reasonable approximation.

3. The Results of Analysis

The model described was analyzed for 448 separate combinations of the parameters T_e, T_b, N_p, and N covering a wide range of values. The analysis was accomplished using the Recursive Queue Analyzer (RQA) [6], a program for rapid numerical solution of the equations of equilibrium probabilities for Markov chains. The results were calculated to three significant figures accuracy, using a total of about 20 minutes of RQA execution time (on an IBM 7090).

The results of the analysis are plotted in Figures 3 and 4, which show the CPU utilization η as a function of the parameters. The CPU utilization as $N \to \infty$ was hand calculated according to the formula.

$$\eta = \begin{cases} 1 & \text{whenever } T_b + N_p T_p \leq T_e \\ \dfrac{T_e}{T_b + N_p T_p} & \text{otherwise} \end{cases}$$

All results are shown with time normalized to units of the mean page transfer time T_p. Thus $T_e = 30$ means that, on the average, the mean total execution time of a job is 30 times the drum-channel time for a page transfer. If it were assumed that the drum channel could service page requests at an average rate of one every 5 milliseconds (ms), then $T_e \approx 30$ would correspond to an average total execution time of

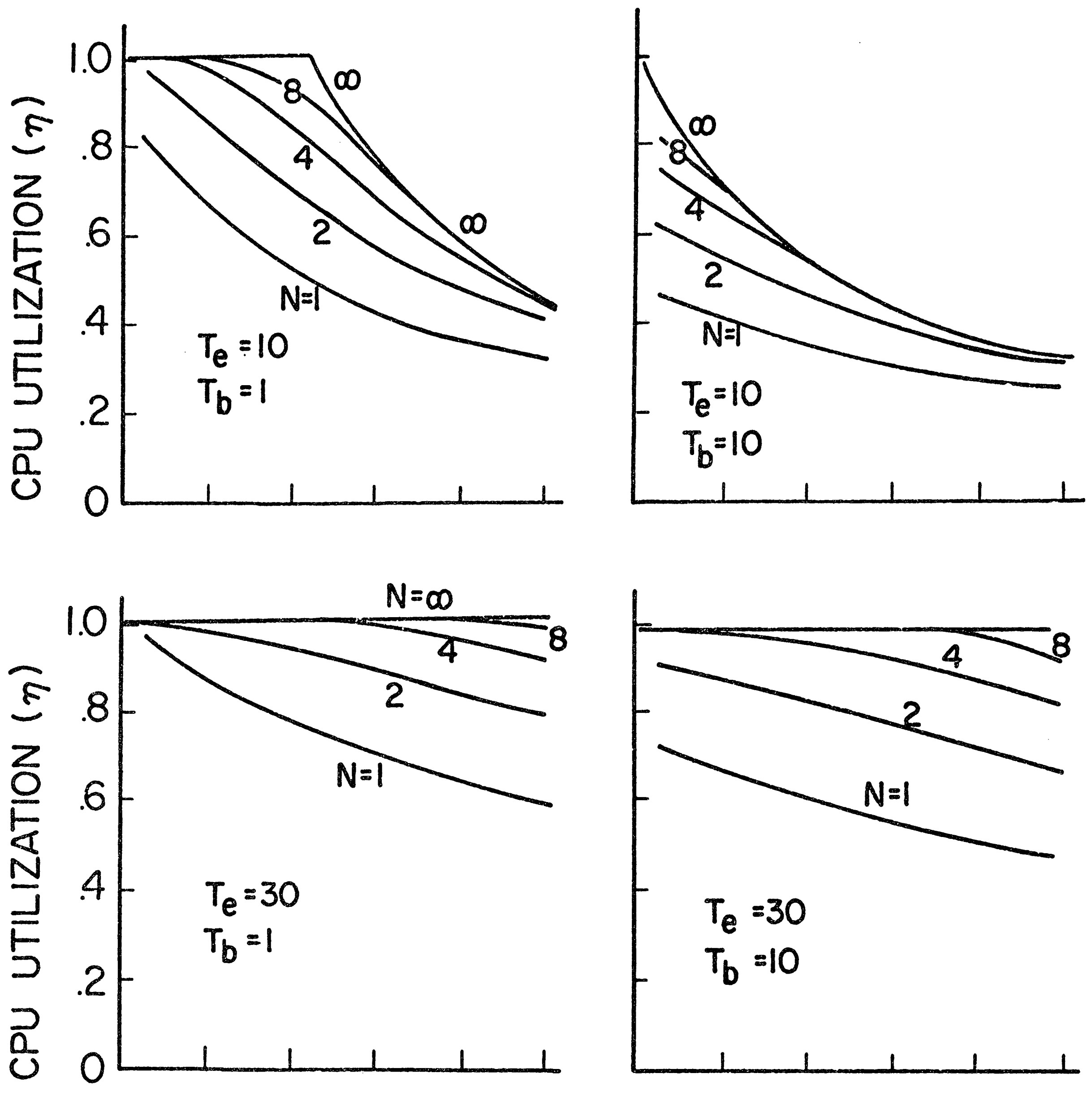


Figure 3

Analysis Results

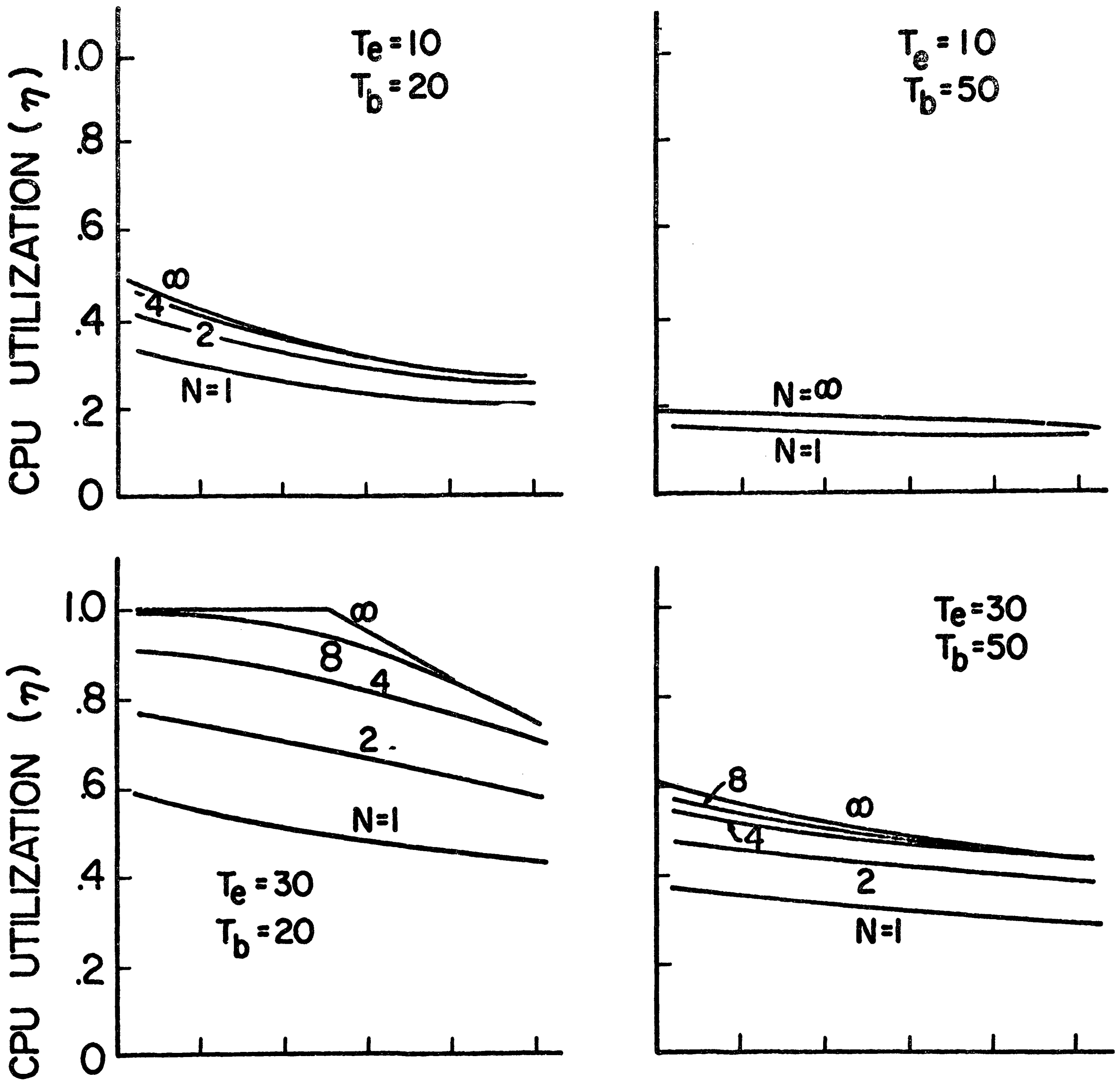


Figure 4

Analysis Results

150 ms.

The point of balance between execution time per job and drum-channel time per job is, in every case, the value of N_p at which the $N = \infty$ curve breaks away from $\eta = 1.00$. It is seen that this value of N_p is critical, for it represents a value at which CPU utilization begins to fall markedly, regardless of the degree of multiprogramming.

The CPU utilization is proportional to the expected rate R of job completions (per unit time), according to the formula

$$R = \eta/T_e,$$

while the expected service interval can be found from the formula

$$W = N/R.$$

Consequently, the results shown readily infer the values of these alternate measures of performance.

The figures show how, other parameters remaining constant, CPU utilization improves as more programs are permitted to share the core memory. This, of course, is the desired effect of multiprogramming, resulting from the improved probability that some one of the jobs is available for execution at any given point in time.

It is to be noted, however, that the improvement becomes increasingly small as N is increased. Indeed, in every case the ratio η/N decreases with increasing N, and we thus deduce that the expected service interval W will always <u>increase</u> with increased N. Nevertheless, the improved rate of service completions R will result in a more than compensating decrease in the time the job must wait to get into the active state, have core assigned, and begin its service interval. Thus, overall delay, as seen by users at terminals, will decrease in spite of the increase of the service interval. Since the waits before entering the service interval will represent the major portion of the delays, they can be expected to be roughly proportional to $1/\eta$.

The figures also show that over a very wide range of the parameters $T_e, T_b, N_p,$ and $T_p,$ there is relatively little to be gained by choosing an N greater than about 8, since those curves already approach the $N=\infty$ curves quite closely. Indeed, when the rate of page requests is such that drum-channel demands (as described by $T_b + N_p T_p$) exceed the CPU demands (T_e) of a job, all performance measures inevitably get worse, and multiprogramming offers less and less advantage.

4. The Optimum Degree of Multiprogramming

The figures offer a picture of the advantages of multiprogramming which is misleadingly optimistic. Because systems generally operate under a core memory limitation (for economic reasons, if no other), it can be expected that an increase in N will cause each job to have, on the average, proportionately less core available to it. This, in turn, will cause the amount of page shuffling to increase, tending to lessen performance, and countering some of the advantage of the increased N. We do not, as yet, have a very good quantitative idea of how these page demands might increase with decreasing core. However, suppose for example that the number of page requests per job, N_p, was known to relate to the degree of multiprogramming in such a manner that

$$N_p = m\,N + b.$$

Then, for $m > 0$ there will be some N which gives best CPU efficiency. The dotted curve in Figure 5 shows the effect of increasing N when $T_e = 30$, $T_b = 20$, $m=2$, and $b=0$. For this example, if one must choose between N=1, 2, 4, or 8, the best choice would be 4, since that gives best CPU utilization.

Figures 6 and 7 show the optima for a number of choices of T_e, T_b, m, and b. What is shown in each case are the regions in an (m, b)-space for which the optimum selection would be either 1, 2, 4, or 8 when these are the only allowed values. The solution of Figure 5 is shown as a heavy black point on the horizontal axis of Figure 7 (T_e=30, T_b=20). Notice that small values of N are favored for conditions where the expansion of demands on the drum is great, and where execution times are small. The size of the burst page demand seems not to be too important to the choice of optimum N, although it is very important to the performance which that optimum gives.

It is remarkable that the selection is so relatively insensitive to b. It is also clear that the value of m for the jobs to be served by the system is an extremely important parameter in determining a suitable choice of N. Hopefully, future published data from real or simulated POD systems can give us a better idea of what values to expect.

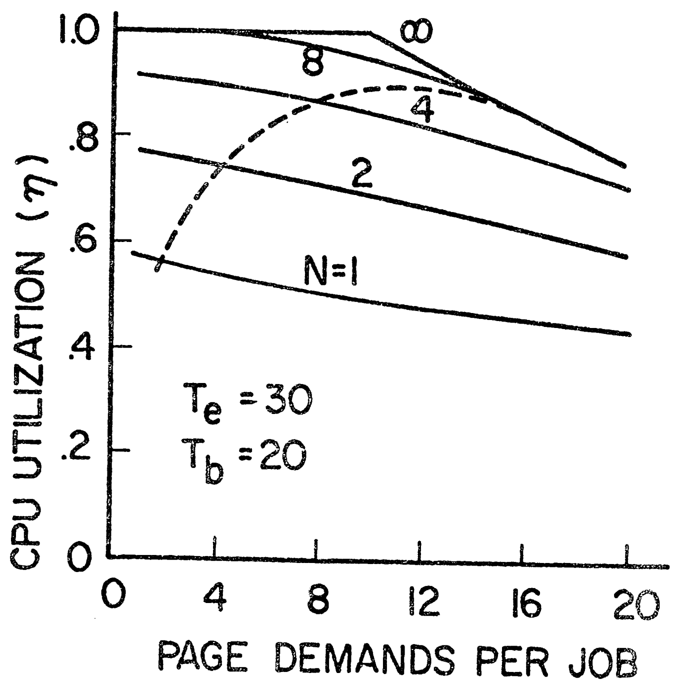


Figure 5

The Effect of Multiprogramming when $N_p = 2N$

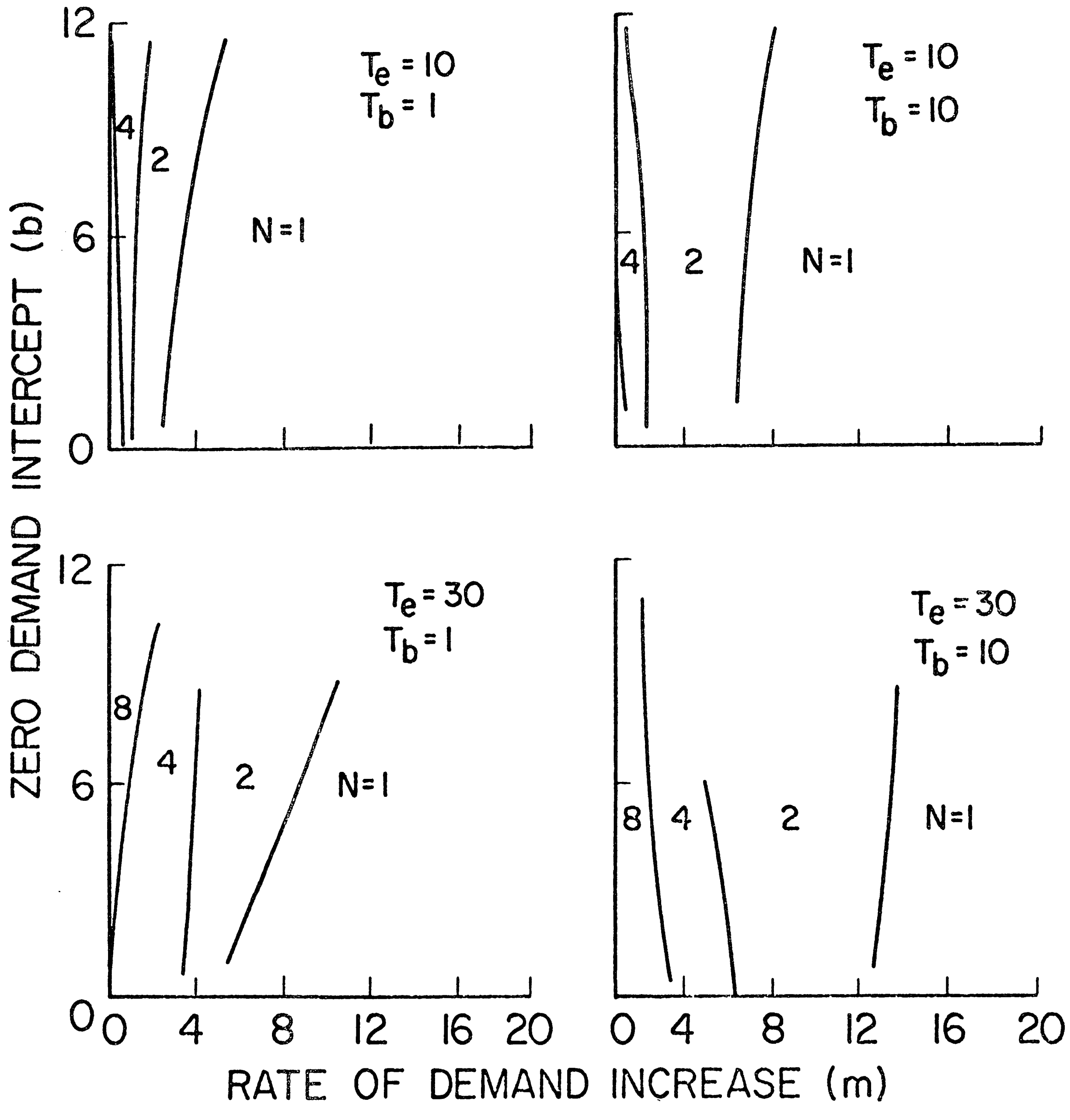


Figure 6

Optimum Degree of Multiprogramming

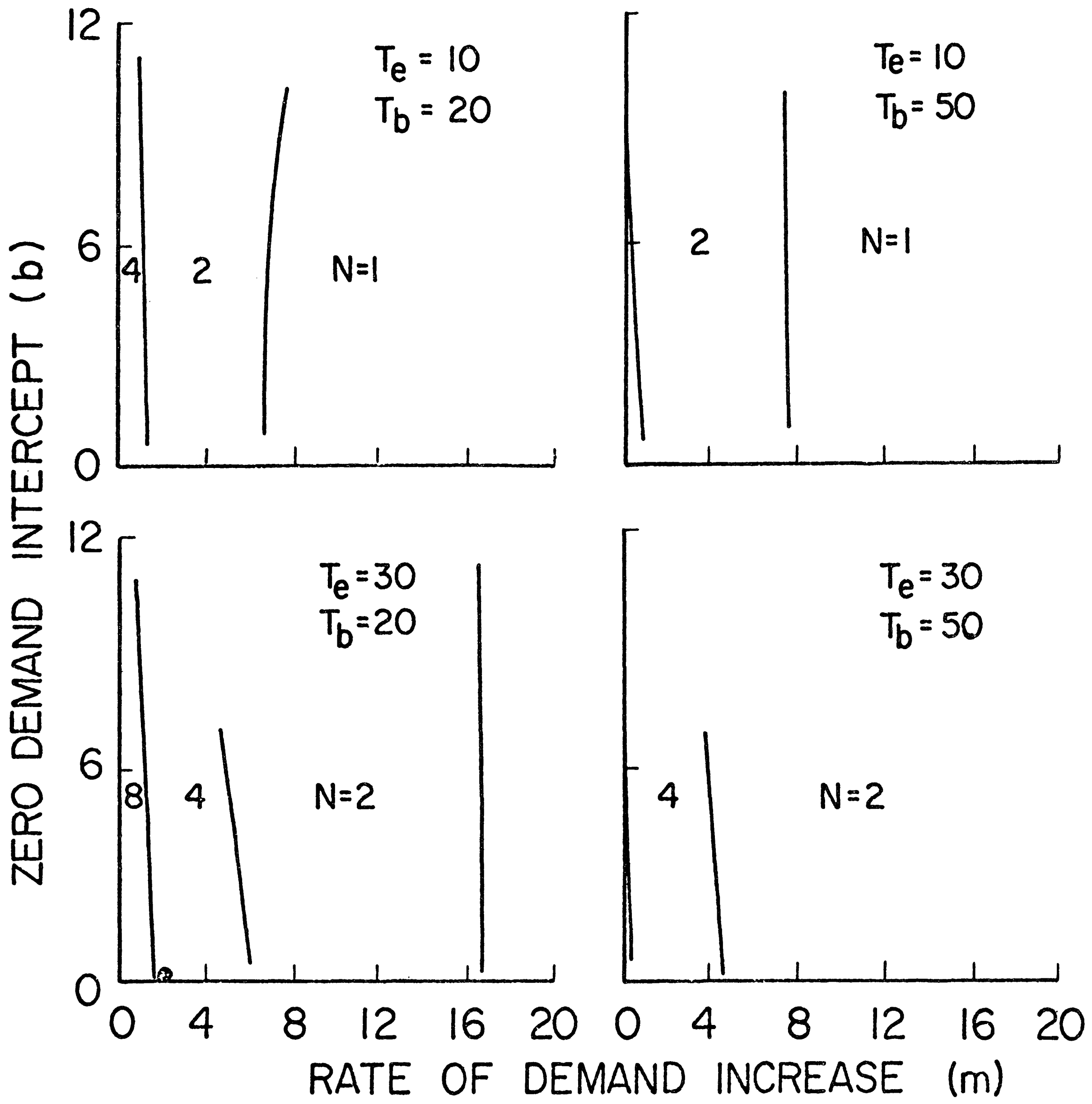


Figure 7

Optimum Degree of Multiprogramming

5. Conclusions

The observations above have been based on a grossly simplified model, and have aided understanding of a complex problem through exploration over a wide spectrum of parameter values. This is the type of analysis most benefited by queueing models, and it is well served by numerical solution technique.

The analysis has provided a more quantitative aid to understanding the role of multiprogramming and of job characteristics in determining CPU utilization in POD systems. While most of the observations are intuitively unsurprising, or perhaps even obvious, the model offers some clearer insight into the questions "how much better?" and "how much worse?". It also dramatizes the fact that the optimal choice of degree of multiprogramming is primarily determined by a few key statistical properties of the job population to be served. While scheduling strategies and equipment selection can strongly affect these parameters, it is still vital that an idea of their values for each choice of strategy and equipment be available for a rational choice of N to be made.

REFERENCES, APPENDIX A

1. Arden, B. W., Galler, B. A., O'Brien, T. C., and Westervelt, F. H., "Program and Addressing Structure in a Time-Sharing Environment," J. ACM, 13, 1 (January 1966), pp. 1-16.
2. Smith, J. L., "Multiprogramming Under a Page on Demand Strategy," Comm. ACM, 10, 10(October 1967), pp. 636-646.

3. Gaver, D. P., Jr., "Probability Models for Multiprogramming Computer Systems," J. ACM, 14, 3(July 1967), pp. 423-438.

4. Nielsen, N. R., "The Simulation of Time-Sharing Systems," Comm. ACM, 10, 7(July 1967), pp. 379-412.

5. Fine, G. H., Jackson, C. W., and McIsaac, P. V., "Dynamic Program Behavior Under Paging," Proc. ACM, National Meeting, 1966, pp. 223-228.

6. Wallace, V. L., and Rosenberg, R. S., RQA-1, The Recursive Queue Analyzer. Technical Report No. 2, Systems Engineering Laboratory, University Of Michigan, Ann Arbor, Michigan, February 1966.

7. Oppenheimer, G., and Weizer, N., "Resource Management for a Medium Scale Time-Sharing Operating System," Comm. ACM, 11, 5(May 1968), pp. 313-322.

8. Coffman, E. G., and Varian, L. C., "Further Experimental Data on the Behavior of Programs in a Paging Environment," Comm. ACM, 11, 7(July 1968), pp. 471-474.

9. Denning, P. J., "The Working Set Model for Program Behavior," Comm. ACM, 11, 5(May 1968), pp. 323-333.

10. Pinkerton, T. B., Program Behavior and Control in Virtual Storage Computer Systems, Technical Report 4, Concomp Project, University of Michigan, Ann Arbor, Michigan, April 1968.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* University of Michigan Systems Engineering Laboratory Ann Arbor, MI 48104	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE
A Study of Information Flow in Multiple-Computer and Multiple-Console Data Processing Systems

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Final Report (Jan 68 to Mar 69)

5. AUTHOR(S) *(First name, middle initial, last name)*

K. B. Irani	J. W. Boyse	V. L. Wallace
A. W. Naylor	D. M. Coleman	A. M. Woolf
J. D. Foley	D. R. Doll	

6. REPORT DATE August 1969	7a. TOTAL NO. OF PAGES 200	7b. NO. OF REFS

8a. CONTRACT OR GRANT NO. AF 30(602)-3953	9a. ORIGINATOR'S REPORT NUMBER(S) Annual Report No. 3
b. PROJECT NO. 5581 c. Task No: 02 d.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* RADC-TR-69-189

10. DISTRIBUTION STATEMENT
This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC (EMIRA), GAFB, NY 13440.

11. SUPPLEMENTARY NOTES RADC Project Engineer: Lawrence W. Odell EMIRA	12. SPONSORING MILITARY ACTIVITY Rome Air Development Center (EMIRA) Griffiss Air Force Base, New York 13440

13. ABSTRACT
This report documents the achievements from January 1968 to March 1969 of continuing research into the application of Queueing Theory and Markov Decision Processes to the design and investigation of multiple-user systems. A summary of the theoretical investigation conducted, the major conclusions reached, and some typical applications are included.

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Data Processing Systems Digital Computers Programming Displays Models (Simulations) Mathematics						