STUDY OF RESOURCE ALLOCATION IN COMPUTER SYSTEMS BY SIMULATION

A Thesis

Presented To

the Faculty of Graduate Studies and Research

The University of Manitoba

In Partial Fulfilment

of the requirements for the Degree

Master of Science

in the Institute for Computer Studies

by

Bruce H. McDonald

February 1969

ABSTRACT

Various job-scheduling algorithms based upon round-robin scheduling with a variable time-slice are studied by simulation under various conditions of system utility. System utility is varied by varying the inter-arrival time of jobs and by varying the quantity of main storage available to jobs. A measure of performance is based upon curves which represent the cost of delay to jobs. This measure of performance is very similar to a performance criterion previously used.

An empirical equation is developed which relates performance to the utilities of the central processor and main storage by means of a variable coefficient. For simulated conditions of utility less than 0.60 the empirical equation is found to agree fairly well with the simulated performance with a constant coefficient value which depends on the number of priority classes. For simulated conditions of higher utility, variability in the coefficient is required to fit the empirical equation to the simulated performance. As utility is increased beyond 0.80, simulated performance becomes a function of the job-scheduling algorithm and when utility reaches unity, simulated performance becomes a function of the number of jobs in the job stream as well.

Those algorithms which tend to select the shorter jobs from the waiting queue for processing during the simulation produce a somewhat better performance.

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Professor T. A. Rourke, my thesis supervisor, for his guidance, direction, advice and criticism which have been so helpful to me in this research.

I wish to thank Professors S. R. Clark and A. Wexler for their time spent in reading this thesis, and for their helpful comments and criticisms.

I wish to thank Mr. Parker (Bud) Caufield and his computer operators for the assistance and cooperation given to me during the simulation studies.

Also, I wish to thank my typist, Mrs. Betty Kimery, for so accurately deciphering my hieroglyphics while preparing this thesis.

TABLE OF CONTENTS

PAGE

PAGE

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

NOMENCLATURE

The following notations are used in this thesis:

A_{jI} : actual interarrival time for job j of priority class I, (page 15).

A_{jI} : arrival time of job j in priority class I, (page 12).

A'_{jI} : earliest time when job j of priority class I can be loaded into storage, sum of arrival time (A_{jI}) and device time (d_j), (page 16).

A^*_{jI} : time when job is loaded into storage. Sum of earliest possible time (A'_{jI}) and time spent in waiting queue, (page 23).

A_s : input parameter determining which queue selection algorithm to be used, (page 45).

b : storage time constant, giving the rate at which mean storage requirement increases with central processor time request, (page 17).

B : maximum mean storage requirement for jobs in the job stream, (page 17).

C_o : coefficient used to fit empirical equation to observed simulated results, (page 52).

d : input parameter device time number, (page 19).

d_j : assumed device time of job j, (page 12).

D_o : time demand on central processor, (page 31).

D_1 : units·time demand on main storage, (page 32).

ε : small quantity used to represent variation from unity where lack of attention actually becomes infinite, (page 52).

g_j : augmented lack of attention used in quantum allocation routine, (page 42).

I_j : priority number of job j in priority class I, (page 12).

job j : the j^{th} job in the job stream, (page 12).

k : a priority constant, (page 13).

L : the number of priority classes I, (page 12).

m_I : mean central processor time request, jobs of priority class I, (page 14).

M : mean storage requirement for jobs j with central processor time request t_j, (page 16).

n_I : the number of jobs of priority class I, (page 13).

N : the number of jobs in the job stream, (page 11).

O_1 : storage occupancy, (page 33).

p_j : penalty of job j, (page 25).

p_j^* : penalty of job J at completion of job j, (page 25).

P : penalty function, (page 24).

Q : length of waiting queue, (page 35).

r_I : mean interarrival time, jobs of priority class I, (page 14).

R_j : storage requirement of job j, (page 12).

$\overline{R}_I$: mean storage requirement of jobs in priority class I, (page 33).

$\overline{R}_M$: sum of mean storage requirements of jobs in all priority classes ($\Sigma\overline{R}_I$), (page 33).

σ_N : standard deviation in system penalty, (page 34).

s : number of slots in the execution list filled by jobs, (page 42).

s_o : time supply of central processor (unity), (page 31).

S_1 : units·time supply of main storage (page 32).

τ_j : time devoted to job j by the computer, includes device time (d_j) and central processor time given, (page 25).

t_j : actual central processor time requires of job j, (page 12).

T : total time (simulated) for simulation run, (page 32).

T_j : time at completion of job j, (page 21).

ΔT_j : duration of job j, elapsed time between job arrival (A_{jI}) and job completion (T_j), (page 21).

T_j^1 : time between arrival of job j in storage and completion of job j, (page 24).

T_R : round robin cycle time, (page 41).

T_{Rj} : time slice allocated to job j by the variable time slicing algorithm, (page 41).

T_{RM} : minimum time-slice, (page 41).

T_{sc} : supervisor cycle time, (page 39).

μ_j : lack of attention of job j, (page 24).

μ_j^* : lack of attention of job j at completion of job j, (page 24).

μ_M : mean lack of attention for simulation run, equals the system penalty when $p_i = \mu_i$, (page 49).

U : utility, defined as time ratio of demand to supply in the competition concerned, (page 50).

U_o : central processor (C.P.U.) utility, (page 27).

U_1 : storage utility, (page 32).

W_o : number of slots in the execution list, (page 34).

W_1 : number of units of main storage initially available to jobs in the job stream, (page 32).

W_t : number of units of main storage available at any time t during the simulation run, (page 42).

CHAPTER I

INTRODUCTION

1.1 Purpose of the Research

In this study use is made of the computer system simulation model which has been used by Chai to investigate some of the effects of a particular time-slicing job-scheduling algorithm (1)[1]. This model is the first stage of a more comprehensive computer system simulation model which will include the operations of logging-in, loading into core, scheduling and outputting of the results. This first stage may be considered as a model of a hypothetical computer system which incurs no overheads to load jobs into core, and which processes to completion jobs which require no input or output during their execution. In the study conducted by Chai (1) it was assumed that the physical size of the main store was large enough to accomodate any three jobs. However, no attempt was made to consider the possibility that this same main store may be able to accomodate more or fewer than three jobs during certain periods of time.

The purpose of this research is to examine the service to computer users as the number of units of main

---

[1] Notation (n) refers to the n^{th} entry in the list of references.

storage available changes for a variety of different job-scheduling algorithms. A job-scheduling algorithm may be separated into two sections, the first is the logic which determines which of the waiting jobs is to be accepted next (queue selection algorithm) and the second is the logic which determines how a job is serviced once it is accepted (quantum allocation routine). More specifically, the purpose of this research is to examine various queue selection algorithms under conditions of varying storage.

1.2 Methods and Scope of the Research

A small modification of the model used by Chai, in order to keep account of how many units of main storage are occupied and how many units are free, makes the model suitable for the study of queue selection algorithms. The main memory is attributed with a size and each job within the job stream requests a particular amount of this main storage. Providing there is sufficient storage available for a particular job, the job can be accepted for execution. No account is taken of the spatial arrangement of jobs within the main storage and therefore the organization within the main memory is taken to be either non-contiguous, that is, the main storage is split into units of an arbitrary size, but jobs need not occupy these units in a contiguous manner, or alternatively, contiguous organization with negligible time of relocation of jobs within the main storage.

The basis of a measure of computer system performance has been suggested by Greenberger (2). This measure of performance involves the summation of the cost of delay to each job requiring service by the system. The measure of performance used in this study and in the previous study by Chai is based upon Greenberger's suggestion with a modification which makes it quite similar to a measure of performance used by Fife (10). Fife considered that the relative response, that is, the ratio of the response time to the amount of processing time required, was a quantity more fundamental than the actual response time to the measurement of performance.

The study of the performance of computer systems for a constant job stream and various job-scheduling algorithms can lead to the choice of a job-scheduling algorithm for a particular computer system and for the job stream considered. However, once selection of a computer system, or configuration, and a job-scheduling algorithm have been made, the job load generally grows rapidly until the system becomes heavily loaded. It is therefore often wise to select a job-scheduling algorithm which functions well under heavy loading, and it is an advantage to study the performance of an algorithm over a range of different job loads.

Many modern computers are modular in the sense that more main storage can be attached upon request, providing some upper limit is not exceeded. The size of main storage can be

considered as a variable within a computer system to be selected according to some economic criterion, generally.

It is reasonable to suppose that, if a computer user is paying to receive a certain maximum relative response either directly, or indirectly by offering a certain proportion of a monthly computing capacity, for example, and that if this maximum relative response is exceeded, a discount amounting to a cost of delay to the user will be given. In this way the measure of performance would be related to the earning power of the computer system.

The decision whether or not to acquire more main storage in order to improve service can be related to economics by studying the improvement in performance as main storage is added to a computer system and balancing this improvement with the extra cost of acquiring the main storage. In this study no assumptions are made concerning the latter cost and the measure of performance is not related specifically to economics, but the effect upon system performance as the size of the main storage is varied is examined.

1.3 Previous Work in the Area of This Research

Most of the published research concerning the performance of time sharing systems has dealt with the performance of scheduling algorithms, and in particular, with the

logic used for allocating execution times to jobs for which sufficient storage is already available. This logic is known as a time-slicing algorithm, or quantum allocation routine.

The primary reference for this study is the thesis of Chai (1) in which the computer system simulation model used in this study is introduced and described, in which a variable time-slice quantum allocation routine is presented, and in which the dynamic job penalty is used as a means of measuring user service. The Monte-Carlo techniques for job stream construction used in this study were also used in the previous study of Chai. In his study both central processor time requests and interarrival times were assumed to be normally distributed, while in this study arrival times are assumed to follow a Poisson distribution.

Kleinrock (3) presents an analytical study of time-shared computer systems in which facilities are treated as stochastic queuing systems under priority service disciplines. The performance measurement of these systems in taken to be the response time expected by the job under consideration. The results presented are for an ideal system. The Priority Processor-shared system analysed includes several priority classes with Poisson arrivals and exponentially distributed service requirements, with a known mean service requirement for jobs of each priority class. For this system Kleinrock states and proves a theorem which relates expected response time to mean central processor utilization in a fashion very similar to the empirical relation developed in the studies

reported here. Other job scheduling algorithms are analysed, with theorems relating performance to utilization being stated and proved, by classical queuing theory, for each algorithm.

Schrage (4) presents an analytical study of the queuing and servicing discipline M/G/1 with feedback. This is a round-robin type of discipline. His performance measurement is the expected response time of the job under consideration. Arrival times are taken to be Poisson and execution times are taken to be exponentially distributed. Analytical relationships are given for the expected response time as a function of job arrival rate and job processing time. A graph is presented showing expected response time as a function of central processor utilization for various job processing times. For certain values of the job processing time, the graph is similar to some graphs presented by Kleinrock (3).

Penny (8) studies the effects of time-shared and non-time-shared computer facilities. He discusses improvements to be made in work load processing times that can be made by time and space sharing. The analysis produces a range of improvements as a function of processor utilization. Results obtained by simulation are shown to be in the range predicted, for the time sharing of two, three and four jobs.

Huesmann and Goldberg (6) present a survey article describing research in time-slicing algorithms with particular emphasis on Scherr's work (13) and the LOMUSS system (5,6) at Lockheed Corporation (Lockheed Multipurpose Simulation System). Huesmann and Goldberg make several very interesting statements. They suggest, for a successful

simulation, that the details of the job-stream must be specified formally, that the constraints of the operation must be specified formally, and that the characteristics for judging system performance must be clearly defined. This is all in complete accord with the philosophy used by Chai and continued in this research. Further, Huesmann stresses the need for 'parameterization' of input to a simulation run and suggests that the simulation approach to time sharing system analysis is popular because there is lack of a viable alternative.

Some details of the LOMUSS system presented by Huesmann and Goldberg are important to this study. The LOMUSS system permits simulation of varying computer systems, or configurations with varying job streams.

For each simulation run two types of output are produced: the state of each resource at different points in time, and what is called an overall profile of each job, from which response times, processor idle time, memory utilization, throughput, and queue behavior may be determined. The specific nature of the scheduling algorithms used is not known, and no attempt has been made to relate user service to resource utilization - at least none has been published.

Nielsen (7) describes simulation studies made of an IBM 360/67 time sharing computer. He suggests that analytic studies (9, 13) are relatively inflexible and simulation studies allow more scope. The model described by Nielsen is responsive to changes in configuration, to changes in the

scheduling and memory allocation algorithms, and to changes in the job stream. Memory is allocated by Nielsen in interchangeable sections rather than contiguously. Nielsen measures performance of his simulation model in terms of job response by priority and type and in terms of central processor utilization and equipment activity - in principle very simularly to the methods used in this research. He considers paging, that is, the rolling in and out of sections of jobs as required (17). Consideration was given to reserving storage for emergency, or heavily loaded operational conditions and found that any benefits to users were absorbed by the resultant increased idle time. Nielsen concluded that reducing the amount of paging, or rolling jobs in and out was the way to reduce overheads and improve user service. This conclusion may not be pertinent to these studies since paging is not used, but the intention of the statement - to reduce supervisor overheads-is one which is considered in this research.

Fife (10) examined the optimization of user service by using Markov model techniques in his study of the time slicing algorithm (quantum allocation routine.) The model he used included the complete rolling of jobs into and out of storage as required to have only the job being processed instantaneously in core. Three queues were used: the first, with jobs awaiting their first quantum of execution time; the second, with jobs awaiting their second quantum; and the third, with jobs awaiting their third

and/or subsequent quanta. The scheduling algorithm used determines quantum sizes and the sequence for servicing the three queues. The criterion used to assess service to the computer user, relative response times, is similar to the criterion used in this study. The objective is to minimize the average response time weighted in relation to the processing time.

Recent technical literature thus contains reports of analytic and simulation studies which are relevant to the objectives of this research. The model used here has been introduced and described (1). In some ways the model used here is similar to other models (5, 6, 13, 7). The criterion of performance used here is similar to one previously used (10). Relationships between performance and computer utilization have been studied analytically (3, 4, 8). In the studies reported here performance is related to storage and central processor utilities in an empirical fashion for varying job interarrival times, for varying quantities of main storage, and for different queue selection algorithms.

CHAPTER II

DESCRIPTION OF THE SIMULATION MODEL

2.1 Introduction

The simulation model used by Chai (1) was designed to simulate a job-scheduling algorithm which could simultaneously schedule up to three jobs. It is a model of a hypothetical computer which incurs no overhead to load jobs into storage and which processes to completion jobs which require no input or output during their execution. This basic simulation model is modified to include a different kind of competition for finite, non- contiguous storage resources. The main storage in attributed with a size in arbitrary units and each job is the job stream requests a certain amount of main storage in these units. No account is taken of the spatial arrangement of jobs within the main storage and therefore either a non-contiguous storage or a contiguous storage with very rapid relocation of jobs is simulated.

A complete description of the simulation model used by Chai is not given here. Rather a general description of the model operation with emphasis on those operations pertinent to the studies of this research is presented in the

following three sections:

1. job generation - which produces a series of jobs, each job being an entity of work that the user wishes the computer to perform. This series of jobs, or job stream, is produced by Monte-Carlo techniques using job-mix parameters specified at the beginning of each simulation run;

2. model operation - in which the generated series of jobs, or job stream, is processed according to algorithms determined by operational parameters also specified at the beginning of each simulation run; and

3. performance measurement, which describes the output from each simulation run by which model performance is measured and evaluated.

The job-mix parameters which determine the details of the job stream and the operational parameters which determine the algorithm by which jobs are processed constitute the input to each simulation model run.

2.2 The Job Stream

This section describes the nature and structure of the job stream. The job stream consists of the N individual entities of computer work, these entities being called jobs,

processed during each simulation. The processing algorithms require the following information about each job j in the job stream:

1. The execution or central processor (cpu) time request, denoted t_j;

2. the priority number of the job, I_j, which is the priority class I of the job;

3. the time of arrival of the job, denoted A_{jI}, where $I = I_j$;

4. the main storage requirement, an integer value denoted R_j; and

5. the assumed device time of the job, denoted d_j.

Although two different jobs may require the same execution time and the same main storage, they may not be equally important. A system of priorities, by which the relative importance of jobs may be specified, is used (1).

The number of priority levels, or classes, is L. Each job belongs to one of these priority classes. If job j belongs to priority class I, that job is said to have priority I; where $I_j = I$. The jobs of greatest importance, or highest priority belong to priority class one $(I = 1)$, and those of the least

importance, to the last priority class $(I = L)$. Thus for any two jobs, for instance, job j and job k, if I_j is smaller than I_k, job j is said to be the more important job and has a higher priority. In certain studies conducted three priority classes were considered $(L = 3)$.

A property of the job streams considered for three priority classes is that :

$$n_1 = kn_2 , \tag{2.2.1}$$

$$n_2 = kn_3 , \tag{2.2.2}$$

where n_I is the mean number of jobs with priority number I, and k is defined as the priority constant. Since in total the job stream consists of N jobs,

$$N = n_1 + n_2 + n_3, \tag{2.2.3}$$

or

$$N = (k^2 + k + 1)\, n_3, \tag{2.2.4}$$

and

$$n_1 = \frac{k^2 N}{k^2 + k + 1} , \tag{2.2.5}$$

$$n_2 = \frac{k N}{k^2 + k + 1} , \tag{2.2.6}$$

$$n_3 = \frac{N}{k^2 + k + 1} . \tag{2.2.7}$$

The mean number of jobs in each of the three priority classes is thus determined from the total number of jobs to be considered, N, and the priority constant, k.

This may be generalized. For L priority classes and with n_I being the mean number of jobs in the I^{th} priority class, n_I is given by:

$$n_I = \frac{k^{(L - I)} N}{\sum_{i=1}^{L} k^{(i-1)}} \quad . \qquad (2.2.8)$$

In simulations with L equal to three, k equal to six, and with N equal to 1500, the mean number of jobs generated for priority classes one, two and three are 1257, 208, and 35 respectively.

A further property of the job streams considered is that the mean interarrival times and the mean central processor (cpu) time requests of jobs of different priority classes are also related by the priority constant k:

$$r_2 = kr_1 \ , \qquad (2.2.9)$$

$$r_3 = kr_2 \ , \qquad (2.210)$$

and

$$m_2 = km_1 \ , \qquad (2.2.11)$$

$$m_3 = km_2 \ , \qquad (2.2.12)$$

where k is the priority constant and r_I and m_I are the mean interarrival time and mean cpu time request, respectively, for jobs of priority class I.

In general, then;

$$r_I = kr_{I-1} , \qquad (2.2.13)$$

$$m_I = km_{I-1} . \qquad (2.2.14)$$

The actual central processor times requested by jobs of priority class I are assumed to be normally distributed about the mean value m_I and the standard deviation is taken to be $m_I/4$.

The actual interarrival times of jobs j of priority class I are assumed to follow an exponential distribution with mean value r_I.

The normal and Poisson generating functions used are described in appendix A. The mean central processor time and interarrival time for jobs of priority class one are specified along with the priority constant at the beginning of each simulation model run. The two generating functions use these parameters to produce the interarrival times and central processor time request for every job in the job stream. The central processor time request constitutes the value t_j for each job j indicated at the beginning of this section.

Arrival times are determined from the interarrival times. The arrival time of the j^{th} job of priority class I, denoted A_{jI}, is determined from that of the last job of the same priority class given by, $A_{j-1,I}$, and the actual interarrival time of the job j, a_{jI}.

The latter is determined from the mean interarrival time for the priority class, r_I, with the Poisson generating function:

$$A_{jI} = A_{j-1,I} + a_{jI} . \qquad (2.2.15)$$

For the first job of each priority class I, the arrival time of the first job is arbitrarily:

$$A_{1I} = \frac{a_{1I}}{2} . \qquad (2.2.16)$$

Thus the arrival time of every job in the job stream is determined.

The assumed device time for job j, denoted d_j, is used to represent time spent in physically transfering the job into a waiting area within the computer system. Transfer from this waiting area to either the waiting queue or to storage is assumed instantaneous. This device time is simulated after the job arrival time, A_{jI}, but before the job is loaded into storage. Thus the earliest time when the job may be loaded into storage and when execution may begin is A'_{jI}, given by:

$$A'_{jI} = A_{jI} + d_j . \qquad (2.2.17)$$

The mean storage requirement, M, is considered to be a function of the central processor time requested by a job j, t_j, and hence:

$$M = M(t_j) . \qquad (2.2.18)$$

Since the mean central processor times are related to the priority classes, the mean storage requirement may be thought

of as being implicitly related to priority. There is, however, no explicit relationship between mean storage requirement of a job and the priority of a job. It is thus possible to determine the mean, M, of the distribution from which the storage requirement R_j is selected, as soon as the central processor time request has been generated.

It is assumed that the distribution of the storage requirements, with M as the mean, is normal and that the standard deviation is M/2.

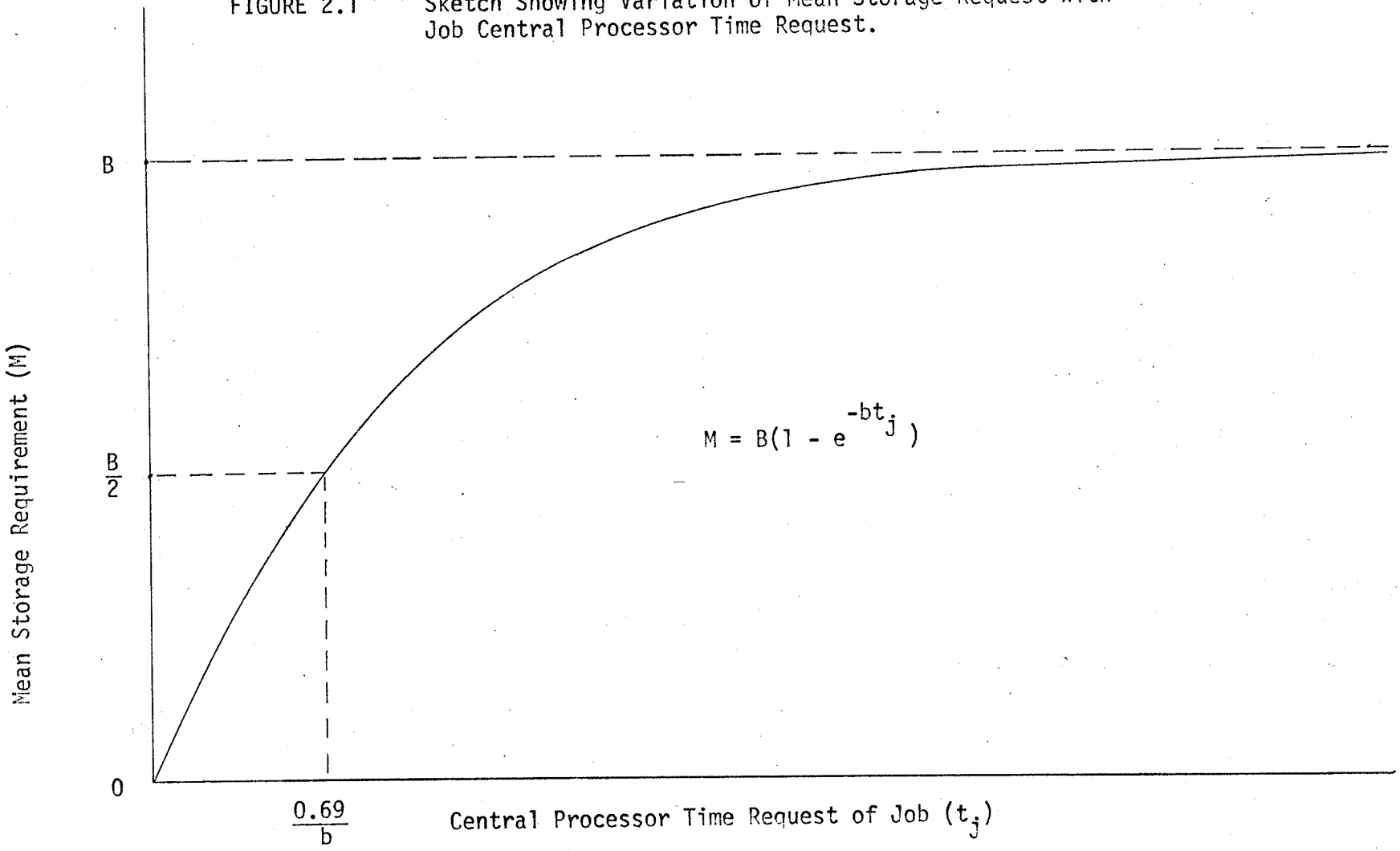
Data obtained from the University of Manchester (12) concerning storage requirements with their Atlas system indicates that the mean storage request for a job j with central processor time request t_j, denoted M_j, can be obtained from an equation of the form:

$$M = B(1 - e^{-bt_j}) . \qquad (2.2.19)$$

The general shape of this curve is shown in Figure 2.1. The parameter B determines the maximum mean storage requirement of all jobs, and the parameter b determines how rapidly the requirement rises to B with the central processor time request t_j. The same normal generating function used for central processor request is employed here, and is described in appendix A. The value generated for the storage request, denoted R_j for job j is in fractional form.

The main storage of some computers is broken

FIGURE 2.1 Sketch Showing Variation of Mean Storage Request With Job Central Processor Time Request.



down into manageable sections, such as pages (17) and these sections are indivisible when storage area is assigned. For this reason, the fractional storage request, R_j, generated by the method described above, is converted to the smallest integer not less than R_j before use in the model. Since every job is assumed to require some storage, R_j is a positive, non zero, integer.

In order that the assumed device time is proportional to the size of the job, the assumed device time is calculated from

$$d_j = R_j \; d \; . \qquad (2.2.20)$$

Where R_j is the integer value of storage requirement and d is an input parameter known as the device time number.

To recapitulate, the job-mix parameters input at the beginning of each simulation run are:

N , the total number of jobs in the job stream;

L , the number of priority classes;

k , the priority constant;

r_1, the mean interarrival time for priority class one jobs;

m_1, the mean cpu time request for priority one jobs;

d , the device time number;

B , the maximum mean storage request;

b , the rate at which mean storage requirements increase with central processor time.

From these job mix parameters, the arrival times, A_{jI}, the central processor time requests, t_j, the priority number I_j, the assumed device time, d_j, and the storage requirements, R_j, are determined for each job j in priority class I in the job stream. The jobs in each priority class I constitute a segment of the job stream, and have a mean central processor time request of m_I, and a mean interarrival time of r_I.

2.3 The Job Stream Generator

The job stream generator is that portion of the simulation model which uses the job-mix parameters and the normal and Poisson generating functions to generate the jobs and to make them available to the other portions of the simulation model as the simulation run proceeds.

Jobs are generated as job sequences or streams by priority class. A generation list is defined with one entry or slot for each priority class, which contains the next available job within the priority class referenced. When the simulated time of the model reaches the arrival time of one of the jobs in the generation list, the job concerned is taken out of the generation list and is referenced elsewhere in the model, leaving a slot free in the generation list. The job-stream generator is then invoked to fill the empty slot with the next job of the priority class concerned, following the methods outlined previously.

A flow chart of the job-stream generator operations

is shown in Figure 2.2.

After the N^{th} job has been generated, and placed in a slot in the generation list (the N^{th} job could be of any priority class), no further jobs are generated, ensuring that only the N jobs generated are processed by the simulation model.

A sample job stream is given in Appendix B.

2.4 Measurement of Performance - General Discussion

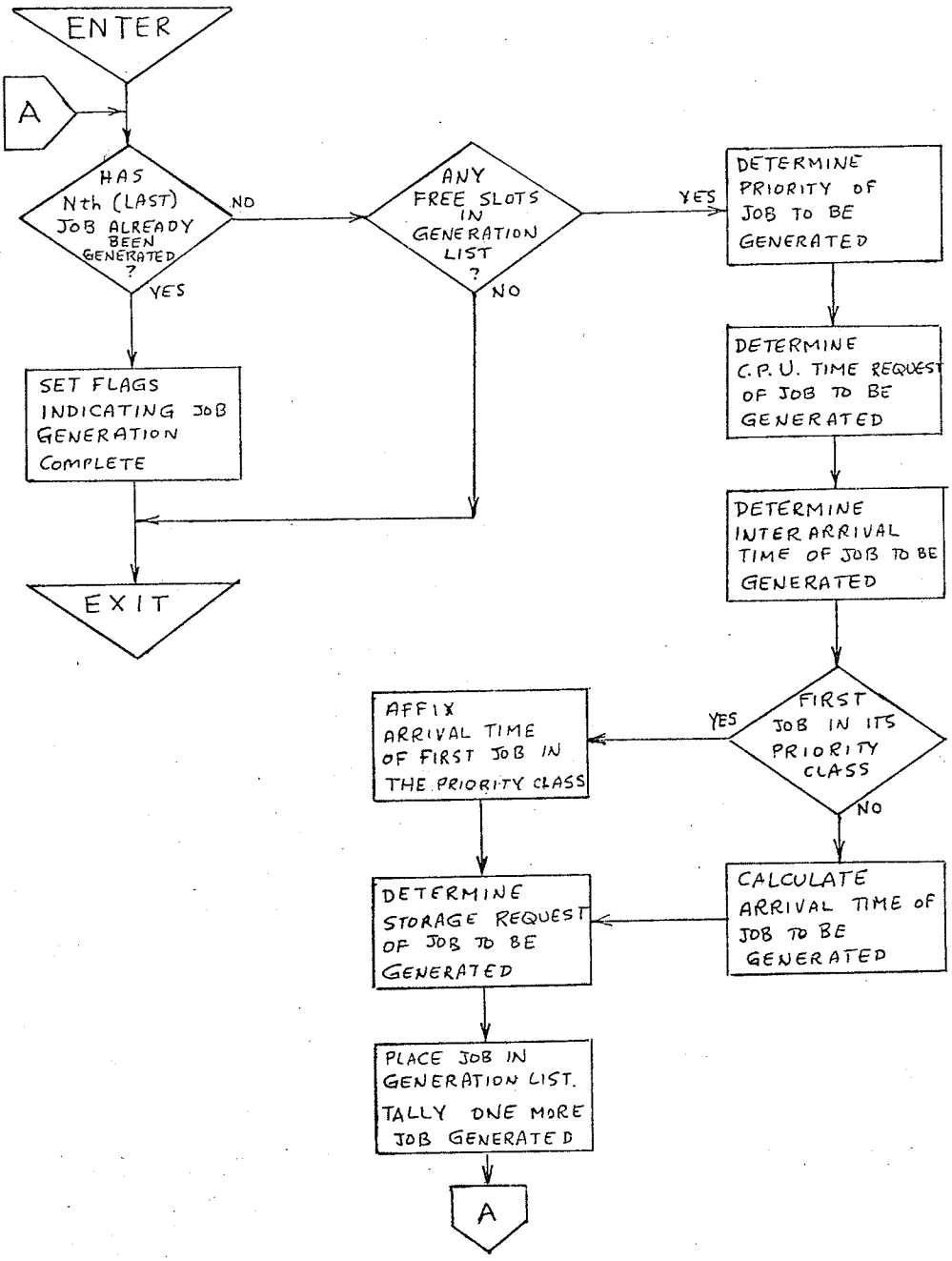
The criterion of performance used in these studies is based upon the principle of cost curves (2) and is very similar to the criterion used by Fife (10). It is the arithmetic mean value of functions of the relative response of each job passing through the simulator. Relative response, here, is the ratio of elapsed time since the beginning of a job, to the sum of the device time and central processor time devoted to the job. The function of relative response used in the measure of performance is a characteristic of the priority class of a job and it can be used to represent the cost of delays to jobs from each priority class.

2.5 Service to the Computer User

The duration of any job j is taken to be the total time elapsed between job arrival, A_{jI} and job completion T_j and is given by:

$$\Delta T_j = T_j - A_{jI} . \qquad (2.5.1)$$

FIGURE 2.2 Flow Chart of Job Generating Operation



The time during which the job j is in storage and is being executed is given by

$$T_j - A^*_{jI} \quad , \tag{2.5.2}$$

where, with d_j being the assumed device time, and with no time spent by the job in the waiting queue,

$$A^*_{jI} = A_{jI} + d_j \quad . \tag{2.5.3}$$

The minimum time which the job will be in storage will be its execution time, t_j. Thus, for minimum processing time:

$$T_j - A^*_{jI} = t_j \, , \tag{2.5.4}$$

or

$$A^*_{jI} = T_j - t_j = A_{jI} + d_j \, , \tag{2.5.5}$$

assuming no time spent in the queue ($A^*_{jI} = A'_{jI}$), and:

$$\Delta T_{j\min} = T_j - A_{jI} = t_j + d_j \, . \tag{2.5.6}$$

The time devoted to the job j, denoted τ_j, is defined at job completion as the sum of the assumed device and the central processor time given, t_j.

Thus

$$\tau_j = t_j + d_j \quad . \tag{2.5.7}$$

The lack of attention of job j at job completion, defined by :

$$\mu_j^* = \frac{\Delta T_j}{\tau_j} \quad . \qquad (2.5.8)$$

The minimum lack of attention at job completion is :

$$\mu_{j\,min}^* = \frac{\Delta T_{j\,min}}{\tau_j} = \frac{t_j + d_j}{t_j + d_j} = 1 \quad (2.5.9)$$

The lack of attention of a job j may be computed prior to job completion. At time T_j^1 during execution of job j,

$$A_{jI}^* \leqslant T_j^1 \leqslant T_j \quad , \qquad (2.5.10)$$

the lack of attention is given by :

$$\mu_j = \frac{T_j^1 - A_{jI}}{d_j + \text{cpu time given to job } j} \quad . \qquad (2.5.11)$$

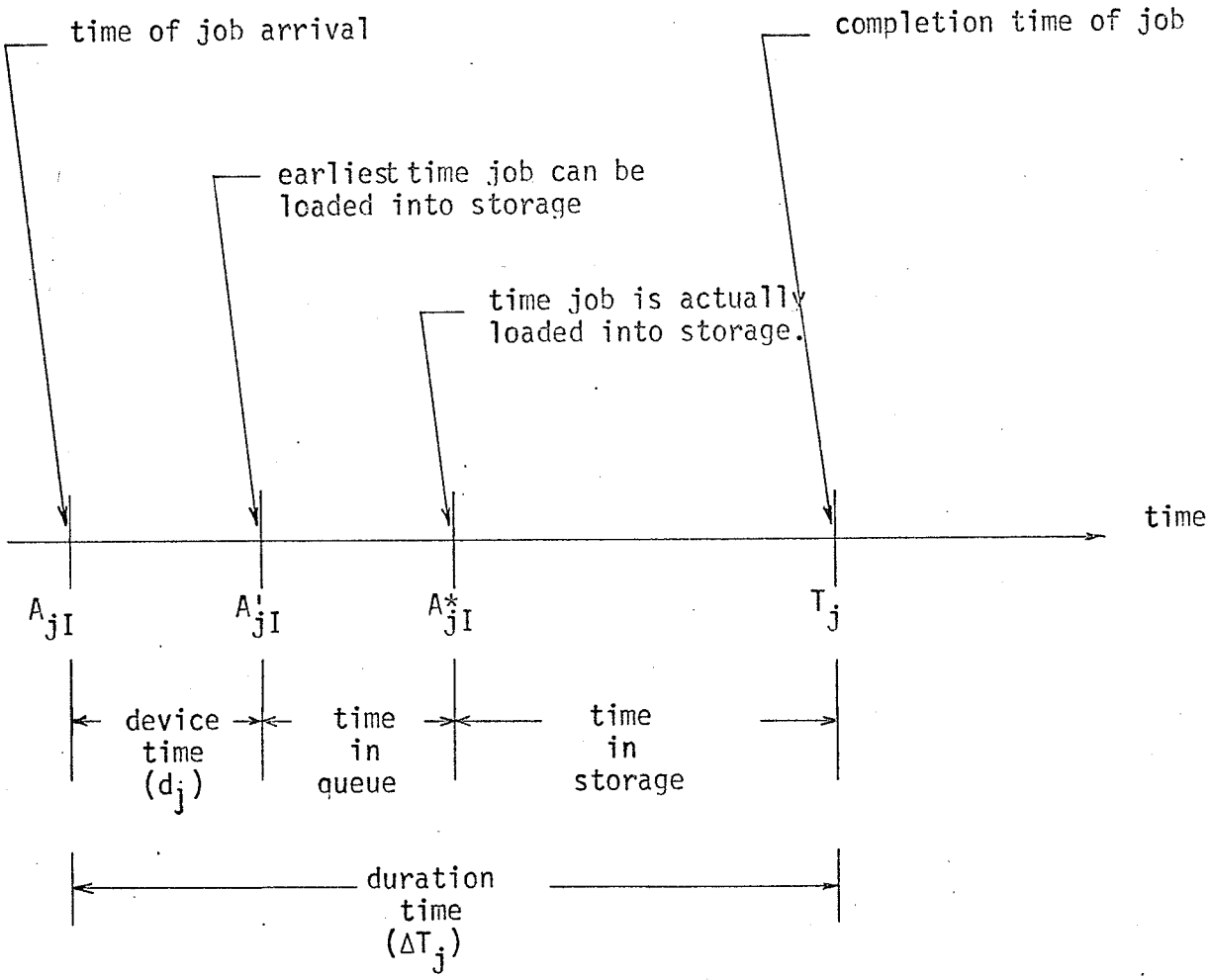
Immediately after loading job j and before any cpu time is given to job j, that is $\tau_j = d_j$, the lack of attention is (with $A_{jI}^* = A_{jI}'$):

$$\mu_j = \frac{A_{jI}^* - A_{jI}}{d_j} = \frac{d_j}{d_j} = 1 \quad . \qquad (2.5.12)$$

Thus lack of attention has a lower limit of one. Figure 2.3 shows diagrammatically the relationships presented here.

In the study of Chai (1) the penalty of a job j was taken to be a function of its lack of attention, μ_j, and its priority, I_j. The penalty function P was then:

FIGURE 2.3 Processing Times Diagram



For minimum duration time, $\Delta T_{j_{min}}$:

the time in queue = 0, i.e., $A'_{jI} = A^*_{jI}$;

the time in storage = t_j;

and the duration time, $\Delta T_{j_{min}} = d_j + t_j$.

$$p_j = P(u_j, I_j). \qquad (2.5.13)$$

The penalty function chosen in these studies is identically equal to the lack of attention. Thus job penalty has a lower limit of unity. The mean lack of attention (system penalty) is then :

$$P_N^* = \frac{1}{N}\sum_{j=1}^{N} \mu_j^* = \frac{1}{N}\sum_{j=1}^{N} \frac{\Delta T_j}{\tau_j} = \frac{1}{N}\sum_{j=1}^{N} \frac{T_j - A_{jI}}{t_j + d_j}, \qquad (2.5.14)$$

and also has a lower limit of unity. This limit is reached when every job is processed immediately to completion without interruption and represents optimal service to the user. Higher system penalties reflect poorer service to the group of users and hence a higher cost of delay.

2.6 Utilization of the System

In this research the utilization of the central processor and main storage are examined. The utility of each of these two resources is defined as the ratio of demand to supply for use of the resource. A utility of one (or one hundred percent) implies complete utilization of the resource, for demand less than or equal to supply.

From another point of view, utility of a resource may be thought of as the percentage of time during which the resource is actually being used by jobs in the job stream.

It is shown that under certain conditions the two approaches to utility produce the same value (appendix D).

To the computer manager a resource is hardware, and to the hardware he attaches a cost or value. Figure 2.4 shows a hypothetical curve relating cost of resources to number of resources. However, if the manager must give a discount for a cost of delay, or penalty to users when service is poor, this cost of delay will increase as the number of resources is reduced with a constant job stream. Figure 2.5 shows a curve expressing this relation and Figure 2.6 shows a superposition of these two cost curves, implying that a certain number of resources may be chosen to achieve minimum costs for a constant job stream. No attempt is made in this study to deal with specific examples of these, however. Of greater interest is the variation of cost of delay(penalty) with the number of resources.

The utility of a resource is inversely related to the number of resources for a constant job stream. Hence the utility can be used in place of the number of resources, for a constant job stream. Further, some properties of the cost of delay (penalty) as a function of utility can be predicted. For this reason utility, as defined, plays a large role in the investigation.

The utility of the central processor, denoted U_o, is defined as the ratio of time demand on the central processor to time supply of the central processor. These quantities can be determined from the job-mix parameters.

FIGURE 2.4 Sketch Showing Variation of Cost of Resources with Number of resources.

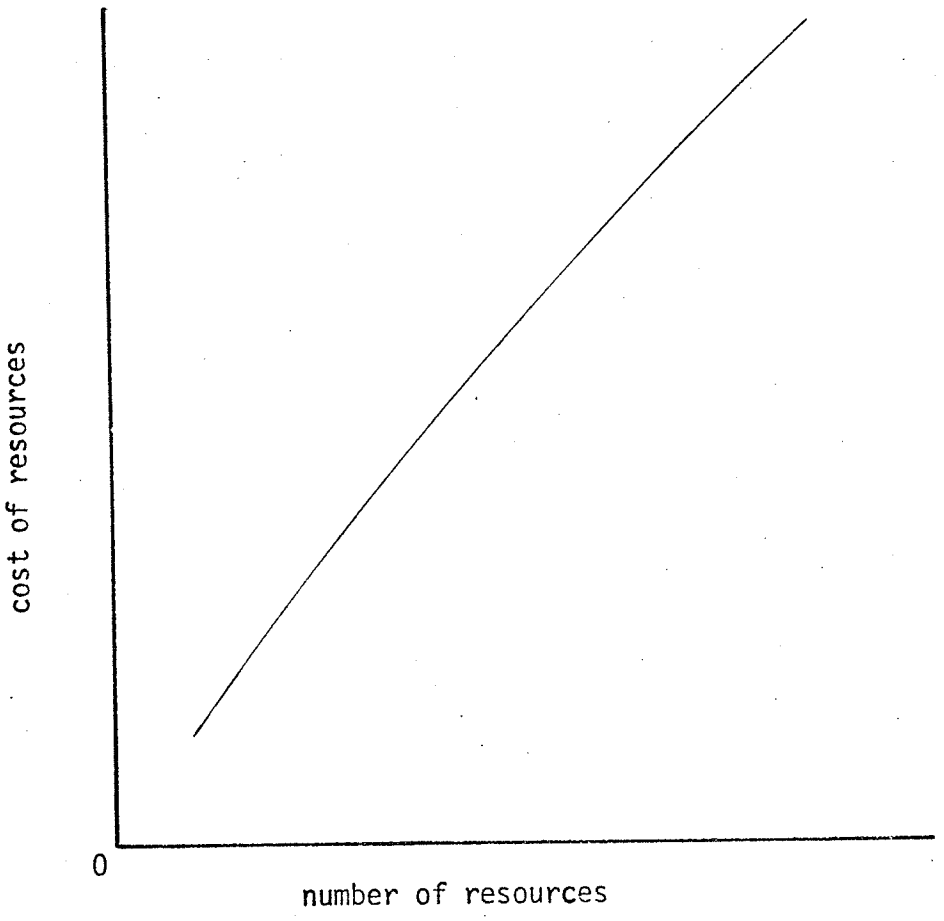


FIGURE 2.5 Sketch Showing Variation of Cost of Delay With Number of Resources

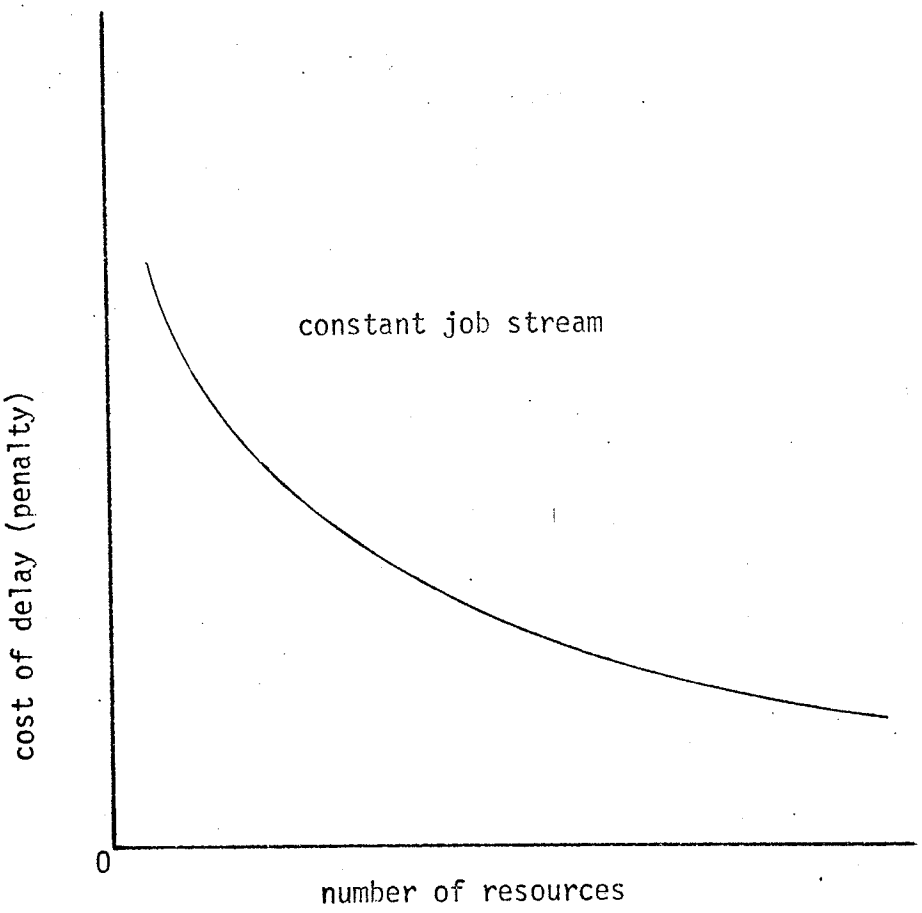
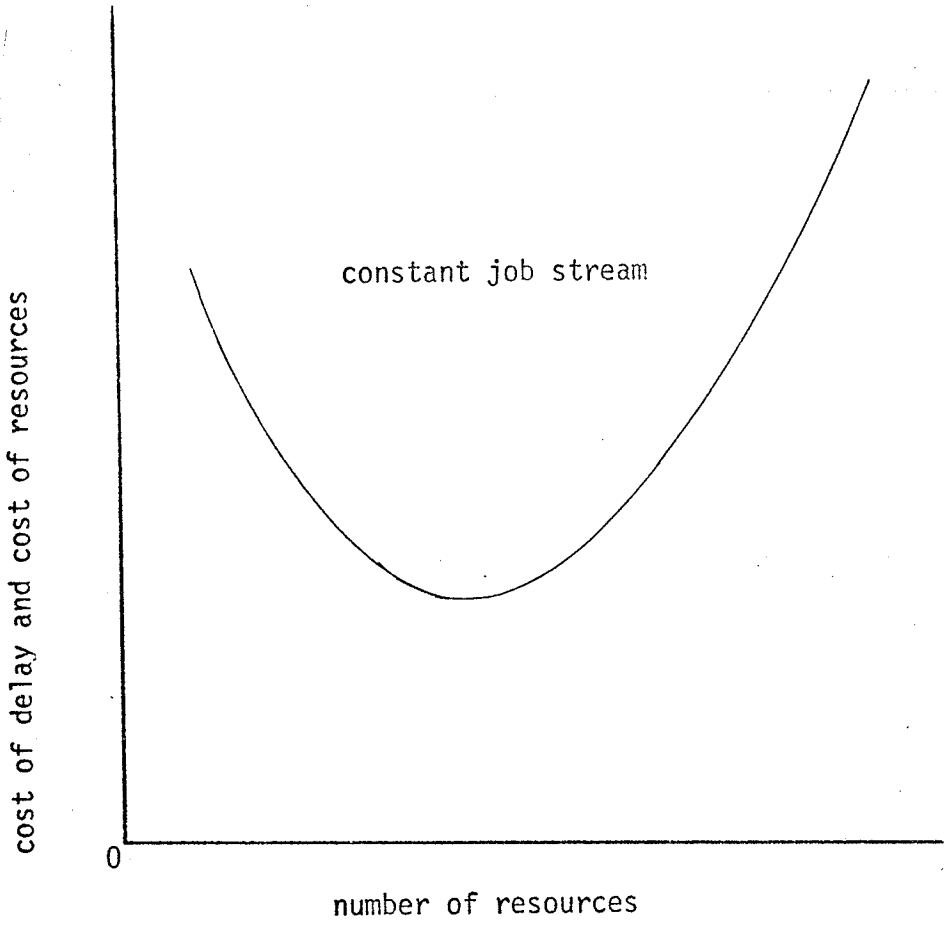


FIGURE 2.6 Sketch Showing Variation of Costs of Delay and Resources With Number of Resources.



The demand on the central processor is for execution time. At any time during the simulation run, the mean demand may be shown to be :

$$D_o = \frac{m_1}{r_1} L , \qquad (2.6.1)$$

where D_o is the demand for execution time per second; m_1, r_1 are the mean central processor time request and mean interarrival time for jobs of priority class one and L is the number of priority classes.

The amount of time which the central processor can allocate to jobs during one second is a time of one second less overheads incurred by the supervisor. If the overhead per second is much less than one second, the supply is approximately one.

Then

$$U_o \simeq \frac{\frac{m_1}{r_1} L}{1} = \frac{m_1}{r_1} L . \qquad (2.6.2)$$

It is to be noted that the cpu utility, U_o, reaches unity when

$$r_1 = m_1 L , \qquad (2.6.3)$$

and is less than unity when

$$r_1 > m_1 L . \qquad (2.6.4)$$

If saturation can be said to occur when U_o becomes unity, the last relationship is the necessary condition to avoid saturation.

The utility of main storage, denoted U_1, is defined as the ratio of demand on, to the supply of, main storage, with dimension [units.time].

For a job stream of N jobs ($j = 1...N$) each with execution time t_j and main storage requirement R_j, the demand, D_1, is:

$$D_1 = \sum_{j=1}^{N} t_j R_j \quad . \tag{2.6.5}$$

If the time required to process these N jobs is T and the available units of main storage is W_1, the supply, S_1, is:

$$S_1 = TW_1 \quad . \tag{2.6.6}$$

The utility of main storage, then, is:

$$U_1 = \frac{D_1}{S_1} = \frac{1}{TW_1} \sum_{j=1}^{N} t_j R_j \quad . \tag{2.6.7}$$

As in the case of central processor utility, the condition required to prevent saturation (or very heavy loading) of the main storage is that

$$TW_1 > \sum_{j=1}^{N} t_j R_j \quad . \tag{2.6.8}$$

In appendix D an equivalent relation for storage utility is derived, namely:

$$U_1 = \frac{\dot{m}_1 \bar{R}_M}{r_1 W_1}, \qquad (2.6.9)$$

where $\bar{R}_M$ is the sum, over the number of priority classes, of the mean storage requirement of jobs of each priority class (appendix D).

Another quantity related to storage utility is used in this research. This quantity is called storage occupancy, O_1, and represents the percentage of time when storage is unavailable to other jobs. For example, a job requesting two seconds of execution time and three sections of storage will have a contribution to utility of six. If this same job resides in storage for ten seconds before it is completed, the utility will remain the same, but the occupancy will be ten multiplied by three, or thirty, representing the resource time monopolized by the job.

If the time of arrival in storage of job j is A^*_{jI} and the completion time is T_j, the storage occupancy is defined by

$$O_1 = \sum_{j=1}^{N} \frac{(T_j - A^*_{jI}) R_j}{W_1 T} . \qquad (2.6.10)$$

In addition to the cpu utility U_0, the storage utility, U_1, the storage occupancy, O_1, and the system penalty P^*_N described, other data are produced at the end of each simulation.

The standard deviation of the final job penalties is determined from

$$\sigma_N = \sqrt{\frac{\sum_{j=1}^{N} (P_j^*)^2 - \frac{1}{N}\left(\sum_{j=1}^{N} P_j^*\right)^2}{N-1}}, \qquad (2.6.11)$$

and is output.

The simulated time for the simulation, T, is output as well.

2.7 Operation of the Simulation Model

A structured but variable job-stream as defined in Section 2.2, is run through the simulation model. This section describes the operation of the simulation model, the handling of jobs and the processing of the results of the simulation run.

The generation list, defined in Section 2.3 is a list of the next jobs to arrive, by priority class. When the simulated time reaches the arrival time of one of the jobs in the generation list, that job is transferred from the generation list to one of two other lists. The execution list, with W_0 slots, contains those jobs which are being executed and are being allocated central processor time. Jobs in the execution list have been assigned resources which remain assigned until the accumulated central processor time allocated equals the central processor time requested for the job. When job j, which occupies one slot in the execution list has received its requested cpu time t_j, the slot is made available to another job, as is storage

R_j . Contributions of job j to system penalty and the utilities and occupancies are stored for processing at the conclusion of the simulation run. Only those jobs occupying slots in the execution list receive central processor time, tie up resources, and can be processed to completion.

The other list to be defined is called the waiting queue, and has Q slots. If a job which has just become available for execution (at time A_{jI}') cannot immediately be placed in the execution list because no slot in the execution list is free or because insufficient storage is available, it is placed in the waiting queue for processing at some later time. As jobs in the execution list are completed, slots in the execution list are freed and resources are released. At such times the waiting queue is scanned for jobs which can be then accomodated. If such jobs are found they are transferred from the waiting queue to the execution list, freeing slots in the queue and filling slots in the execution list. These operations continue until all the N jobs in the job-mix are completed.

The transfer of a job from one list to another, that is from the generation list to the execution list, from the generation list to the waiting queue, from the waiting queue to the execution list, and removal upon completion, of jobs from the execution list, is a simple and straightforward operation. Appendix C lists the data kept for each job in each list.

Of more interest are the algorithms used for selecting jobs to be transferred from the waiting queue to the execution list, and for allocating central processor time to those jobs in the execution list. In this research four algorithms were studied for the former operation and a round robin variable time slicing algorithm similar to the one used by Chai (1) is used for allocation of central processor time.

A general flow diagram of the model used by Chai is shown in Figure 27. As seen in the diagram, operation of the model may be considered in terms of the following routines:

- a switching routine is used to determine whether the quantum allocation routine or the search routine is to be executed and selection is made depending upon which of the two was executed last and which of the three paths was taken on the previous cycle. The switch table is presented in Table 2.1. Vacant entries occur because only certain combinations of last routines and last path-numbers occur during execution of the simulator (1);
- the quantum-allocation routine, which allocates central processor time to the jobs currently in the execution list;
- the event control routine, which selects arriving jobs from the generation list, places them into either the execution list or waiting queue, selects jobs from the waiting queue for transfer to the execution list, and which includes supervisor overhead time, representing the time spent by the computer system handling the job-stream, allocating, accumulating and so on;

FIGURE 2.7 Flow Chart of Simulation Model Operation

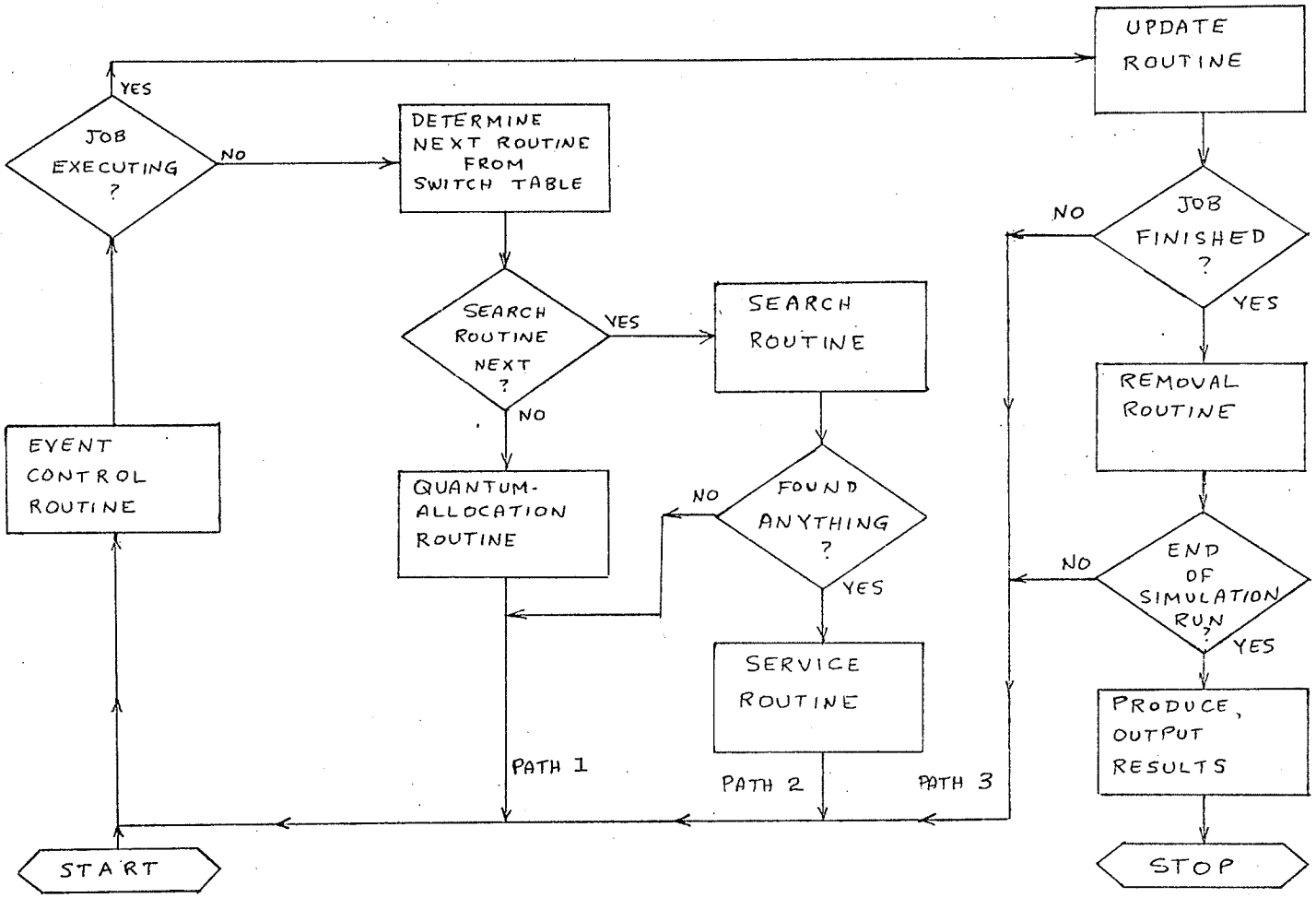


TABLE 2.1

SWITCH TABLE CONTAINING THE DESCRIPTION OF THE NEXT CYCLES

Last Primary Routine Executed	Path Number for the Previous Cycle		
	1	2	3
Search	Quantum - Allocation	-	Search
Quantum-Allocation	Search	-	-

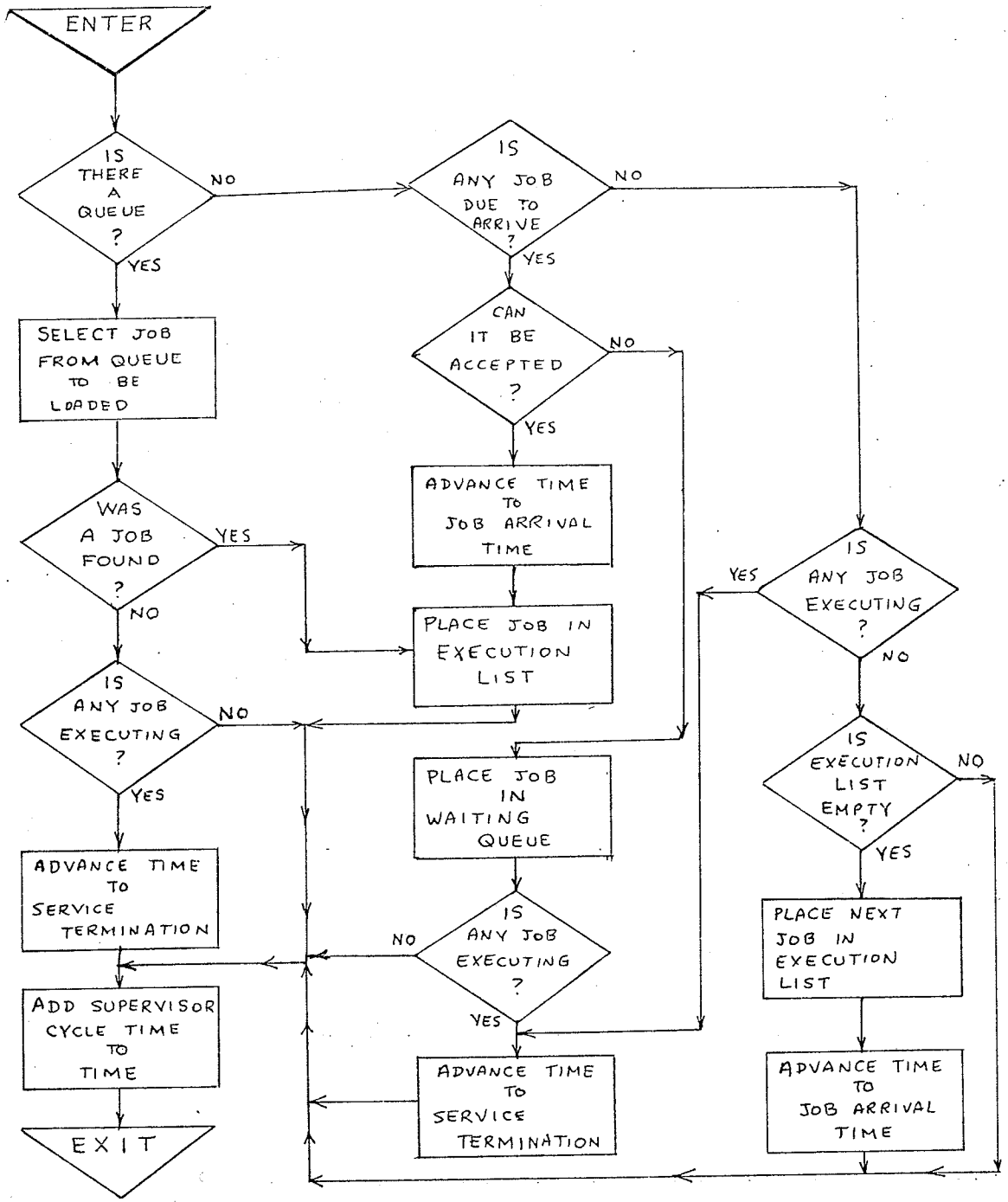
- the search routine, which determines the next job in the execution list to be serviced in the round robin cycle;
- the service routine, which administers the the central processor time allocated to the job found by the search routine;
- the update routine, which updates the accumulated central processor time received by the job serviced; and
- the removal routine, which removes a job from the execution list when its central processor requirements have been met and calculates the job contributions to system penalty and the utilities and occupancies.

Of these routines, the event control routine is of greatest interest. A flow chart of the event control routine used by Chai is shown in Figure 2.8. This diagram shows generally the logical operations required in handling arriving jobs, the waiting queue, and shows the inclusion of an overhead time for supervisor operations. It was considered by Chai that a time of 0.01 seconds for a supervisor cycle, representing about three thousand instructions on an IBM 360/65 computer was realistic. This same time is used in this research, and the parameter for supervisor cycle time is denoted T_{sc}.

2.8 Allocation of Central Processor Time - The Variable Time Slice

Central processor time is allocated to jobs in the

FIGURE 2.8 Flow Chart of Event Control Routine



execution list on a round robin basis. A round robin time of T_R is defined such that each job in the execution list receives some execution time every T_R seconds, approximately. Interruptions occurring from time to time during a job's time-slice cause the supervisor to incur overhead, resulting in the cycle time being slightly greater than T_R. Also, a job could be completed during its time slice, resulting in a slight decrease in the cycle time. No account is taken of the fact that a job is close to completion when the job's time slice is being calculated.

Slots in the execution list are considered as resources. Each job requires one slot in the execution list, of which W_o are initially available.

The variable time slice algorithm used allocates a time slice to job j, T_{Rj} according to the equation

$$T_{Rj} = \frac{g_j T_R}{\sum_{i=1}^{s} g_i} \quad \text{for} \quad T_{Rj} \geqslant T_{RM} , \tag{2.8.1}$$

and

$$T_{Rj} = T_{RM} \quad \text{for} \quad \frac{g_j T_R}{\sum_{i=1}^{s} g_i} < T_{RM} , \tag{2.8.2}$$

where s is the number of jobs being simultaneously processed (occupying slots in the execution list).

The quantity T_{RM} is called the minimum time slice and represents the value of a time slice below which overhead becomes crucial. In this research T_{RM} is taken to be $0.01\ T_R$ (1).

The quantity g_j is an augmented lack of attention, defined by :

$$g_j = \frac{\Delta T_j + d_j}{\tau_j} = \mu_j + \frac{d_j}{\tau_j} . \quad (2.8.3)$$

Since d_j is proportional to the size of the job, the algorithm favors larger jobs by giving them larger time slices than jobs with smaller storage requirements, all other things being equal.

2.9 Resource Handing in the Simulation Model

At the beginning of each simulation the parameter W_1, representing the number of storage sections available, is input. During the simulation run a secondary quantity, W_t is maintained which represents the number of storage sections available at any time t during the run.

When some job j is to be tested to see if it can be loaded, into core, two conditions must be satisfied. A slot in the execution list must be free, ie:

$$W_o - s \geq 1 , \quad (2.9.1)$$

and sufficient storage must be available::

$$W_t \geq R_j . \quad (2.9.2)$$

If these conditions are satisfied, the job is considered as a possibility for acceptance into the execution list. The final selection of one job from the list of possibilities is made by applying the logic of the queue selection algorithm. Upon acceptance of a job j, s is incremented by one and R_j is subtracted from W_t.

When some executing job j has received its full central processor time, it is removed from the execution list, s is decremented by one, and R_j is added to W_t. Thus a slot is now available for another job and R_j more storage sections are available as well.

Upon job completion the contributions to storage utility and occupancy are stored - the products $t_j R_j$ and $(T_j - A_{jI}^*) R_j$ respectively.

2.10 Algorithms for Selection of Jobs from the Waiting Queue

Jobs which arrive and cannot be immediately accomodated are placed in the waiting queue as described in Section 2.7. When the waiting queue is full and a job is scheduled to arrive, the arriving job is left in the generation list until a slot in the waiting queue is free, at which time that job is placed in the waiting queue. For this study, the queue length was set at twenty. The queue was found to fill up only as the utilities approached unity,

that is, as

$$r_1 \rightarrow m_1 L \ , \tag{2.10.1}$$

or as

$$TW_1 \rightarrow \sum_{j=1}^{N} t_j R_j \ . \tag{2.10.2}$$

For utilities less than 0.60, the number of waiting jobs did not exceed fifteen.

If there is only one job in the waiting queue which can be accomodated in the execution list and by the currently available resources, the task of selection is trivial. If, however, there are two or more jobs which could be accomodated, some algorithm must exist to decide which one is to be transferred from the waiting queue to the execution list, as mentioned in Section 2.9.

The four queue selection algorithms included in the model are:

1. First - come - first - served:

   selection of job i such that

$$A^{*}_{jI_i} \leqslant A^{*}_{jI_j} \ , \tag{2.10.3}$$

   for j equal to each member of the waiting list which can be accomodated ;

2. First - come - first - served - by - priority - class:

   selection of job i such that

$$I_i \leqslant I_j \ , \tag{2.10.4}$$

   for j equal to each member of the waiting list except i;

and

$$A^{*}_{iI_i} \leqslant A^{*}_{jI_j} , \qquad (2.10.5)$$

for all j with $I_j = I_i$;

3. Highest - penalty - first :

selection of job i such that

$$P_i \geqslant P_j , \qquad (2.10.6)$$

for j equal to each member of the waiting list which can be accomodated except i;

4. Shortest - job - with - highest - storage - requirements - first:

selection of job i such that

$$\frac{1}{t_i(W_t-R_i+1)} \geqslant \frac{1}{t_j(W_t-R_j+1)} , \qquad (2.10.7)$$

for all j in the waiting list which can be accomodated except i.

For any of the four queue selection algorithms, only those jobs in the waiting list which are known to be possibilities for acceptance are tested. In the event that two jobs both satisfy the logic of the queue selection algorithm, the job tested first is accepted from the waiting queue.

An input parameter, A_s, is set to the number of the queue selection algorithm to be used in the simulation.

For instance, if jobs are to be selected by queue selection algorithm three, highest-priority-first, A_s is set to three.

2.11 Summary

This chapter has described the job stream, the operations of the simulation model, and the output statistics from the model.

The output statistics are:

system penalty $P_N^* = \frac{1}{N} \sum_{i=1}^{N} P_i^* = \frac{1}{N} \sum_{i=1}^{N} (\mu_i^*)$

cpu utility $U_0 = \frac{m_1}{r_1} L$

storage utility $U_1 = \frac{1}{TW_1} \sum_{i=1}^{N} t_i R_i = \frac{m_1}{r_1} \frac{\bar{R}_M}{\bar{W}_1}$

storage occupancy $O_1 = \frac{1}{TW_1} \sum_{i=1}^{N} (T_i - A_{iI}) R_i$

The job-mix parameters are:

number of jobs	N
number of priority classes	L
priority constant	k
mean interarrival time, priority one	r_1
mean cpu time, priority one	m_1
maximum mean storage constant	B
storage time constant	b
device time number	d

Simulation model operational parameters:

number of slots in execution list	W_o
number of slots in waiting queue	Q
round robin cycle time	T_R
minimum time slice	T_{RM}
supervisor Cycle Time	T_{sc}
storage Resource available	W_1
queue selection algorithm to be used	$A_s = 1,2,3$ or 4

These listed parameters are input at the beginning of each simulation. A job stream developed from the job-mix parameters is run through the simulation model, whose operations are a result of the operational parameters, and the indicated results are produced at the conclusion of the simulation run.

CHAPTER III

A STUDY TO DETERMINE A RELATIONSHIP BETWEEN LACK OF ATTENTION AND UTILITY

3.1 Introduction

Each simulation run produces a system penalty, a central processor (cpu) utility and a storage utility. If a series of simulation runs is made with a constant job stream and diminishing storage, it might be expected that the system penalty would increase, since the same jobs are being processed by a computer which is continually growing smaller. Or if a series of simulation runs is made with the interarrival times of the jobs monotonically decreasing, the system penalty might be expected to increase, since the same computer would have to process more and more jobs per unit of time.

From the definition of cpu and storage utilities,

$$U_o = \frac{m_1}{r_1} L ,$$

$$U_1 = \frac{1}{TW_1} \sum_{j=1}^{N} t_j R_j \quad \text{or} \quad U_1 = \frac{M_1}{r_1} \frac{\bar{R}_M}{W_1} , \text{ (Appendix D)},$$

it is clear that both U_o and U_1 will increase with decreasing interarrival time r_1, and that U_1 will increase with decreasing available storage, W_1.

It can then be suggested at this point, without proof, that system penalty increases with utility.

3.2 An Empirical Relation Between Lack of Attention and Utility

The penalty function $p_i = \not{U}_i$ is used throughout these studies. In this case the system penalty p_N^* becomes the mean lack of attention, $\not{U}_M$. In this section the mean lack of attention is related to the utilities in an empirical manner. The purpose of deriving and testing an empirical relationship between mean lack of attention, $\not{U}_M$ and the utilities U_o and U_1 is to try to establish the characteristics of the variation in a manner which is independent of the job-scheduling algorithm being used. It is thought that the observed variations of $\not{U}_M$ with U_o and U_1, for various job-scheduling algorithms, might be explained more simply by reference to the deviations from the derived empirical relationship.

Both central processor (cpu) and storage utilities, U_o and U_1 respectively, are defined as a ratio of mean demand to supply in the competition concerned. For the derivation of an empirical relationship, it is assumed that the mean lack of attention, $\not{U}_M$ results from such competitions as those for central processor time and for storage. Two stipulations on each competition are made:

1. As the demand approaches zero or the supply becomes infinite in any competition, the

contribution made by that competition to the mean lack of attention approaches zero;

2. As the supply approaches the demand, or vice versa, in any competition, the contribution made by that competition to the mean lack of attention approaches infinity.

These two stipulations are satisfied by a form:

$$\frac{1}{\frac{S}{D} - 1} \quad , \tag{3.2.1}$$

for each competition, where S is the supply and D is the demand. Since utility is defined as:

$$U = \frac{D}{S} \quad , \tag{3.2.2}$$

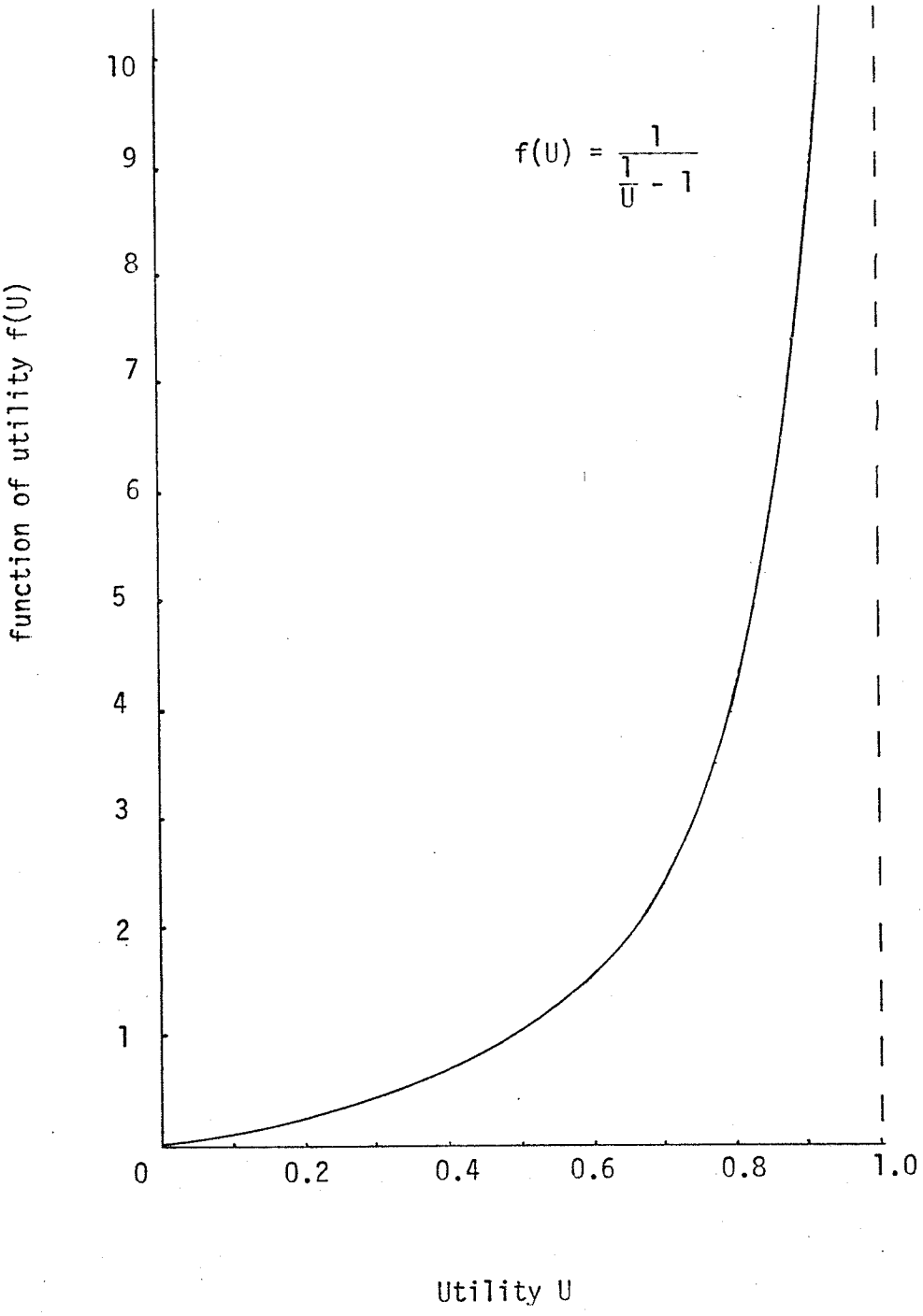
the form becomes

$$\frac{1}{\frac{1}{U} - 1} \quad . \tag{3.2.3}$$

This form (see Figure 3.1) is similar to one by Kleinrock (3).

It is assumed that the contributions made by the competitions for the different resources to the mean lack of attention are additive. An additive relationship is justified on the grounds that it is the simplest relationship allowing contributions from different competitions to be independent. Since lack of attention has a lower limit of one, the following equation may be written:

FIGURE 3.1 Graph Showing the Function of Utility, f(U), Used in the Empirical Relation.



$$\mu_M = 1 + C_o \sum_{j=1}^{K} \frac{1}{\frac{1}{U_j} - 1}, \quad (3.2.4)$$

for K competitions. The coefficient C_o is included to account in an empirical manner for contributions to the mean lack of attention, μ_M, made by parameters and effects not explicitly included in the definition of the utilities. In the particular study of this research two competitions, for cpu time and for storage, represented by utilities U_o and U_1 respectively, are considered:

$$\mu_M = 1 + C_o \left(\frac{1}{\frac{1}{U_o} - 1} + \frac{1}{\frac{1}{U_1} - 1} \right). \quad (3.2.5)$$

The assumptions and conditions governing the use of this relationship must be stated. A utility of one, or one hundred percent, represents the real situation in which jobs are demanding, on the average, everything within the particular competition. The lack of attention for some jobs may be very high, but the mean lack of attention will probably not be infinite. For a utility greater than unity, the waiting queue will build up without limit and the mean lack of attention will become very high. The relation predicts infinite lack of attention at utility identically equal to one. The mean lack of attention will rise toward infinity when $U_j = 1 + \epsilon$, for any competition j, where $\epsilon \simeq$ zero. The empirical relation is quite good, in this region.

The empirical relation, Equation (3.2.5) is probably dependent on the job-scheduling algorithm for heavily loaded simulations, that is with any utility approaching one, and is probably independent of the job scheduling algorithm for lightly loaded simulations, where all the utilities are much less than unity. Variation in the coefficient, C_o, may be used to account for any algorithm dependence and therefore C_o will not vary until high values of utility ($U_j \rightarrow 1$, any j) are reached.

When any utility is equal to unity, the waiting queue will build up very quickly. With a finite number of jobs, the mean lack of attention will not become infinite, but will, in fact be related to the number of jobs in the simulation, since the series of job penalties, p_i^* becomes time dependent as either the central processor utility, U_o, or the storage utility, U_1, reaches a value of unity. When any utility is equal to unity, mean lack of attention becomes a function of the number of jobs being processed.

3.3 Experimentation to Test the Derived Empirical Relation.

The relationship between mean lack of attention μ_M and the utilities, U_o and U_1 developed in the previous section, Equation (3.2.5), is of an empirical nature and its validity can be tested by comparison with simulated results.

The coefficient, C_o, may be determined for each simulation run from the values of μ_M, U_o, U_1, the mean lack of attention, the cpu and storage utilities respectively. Equation (3.2.5) may be rewritten:

$$C_o = \frac{\mu_M - 1}{\frac{U_o}{1 - U_o} + \frac{U_1}{1 - U_1}} , \qquad (3.3.1)$$

to show the calculation. From these values of C_o it is possible to test the validity of Equation (3.2.5) over a range of utilities.

It is desirable to use the same set of jobs in each simulation run for each study wherever possible. In the case where a job's storage request, R_j, exceeds that initially available, W_1, the storage request is reduced from R_j to W_1. The job will then monopolize storage while it is being processed.

The empirical relation is tested in three ways in this research. In the first investigation the interarrival times of jobs in the job stream are systematically varied, while everything else is held constant. In the second investigation, the number of units of available storage is systematically varied while the operational and the job stream parameters are held constant. In the third investigation

only the queue selection algorithms are varied.

The system penalty is the arithmetic mean of the final job penalties of the fifteen hundred jobs processed in each run and in these investigations is the same as the mean lack of attention. The reproducibility of this mean value is dependent upon the distribution of the final job penalties of the jobs processed. The range around the system penalty, P_N^* within which the population mean lies with 95 percent probability (for normally distributed final job penalty values) is approximately (11):

$$\pm \frac{2\sigma_N}{\sqrt{N}-\sqrt{2}},$$

where is the standard deviation of N final job penalty values.

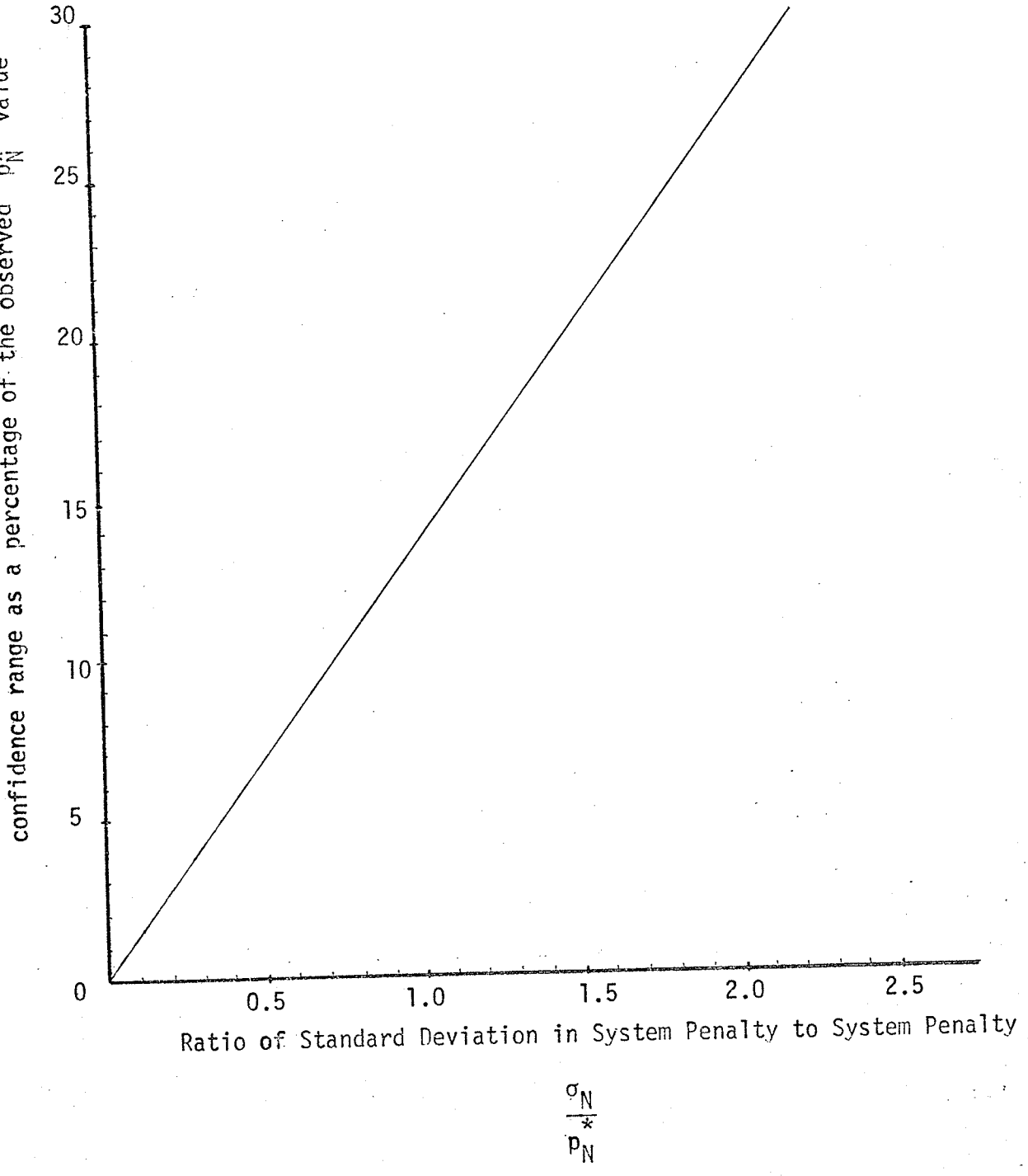
Because of certain amount of autocorrelation exists in the final job penalty values, a deviation from this normal situation occurs. This deviation has been found to be a function of the number of jobs, N, but for values of N greater than 50, the observed range around the system penalty within which the population mean lies has been found (11) not to exceed:

$$\pm \frac{5\sigma_N}{\sqrt{N}-\sqrt{2}}.$$

Figure 3.2 shows the reproducibility limit for various ratios of standard deviation to system penalty.

When the standard deviation in system penalty equals the system penalty, the range of confidence in the system

FIGURE 3.2 Confidence Range of System Penalty as a Function of the Ratio of the Standard Deviation in the System Penalty to the System Penalty with N = 1500



penalty is $\pm$ 13.5 percent. of the observed system penalty.

3.4 The Validity of the Empirical Equation As Interarrival Time is Varied.

This section describes and discusses two simulation studies conducted in each of which nothing but the interarrival time of arriving jobs in the job stream is varied between test runs.

For each test run a value of C_o is calculated from

$$C_o = \frac{\mu_M - 1}{\frac{U_o}{1 - U_o} + \frac{U_1}{1 - U_1}} \qquad (3.3.1)$$

Thus the empirical equation gives the value of μ_M actually observed. A constant value of C_o indicates that the empirical equation fits the simulated results exactly throughout the whole variation of U_o and U_1. It is to be noted that system penalty and mean lack of attention can be used interchangeable because of the penalty function used ($P_i = \mu_i$).

In the first of the studies described (3.4 - 1) in this section one priority class is considered. Job-mix and operational parameters held constant between simulation runs are listed in Table 3.1. The results obtained and the coefficient C_o calculated for each simulation run as the mean interarrival time for priority one jobs is varied from 100.0 to 1.0 seconds are given in Table 3.2. (See Appendix E.).

TABLE 3.1

SIMULATION MODEL PARAMETERS HELD CONSTANT FOR STUDY 3.4-1

JOB-MIX PARAMETERS:

Priority Constant	k	6.0
Number of Jobs	N	1500
Mean C.P.U. time, priority one	m_1 (seconds)	1.0
Mean Interarrival time, priority one	r_1 (seconds)	varied
Number of Priority Classes	L	1
Resource Request Size Constant	B	6.0
Resource Request Rate Constant	b	0.46

OPERATIONAL PARAMETERS:

Length of Execution List	S	10
Length of Waiting Queue	Q	20
Round Robin Cycle Time	T_R (seconds)	6.0
Mininum Time Slice	T_{RM} (seconds)	0.06
Supervisor Cycle Time	T_{SC} (seconds)	0.01
Device Time	d (seconds)	1.0
Queue Selection Algorithm	A_S	1
Storage Available	W_1	9

TABLE 3.2

RESULTS OF STUDY 3.4-1

Interarrival time r_1 (seconds)	Simulated Run Time (seconds)	System Penalty p_N *	Standard deviation, System penalty	C.P.U. Utility U_ox100	Storage Utility U_1x100	Storage Occupancy O_1x100	Calculated Coefficient C_o
100.0	141755.40	1.02	0.19	1.0	0.33	0.34	- *
8.0	11343.61	1.09	0.29	12.5	4.12	4.64	0.44
2.0	2840.20	1.48	0.69	50.0	16.44	27.13	0.40
1.8	2556.80	1.57	0.76	55.5	18.26	32.07	0.39
1.6	2273.40	1.68	0.82	62.5	20.53	39.68	0.34
1.4	1991.31	1.89	0.93	71.5	23.44	51.08	0.32
1.2	1709.95	2.55	1.36	83.4	27.30	71.40	0.30
1.1	1575.15	5.12	3.21	91.0	29.64	93.17	0.39
1.0	1543.75	21.99	11.85	100.0	30.24	97.34	0.00

* See Appendix E.

In the second study (3.4-2), two priority classes are considered. Table 3.3 lists the invariate job-mix and operational parameters, and Table 3.4 lists the results obtained and the coefficient C_o calculated for each value of mean interarrival time (for jobs of priority class one) as that time is varied from 100.0 to 2.0 seconds.

In both studies the first-come-first-served queue selection algorithm is used $(A_s = 1)$.

Figure 3.3 shows the coefficients C_o determined in these studies (3.4-1 and 3.4-2 respectively) as functions of the cpu utility, U_o, with a range determined from the standard deviation (Section 3.4). It is seen that for both studies, the coefficients C_o are approximately constant for :

$$U_o \leq 0.60 .$$

An approximate value of C_o for study 3.4-1 valid in the region indicated, would be:

$$C_o = 0.40,$$

and for study 3.4-2 :

$$C_o = 0.80.$$

Figures 3.4 and 3.5 show the mean lack of attention measured as a function of cpu utility and that predicted by the analytic relation for studies 3.4-1 and 3.4-2, with

TABLE 3.3

SIMULATION MODEL PARAMETERS HELD CONSTANT FOR STUDY 3.4-2

JOB-MIX PARAMETERS:		
Priority Constant	k	6.0
Number of Jobs	N	1500
Mean C.P.U. time, priority one	m_1 (seconds)	1.0
Mean Interarrival Time, priority one	r_1 (seconds)	varied
Number of Priority Classes	L	2
Resource Request Size Constant	B	6.0
Resource Request Rate Constant	b	0.46
OPERATIONAL PARAMETERS:		
Length of Execution List	S	10
Length of Waiting Queue	Q	20
Round Robin Cycle Time	T_R (seconds)	6.0
Minimum Time Slice	T_{RM} (seconds)	0.06
Supervisor Cycle Time	T_{SC} (seconds)	0.01
Device Time	d (seconds)	1.0
Queue Selection Algorithm	A_S	1
Storage Available	W_1	25

TABLE 3.4

RESULTS OF STUDY 3.4-2

Interarrival time r_1 (seconds)	Simulated Run Time (seconds)	System Penalty * p_N	Standard deviation, System penalty	C.P.U. Utility U_ox100	Storage Utility U_1x100	Storage Occupancy O_1x100	Calculated Coefficient C_o
100.0	121840.88	1.04	0.26	2.0	0.38	0.39	- *
8.0	9758.57	1.33	0.99	25.0	4.66	6.77	0.84
4.0	4912.38	1.92	1.29	50.0	8.95	14.41	0.82
3.0	3669.96	2.50	1.59	66.7	11.98	35.76	0.71
2.8	3429.56	2.68	1.68	71.5	12.82	41.58	0.63
2.6	3189.52	2.95	1.73	77.0	13.87	50.00	0.56
2.4	2960.66	3.47	1.81	83.5	14.85	62.16	0.48
2.2	2752.96	4.63	2.48	91.0	15.98	76.79	0.35
2.0	2583.24	9.54	5.72	100.0	17.02	97.06	0.00

* See Appendix E.

FIGURE 3.3 Variation of the Coefficient C_o with C.P.U. Utility U_o for Studies 3.4-1 and 3.4-2

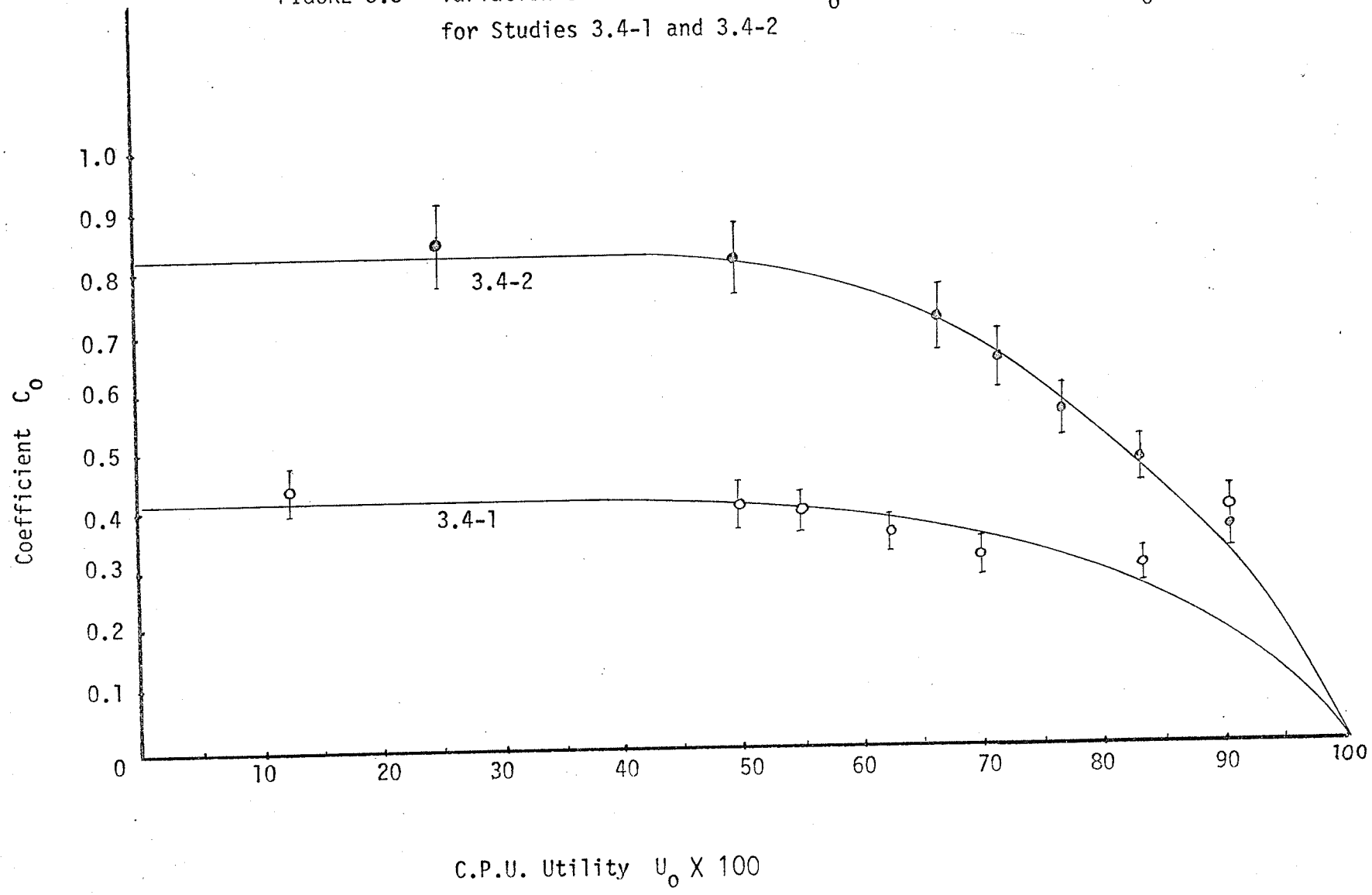


FIGURE 3.4 Variation of the Lack of Attention with C.P.U. Utility for Study 3.4-1

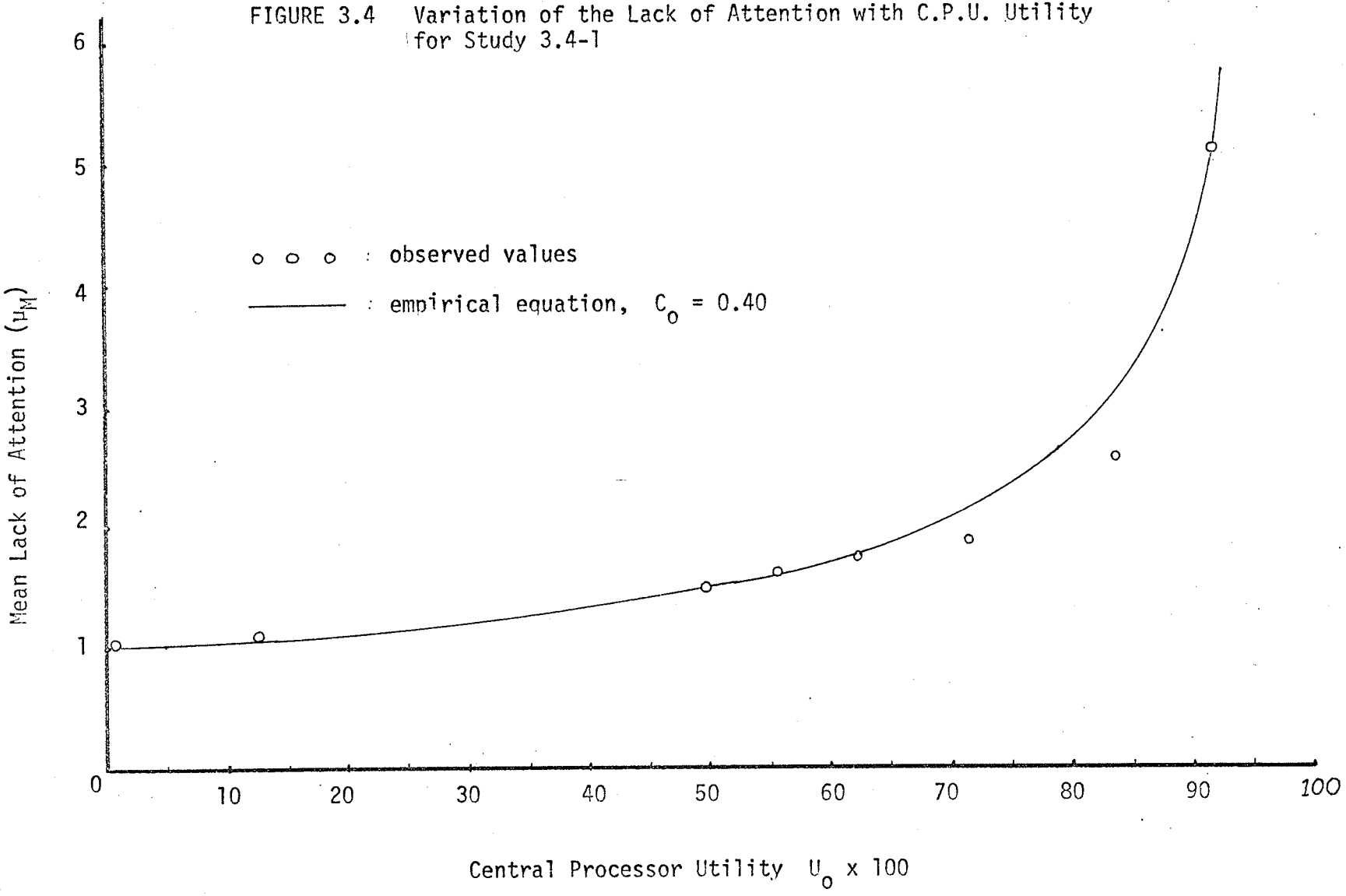
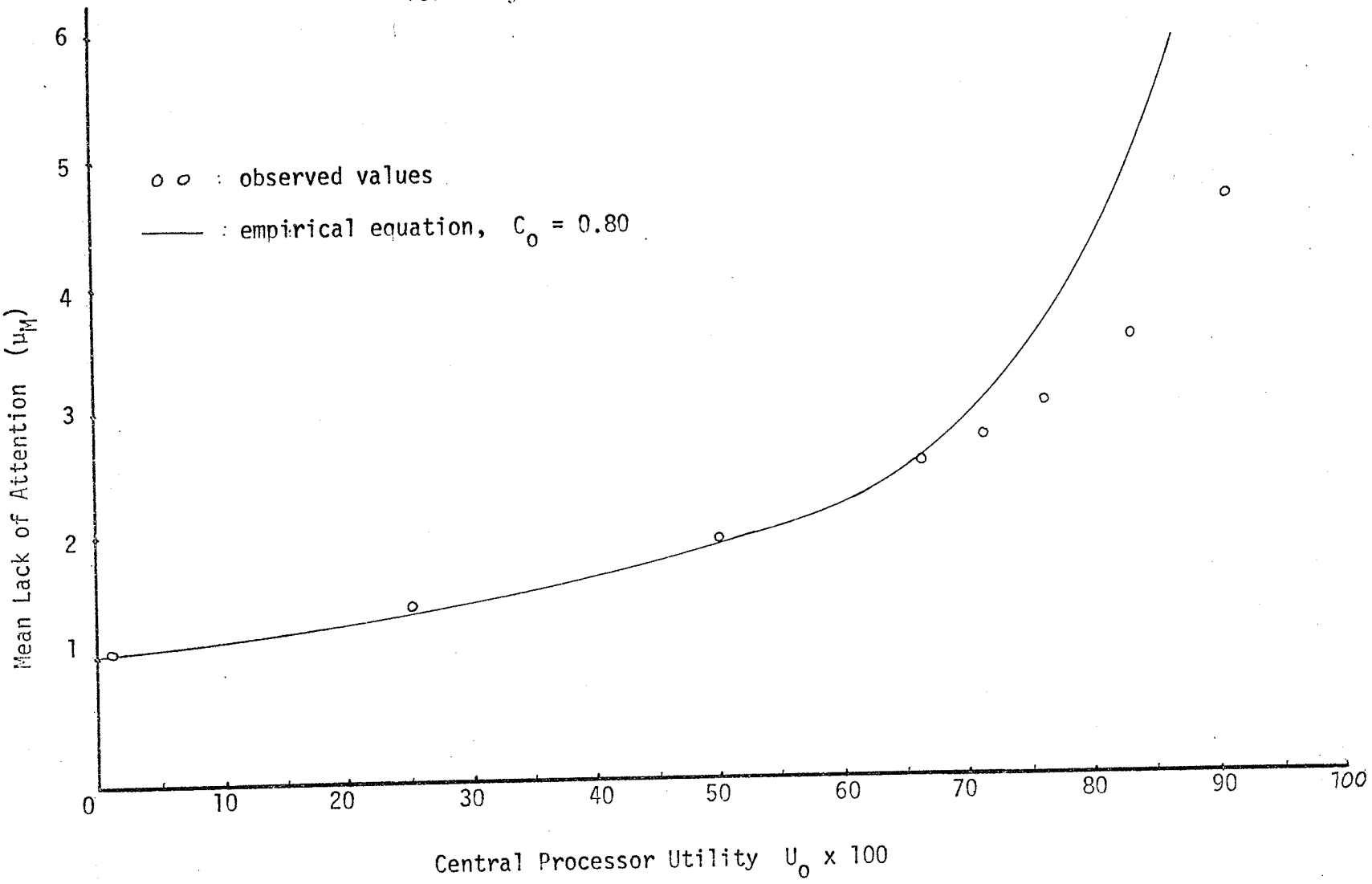


FIGURE 3.5 Variation of the Mean Lack of Attention with C.P.U. Utility for Study 3.4-2



$C_o =$ 0.40 and 0.80, respectively. From these curves it is seen that the analytic relation holds well for $U_o \leq$ 0.60, and there is a strong indication that in this range of cpu utility, performance characteristics of the simulation model are determinable from the utilities. The cpu utility is given from the input parameters r_1, m_1, and L :

$$U_o = \frac{m_1 L}{r_1} ,$$

and the storage utility, using the equivalent definition (appendix D), is given by :

$$U_1 = \frac{m_1 \bar{R}_M}{r_1 W_1} .$$

The value $\bar{R}_M$ may be determined from:

$$\bar{R}_M = \sum_{e=1}^{L} \bar{R}_I = \sum_{I=1}^{L} B(1 - e^{-bm_I}),$$

and W_1 is the number of units of storage available. Thus the utilities and therefore the mean lack of attention are determinable from input parameter values for the range of cpu utility indicated.

Figures 3.6 and 3.7 show mean lack of attention U_M related to the storage utility, U_1, for these two studies, with C_o equal to 0.40 and 0.80 respectively.

FIGURE 3.6 Variation of the Mean Lack of Attention with Storage Utility for Study 3.4-1

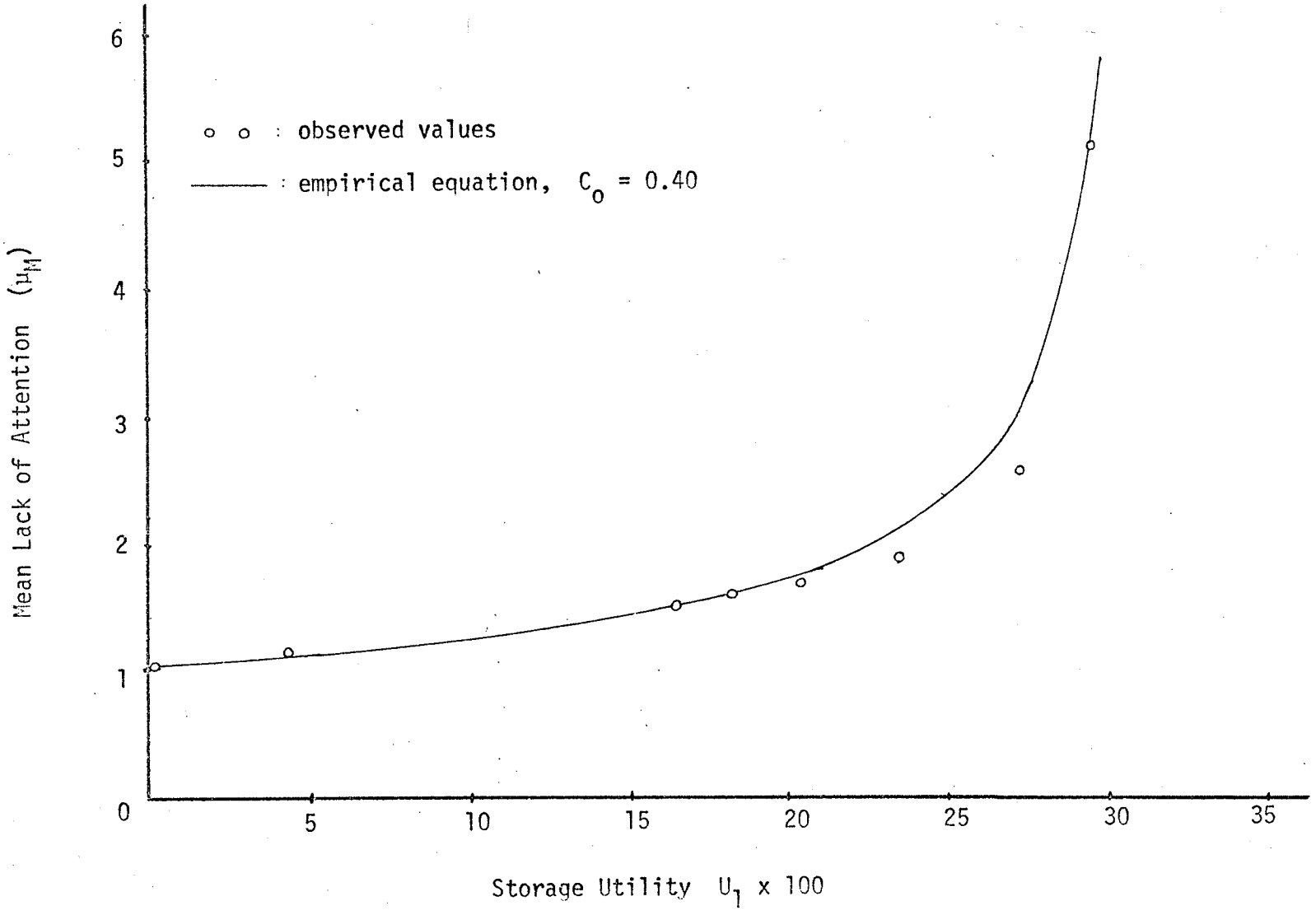
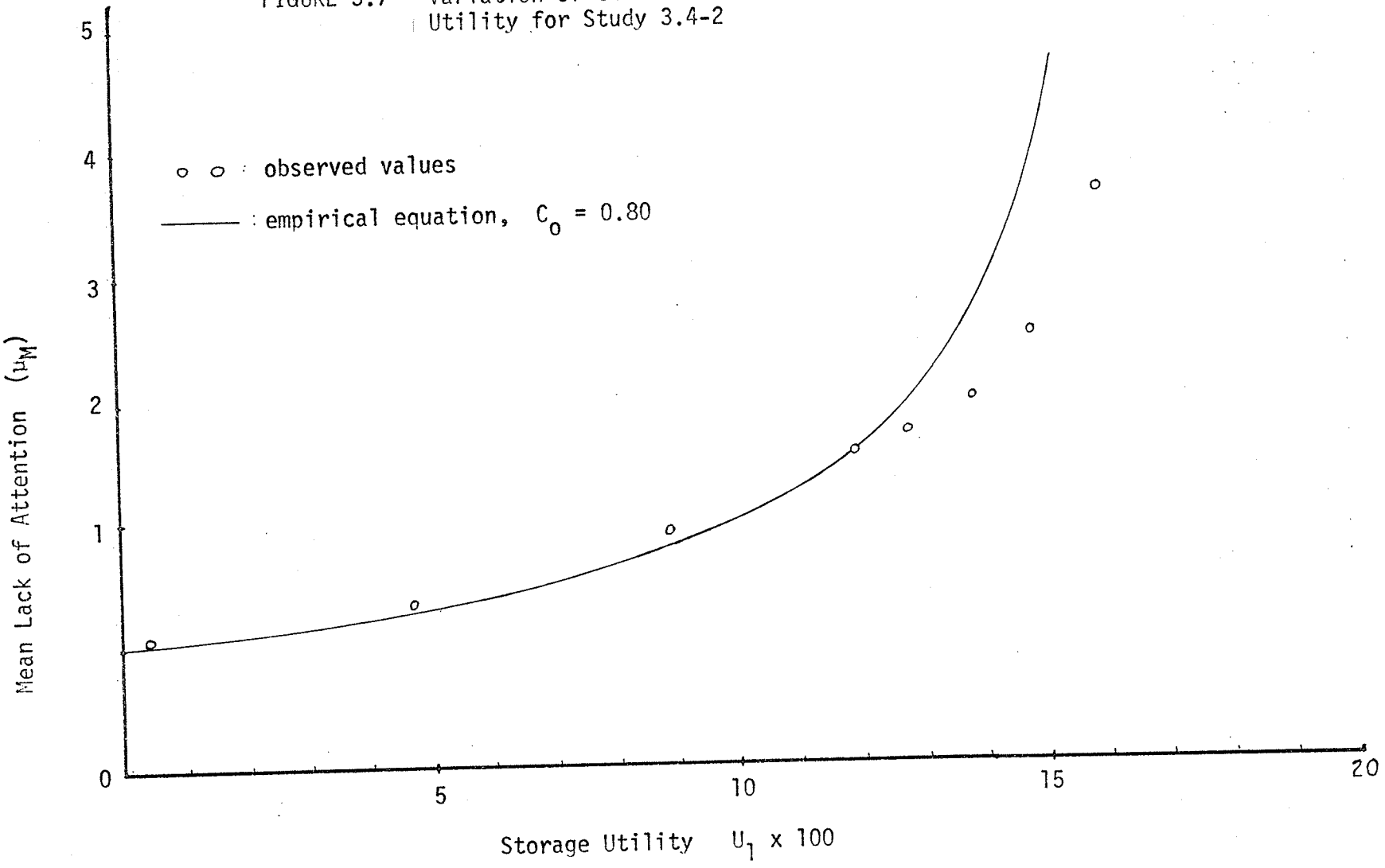


FIGURE 3.7 Variation of the Mean Lack of Attention with Storage Utility for Study 3.4-2



For cpu utility greater than 0.60 a variable coefficient C_o is required if the empirical equation is to fit the observed results. This suggests that the equation, and hence the coefficient C_o become functions of the job-scheduling algorithm as any of the utilities exceeds 0.60.

3.5 Validity of the Empirical Equation as Storage Size is Varied

This section describes and discusses two simulation studies conducted in each of which nothing but the size of the available storage (W_1) is varied between test runs.

As in the studies described in Section 3.4, a value of the coefficient C_o is calculated for each test run to make the empirical Equation (3.2.5) fit the observed results.

The only difference between the two studies of this section, labelled studies 3.5-1 and 3.5-2 respectively, is in the storage requests of each job in the job stream. In study 3.5-1, storage requests are determined from the input parameters B and b and from the cpu time request of each job j, in the manner described in section 2.2. In study 3.5-2 the storage request of every job in the job stream is identically one unit of storage.

The job-mix and operational parameters held constant during both studies 3.5-1 and 3.5-2 are given in

Table 3.5. The first-come-first-served queue selection algorithm is used $(A_s = 1)$.

The results obtained and the coefficient C_o calculated for each value of W_1 as W_1 is varied from 99 to 1 are shown for studies 3.5-1 and 3.5-2 in Tables 3.6 and 3.7 respectively. The range of confidence in the observed system penalty (mean lack of attention) varies from $\pm$ 8 percent to $\pm$ 20 percent in both studies 3.5-1 and 3.5-2, with the higher value occurring in both when $W_1 = 1$.

Figure 3.8 shows the coefficient C_o as a function of W_1 for studies 3.5-1 and 3.5-2. Two features of the curves shown are of interest, the extremely high value of C_o at $W_1 = 1$ and the difference in slope of the curves in the range of W_1 from 6 to 15. In Section 3.4 it was stated that the coefficient C_o could include the contributions to mean lack of attention of all parameters and effects not explicitly included in the definition of the utilities. The two effects which may explain the curves of figure 3.8 are supervisor overhead and long-job-loading, neither of which appear explicitly in the utilities.

With each job in the job stream requiring all of the storage available (See page 54), that is:

$$R_i = W_1, \quad \text{all jobs } i,$$

and with the first-come-first-served algorithm, from time to time a priority class three job will monopolize the

TABLE 3.5

SIMULATION MODEL PARAMETERS HELD CONSTANT FOR STUDY 3.5-1 and (3.5-2)

JOB-MIX PARAMETERS:

Priority Constant	k	6.0
Number of Jobs	N	1500
Mean C.P.U. time, priority one	m_1 (seconds)	1.0
Mean Interarrival time, priority one	r_1 (seconds)	6.0
Number of Priority Classes	L	3
Resource Request Size Constant	B	6.0 (0.0)
Resource Request Rate Constant	b	0.46 (0.0)

OPERATIONAL PARAMETERS:

Length of Execution List	S	10
Length of Waiting Queue	Q	20
Round Robin Cycle Time	T_R (seconds)	6.0
Minimum Time Slice	T_{RM} (seconds)	0.06
Supervisor Cycle Time	T_{SC} (seconds)	0.01
Device Time	d (seconds)	1.0
Queue Selection Algorithm	A_S	1
Storage Available	W_1	varied

TABLE 3.6

RESULTS OF STUDY 3.5-1

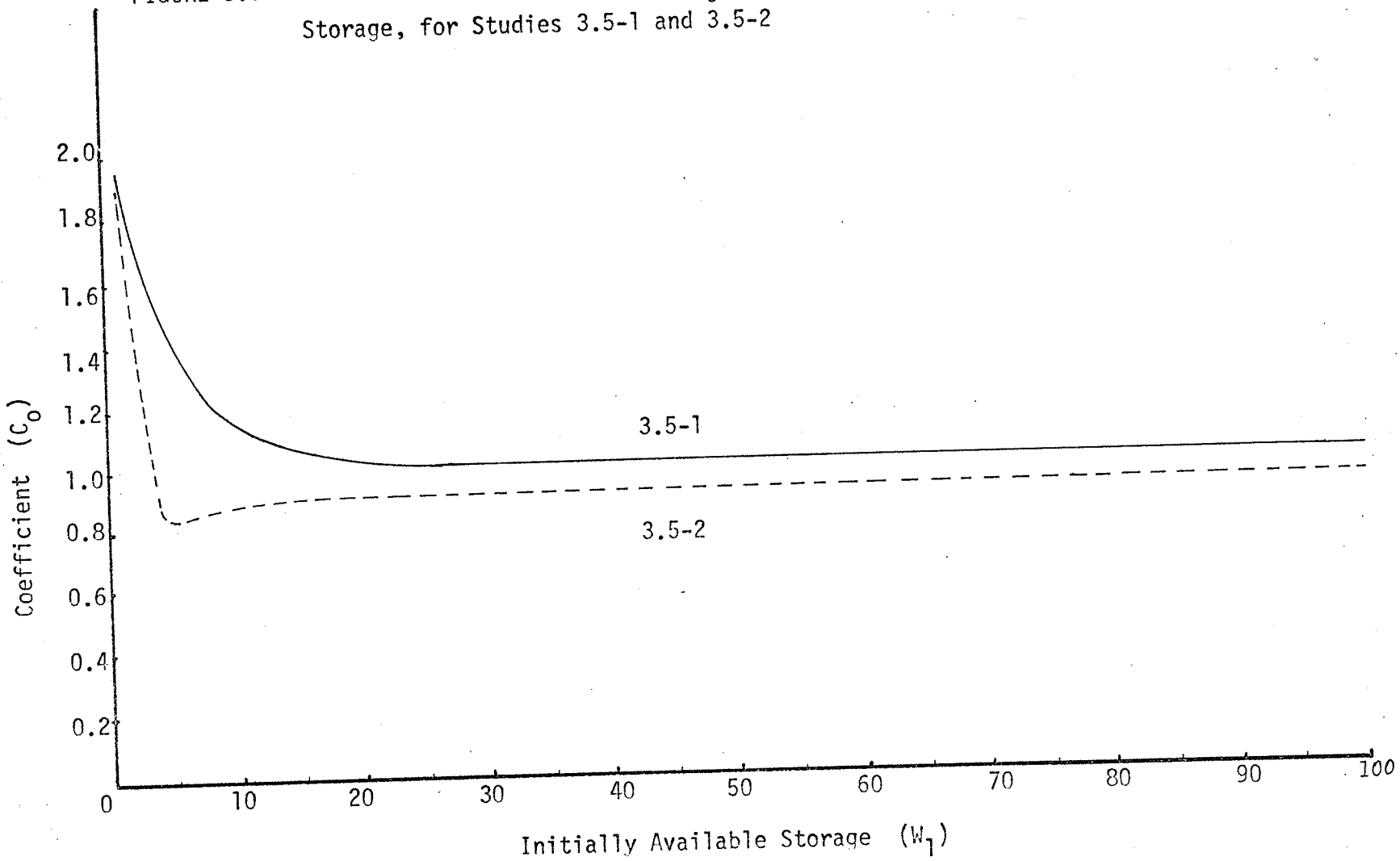
Available Storage W_1	Simulated Run Time (seconds)	System Penalty p_N^*	Standard deviation, System penalty	C.P.U. Utility U_ox100	Storage Utility U_1x100	Storage Occupancy O_1x100	Calculated Coefficient C_o
99	7442.31	2.05	1.37	50.0	2.87	6.95	1.00
50	7442.31	2.06	1.37	50.0	5.68	13.77	1.00
25	7442.31	2.13	1.79	50.0	11.37	25.28	1.00
20	7442.31	2.17	1.87	50.0	14.21	29.43	1.01
15	7442.31	2.31	2.33	50.0	18.94	33.66	1.06
10	7442.31	2.58	2.83	50.0	28.06	38.83	1.14
8	7442.31	3.01	3.73	50.0	33.37	41.49	1.35
7	7442.31	3.25	4.29	50.0	36.26	43.09	1.44
6	7442.31	3.13	3.83	50.0	39.38	44.78	1.30
1	7442.31	5.25	9.43	50.0	53.60	54.49	1.95

TABLE 3.7

RESULTS OF STUDY 3.5-2

Available Storage W_1	Simulated Run Time (seconds)	System Penalty P_N^*	Standard deviation, System penalty	C.P.U. Utility $U_o x100$	Storage Utility $U_1 x100$	Storage Occupancy $O_1 x100$	Calculated coefficient C_o
99	7641.01	1.92	1.20	50.0	0.52	1.14	0.92
50	7641.01	1.92	1.20	50.0	1.02	2.25	0.91
25	7641.01	1.92	1.20	50.0	2.05	4.51	0.90
20	7641.01	1.92	1.20	50.0	2.56	5.63	0.90
15	7641.01	1.92	1.20	50.0	3.41	7.51	0.89
10	7641.01	1.92	1.20	50.0	5.12	11.27	0.87
8	7641.01	1.92	1.21	50.0	6.39	14.09	0.86
7	7641.01	1.92	1.20	50.0	7.31	16.10	0.85
6	7641.01	1.91	1.19	50.0	8.53	18.60	0.83
1	7641.01	4.78	7.09	50.0	51.16	52.01	1.89

FIGURE 3.8 Variation of the Coefficient C_o with the Intitally Available Storage, for Studies 3.5-1 and 3.5-2



computer, holding up shorter jobs of higher priority (lower priority class number). This will result in a higher mean lack of attention and hence higher coefficients C_o. For study 3.5-1, where the mean storage demand of all jobs (mean R_i) can be shown to be about five units or sections*, the effect will persist until W_1 is equal to at least five, perhaps ten. The coefficient C_o calculated for lower values of W_1 in study 3.5-1 will be higher than one might otherwise expect. It is not unreasonable to suggest that the effect will fall off not abrubtly, but gently with W_1. The observed effect does not disappear until W_1 is approximately 15 (see Figure 3.8) that is, three times the mean storage request.

In the case of study 3.5-2, with the mean storage requirement equal to one, and by the argument above, the value of the coefficient at $W_1 = 1$ should be the same as that for study 3.5-1, and the effect of long-job-loading should disappear at three times the mean storage request, or at $W_1 = 3$. From the curve for study 3.5-2 in Figure 3.8, it is seen that the coefficients at $W_1 = 1$ are very close for the two studies and that for study 3.5-2 the coefficient falls off very rapidly and reaches a minimum in the neighborhood of $W_1 = 5$. This does not justify the preceeding argument, but does help to support it.

Supervisor overhead is almost directly proportional to the number of jobs being simultaneously processed (1). It

---

* the mean storage requirement is approximately given by

$$\text{Mean } R_i = \frac{1}{N} \sum_{I=1}^{L} n_I \cdot B \left(1 - e^{-bm_I}\right)$$

is reasonable to suggest that higher overheads will cause the system penalty to increase, and as a result, the coefficients C_o, to increase, since overhead reduces the amount of cpu time available for the fulfilment of requests by jobs for cpu time. Very simply, then, the more jobs in storage being executed simultaneously, the higher the coefficient C_o, all other things being equal. With R_i equal to one for all jobs i, the number of jobs that can be simultaneously executed is related directly to the number of units or sections of storage, W_1. This may account for the positive slope in the curve for study 3.5-2 from $W_1 = 6$ to $W_1 = 10$.

Occupancy (Equation 2.6.10) may be used to show some effects of space sharing in study 3.5-2, where $T = 7641.01$ (Table 3.7) and $R_j = 1$ for all jobs. The total time spent in storage by jobs is then given by the product $(7641.01\ O_1\ W_1)$ for each run. With $W_1 = 1$, the product is about 3,980 seconds, representing serial processing. With $W_1 \geqslant 7$, the product is constant at about 8,600 seconds (see Table 3.7), implying a maximization of space sharing benefits for this study. Therefore, at most seven jobs are processed simultaneously, with a mean of about two (86/39.8) when $W_1 \geqslant 7$.

From Tables 3.6 and 3.7, and from Figure 3.8, the constant values of the coefficient C_o may be taken as 1.00 for study 3.5-1 and 0.90 for study 3.5-2. The range of confidence in these values is at least 8 percent (Tables 3.6 and 3.7, with Section 3.4). Thus a coefficient of 0.95 may be used for both studies, since it lies within the range

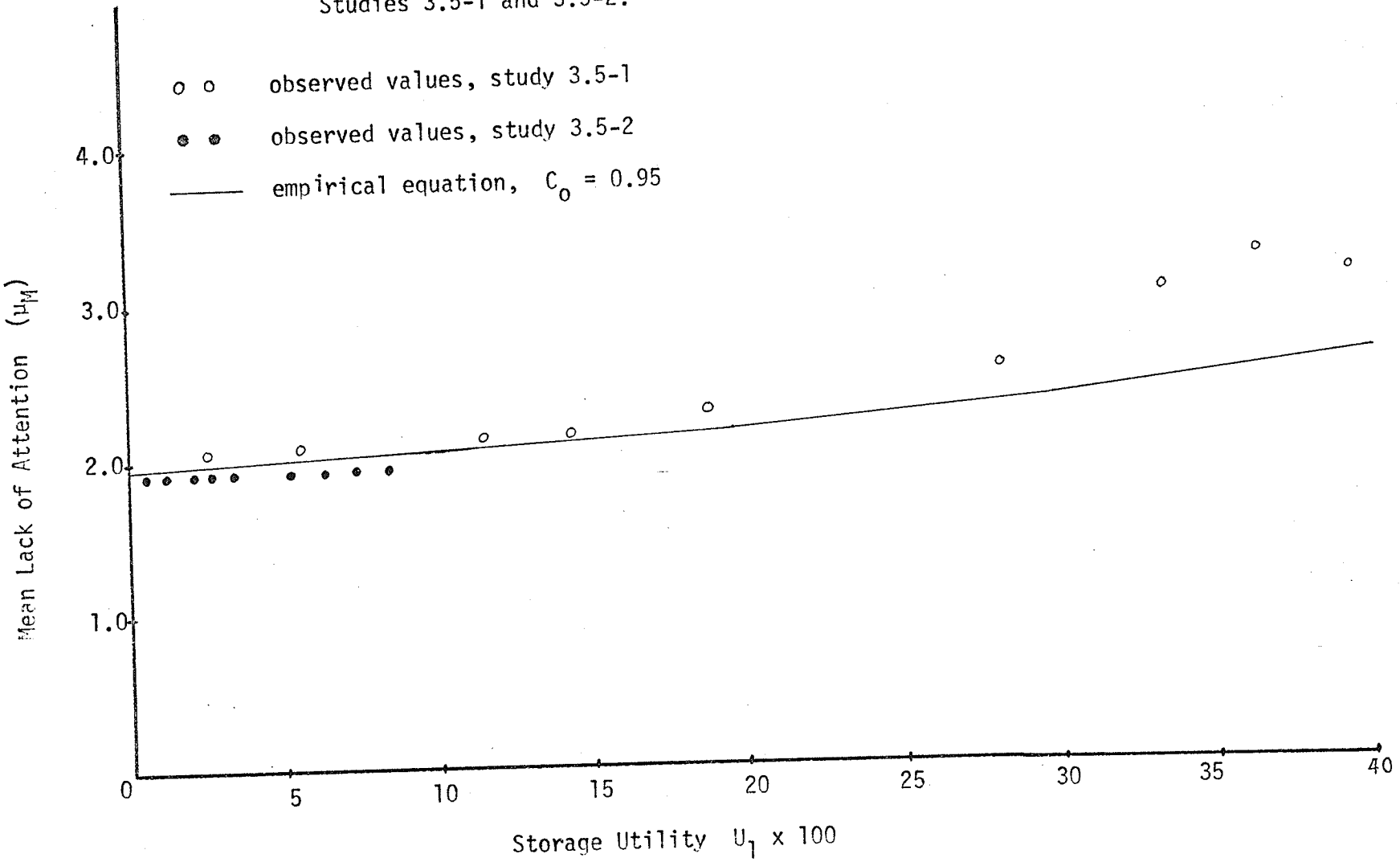
of confidence for both. Figure 3.9 shows system penalty (mean lack of attention, μ_M) as a function of storage utility, U_1, with $C_o = 0.95$, with the observed data from studies 3.5-1 and 3.5-2. For the storage utility less than 0.40 the empirical equation is seen to hold reasonably well.

Divergence of the observed mean lack of attention from that given by the empirical equation for storage utility larger than 0.40 has been attributed to the effect of long-job-loading. At a storage utility of approximately 0.50, for both studies 3.5-1 and 3.5-2 (Tables 3.6, 3.7) the processing becomes serial with jobs being processed in their arrival sequence. For the first-come-first-served queue selection algorithm and for the particular job stream studied, the acquisition of sufficient storage to permit time and space sharing produces a clear improvement in simulation model performance.

3.6 Validity of the Empirical Equation With Various Queue Selection Algorithms.

This section describes and discusses four studies conducted which differ only in the queue selection algorithms used. The quantum allocation logic is constant and corresponds to that used previously (Sections 3.4,3.5). The queue selection algorithms, which are fully described in Section 2.5, are: one, first-come-first-served; two, first-come-first-served-by-priority; three, highest-penalty-first,

FIGURE 3.9 Variation of the Mean Lack of Attention with Storage Utility for Studies 3.5-1 and 3.5-2.



and four, shortest-job-with-highest-main-storage-requirement-first. Each study involves varying the interarrival time of the jobs and observing the change in the system penalty (mean lack of attention).

As in the studies previously described (Sections 3.4,3.5) a value of the coefficient C_o is calculated for each test run to make the empirical Equation (3.2.5) fit the observed results.

The job-mix and operational parameters held constant during these four studies are listed in Table 3.8. The results and calculated coefficients are given in Tables 3.9, 3.10, 3.11, and 3.12, corresponding, respectively, to studies 3.6-1 (first-come-first-served), 3.6-2 (first-come-first-served-by-priority), 3.6-3 (highest-penalty-first) and 3.6-4 (shortest-job-with-largest-storage-requirement-first). Table 3.9 is a repeat of Table 3.3 (Section 3.4). The coefficients calculated for these studies are summarily listed in Table 3.13.

Examination of Tables 3.9,3.10,3.11, and 3.12 reveals that for any particular value of interarrival time greater than 2.2 seconds, the results are largely independent of the queue selection algorithms. Run time, system penalty (mean lack of attention), standard deviation in the system penalty, the utilities, storage occupancy and coefficient C_o vary from algorithm to algorithm by less than five percent. Since reproducibility of results cannot be guaranteed within five percent, there seems to be no significant dependence of results on the particular queue selection algorithm chosen in this region of utility.

TABLE 3.8

SIMULATION MODEL PARAMETERS HELD CONSTANT
FOR STUDY 3.6-1, 3.6-2, 3.6-3, 3.6-4

JOB-MIX PARAMETERS:

Priority Constant	k	6.0
Number of Jobs	N	1500
Mean C.P.U. time, priority one	m_1 (seconds)	1.0
Mean Interarrival time, priority one	r_1 (seconds)	varied
Number of Priority Classes	L	2
Resource Request Size Constant	B	6.0
Resource Request Rate Constant	b	0.46

OPERATIONAL PARAMETERS:

Length of Execution List	S	10
Length of Waiting Queue	Q	20
Round Robin Cycle Time	T_R (seconds)	6.0
Minimum Time Slice	T_{RM} (seconds)	0.06
Supervisor Cycle Time	T_{SC} (seconds)	0.01
Device Time	d (seconds)	1.0
Queue Selection Algorithm	A_S	varied
Storage Available	W_1	25

TABLE 3.9

RESULTS WITH QUEUE SELECTION ALGORITHM ONE (3.6-1)

FIRST-COME-FIRST-SERVED

Interarrival time r_1 (seconds)	Simulated Run Time (seconds)	System Penalty * p_N	Standard deviation, System penalty	C.P.U. Utility U_ox100	Storage Utility U_1x100	Storage Occupancy O_1x100	Calculated Coefficient C_o
100.0	121840.88	1.04	0.26	2.0	0.38	0.39	- *
8.0	9758.57	1.33	0.99	25.0	4.66	6.77	0.84
4.0	4912.38	1.92	1.29	50.0	8.95	14.41	0.82
3.0	3669.96	2.50	1.59	66.7	11.98	35.76	0.71
2.8	3429.56	2.68	1.68	71.5	12.82	41.58	0.63
2.6	3189.52	2.95	1.73	77.0	13.87	50.0	0.56
2.4	2960.66	3.47	1.81	83.5	14.85	62.16	0.48
2.2	2752.96	4.63	2.48	91.0	15.98	76.79	0.35
2.0	2583.24	9.54	5.72	100.0	17.02	97.06	0.00

* See Appendix E.

TABLE 3.10

RESULTS WITH QUEUE SELECTION ALGORITHM TWO (3.6-2)

FIRST-COME-FIRST-SERVIED BY PRIORITY CLASS

Interarrival time r_1 (seconds)	Simulated Run Time (seconds)	System Penalty * P_N	Standard deviation, System penalty	C.P.U. Utility U_ox100	Storage Utility U_1x100	Storage Occupancy O_1x100	Calculated Coefficient C_o *
100.0	121840.88	1.04	0.26	2.0	0.38	0.39	-
8.0	9758.57	1.33	0.99	25.0	4.66	6.77	0.84
4.0	4912.38	1.90	1.29	50.0	8.95	19.41	0.83
3.0	3669.96	2.49	1.58	66.7	11.98	35.82	0.71
2.8	3429.56	2.64	1.59	71.5	12.82	41.55	0.63
2.6	3189.52	2.95	1.72	77.0	13.78	49.98	0.56
2.4	2960.32	3.36	1.76	83.5	14.85	61.73	0.46
2.2	2752.64	4.22	2.19	91.0	15.97	76.28	0.31
2.0	2584.63	6.49	5.82	100.0	17.01	95.68	0.00

* See Appendix E.

TABLE 3.11

RESULTS WITH QUEUE SELECTION ALGORITHM THREE (3.6-3)

HIGHEST PENALTY FIRST

Interarrival time r_1 (seconds)	Simulated Run Time (seconds)	System Penalty p_N^*	Standard deviation, System penalty	C.P.U. Utility $U_o \times 100$	Storage Utility $U_1 \times 100$	Storage Occupancy $O_1 \times 100$	Calculated Coefficient C_o
100.0	121840.88	1.03	0.26	2.0	0.38	0.39	- *
8.0	9758.57	1.35	0.98	25.0	4.66	6.78	0.85
4.0	4912.38	1.90	1.26	50.0	8.95	19.46	0.84
3.0	3669.96	2.47	1.52	66.7	11.98	35.76	0.70
2.8	3429.58	2.62	1.54	71.5	12.82	41.54	0.61
2.6	3189.50	2.88	1.61	77.0	13.78	49.46	0.54
2.4	2960.66	3.46	1.78	83.5	14.85	62.16	0.48
2.2	2753.16	4.34	2.12	91.0	15.97	76.41	0.32
2.0	2584.35	9.18	5.14	100.0	17.01	96.47	0.00

* See Appendix E.

TABLE 3.12

RESULTS WITH QUEUE SELECTION ALGORITHM FOUR (3.6-4)

SHORTEST JOB, HIGHEST STORAGE NEEDS FIRST

Interarrival time r_1 (seconds)	Simulated Run Time (seconds)	System Penalty p_N^*	Standard deviation, System penalty	C.P.U. Utility U_ox100	Storage Utility U_1x100	Storage Occupancy O_1x100	Calculated coefficient C_o
100.0	121840.88	1.04	0.26	2.0	0.38	0.39	- *
8.0	9758.57	1.32	0.99	25.0	4.66	6.77	0.84
4.0	4912.38	1.91	1.26	50.0	8.95	19.39	0.84
3.0	3669.96	2.45	1.53	66.7	11.98	35.64	0.69
2.8	3429.56	2.62	1.64	71.5	12.82	41.28	0.61
2.6	3189.48	2.88	1.76	77.0	13.78	49.57	0.54
2.4	2960.04	3.34	1.94	83.5	14.85	61.69	0.46
2.2	2752.92	4.18	2.37	91.0	15.97	76.03	0.31
2.0	2584.79	6.19	5.58	100.0	17.01	95.98	0.00

* See Appendix E.

TABLE 3.13

SUMMARY DATA FROM THE FOUR QUEUE SELECTION ALGORITHMS.

cpu utility U_0 x 100	Storage utility U_1 x 100	Coefficient - C_0 Algorithm one (study 3.6-1)	Algorithm two (3.6-2)	Algorithm three (3.6-3)	Algorithm four (3.6-4)
25.0	4.66	0.84	0.84	0.85	0.84
50.0	8.95	0.84	0.83	0.84	0.84
66.7	11.98	0.71	0.71	0.70	0.69
71.5	12.82	0.63	0.63	0.61	0.61
77.0	13.78	0.56	0.56	0.54	0.54
83.5	14.85	0.48	0.46	0.48	0.46
91.0	15.97	0.35	0.31	0.32	0.31
100.0	17.01	0.00	0.00	0.00	0.00

Table 3.13 shows the coefficients C_o determined from the utilities listed. For the range of cpu utility for which the coefficient C_o could be assumed constant according to the discussion of chapter three, namely cpu utility less than about 0.60, the same constant coefficient C_o, 0.80, can be used for all four algorithms. Figure 3.10 shows mean lack of attention measured for each algorithm and calculated with C_o equal to 0.80, for cpu utility values from zero to unity. Figure 3.11 shows μ_M versus storage utility. The measured data is well fitted by the analytic curve for cpu utility less than 0.60.

The unusual conclusion that may be drawn from this chapter is that the performance of the simulation model is independent of the queue selection algorithm for values of cpu utility up to 91%. At a cpu utility of 100%, the measured mean lack of attention for the queue selection algorithms one to four, are 9.54, 6.49, 9.18, and 6.19 respectively, representing a variation of nearly fifty percent. Thus the queue selection algorithm affects performance only in extremely heavily loaded or saturated systems, with cpu utility very close to unity. Under these conditions the algorithms choosing jobs earliest in arrival by priority class or choosing the shortest job with highest storage requirement appear to produce better performance than those algorithms choosing jobs on a first-come-first-served basis or choosing the job with the highest penalty. One feature in common with the two more successful queue selection algorithms is that they both choose short jobs in preference to long jobs, leaving

FIGURE 3.10 Variation of the Mean Lack of Attention with Central Processor Utility, for the Four Queue Selection Algorithms

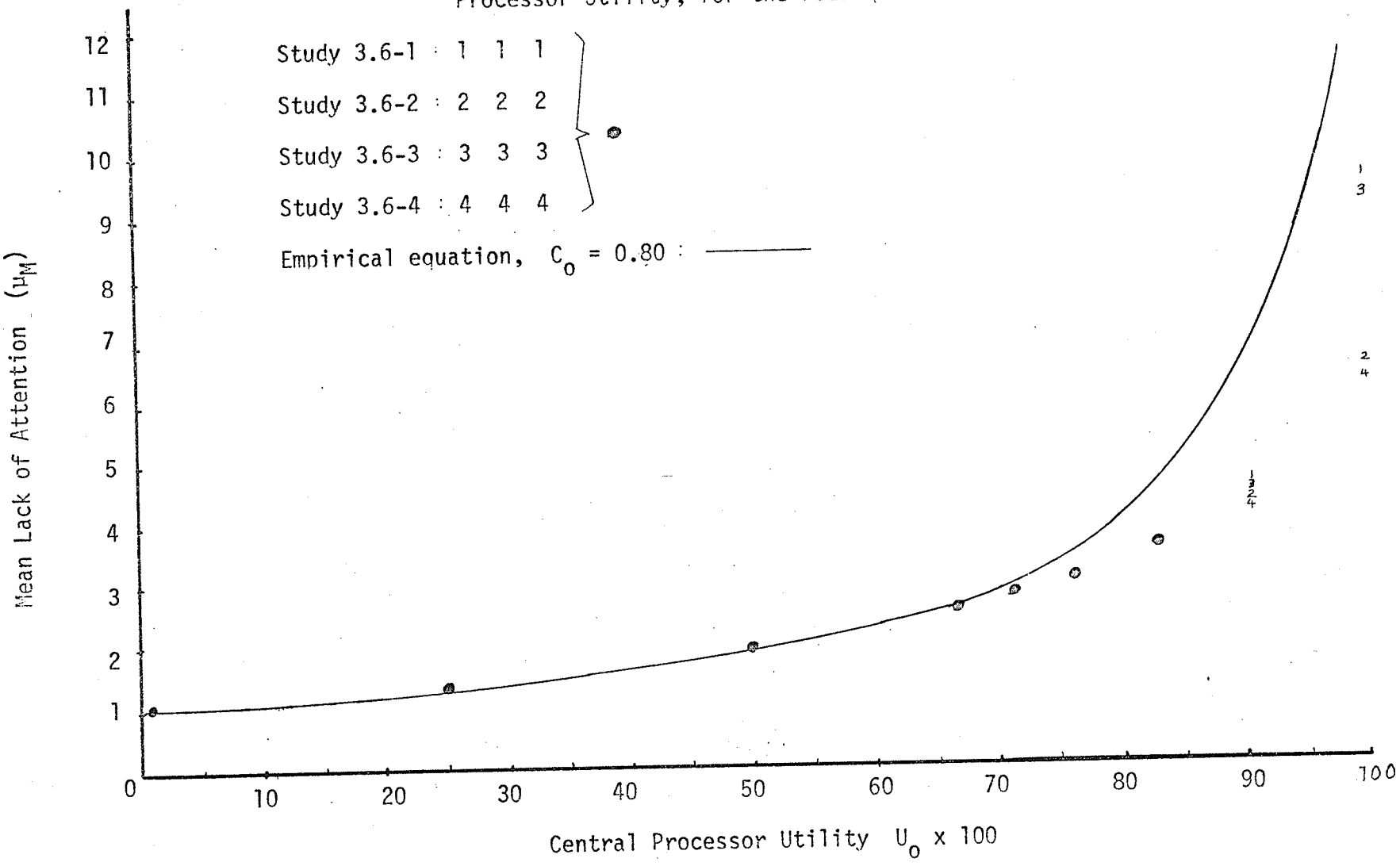
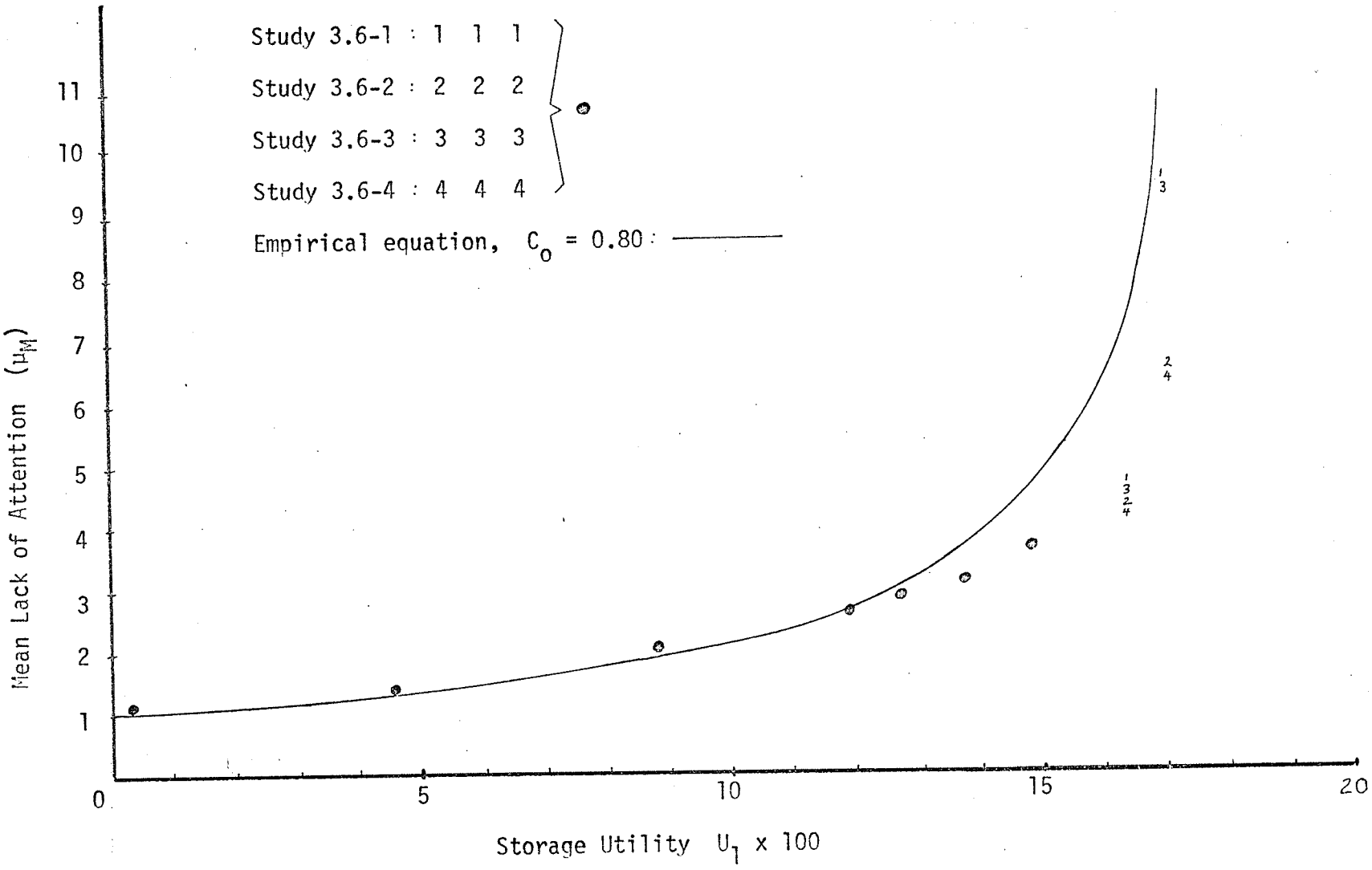


FIGURE 3.11 Variation of the Mean Lack of Attention with Storage Utility for the Four Queue Selection Algorithms.



the longer jobs to be processed during periods of time when shorter jobs are not arriving, in this case, probably just prior to completion of the simulation run.

3.7 Conclusions of the Simulation Studies

An empirical equation was suggested to explain the variation of μ_M in a manner which is independent of the details of the job stream or the job scheduling algorithm. The equation is:

$$\mu_M = 1 + C_o \left(\frac{1}{\frac{1}{U_o} - 1} + \frac{1}{\frac{1}{U_1} - 1} \right). \quad (3.2.5)$$

The value of the coefficient C_o, which was appropriate in the range of utility from 0 to 0.60, appears to depend on the number of priority classes in the job stream. With the main storage large enough to permit space sharing, the values of C_o are 0.40, 0.80 and 0.95 for one, two and three priority classes, respectively.

Values of μ_M calculated with a constant coefficient were found not to fit the observed values of μ_M in the range of utility 0.6 to 1.0. The equation appears to be dependent on the job scheduling algorithm in this range of utility, with marked differences occuring for utility values exceeding 0.80. Those algorithms which tended to select shorter jobs from the waiting queue in preference to longer jobs produced up to 50 percent better performance than other

algorithms studied. These results indicate that the selection of a job-scheduling algorithm is only significant for computer systems which are heavily loaded for at least part of their operating time (that is U_o or $U_1 > 0.8$).

For utility values of unity, a coefficient value of zero was required, since the observed values of $\not{U}_M$ remained finite. It was suggested that $\not{U}_M$ depends on the number of jobs in the job stream since the series of final job penalties, p_j^* becomes time dependent at utility values of unity.

Values of $\not{U}_M$ were found to become dependent on the main storage size, W_1, as the latter was reduced toward unity. This dependence was attributed to loss of the space sharing properties of the model as the main storage size was reduced (serial processing at $1 = W_1 = R_j$, all j). The space sharing properties of the model could also have been removed by appropriately modifying the quantum-allocation routine to process jobs serially. It is quite, therefore, reasonable to suggest that selection of the quantum allocation routine is significant in at least the range of utility for which $\not{U}_M$ was found to be dependent on W_1, that is, the range of utility from 0.20 to 0.50. Further investigation of this suggested significance might be fruitful.

APPENDIX A

MONTE CARLO TECHNIQUES USED

A. 1 Introduction

In the studies described, the central processor time requests and storage requests are assumed to be distributed normally about given mean values, with given standard deviations. Arrival times are assumed Poisson, requiring interarrival times to follow exponential distributions with given mean values (14). The power residue or multiplicative congruential method is used to generate random numbers following a rectangular distribution and these numbers are transformed to either a normal distribution or an exponential distribution, depending upon the particular requirement.

A. 2 Power Residue Method

The method begins with a constant k, a starting value n_o and a modulus m. A sequence $\{n_i\}$ of non-negative integers, randomly distributed, with each less than m, is generated by means of the recursive formula

$$n_{i+1} = k\, n_i \pmod{m}, \qquad \text{(A.2.1)}$$

where $k = 65539$ and $m = 2^{31}$.

Any positive odd integer less than 2^{31} may be used as the starting value n_o. In all of the studies reported,

a value of n_o is chosen to be 123.

To obtain a real number, y_i, randomly distributed in the interval (0, 1) a further calculation is necessary using the formula

$$y_i = n_i / (2^{31} - 1) , \qquad (A.2.2)$$

where n_i is an integer randomly distributed and is obtained by equation (A.2.1).

Further discussion of the power residue method may be found in (16).

A.3 Method Used for Generating Normal Random Variates.

The Central Limit Theorem is used in the generation of a normal random variates x_i with a given mean, m_x, and a standard deviation s_x, in the following formula:

$$x_i = s_x(12/K)^{\frac{1}{2}} \left(\sum_{i=1}^{K} y_i - K/2 \right) + m_x ,, \qquad (A.3.1)$$

where y_i is a uniformly distributed random number between 0 and 1, determined by the power residue method described in Section A.2., and K is the number of new values of y_i to be used in the generation of x_i. To reduce execution time, K is chosen as 12. Thus Equation (A.3.1) becomes

$$x_i = s_x \left(\sum_{i=1}^{12} y_i - 6.0 \right) + m_x .. \qquad (A.3.2)$$

According to the Central Limit Theorem, as K approaches infinity, the set of values of x_i approaches a true normal distribution asymptotically (16).

A.4. Method Used For Generating Exponentially Distributed Variates

Exponentially distributed variates are generated by taking a uniformly distributed random number y_i between 0 and 1, determined by the power residue technique (described in Section A.2) and transforming it to an exponentially distributed variable x_i according to the equation (15):

$$x_i = m_x \log_e \left(\frac{1}{y_i} \right) , \qquad (A.4.1)$$

where m_x is the mean value of the exponentially distributed variates x.

The probability that y_i will occur in a particular range of size Δy is equal to the area beneath the constant probability distribution, and therefore this may be expressed as $C \Delta y$. All the values of y_i occurring in this range are transformed to values of x_i and they will occur in a particular range of size Δx, and the probability density function at this range may be represented $P(x)$. It follows that

$$P(x) \Delta x = C \Delta y , \qquad (A.4.2)$$

and therefore

$$\lim_{\Delta y \to 0} \frac{\Delta y}{\Delta x} = \frac{P(x)}{C} = \frac{dy}{dx} . \qquad (A.4.3)$$

From equation (A.4.1)

$$\frac{dy}{dx} = - \frac{1}{m_x} e^{- \frac{x}{m_x}} , \qquad (A.4.4)$$

and since the integal of P (x) over the range $(0, \infty)$ must be unity, C must assume a value of - 1.

Therefore

$$P(x) = \frac{1}{m_x} e^{-x/m_x} . \qquad (A.4.5)$$

The exponential distribution expressed in Equation (A.4.5) describes the probability of a value of x less than T which can be calculated from:

$$P(x < T) = \int_0^T P(x)\, dx = \left[-e^{-x/m_x} \right]_0^T$$

$$= 1 - e^{-T/m_x} . \qquad (A.4.6)$$

The transformation expressed in Equation (A.4.1) may be used to convert rectangularly distributed variables to exponentially distributed variables.

A.5 Summary

Generation of a normally distributed random variate with a given mean and standard deviation (a central processor time or a storage request) is accomplished by applying the Central Limit Theorem (Section A.3) to twelve uniform random numbers in the range (0,1) (Section A.2). Generation of an exponentially distributed variate with a given mean (an interarrival time) is accomplished by applying the transformation described in Section (A.4) to a uniform random number

in the range (0,1) (Section A.2). The starting value for the random number generation routine is 123 throughout.

APPENDIX B

SAMPLE JOB STREAM DATA

The first few jobs generated in each of the three priority classes for study 3.5-1 are shown. Parameters used to generate these jobs are presented in table 3.5.

Priority Class One Jobs:

Arrival Time (seconds)	CPU Time Request (seconds)	Storage Request (units)
5.885	0.608	2
8.489	0.725	2
10.621	1.042	4
12.137	0.803	2
29.166	1.028	5
33.290	0.750	2
36.654	0.662	2
43.890	1.239	3
45.215	0.827	3
46.338	1.093	3
47.113	1.201	4
57.903	0.853	2
58.053	1.254	5
66.211	0.605	1
66.816	0.918	4
...	...	...

Priority Class Two Jobs

Arrival Time (seconds)	CPU Time Request (seconds)	Storage Request (units)
4.621	7.114	9
4.684	6.812	5
100.671	4.745	3
129.363	5.307	4
149.155	9.086	8
206.949	5.253	5
248.853	7.716	6
...	...	...

Priority Class Three Jobs

Arrival Time (seconds)	CPU Time Request (seconds)	Storage Request (units)
144.254	39.008	9
223.692	33.261	10
294.969	35.146	7
510.519	46.566	12
...	...	...

APPENDIX C

DESCRIPTION OF WAITING-QUEUE AND EXECUTION-LIST ENTRIES

The items of description recorded for a job while it is in the waiting queue are the following:

(1) job identification number
(2) job priority class number
(3) job central processor time request
(4) job arrival time
(5) job device time
(6) job storage requirement

The items of description recorded for a job while it is in the execution list are those listed above plus the following:

(7) central processor time so far allocated
(8) actual time of job entry to the execution list.

APPENDIX D

AN EQUIVALENT DEFINITION OF STORAGE UTILITY

Storage utility, U_1, is defined by

$$U_1 = \frac{1}{TW_1} \sum_{j=1}^{N} t_j R_j \ , \qquad \text{(D-1)}$$

where T is the simulated run time, W_1 is the available storage in arbitrary units, t_j is the execution time of each job j and R_j is the storage request of each job j in the same units.

An equivalent expression may be derived. The mean cpu time request is the same for all jobs of the same priority class I, and is m_I. The mean storage requirement of all jobs with the same execution time is the same - hence to each priority class a mean storage requirement, $\bar{R}_I$, may be assigned. The storage time product for jobs of each priority class is then $n_I m_I \bar{R}_I$ and for the sum,

$$\sum_{j=1}^{N} t_j R_j = \sum_{I=1}^{L} n_I m_I \bar{R}_I \ . \qquad \text{(D-2)}$$

In Section 2.2, an expression for n_I is given

$$n_I = \frac{k^{(L-I)}}{\sum_{i=1}^{L} k^{(i-1)}} \ , \qquad \text{(D-3)} \quad \text{(2.2.8)}$$

also

$$m_I = km_{I-1} = k^{(I-1)} m_1 \quad . \qquad (D-4)$$

Then, using Equations (D-3) and (D-4)

$$\sum_{I=1}^{L} n_I m_I \bar{R}_I = \frac{k^{(L-1)} Nm_1}{\sum_{i=1}^{L} k^{(i-1)}} \sum_{I=1}^{L} \bar{R}_I \quad . \qquad (D-5)$$

For a large number of jobs (large N) the time of the simulation, T, may be approximated by

$$T \simeq n_I r_I \quad . \qquad (D-6)$$

This expression, (D-6), is independent of I, since

$$n_I r_I = \frac{1}{k} n_{I-1} \; kr_{I-1} = n_{I-1} r_{I-1} \; , \qquad (D-7)$$

and then T may be written:

$$T \simeq n_1 r_1 = \frac{k^{(L-1)} N}{\sum_{i=1}^{L} k^{(i-1)}} r_1 \quad . \qquad (3-8)$$

Using Equations (D-5) and (D-8), storage utility becomes

$$U_1 = \frac{m_1 \sum_{I=1}^{L} \bar{R}_I}{r_1 W_1} \quad . \qquad (D-9)$$

Since $\bar{R}_M$ is defined by:

$$\bar{R}_M = \sum_{I=1}^{L} \bar{R}_I \quad ; \qquad \text{(D-10)}$$

it follows that:

$$U_1 \simeq \frac{m_1 \bar{R}_M}{r_1 W_1} \quad . \qquad \text{(D-11)}$$

The basic assumption in this derivation was that $T \simeq n_1 r_1$, and for sufficiently large N (1500) this is found to hold within five percent when $m_i L/r_i \leq 1$. Equations (D-1) and (D-11) are very nearly equivalent in this region.

APPENDIX E

THE COEFFICIENT C_o AT LOW UTILITY VALUES

The coefficient C_o is not calculated for simulation runs made with the parameter r_1 of 100 seconds in studies 3.4-1, 3.4-2, 3.6-1, 3.6-2, 3.6-3 and 3.6-4. This is because the range of confidence in such calculated values of C_o is very large.

The numerator of the equation used for calculation of the coefficient, Equation (3.3.1), is $\mu_M - 1$. In the simulation runs indicated, the value of $\mu_M - 1$ varies from 0.02 to 0.04, while the standard deviation in the system penalty (or μ_M) varies from 0.19 to 0.26 (see Tables 3.2, 3.4, 3.9, 3.10, 3.11 and 3.12).

The techniques described in Section 3.3 may be used to show, for the simulation runs indicated, that the confidence range for values of the coefficient calculated would vary from $\pm$ 84.4 percent to $\pm$ 128.8 percent.

REFERENCES

(1) Chai, L. A., Study of the Performance of a Scheduling Algorithm for a Time Slicing Supervisor M.Sc. Thesis, University of Manitoba (1968)

(2) Greenberger, M., The Priority Problem and Computer Time Sharing, Management Science 12, (July 1966), pp 888-906.

(3) Kleinrock, L., Time Shared Systems, A Theoretical Treatment, J.A.C.M., Vol. 14, No.2, (April 1967), pp 242-261.

(4) Schrage, L. E., The Queue M/G/1 with Feedback to Lower Priority Queues, Management Science, Vol. 13, No. 7 (1967). pp. 466-474.

(5) Hutchinson, G., and Maguire, J., Computer Systems Design and Analysis Through Simulation, A.F.I.P.S. Conference Proceedings, Vol. 27 (1965), pp 161-167

(6) Huesmann, L., and Goldberg, R., Evaluating Computer Systems Through Simulation, Computer Journal, Vol. 10, No. 2 (August 1967), pp. 150-156

(7) Nielsen, N., The Simulation of Time Sharing Systems, C.A.C.M. Vol. 10, No. 7 (July 1967), pp 397-412.

(8) Penny, J. P. An Analysis Both Theoretical and by Simulation of a Time-Shared Computer System, The Computer Journal, Vol. 9, No. 1, (May 1966) pp. 53-59.

(9) Smith, J., An Analysis of Time Sharing Computer Systems Using Markov Models, A.F.I.P.S. Conference Proceedings, Vol. 28 (1966), pp 87-95.

(10) Fife, D., An Optimization Model for Time Sharing, A.F.I.P.S. Conference Proceedings, Vol. 28 (1966) pp 97-104.

(11) Wren, J., Study of the Performance of Scheduling Algorithms by Simulation, M.Sc.Thesis, University of Manitoba (to be submitted).

(12) Riding, G., (private communication)

(13) Scherr, A. L., An Analysis of Time Shared Computer Systems, MAC-TR-18, M.I.T. Project MAC, Cambridge, Mass., (1965)

(14) Analysis of Some Queuing Models in Real Time Systems, I.B.M. Corporation, Manual C20-0007-1, White Plains, New York (1966)

(15) Blatny, J. (private communication)

(16) Naylor, T. H., Balintfy, J. L., Burdick, D. S., and Chu, K., Computer Simulation Techniques (New York: John Wiley & Sons, Inc., 1966), pp 49-54.

(17) Kilburn, T., Payne, R. B., and Howarth, D. J., The Atlas Supervisor, A.F.I.P.S. Eastern Joint Computer Conference Proceedings, Vol. 20 (1961), pp 279-294.