OPTIMIZATION TECHNIQUES IN COMPUTER SYSTEM DESIGN AND LOAD CONTROL

by

PREM SWARUP SINHA

Technical Report TN 81-16
September 1981

Department of Computer Science
University of British Columbia
Vancouver, B. C., V6T-1W5

This report was submitted as a thesis in partial fulfillment of the requirement for the degree of Doctor of Philosophy.

OPTIMIZATION TECHNIQUES IN COMPUTER SYSTEM DESIGN AND LOAD CONTROL

by

PREM SWARUP SINHA

B.Sc., Delhi University, India, 1971

M.Sc., Indian Institute of Technology, New Delhi, India, 1973

DIIT, Indian Institute of Technology, New Delhi, India, 1974

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming

to the required standard

.....

.....

.....

.....

THE UNIVERSITY OF BRITISH COLUMBIA

SEPTEMBER 1981

ABSTRACT

Analytic modelling has proven to be cost-effective in the performance evaluation of computer systems. So far, queueing theory has been employed as the main tool. This thesis extends the scope of analytic modelling by using optimization techniques along with queuing theory in solving the decision-making problems of performance evaluation. Two different problems have been attempted in this thesis.

First, a queueing network model is developed to find the optimal capacities and speeds of the memory levels in a memory hierarchy system operating in a multiprogrammed environment. Optimality is defined with respect to mean system response time under a fixed cost constraint. It is assumed that the number of levels in the hierarchy as well as the capacity of the lowest level are known. The effect of storage management strategy and program behaviour are characterised by the miss ratio function which, together with the device technology cost function, is assumed to be represented by power functions. It is shown that the solution obtained is globally optimal.

Next, two adaptive schemes, SELF and MULTI-SELF, are developed to control the flow of jobs in a multiprogrammed computer system. They periodically determine the number of jobs from each class that should be activated to minimize the mean system residence time without saturating the system. The computation is based on the estimated system workload in the next interval. An exponential smoothing technique is used to reduce the error in estimating the values of the model parameters. The service provided to each class (specifically, the mean response time) may be adjusted by changing the weight associated with the job class. From our simulation results, the schemes appear to be both stable and robust. Performance improvement over some of the existing schemes (50%, L=S and the Knee criteria) is significant under some workloads. The overhead involved in its implementation is acceptable and the error due to some of the assumptions in the formulation and solution of the model are discussed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

I would like to thank my advisor, Dr. Samuel T. Chanson, for his guidance and constant encouragement during the past three years of my graduate studies. I am also thankful to the members of my committee, Dr. David R. Cheriton, Dr. David G. Kirkpatrick, Dr. Uri Ascher and Dr. Mabo R. Ito for their interest and constructive comments.

I would also like to thank my colleagues Ezio Catansariti and Mark C. Fox for carefully reading an early version of the thesis. A very personal note of gratitude to my parents, brothers and sisters. It was almost impossible for me to accomplish this achievement without their constant support and encouragement from half way round the globe.

Finally I am grateful to the Department of Computer Science, University of British Columbia for the financial help in the form of Teaching Assistantship and University Summer Research fellowship.

CHAPTER 1

INTRODUCTION AND PROBLEM DEFINITION

1.1 The Importance of Performance Evaluation

Performance criteria are major consideration in the design of computer systems. Therefore, knowledge about program characteristics, system load and optimal design theory are important aids to the prevalent intuitive and *ad hoc* methods. The increasing demand for computer resources has made computer system designers, data processing and corporate managers, system analysts, programmers as well as the user community at large more concerned about system efficiency and utilization. Despite continual reduction in the cost of hardware, inefficient resource utilization represents unnecessary wastage. Moreover, there is an increasing interdependency among users. Whether in a large centralized system or in a distributed computing environment where resources are shared, poor performance of

one component affects many users.

There are different measures of performance such as response time, throughput rate and various resource utilizations. Optimization of one measure does not necessarily optimize other measures. Therefore it is important to select a proper measure or a combination of measures to be consistent with the objectives of a particular study.

Once the system is in production, structural problems, particularly those related to the computer architecture are often difficult to remove. A significant change in the system structure may be required to remove bottlenecks in the system. Therefore it is desirable to study a model of the system before it is configured. This also allows the designers to study different models of the system and select the best suited for the specific application.

1.2 Approaches to Computer System Modelling

The study of computer system modelling and performance analysis employs three different methods viz., measurement, simulation and analytic techniques. Direct measurement techniques are not always applicable, particularly to design problems because they require the system to be in existence. Also production systems may not be available

for experimentation. Moreover, it may not be practical to implement certain control mechanisms if the system was not initially designed to facilitate extensions or modifications.

Simulation techniques are the most popular, since they are able to represent the characteristics of the modelled system more closely than is possible with currently available analytic techniques. However simulation models are expensive to build, validate and use.

Although analytic models are more cost-effective for evaluating and predicting computer system performance, they are harder to formulate and to solve in general. Recent advances have made analytic models increasingly capable of representing more closely the characteristics of the modelled system.

1.3 Analytic Modelling

Queueing theory has been the major tool used in analytic modelling of computer systems. A queueing network is a network of service centres. Each centre has one or more queues associated with it. Customers wait in queues for their turn to be served by the server. At each service centre the distribution of the service times for each class of customers and the scheduling algorithm are known and

fixed. After being served at a centre the customer moves to other service centres or exits from the system according to a fixed set of transition probabilities which may or may not be class and/or workload dependent.

A queueing network is said to be open if job arrivals are independent of job departures and the number of jobs present in the system. In a closed queueing network, the number of jobs remain constant. Whenever a job leaves the system, a statistically identical job is injected into the system to replace it. A computer system can be considered as a network of queues with various processing devices as the service centres.

The analysis of a queueing network can be classifed as:

(i) exact analysis using classical queueing theory,

(ii) approximate techniques,

(iii) operational analysis.

As discussed below, none of the three approaches is superior to the others under all circumstances.

Exact solutions were first given by Jackson [Jack63].

He showed that in a queueing network composed of exponential service centres, the equations governing the equilibrium distribution of the system states exhibit a product form. This means the probability that the queueing network is in a particular state is equal to the product of the probabilities that each individual service centre is in its corresponding state divided by a normalizing constant. Gorden and Newell [GoNe67] simplified the product form solution for closed queueing networks by developing a separation of variables solution technique. Buzen [Buze73] introduced the widely used central server model and provided efficient algorithms for computing the normalizing constants, the equilibrium distribution of system states, throughput rates and queue length distributions. Most of the work in queueing theory assumed identical jobs until Basket et al. [BCMP75] extended these results to cover multiple classes of jobs, different queueing disciplines and non-exponential service distributions.

Approximate techniques give either an approximate solution to the original network or an exact solution of an approximate model. Norton's theorem as defined by Chandy et al. [ChHW75b] (analogous to Norton's theorem in electrical engineering) plays a major role in the development of approximate techniques. The model involves replacing a subnetwork that has a single input stream and a single output stream with a single composite service

centre. The resulting queueing network is repeatedly simplified until it permits analysis by global balance techniques. It is shown that, if the original system has the local balance property, then the reduced model is exact in the sense that the queue length distribution at the service centres (not the system) are identical in the original and the reduced network. Good references for approximate techniques can be found in Chandy et al. [ChHW75b] and Courtois [Cour77].

Operational analysis was first introduced by Buzen [Buze76] and a good survey can be found in [DeBu78]. With this approach, one can obtain most of the results of stochastic analysis without making the assumptions (often unrealistic) required by queueing theory. The main drawback of this technique is that it is not suitable for performance prediction and does not lend itself to the study of transient system behaviour.

Although queueing theory is an essential tool for computer system performance modelling, it does not solve the problems of decision-making. For example, it can provide the values of various system performance measures for a combination of system parameters but it does not provide a best combination of system parameters that will optimize certain performance measures. Recently, some work has been done to apply optimization techniques along with

queueing theory to solve such problems ([ChSi80a], [ChSi80b], [HiMT79], [TrWa80], [TrWS80]). In this thesis, optimization techniques are used along with time series analysis and queueing theory in solving the decision-making problems of large computer systems. The techniques developed have been applied to the two problems of (a) the design of a memory hierarchy, and (b) the job flow control in a multiprogrammed system.

1.4 Thesis Overview

The thesis consists of five more chapters. In chapter 2, the problem of memory hierarchy design in a multiprogrammed system is studied. First a queueing network model of a multiprogrammed memory hierarchy system is developed. An expression for its response time is then computed in terms of the capacities and speeds of various levels of memory. The optimal capacities and speeds are then obtained by minimizing the response time subject to a fixed cost constraint. A modified version of geometric programming is used to solve the minimization problem. A closed form solution is derived and shown to be globally optimal.

Chapter 3 describes and compares the relative merits of various existing load control mechanisms. An expression for the estimate of the saturation point is then derived

using operational analysis. An optimal selection of batch and terminal jobs that should be maintained in the system is also computed. This chapter also describes how time series analysis can be used in the estimation of the system parameters required for computing the saturation point estimate. One of the main assumptions of the control scheme developed in this chapter is that all the batch and terminal jobs are statistically identical in their resource demands.

Chapter 4 relaxes the identical job assumption of the model developed in chapter 3. The scheme developed here can handle any practical number of multiple classes of jobs. Jobs within a class are statistically identical but jobs in different classes can be different. In the extreme case, each job can belong to a different class.

Chapter 5 describes a simulator for a central server model. The two schemes described in chapters 3 and 4 were simulated and their performance compared to some of the existing schemes. The assumptions made throughout the development of the two control schemes and the errors introduced by them are discussed.

Chapter 6 summarizes the thesis and suggests further research directions.

CHAPTER 2

MEMORY HIERARCHY DESIGN

2.1 Introduction and Review

A major criterion of computer system design is the efficiency of its memory resource utilization. Although the cost of memory is decreasing, the demand for computer resources is increasing even more rapidly, and in some applications the problems are getting larger. Therefore, it is important to have an optimal design for the memory system. The memory system is generally a combination of a variety of technologies with different cost-performance characteristics. Such an assembly of interconnected devices is generally called a memory hierarchy. Management of the memory hierarchy requires determining where to store specific information, how to retrieve it and finally when to move it from one level to another. The objective is to maintain the more frequently used data in the fastest (and therefore most expensive) device in order to minimize the

retrieval time. It is also necessary to recognize when the data are no longer needed so that they can be moved to a slower (and cheaper) device. The design problem considered in this chapter is to find the optimal sizes and speeds of different memory devices given a fixed cost constraint. In the past, this problem was treated using heuristic approaches rather than quantitative methods. Recently, quantitative methods have been developed which yield a better understanding of memory hierarchy configuration evaluation.

The optimization of a memory hierarchy is recognized as an important research area [Triv78] and has been approached from several directions. Various solutions, optimal under certain constraints, have been obtained.

Ramamoorthy and Chandy [RaCh70] have considered the problem of finding the type and size of each level of memory under certain assumptions. Their method involves solving a linear programming problem with the average access time of an information block in a program as the objective function and a given hierarchy cost as the constraint. The results are then extended to a general case of multiprogramming. The approach presupposes the knowledge of frequency of access for each information block. A drawback of the model is that it uses average access time as the objective function while ignoring the

delays due to queues formed in front of the lower levels of memory.

MacDonald and Sigworth [MaSi75] have dealt with various combinations of optimization criteria such as fixed cost constraint, fixed and variable page size etc. They too assume knowledge of the storage address sequence and have used its statistical properties extensively in their work. The objective function to be minimized is again the average access time, or a function of it (ignoring the queueing delays), which implies that even with available program reference strings the scheme cannot be applied to multiprogramming.

Chow [Chow74] has very nicely applied geometric programming to obtain not only the optimal size and speed of each memory level, but also the optimal number of levels of hierarchy for a given cost constraint. The unit of information transfer is a fixed-size page and the page size is the same for all levels. The effect of the page replacement algorithms, the program behaviour (and hence the workload) and the page size are captured by a hit-ratio function. A hit-ratio function is defined as the probability of successfully retrieving the needed information from a particular level of memory. Furthermore, the hit-ratio function and device technology cost function are taken as a power function of the capacity

and access time respectively of each level of memory hierarchy. Chow's analysis ignores queueing delays and hence is also restricted to a uniprogramming system.

Welch [Welc78] gives a very simple and straightforward analysis of a memory hierarchy for speed-cost trade-off with the assumption that the size and access probability of each level of memory are known and fixed. Rege [Rege76] uses a very simple two-server queueing network model to analyze the cost-performance trade-off by using different sizes and speeds at different memory levels. There is no optimization study in either case.

Foster and Browne [FoBr76] are among the first to explicitly account for queueing at devices in the memory hierarchy to study a related but different problem - that of file assignment in the hierarchy. Traiger and Mattson [TrMa71] consider the design problem of a storage hierarchy using a central server model. But their model is restricted to three levels. In a later study, Lin and Mattson [LiMa72] extend the technique to four levels. Because of the exhaustive nature of the search for the optimal solution, the technique seems to be impractical when the number of levels in the hierarchy increases beyond four. Gecsei and Lukes [GeLu74] reduced the complexity of the model to some extent but in doing so they had to approximate each stage of the network as an open-loop queue

with random arrival.

Trivedi, Wagner and Sigmon [TrWS80] have used a closed queueing network model to find optimal CPU speed, device capacities and allocation of a set of files across the secondary storage devices by maximizing the throughput rate. In another work, Trivedi and Wagner [TrWa79] have obtained the speeds of various secondary devices for a given set of transition probabilities in a closed queueing network.

Most previous work in optimization of memory hierarchies used the mean memory access time as the objective function for minimization. With a few exceptions (e.g.,[RaCh70], [TrWa79] and [TrWS80]-- the latter two were done in parallel with this work [ChTr79]), they dealt only with the uniprogrammed environment where only one process is active at any time and the processor is idle when a request is made to any memory level. It is not clear that in a multiprogrammed environment, minimizing the average memory access time is meaningful since a process may be blocked while it is referencing information in a certain level of memory. The following analysis combines performance evaluation techniques and optimization methods to extend the analysis of Chow [Chow74] to cover multiprogrammed systems. Mean response time is chosen as the objective function. With the number of memory levels

fixed and the capacity of the lowest level known, an expression for response time is obtained in terms of the capacity and speed of each level of memory. The optimal expression of memory sizes and speeds are then obtained using the Lagrangian function under a fixed cost constraint.

Notice that in the uniprogramming environment,

Average response time = c_1 * Average access time of the memory hierarchy + c_2

where c_1 = average number of accesses to the memory hierarchy per interaction,

and c_2 = mean CPU time demand of the process per interaction.

The parameters c_1 and c_2 are constants for a given process. Hence the average response time and the average memory hierarchy access time are equivalent objective functions in the uniprogrammed environment.

2.2 System Description, Assumptions and Notation

The memory hierarchy consists of N levels, $M_1, M_2, \ldots, M_N$, where N is known and fixed. Generally, the higher the level (i.e., the smaller the index) the smaller is the capacity, the faster its speed and the more expensive is its unit cost. It is assumed that information

present in any level is also present in all subsequent lower levels. This assumption may not be true for all levels (particularly for lower levels of memory). In the case of a uniprogramming system, whenever the needed information is not found in the highest level M_1, a request is made to each of the lower levels successively until it is found in a level M_i; i = 2, ..., N. The processor is held waiting all the time until the information is retrieved from M_i. As i increases, the time required to fetch the information goes up. When i exceeds a certain value, it becomes uneconomical to keep the processor idle while the information is being retrieved from M_i, particularly when there are other processes waiting for the processor. Thus, in the case of multiprogramming, we have two types of memory - A and B. While the processor waits for access to type A memory, it does not do so for access to type B memory, but releases the current process and takes the next process ready to run, if one exists. It is therefore possible for several requests to queue up at a type B memory level but there is at most one request at any one time for a type A memory level. The model of such a system is shown in Figure 2.1 where n_1 and n_2 are the number of type A and type B memory levels respectively. x_i, i = 1, 2, ..., N ($N = n_1 + n_2$) is the capacity of memory level M_i in the hierarchy. n_1, n_2 and x_N are assumed to be given.

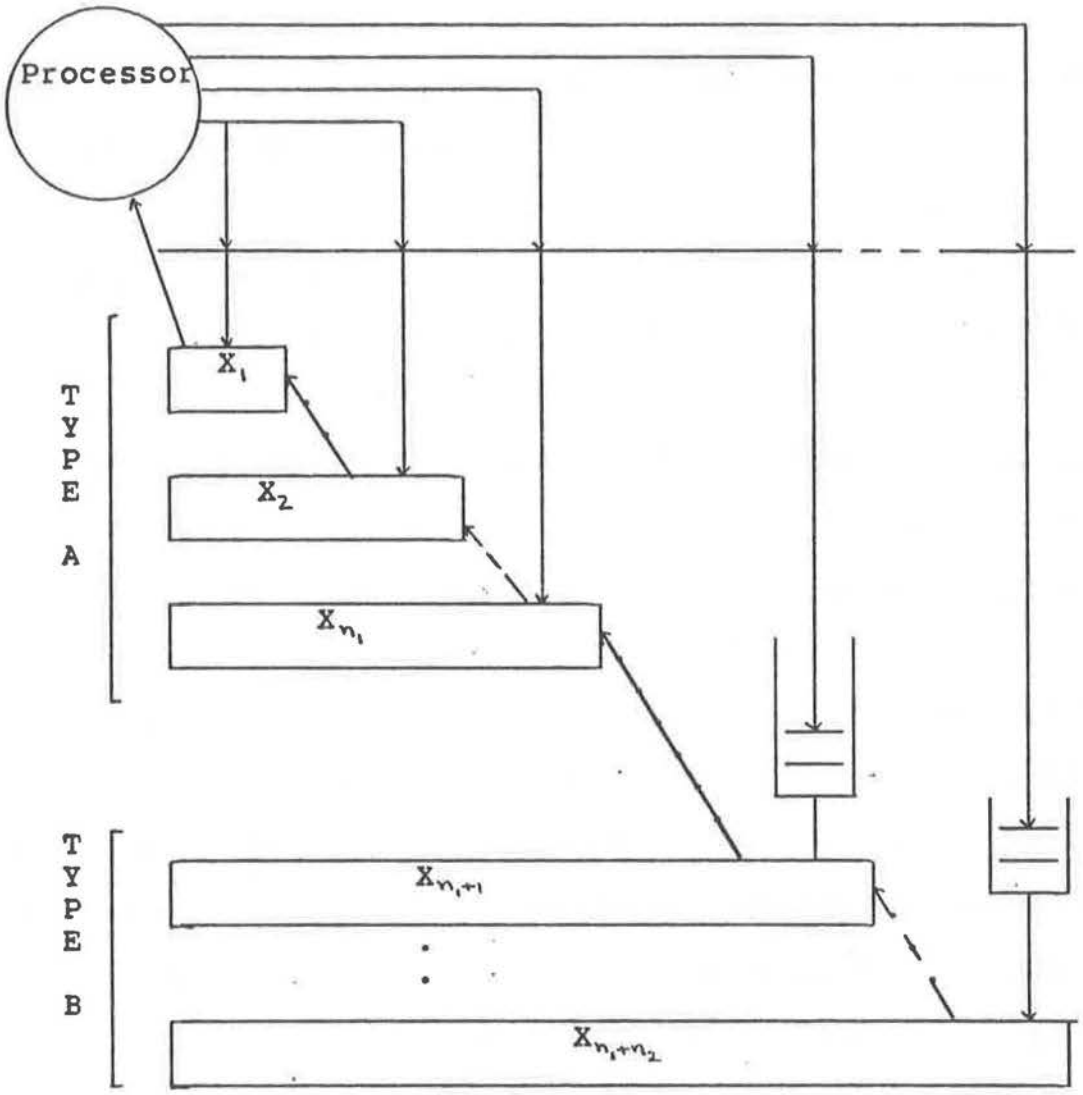

FIGURE 2.1 STORAGE HIERARCHY IN A MULTIPROGRAMMED SYSTEM

Following Chow's [Chow74] terminology, we define the following:

y_i, $i = 1, 2, \ldots, N$ is the mean transfer time of a unit of information from level M_i to M_{i-1} (this does not include the queue wait time for type B memory levels) and y_1 is simply the mean access time of the fastest memory.

$H(x)$ is the probability of finding the required information in a memory level with capacity x.

The hit-ratio function p_i is therefore given by the difference in probability of finding the information in M_i but not finding it in M_{i-1}.

$$\text{i.e.,} \quad P_i = H(x_i) - H(x_{i-1}),$$
$$i = 1, 2, \ldots, N; \quad H(x_0) = 0. \qquad (2.1)$$

The miss ratio $F(x)$ is simply $1 - H(x)$ and is assumed to be a power function of capacity x, a positive constant K_1 and α, defined as:

$$F(x) = K_1 x^{-\alpha} \qquad (2.2)$$

The technology cost function (i.e., unit cost of a storage level with transfer time y to the next higher level) is assumed to take the form:

$$b(y) = K_2\, y^{-\beta} \qquad (2.3)$$

where β and K_2 are positive constants. Without loss of generality, we take $K_1 = K_2 = 1$, i.e., equations (2.2) and (2.3) become

$$F(x) = x^{-\alpha} \qquad (2.2')$$

and

$$b(y) = y^{-\beta}. \qquad (2.3')$$

This means $K_1^{-1/\alpha}$ is the unit for storage capacity and K_2 is the unit for cost. Empirical data have shown that equations (2.2) and (2.3) are good approximations for the hit-ratio function and technology cost function respectively (see [Chow74], [Mats71],[Rege76], [Welc78]). Mattson's [Mats71] empirical hit-ratio data , for example, can be approximated by a power function , Rege's data [Rege76] support a $\beta = 0.5$ in (2.3) if system costs rather than component costs are used. However, as stated in [Welc78], because of ambiguities due to 1) system costs versus component costs, 2) rapid change in technology, and 3) approximations needed to estimate the average access times on non-random access devices such as discs, tapes etc., β (in (2.3')) may take on widely different values (the range of 0.2 to 0.6 has been used in [LiMa72]) depending on the particular application being analyzed. Similarly, due to different program behaviour and storage management strategies, the values of α in (2.2') will vary from application to application.

2.3 Queueing Model

The queueing model of the system described in the previous section is shown in Figure 2.2.

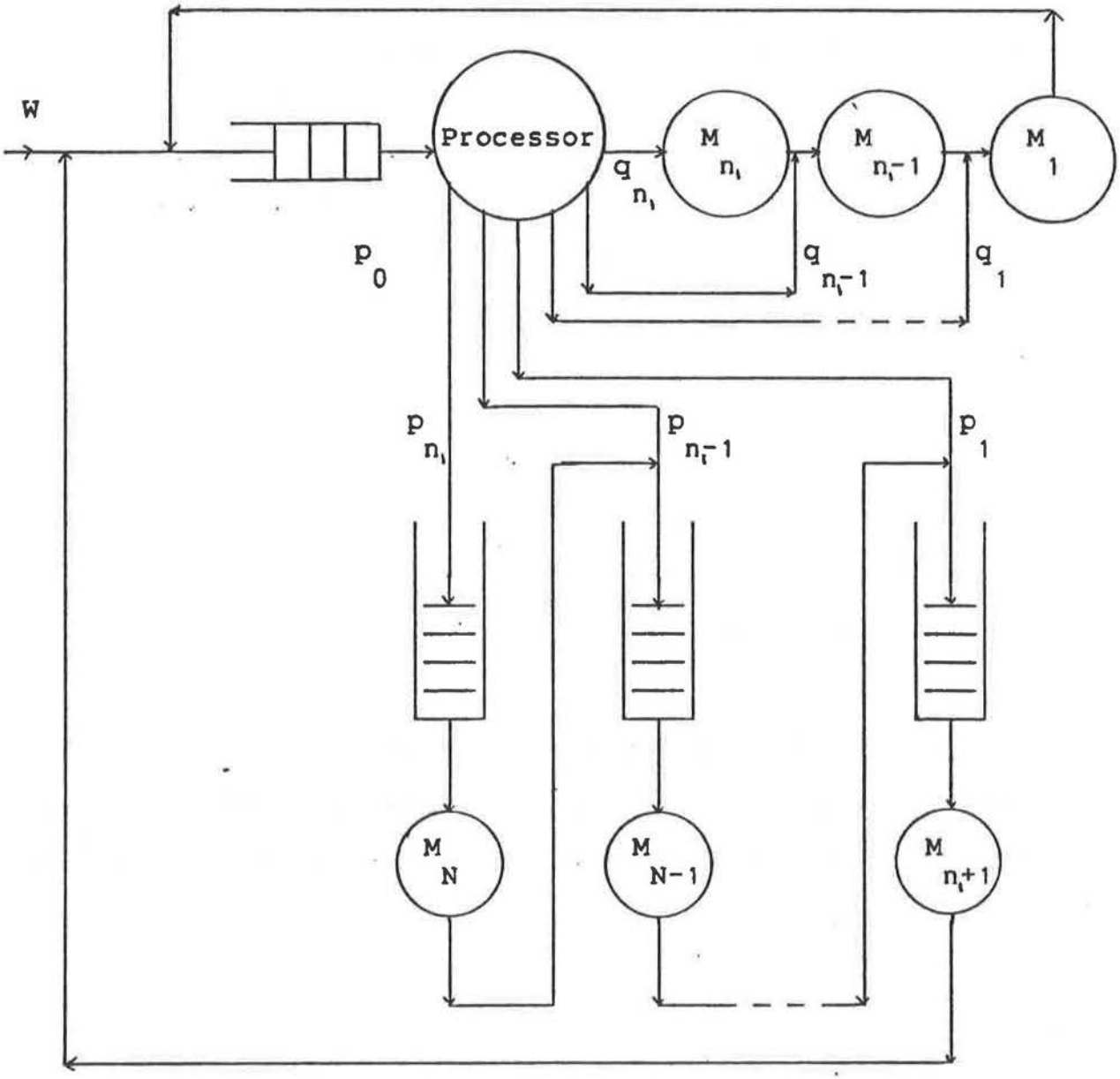

FIGURE 2.2: QUEUEING NETWORK MODEL "A" OF SYSTEM

The arrival pattern of requests is assumed to follow a Poisson distribution with mean arrival rate ω. $q_1, q_2, \ldots, q_{n_1}$ are the probabilities of referencing memory levels $M_1, M_2, \ldots, M_{n_1}$ respectively and $p_1, p_2, \ldots, p_{n_2}$ are the probabilities of referencing memory levels $M_{n_1+1}, \ldots, M_{n_1+n_2}$ respectively. The probability of exit (i.e., termination of task or completion of a request) is p_0.

Define

$$Q = \sum_{i=1}^{n_1} q_i,$$

$$P = \sum_{i=0}^{n_2} p_i,$$

then clearly, $P + Q = 1$.

For type A memory, the mean effective hierarchy access time T_i to level $M_i (i \leq n_1)$ is the sum of the mean individual transfer time between two consecutive levels from M_i up to M_1

$$\text{i.e.,} \quad T_i = \sum_{j=1}^{i} y_j .$$

In the case of type B memory the mean transfer times y_i's are taken as the inverse of the service rates of the memory levels. Furthermore, it is assumed that the service rates are exponentially distributed with mean $1/y_i$, $i = n_1 + 1, \ldots, n_1 + n_2$. The mean effective hierarchy access time of type B memory levels are not so easily obtained because of possible queueing of requests at these levels. Furthermore, since a process may be blocked while accessing these levels, it is more reasonable to use mean response time (or mean request completion time) as the criterion of optimization. To do so, we first transform the model in Figure 2.2 to the model in Figure 2.3.

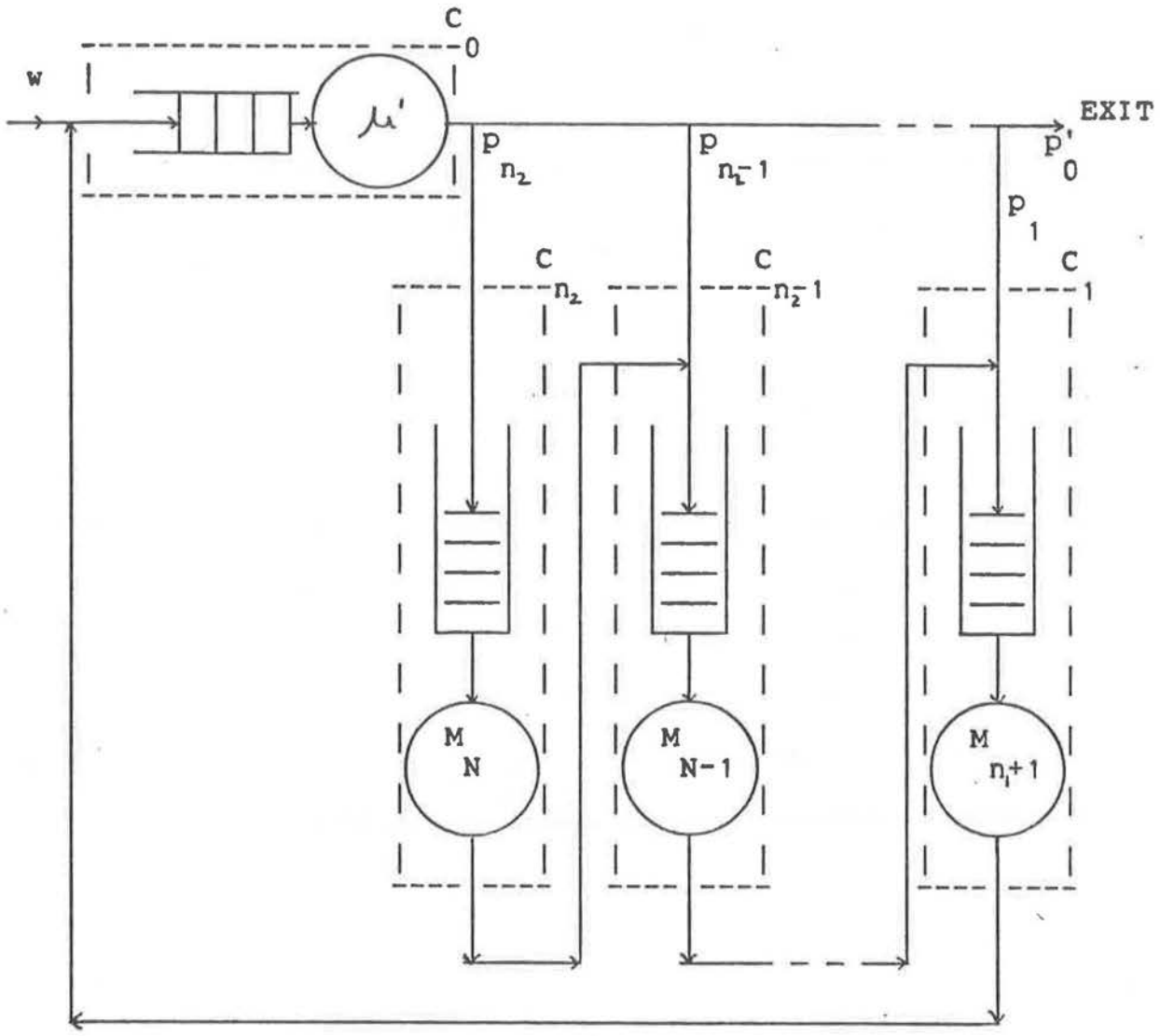

FIGURE 2.3: QUEUEING NETWORK MODEL "B" OF SYSTEM

Here $p_i' = p_i/P$ and the mean service time of centre C_0 is

$$\mu' = P.K$$

with $$1/K = \sum_{i=1}^{n_1} q_i/\mu_i$$

and $$\mu_i = 1/T_i \quad i = 1, 2, \ldots, n_1 \qquad (2.4)$$

Taking $\omega + \omega_1, \omega_1, \omega_2, \ldots, \omega_{n_2}$ to be the arrival rates of centres $C_0, C_1, \ldots, C_{n_2}$, respectively, and assuming the service disciplines of the centres do not depend on the future service time requirements or on the future path of a job through the network, we now compute the response time of the network in model B.

Assuming that the interarrival-time distribution and all service time distributions are exponential, the model can be analyzed following Jackson's [Jack63] approach of considering each service centre as an M/M/1 queueing model.

Using Little's Law, the mean response time of the network is given by

$$R = \frac{1}{\omega}\left[\sum_{i=1}^{n_2} \text{mean number of jobs in centre } i\right]$$

$$= \frac{1}{\omega}\sum_{i=1}^{n_2} \frac{\rho_i}{1-\rho_i}$$

where

ρ_i = (mean job arrival rate of service centre C_i / mean service rate of centre C_i)

Now the mean job arrival rate for service centre C_i

$$= \begin{cases} \omega + \omega_1 & i = 0 \\ \omega_i & i = 1, 2, \ldots, n_2 \end{cases}$$

and the mean service rate of service centre C_1

$$= \begin{cases} \mu' & i = 0 \\ y_{1+i}^{-\beta} & i = 1, 2, \ldots, n_2 \end{cases}$$

Hence
$$R = \frac{1}{\omega} \left[\frac{(\omega+\omega_1)\rho'}{1-(\omega+\omega_1)/\mu'} + \sum_{i=n_1+1}^{n_1+n_2} \frac{\omega_{i-1} y_i}{1-\omega_{i-1} y_i} \right] \qquad (2.5)$$

From Figure 2.3, job arrival rate for service centre C_i is given by

$$\omega_{n_2-i+1} = (\omega + \omega_1)\, p'_{n_2-i+1} + \omega_{n_2-i+2}$$
$$i = 1, 2, \ldots, n_2$$

which, after some algebraic manipulation, can be shown to be equivalent to

$$\omega_i = (\omega + \omega_1) \sum_{i=1}^{n_2} p_i'$$

$$= \frac{\omega}{p_0} \sum_{j=i}^{n_2} p_i \, . \qquad (2.6)$$

From equation (2.1), for type B memory, we have

$$\sum_{j=i}^{n_2} p_i = \sum_{j=i}^{n_2} [\, H(x_{j+n}) - H(x_{i+n_1-1}) \,]$$

(taking $H(x_N) = 1$)

$$= 1 - H(x_{i+n_1-1})$$

$$= x_{i+n_1-1}^{-\alpha}$$

Substituting in (2.6), we get

$$\omega_i = \mu\, x_{i+n_1-1}^{-\alpha}$$

where $\mu = \dfrac{\omega}{p_0}$

$$\Rightarrow \quad \omega_{i-1} = \mu\, x_{i-1}^{-\alpha} \tag{2.7}$$

Also $$(\omega+\omega_1)/\mu' = \frac{\omega}{p_0} \sum_{i=1}^{n_1} q_i \sum_{j=1}^{i} y_j.$$

Substituting

$$q_i = H(x_i) - H(x_{i-1})$$

and $$F(x_i) = 1 - H(x_i) = x_i^{-\alpha},$$

we have, $$(\omega+\omega_1)/\mu' = \mu \sum_{i=1}^{n_1} x_{i-1}^{-\alpha}\, y_i. \tag{2.8}$$

Substituting (2.7) and (2.8) into (2.5) we obtain

$$R = \frac{1}{p_0}\left[\frac{\sum_{i=1}^{n_1} x_{i-1}^{-\alpha}\, y_i}{1 - \sum_{i=1}^{n_1} \mu\, x_{i-1}^{-\alpha}\, y_i} + \sum_{i=n_1+1}^{n_1+n_2} \frac{x_{i-1}^{-\alpha}\, y_i}{1 - \mu\, x_{i-1}^{-\alpha}\, y_i} \right] \tag{2.9}$$

where $x_0 = 1$.

Notice that the degree of multiprogramming is not explicitly represented in the model. It is generally

recognized that, for a given size of the main memory, the degree of multiprogramming affects the performance of the system. Some systems try to maintain the degree of multiprogramming at a fixed level. However, except for some simple systems where fixed partition memory management is used, the resident sets of active processes change drastically with time and even the average size differs from application to application. The influence of the degree of multiprogramming on performance is largely due to its effect on the miss ratio function. In fact, in a paging system using a fixed allocation strategy, the mean CPU time interval between consecutive page faults $\hat{e}$ was found to be proportional to a power function of the size of the resident set m [BeKu69]

$$\hat{e} \propto m^{\kappa}$$

If main memory is shared equally by the active processes then there is a direct relationship between the values of parameter α of the miss-ratio function and K.

The system designer is not as much interested in the absolute values of the degree of multiprogramming as in its effect on system performance (which in this model is represented by the value of α in (2.2)).

2.4 Formulation of the Optimization Problem

Since the technology cost per unit of information is given by

$$b(y) = y^{-\beta}$$

the system cost with storage sizes $x_1, x_2, \ldots, x_N$ for levels $M_1, M_2, \ldots, M_N$ with average transfer time $y_1, y_2, \ldots, y_N$ respectively, is given by

$$S = \sum_{i=1}^{N} x_i\, y_i^{-\beta} \tag{2.10}$$

Given N, x_N, and that the memory system cost is not to exceed S_0, the optimization problem becomes:

$$\text{Min} \quad \frac{\sum_{i=1}^{n_1} x_{i-1}\, y_i^{-\alpha}}{1-\sum_{i=1}^{n_1} \mu\, x_{i-1}\, y_i^{-\alpha}} + \sum_{i=n_1}^{n_1+n_2} \frac{x_{i-1}\, y_i^{-\alpha}}{1-\mu\, x_{i-1}\, y_i^{-\alpha}}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} x_i\, y_i^{-\beta} \le S_0,$$

$$x_0 = 1; \quad x_i > 0; \quad y_i > 0 \quad i = 1, 2, \ldots, N \tag{2.11}$$

The problem (2.11) will have a solution only in the region where

$$\sum_{i=1}^{n_1} \mu\, x_{i-1}\, y_i^{-\alpha} < 1$$

and $$\mu\, x_{i-1}\, y_i^{-\alpha} < 1 \quad i = n_1 + 1, \ldots, n_1 + n_2$$

This restriction meets one of the assumptions made while calculating the equilibrium state probability, i.e., the

traffic intensity[1] has to be strictly less than one [Ferr78].

Now by multiplying the objective function by μp_0 (a constant) and adding $1 + n_2$ to it, the problem (2.11) reduces to

$$\text{Min} \quad \frac{1}{1-\sum_{i=1}^{n_1} \mu^{-\alpha} x_{i-1}^{-\alpha} y_i} + \sum_{i=n_1}^{n_1+n_2} \frac{1}{1-\mu^{-\alpha} x_{i-1}^{-\alpha} y_i}$$

$$\text{s.t.} \quad \frac{1}{S_0} \sum_{i=1}^{N} x_i^{-\beta} y_i^{-\beta} \leq 1 \qquad (2.12)$$

The natural constraints $x_i > 0$ and $y_i > 0$ can be ignored in the calculation by looking at a solution only in the positive region of x and y.

Introducing new variables r_0 and r_i's such that

$$r_0 \leq 1 - \sum_{i=1}^{n_1} \mu^{-\alpha} x_{i-1}^{-\alpha} y_i$$

and $$r_{i-1} \leq 1 - \mu^{-\alpha} x_{i-1}^{-\alpha} y_i; \quad i = n_1 + 1, \ldots, n_1+n_2$$

The problem (2.12) is equivalent to

$$\text{Min} \quad r_0^{-1} + \sum_{i=1}^{n_2} r_i^{-1}$$

---

[1] Traffic intensity is defined as the ratio of the arrival rate to the service rate at a service centre.

$$\text{s.t.} \qquad r_0 \le 1 - \sum_{i=1}^{n_1} \mu\, x_{i-1}^{-\alpha}\, y_i$$

$$r_{i-1} \le 1 - \mu\, x_{i-1}^{-\alpha}\, y_i; \quad i = n_1+1, \ldots, n_1+n_2$$

$$\text{and} \quad \frac{1}{S_0} \sum_{i=1}^{N} x_i^{-\beta}\, y_i < 1. \qquad (2.13)$$

Seemingly the above problem (2.9) falls under the framework of the standard Geometric Programming. However, it does not satisfy one of the conditions required, i.e., the number of variables minus the number of constraints equals one. The difference here is $n_1 + 2$. Therefore we expect an $n_1 + 1$ parametric solution. Now when we write the Lagrangian equations for this problem, we find that exactly $n_1 + 1$ equations can be derived from the rest of the equations. Therefore in theory we should be able to eliminate the above $n_1 + 1$ parameters. However, in doing so, the orthogonality equations thus obtained become non-linear and we are forced to solve a system of non-linear equations. Thus the very advantage of using Geometric Programming is lost. We shall therefore turn to the Lagrangian multiplier method supplemented with a technique similar (but not exactly equal) to Geometric Programming. This is why detailed derivation of the results are given rather than simply stating the results.

Before solving the problem (2.13) we shall first show that any solution to this problem is globally optimal.

Let (X^0,Y^0) be any stationary point of the problem

and $$R(X,Y) = \mu\, p_0\, \bar{R}(X,Y) + n_2 + 1$$

i.e., $$\left.\frac{\partial R(X,Y)}{\partial x_i}\right|_{(X^0,Y^0)} = 0 \quad \text{and} \quad \left.\frac{\partial R(X,Y)}{\partial y_i}\right|_{(X^0,Y^0)} = 0,$$

$$i = 1, 2, \ldots, N$$

where $\bar{R}$ is as defined in (2.5). We shall show that (X^0,Y^0) is a global minimum of $R(X,Y)$.

The following two theorems [Hadl72] are used in the proof:

Theorem 1. The sum of convex functions is convex.

Theorem 2. If g is a monotonically nondecreasing convex function defined on the convex subset $S_1 \subseteq R^1$ and if f is a convex function defined on the convex subset $S_n \subseteq R^n$ then the composite function g(f) is a convex function on S_1.

We now transform the original function $R(X,Y)$ to $R'(U,V)$ using the transformation $x_i = e^{-u_i}$ and $y_i = e^{-v_i}$ and obtain

$$R'(U,V) = \sum_{i=n_1+1}^{N} \frac{-1}{1 - \mu\, e^{-\alpha u_{i-1}+v_i}} + \frac{1}{1 - \sum_{i=1}^{n_1} \mu\, e^{-\alpha u_{i-1}+v_i}}$$

Since each of the terms e^{-u_i} and e^{-v_i} are convex in their respective variables, it is easy to show, by repeated application of Theorems 1 and 2 that the function $R'(U,V)$ is convex. It now remains to be shown that the point (U^o,V^o) in $R'(U,V)$, which corresponds to the stationary point (X^o,Y^o) in $R(X,Y)$, is also a stationary point of $R'(U,V)$ and hence a global optimal point since $R'(U,V)$ is convex.

Case 1 $\qquad i = n_1+1,\ n_1+2,\ \ldots,\ N$

$$\left.\frac{\partial R'(U,V)}{\partial u_i}\right|_{(U^o,V^o)} = \left.\frac{\partial}{\partial u_i}\left[\frac{-1}{1 - \mu\, e^{-\alpha u_i+v_{i+1}}}\right]\right|_{(U^o,V^o)}$$

$$= \left.\frac{(-\mu)(-\alpha)\, e^{-\alpha u_i+v_{i+1}}}{(1 - \mu\, e^{-\alpha u_i+v_{i+1}})^2}\right|_{(U^o,V^o)}$$

$$= \left.\frac{(-\mu)(-\alpha) x_i\, y_{i+1}}{(1 - \mu\, x_i^{-\alpha}\, y_{i+1})^2}\right|_{(X^o,Y^o)}$$

$$= \left.\frac{\partial R(X,Y)}{\partial x_i}\right|_{(X^o,Y^o)}$$

$$= 0$$

Case 2. $i = 1, 2, \ldots, n_1$

$$\frac{\partial R'(U,V)}{\partial u_i} = \frac{-\mu}{(1 - \Sigma_i \mu e^{-\alpha u_i + v_{i+1}})^2} \frac{\partial}{\partial u_i}(e^{-\alpha u_i + v_{i+1}})$$

$$= \frac{\partial R(X,Y)}{\partial x_i}\Bigg|_{(X^0,Y^0)}$$

$$= 0 \text{ (following same arguments as in Case 1)}$$

Similarly $$\frac{\partial R'(U,V)}{\partial v_i}\Bigg|_{(U^0,V^0)} = 0$$

However, (U^0,V^0) is the global minimum of $R'(U,V)$, and the transformation between (X,Y) and (U,V) is unique. Therefore (X^0,Y^0) is the global minimum of $R(X,Y)$.

Using similar arguments it can be shown that the constraint problem (2.9) has a global minimum.

Trivedi and Sigmon [TrSi80] have independently shown [ChTr79] that this design problem in general has a global optimal solution although their objective function was different and they used a closed queueing network model.

2.5 Solution of the Optimization Problem

$$F(R,X,Y,\lambda) = r_0^{-1} + \sum_{i=1}^{n_2} r_i^{-1} + \lambda_0\left(r_0 + \sum_{i=1}^{n_1} x_{i-1}^{-\alpha}\, y_i - 1\right)$$

$$+ \sum_{i=n_1+1}^{n_1+n_2} \lambda_{i-1}\left(r_{i-1} + \mu\, x_{i-1}^{-\alpha}\, y_i - 1\right)$$

$$+ \lambda'\left(\frac{1}{S_0} \sum_{i=1}^{N} x_i\, y_i^{-\beta} - 1\right) \qquad (2.14)$$

Differentiating with respect to R, X, Y and λ respectively and equating to zero, we obtain,

$$\frac{\partial F}{\partial r_0} = (-1)\, r_0^{-1} + \lambda_0 r_0 = 0 \qquad (2.15)$$

$$\frac{\partial F}{\partial r_i} = (-1)\, r_i^{-1} + \lambda_i r_i = 0;\ i = 1, 2, \ldots, n_2 \qquad (2.16)$$

$$\frac{\partial F}{\partial x_i} = \begin{cases} (-\alpha)\mu\lambda_0 x_i^{-\alpha}\, y_i + \dfrac{\lambda'}{S_0}\, x_{i-1}\, y_{i-1}^{-\beta} = 0; \\ \qquad i = 1, 2, \ldots, n_1+n_2 \qquad (2.17) \\ (-\alpha)\mu\lambda_{i-1}\, x_i^{-\alpha}\, y_i + \dfrac{\lambda'}{S_0}\, x_{i-1}\, y_{i-1}^{-\beta} = 0; \\ \qquad i = n_1+1, \ldots, n_1+n_2 \qquad (2.18) \end{cases}$$

$$\frac{\partial F}{\partial y_i} = \begin{cases} \mu\lambda_0 x_{i-1}^{-\alpha}\, y_i + (-\beta)\dfrac{\lambda'}{S_0}\, x_i\, y_i^{-\beta} = 0; \\ \qquad i = 1, 2, \ldots, n_1 \qquad (2.19) \\ \mu\lambda_{i-1} x_{i-1}^{-\alpha}\, y_i + (-\beta)\dfrac{\lambda'}{S_0}\, x_i\, y_i^{-\beta} = 0; \\ \qquad i = n_1+1, \ldots, n_1+n_2 \qquad (2.20) \end{cases}$$

$$\frac{\partial F}{\partial \lambda_0} = r_0 + \sum_{i=1}^{n_1} \mu\, x_{i-1}^{-\alpha}\, y_i - 1 = 0 \qquad (2.21)$$

$$\frac{\partial F}{\partial \lambda_{i-1}} = r_{i-1} + \mu x_{i-1}^{-\alpha} y_i - 1 = 0$$

$$i = n_1+1, \ldots, n_1+n_2 \qquad (2.22)$$

$$\frac{\partial F}{\partial \lambda'} = \frac{1}{S_0} \sum_{i=1}^{N} x_i y_i^{-\beta} - 1 = 0 \qquad (2.23)$$

Define $$f^0 = \lambda_0 \sum_{i=1}^{n_1} \mu x_{i-1}^{-\alpha} y_i + \sum_{i=n_1+1}^{n_1+n_2} \lambda_{i-1} \mu x_{i-1}^{-\alpha} y_i \qquad (2.24)$$

$$\delta_0 = r_0^{-1} \qquad (2.25)$$

$$\delta_i = r_i^{-1} \qquad i = 1, \ldots, n_2 \qquad (2.26)$$

$$\delta_{1i} = \begin{cases} \lambda_0 \mu x_{i-1}^{-\alpha} y_i / f^0 \\ \qquad i = 1, 2, \ldots, n_1 \qquad (2.27) \\ \lambda_{i-1} \mu x_{i-1}^{-\alpha} y_i / f^0 \\ \qquad i = n_1+1, \ldots, n_1+n_2 \qquad (2.28) \end{cases}$$

$$\delta_{2o} = \lambda_0 r_0 \qquad (2.29)$$

$$\delta_{2i} = \lambda_i r_i \qquad i = 1, 2, \ldots, n_2 \qquad (2.30)$$

$$\delta_{3i} = \frac{\lambda'}{S_0 f^0} x_i y_i^{-\beta} \qquad i = 1, 2, \ldots, n_1+n_2 \qquad (2.31)$$

Now clearly,

$$\sum_{i=1}^{n_1+n_2} \delta_{1i} = 1 \qquad \text{(normality)} \qquad (2.32)$$

$$(2.15) \Rightarrow (-1)\ \delta_0 + \delta_{2o} = 0 \qquad (2.33)$$

$$(2.16) \Rightarrow (-1)\ \delta_i + \delta_{2i} = 0 \quad i=1, 2, \ldots, n_2 \qquad (2.34)$$

$$(2.17) \text{ and } (2.18) \Rightarrow (-\alpha)\ \delta_{1i} + \delta_{3i-1} = 0$$

$$i=2, \ldots, n_1+n_2 \qquad (2.35)$$

$$(2.19) \text{ and } (2.20) \Rightarrow \delta_{1i} + (-\beta)\,\delta_{3i} = 0$$

$$i = 1, 2, \ldots, n_1+n_2 \qquad (2.36)$$

Solving (2.32), (2.35), and (2.36) simultaneously, we obtain

$$\delta_{1i} = (\alpha\beta)^{N-i} \frac{\alpha\beta - 1}{(\alpha\beta)^{N} - 1}$$

$$\delta_{3i} = \frac{1}{\beta}\,\delta_{1i}$$

$$N = n_1+n_2; \qquad i = 1, 2, \ldots, n_1 + n_2 \qquad (2.37)$$

Now in order to obtain optimal values for x_i's and y_i's we first obtain values for f^0 and λ_i's.

Raising (2.27) and (2.28) to the power δ_{1i} and (2.31) to the power δ_{3i} for $i = 1, 2, \ldots, n_1 + n_2$ respectively, and then multiplying we obtain:

$$(f^0)^{\sum_{1}^{n_1+n_2} \delta_{1i}} = \lambda_0^{a_{n_1}} \prod_{i=1}^{n_2} \lambda_i^{b_i} \cdot c \qquad (2.38)$$

where $a_{n_1} = \sum_{i=1}^{n_1} \delta_{1i}$

$$b_i = \delta_{1i+1}, \qquad i = 1, 2, \ldots, n_2$$

$$c = \mu(1/\beta S_0)^{1/\beta} \cdot x_N \cdot \prod_{i=1}^{n_1+n_2} (1/\delta_{1i})^{\delta_{1i}} (1/\delta_{3i})^{\delta_{3i}}$$

$$\frac{\lambda'}{f^0} = \frac{1}{\beta}$$

From (2.21), (2.25), (2,27), (2.29), and (2.33)

$$\lambda_0 = (2 f^0 a_{n_1} + \sqrt{4 f^0 a_{n_1} + 1}) / 2 \qquad (2.39)$$

and from (2.22), (2.26), (2,28), (2.30), and (2.34)

$$\lambda_i = (2 f^0 b_i + 1 + \sqrt{4 f^0 b_i + 1}) / 2 \qquad (2.40)$$

Substituting (2.39) and (2.40) in (2.38)

$$f^{0*} = c \cdot (2 f^0 a_{n_1} + 1 + \sqrt{4 f^0 a_{n_1} + 1})^{a_{n_1}} \cdot \prod_{i=1}^{n_2} (2 f^0 b_i + 1 + \sqrt{4 f^0 b_i + 1})^{b_i} / 2 \qquad (2.41)$$

Substituting the values of f^0 in (2.39) and (2.40) we obtain values for the λ_i's.

From (2.28), (2.31), (2.25) and (2.36)

$$x_i \, y_i^{-\beta} = (\alpha\beta) \, x_{i+1} \, y_{i+1}^{-\beta}$$

$$i = 1, 2, \ldots, n_1+n_2-1 \qquad (2.42)$$

$$x_{i-1}^{-\alpha} \, y_i = (\alpha\beta) \, x_i^{-\alpha} \, y_{i+1}$$

$$i = 1, 2, \ldots, n_1-1$$

$$\lambda_0 \, x_{n_1-1}^{-\alpha} \, y_{n_1} = (\alpha\beta) \, \lambda_1 \, x_{n_1}^{-\alpha} \, y_{n_1+1}$$

$$\lambda_{i-1} \, x_{i-1}^{-\alpha} \, y_i = (\alpha\beta) \, \lambda_{i-1+1} \, x_i^{-\alpha} \, y_{i+1}$$

$$i = n_1+1, \ldots, n_1+n_2-1 \qquad (2.43)$$

From (2.42) and (2.43)

$$\left(\frac{x_{i-1}}{x_i}\right)^{\alpha\beta} = \left(\frac{x_i}{x_{i+1}}\right)^{\alpha\beta} (\alpha\beta)^{-(\beta+1)} \quad i = 1, 2, \ldots, n_1-1$$

$$\lambda_{i-1}\left(\frac{x_{i-1}}{x_i}\right)^{\alpha\beta} = (\alpha\beta)^{-(\beta+1)}\,\lambda_{i-1+1}\left(\frac{x_i}{x_{i+1}}\right)^{\alpha\beta}$$

$$i = n_1, \ldots, n_1+n_2-1 \qquad (2.44)$$

Taking log

$$a\, z_{i-1} = K + (1+a)\, z_i - z_{i+1}$$

$$i = 1, 2, \ldots, n_1-1$$

$$a\, z_{i-1} = K + (1+a)\, z_i - z_{i+1} + h_{i-1}$$

$$i = n_1, \ldots, n_1+n_2-1 \qquad (2.45)$$

where $z_i = \log x_i$; $\quad a = \alpha\beta$;

$K = -(\beta+1) \log a$; $h_i = \beta\,(\log \lambda_i - \log \lambda_{i+1})$

Adding (2.45) for $i = 1, 2, \ldots, n_1+n_2-1$

$$\sum_{i=1}^{N-1} a\, z_{i-1} = K(N-1) + (1+a) \sum_{i=1}^{N-1} z_i - \sum_{i=1}^{N-1} z_{i+1} + \sum_{i=n_1}^{N-1} h_{i-1} \qquad (2.46)$$

Simplifying (2.46)

$$a\, z_0 = k(N-1) + z_1 + a\, z_{N-1} - z_N + L \qquad (2.47)$$

where $L = \sum_{i=n_1}^{N-1} h_{i-n_1} = \sum_{i=0}^{n_2-1} h_i = \beta(\log \lambda_0 - \log \lambda_{n_2})$

Also multiplying (2.45) by $(1/a^i)$ and then adding for $i = 1, 2, \ldots, N-1$

$$\Rightarrow \quad a^N z_0 = K \sum_{i=0}^{N-1} a^i + a^N z_1 + a\, z_{1-1} - a\, z_1 + M \qquad (2.48)$$

where $M = a^N \sum_{i=0}^{n_2-1} h_i / a^{n_1+i} = \sum_{i=0}^{n_2-1} h_i\, a^{n_2-i}$

From (2.47) and (2.48) (taking $z_0 = 0$)

$$0 = K[(N-1) - \sum_{i=1}^{N-1} a^i] + (1-a^N)z_1 - (1-a)z_N + (L-M)$$

$$\Rightarrow \quad z_1 = \left[\frac{1}{1-a} - \frac{N}{1-a^N}\right] K + \frac{1-a}{1-a^N} z_1 + \frac{L-M}{1-a^N} \qquad (2.49)$$

Again from (2.44) it can be shown that

$$z_i = \left[\frac{i}{1-a} - \frac{(1-a^i)N}{(1-a)(1-a^N)}\right] K + \frac{1-a^i}{1-a^N} z_N$$

$$+ \frac{1-a^i}{1-a}\left[\frac{L-M}{1-a^N}\right] \qquad i = 1, 2, \ldots, n_1-1 \quad (2.50a)$$

Similarly it can be shown that

$$z_i = \left[\frac{i}{1-a} - \frac{(1-a^i)N}{(1-a)(1-a^N)}\right] K + \frac{1-a^i}{1-a^N} z_N$$

$$+ \frac{1-a^i}{1-a}\left[\frac{L-M}{1-a^N}\right] + \sum_{j=0}^{i-n_1} h_j a^{i-j}$$

$$i = n_1, \ldots, N-1; \qquad (2.50b)$$

Hence all z_i's can be computed using (2.50a) and (2.50b). From (2.37)

$$\delta_{3i} = (1/\beta)\delta_{1i}$$

$$\Rightarrow \qquad y_i = (x_i/S_0\delta_{1i})^{1/\beta} \qquad i = 1, 2, \ldots, n; \qquad (2.51)$$

Hence knowing all x_i's all y_i's can be computed from (2.51).

If we substitute $\mu = 0$ in the expression for the steady state residence time (2.9) of a multiprogrammed system, we obtain the expression for the uniprogrammed system. Intuitively also, the number of jobs in the system

is reduced as the arrival rate in an open network decreases. As a result, the interaction among the jobs also goes down. In the limiting case, as the arrival rate tends to zero, the mean response time coincides with that of the uniprogrammed case. Now letting $\mu = 0$ in (2.50a) and (2.50b) we obtain,

$$\log x_i = \left[\frac{i}{1-a} - \frac{(1-a^i)N}{(1-a)(1-a^N)}\right] \cdot K + \frac{1-a^i}{1-a^N} \log x_N; \qquad i = 1, 2, \ldots, N.$$

This is the same result obtained by Chow [Chow74]. The following example illustrates how the optimal solution for a multiprogrammed environment approaches the ones for a uniprogrammed environment as μ tends to zero.

Example

For the case $N = 4$, $n_1 = n_2 = 2$, and $\alpha = \beta = 1$, the expression for optimal x_i's are

$$x_1 = x_4^{0.25} \left[\frac{\lambda}{\lambda_1}\right]^{0.5}$$

$$x_2 = x_4^{0.5} \left[\frac{\lambda_2}{\lambda_1}\right]^{2}$$

$$x_3 = x_4^{0.75} \left[\frac{\lambda_2}{\lambda_1}\right]^{2.5}$$

Taking $x_4 = 10^8$ and $S_0 = 4 * 10^{10}$, (the actual units depend upon the normalization factors used in equations (2.2) and (2.3)) the values of x_i's are given in Table 1 for different values of μ. The values of the speeds of the

memory hierarchy are obtained by dividing the values of x in Table 1 by 10^{10}. Thus for $\mu = 0$, the speeds for M_1, M_2, M_3, and M_4 are 10ns, 1μs, 100μs and 10ms, respectively.

μ	x_1	x_2	x_3
100	100.08	10031	1003905
1000	100.77	10313	1039305
10000	107.24	13224	1418069
0	100.00	10000	1000000

Table 2.1

Optimal Storage Sizes for N=4, $x_4=10^8$, $S_0=4*10^{10}$, $\alpha=\beta=1$

The values of x's for $\mu = 0$ are the ones obtained by Chow for a uniprogrammed system.

From the data gathered from an Amdhal 470 v/6 model II running the Michigan Terminal System, we estimate that $\omega \simeq 5$ request/sec. in a moderately heavy load and p_0, which differs widely from process to process, is in the range of 1/1000 to 1/10,000. Hence $\mu(=\omega p_0) = 10{,}000$ is not unreasonable. From Rege's empirical data [Rege76], α is roughly equal to 1 and the storage unit K_1 is about 200 bytes. Thus the optimal solution for $\mu = 10{,}000$ (Table 1) is $x_1 \simeq$ 21K bytes, $x_2 \simeq$ 2.6M bytes, $x_3 \simeq$ 280 M bytes and $x_4 \simeq$ 20 B bytes. If the unit of cost is 0.00055¢[2], S_0 will be approximately $200,000.

Because of the closed form results we are able to make the following important observations. From equations (2.50a) and (2.50b) we find that the optimal sizes of the various levels of memory are independent of the given system cost constraint S_0. An intuitive explanation lies in the often cited observation that about 10% of the memory is accessed 90% of the time [Welc78]. Thus, after a certain point, the miss-ratio will decrease only marginally with an increase in memory size. Thus one can conclude that once each level in the hierarchy has the optimal size given by (2.50a) and (2.50b), any additional money should be invested in acquiring faster memory rather than more memory. On the other hand, the optimal sizes of memory hierarchy is rather sensitive to the miss-ratio parameter α. The smaller the value of α in (2.2) (and hence the smaller the values of a, K, and M in (2.50a) and (2.50b)) the larger the values of the x_i's.

Furthermore, we observe that there is a considerable difference between uniprogrammed and multiprogrammed results when μ is large (i.e., when arrival rate of requests is large and/or the exit probability of a process is small which implies a large number of active process competing for limited system resources simultaneously).

---

[2] The cost unit is obtained by using 1979 DEC prices and assuming $\beta = 1$. Although $\beta = 1$ gives a poor fit of the available data, it is used so that the results can be compared to Chow's [Chow74].

Finally, Chow [Chow74] reported that in a uniprogrammed system, the ratios of the capacities as well as the speeds of adjacent levels of memory for the optimal configuration are constant if $\alpha\beta = 1$. This constancy in the ratio of capacities was also emperically observed (see reference 11 in [Chow74]). Equations (2.50a) and (2.50b) show that in a multiprogrammed environment this is true for the first n_1 levels and for the next n_2 levels separately.

The results presented in this chapter are not intended to be used directly by the designers and evaluators in the final configuration. This is because memory may not be available with the capacity/speed characteristics computed. Also some of the assumptions used in the formulation and the solution of the problem may not be satisfied in practice. However, they are useful as guidelines and as an configuration in an iterative design technique (see for example, chapter 8 of [Ferr78]).

CHAPTER 3

ADAPTIVE LOAD CONTROL

3.1 Introduction and Review

One of the principle ideas behind multiprogramming is to make more effective use of the system resources, many of which can be simultaneously utilized. However, in order to avoid excessive interactions among the competing jobs, which will result in general degradation of system performance, the number and composition of jobs in a multiprogramming set should be carefully controlled. The policy that controls the flow of jobs in a multiprogramming system is called load control policy. Therefore, in order to provide an acceptable level of service, most multiprogramming computer systems employ some form of load control policy.

Load control policies are typically built around maintaining two sets of queues, often called the eligible queues and the multiprogramming queues. Jobs in the

eligible queues must wait until the control policy decides (depending upon the system state or some other criteria) to move them to the multiprogramming queues. Only jobs in the multiprogramming queues are allowed to actively share the system's resources.

In a large computer installation or a distributed computing environment, generally a large number of jobs with different characteristics and resource demands are executing at the same time. The number of jobs competing for limited system resources changes from time to time. Their resource demands also change with time. For such a system , a static control policy (one which is insensitive to job characteristics and job-mix) cannot be expected to give good performance at all times.

There are various load control mechanisms available which are optimal under various conditions. The optimality of a control mechanism is always relative to the objective function selected (i.e., the performance measure) and a set of constraints. A control mechanism that is optimal with respect to one performance measure may not be optimal with respect to others. Furthermore, a control mechanism may or may not be realizable because of the underlying assumptions. The one that is unrealizable cannot be implemented in practice but can be used as a standard for comparison.

A natural load control mechanism, particularly for paging systems, is accomplished by controlling the degree of multiprogramming. Previous work directed towards this end is primarily represented by the development of the working set policy[1]([Denn68b],[Denn70],[Denn71],[RoDu73]). More recently, efforts to optimize the system work capacity lie mainly in keeping some measure related to program behaviour (usually paging behaviour) within some predetermined bounds ([DKLP76],[DeKh76],[LePo76],[GrDe77]). The 50% criterion [LePo76] for example, aims at maintaining the utilization of the paging device to around 0.5. The L=S criterion [DeKh76] proposes to keep the system lifetime approximately equal to that of the page swap time. The Knee criterion ([DKLP76],[GrDe77]) suggests that the mean resident set of each process should be maintained at the value associated with the primary knee[2] of its life-time function, where life time is defined to be the mean virtual time between two successive page faults [DKLP76]. Though the most robust of the three, the Knee criterion also involves considerable amount of overhead.

The above mentioned criteria are not based on

---

[1] According to this policy, the number of jobs allowed in the multiprogramming queue is limited such that the sum of their working sets can be accomodated in main memory. The working set $W(t,\tau)$ of a process at any time t, with window size τ, is the set of distinct pages referenced by the process during the most recent time interval of length τ.

[2] The point where the curve of the system life time vs number of active jobs has maximum slope.

mathematical models and are not proven optimal. There are conditions under which they may perform poorly. If the system is execution bound, the 50% criterion does not work well. If the system is both execution and I/O bound, the L=S criterion does not give good results. These criteria mainly aim at increasing the throughput rate by loading the system up to the point when the measured indicator suggests that further increase in system load may cause "thrashing" [Denn68a]. The methods are applicable only to paging systems. Furthermore, for interactive systems and combined batch-interactive systems, one is interested not only in maximizing the system throughput rate but also in guaranteeing good response times to the interactive jobs (possibly at the expense of the batch jobs).

Landwehr [Land76] studied a combined batch-interactive system and proposed a scheme to activate batch jobs depending on the terminal load. The emphasis of the study, however, was on model formulation and its validation. There was no attempt to prevent the system from saturation or to optimize performance.

Hine et al. [HiMT79] studied the problem from a slightly different viewpoint. Their goal was to control the main memory allocation for each class of job in order to provide different response time to each class while maximizing the CPU utilization. They employed a

mathematical model but optimization was achieved by an exhaustive search technique. A heuristic extension of the method was also given which provided good but not optimal results. Moreover, their model assumes that the distribution of service time for various service centres and the values of various system parameters are known.

In the following example we explain, using a simple queueing network model, how some of the schemes discussed above, which employ static criteria for saturation estimation, may not produce satisfactory results under various load conditions.

Example 3.1

Consider a simple closed queueing network model. In order to reduce the complexity of the model so that a graphical explanation can be given we consider a system having only one I/O unit and a CPU.

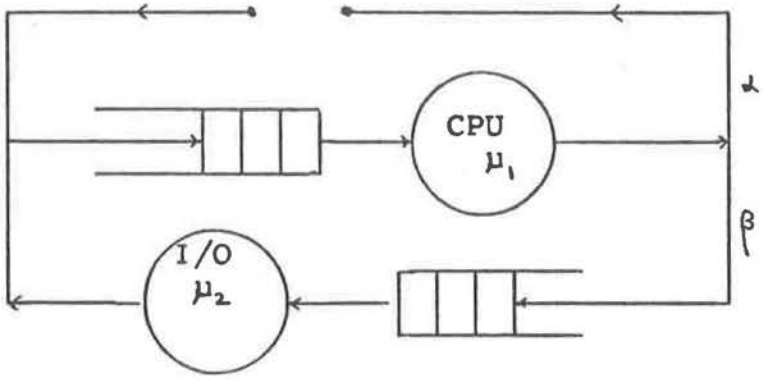

Figure 3.1 Simple Central Server Model

The response time for such a system with N jobs is given by [CoDe73]

$$R(\alpha,\mu_1,\mu_2,N) = \frac{1}{\alpha\mu_1} \cdot \frac{1-(\mu_2/\beta\mu_1)^{N+1}}{\mu_2/\beta\mu_1-(\mu_2/\beta\mu_1)^{N+1}} \cdot N \qquad (3.1)$$

where, μ_1 and μ_2 are the mean service rates of the CPU and the I/O device respectively and $\alpha(=1-\beta)$ is the probability that a job leaves the system after being served by the CPU.

The equation of the asymptote of R (i.e., as $N \rightarrow \infty$) can be shown to be

$$R = (1/\alpha\mu_1) * N. \qquad (3.2)$$

As the system load increases the slope of the response time vs system load curve also increases. This increase is initially small. A system is considered to be saturated when the slope becomes significant. The saturation point

N* of the system can be approximated by the point of intersection of the asymptote with the horizontal line R(1) (see Figure 3.2) and is given by

$$N^* = 1 + \rho(\mu_1/\mu_2). \tag{3.3}$$

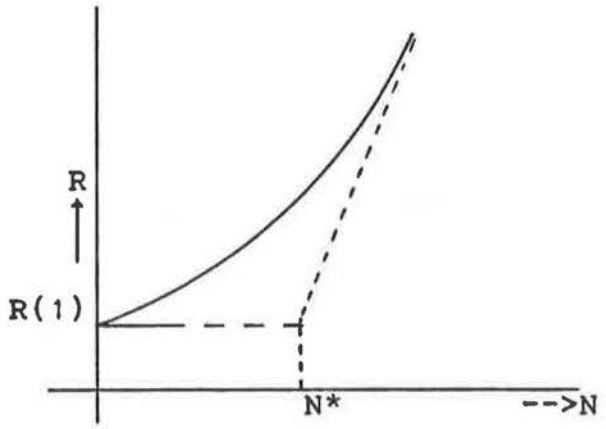

Figure 3.2 Saturation point

Thus the system is saturated when the condition

$$N \geq 1 + \rho(\mu_1/\mu_2) \tag{3.4}$$

holds.

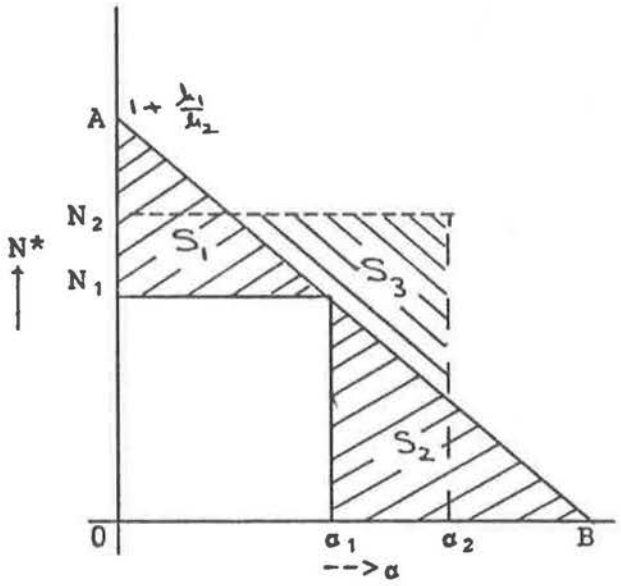

Figure 3.3 Under-saturated and Over-saturated Regions

In order to show pictorially that static schemes at times leave the system either underutilized or supersaturated, we shall assume that the system parameters μ_1 and μ_2 are constant.

Figure 3.3 is a graph of equation (3.3), relating the remaining system parameters N* and $\alpha(=1-\beta)$. Thus if the operating point of the system lies within the area enclosed between the point AOB, the system is not saturated.

The values of parameters like N* or α are fixed in most control schemes. Whenever the values of the observed parameters exceeds some predetermined fixed value, normally obtained empirically, the system is assumed to be saturated. If the control scheme uses (N_1, α_1) as the saturation point then this scheme will leave the system

under-utilized because it cannot operate in the unsaturated region $S_1 \cup S_2$. If the control parameters are (N_2, α_2) then the system is allowed to operate in the saturated region S_3.

MTS has five parameters which control the system load. If any one of the observed values exceeds some precalculated fixed value the system is considered to be saturated (for more detail see [Chan79]). If we consider this in two dimensions only (i.e., only two load parameters), then the analogy of Figure 3.3 discussed earlier can be applied directly.

The control algorithm described by Landwehr [Land76] is also insensitive to variations in job characteristics because the values of the break points b_i's (defined in the scheme [Land76]) are fixed and do not depend on the instantaneous workload characteristics.

The control scheme described by Hine et al. [HiMT79] characterizes jobs only by their page fault rate. The scheme does not take into consideration the variation in the I/O requirements of the jobs. Moreover, the use of life-time function gives only a global picture of the system paging characteristics and not the local behaviour.

It should now be clear that a control policy which

does not base its decision on job characteristics and instantataneous system conditions will not perform optimally under all circumstances. We need an adaptive control policy which is able to dynamically recompute the system saturation point as the characteristics of the workload change. From among the waiting jobs, several sets of job-mix can be selected to attain the calculated saturation point. Therefore the scheme should also be able to select the one that optimizes some system performance measure(s). Listed below are some recent work in adaptive load control policy.

Badel and Leroudier [BaLe78] have proposed an adaptive load control policy by introducing a system "dilatation" function. The system "dilatation" function is defined as the ratio of the real execution time of N programs running one at a time to the real execution time of the N programs running simultaneously (i.e., degree of multiprogramming equal to N). They observe that the "dilatation" function attains its maximum value when certain principles are satisfied. The principles are equivalent to L=S and 50% criteria. The implementation of their scheme is, in fact, implementation of these criteria. We shall see in chapter 5 that the schemes developed in this thesis are better than these criteria.

Schonbach [Scho80] describes a macro scheduler for

high productivity. It is assumed that the "system balance" point is already specified. Here, system balance is a state in which various processor utilizations are at some prespecified levels. The macro-scheduler then chooses, from among the waiting jobs, a job-mix which maintains system balance. The scheme does not include external priorities and is applicable only to non-paged systems.

Lo [Lo80] describes a load control policy using stochastic control theory. The policy is shown to give optimal results. The main weakness of the scheme is that its implementation requires the job parameter values to be known. It also makes the usual queueing theory assumptions, such as exponential interarrival and service time distributions which may not be satisfied in practice.

With the exception of Lo's and Schonbach's schemes, most of the control policies mentioned above are not based on mathematical models and are not expected to give optimal results in all situations. On the other hand, Lo and Schonbach make such strong assumptions in their formulation that it is impracticable and sometimes impossible to meet those assumptions in practice.

In the next three sections we develop a dynamic load control scheme. First, we derive an expression for the system saturation point. The expression depends upon

parameters that are directly measurable from the system, thereby reflecting instantaneous change in the state of the system. The control policy computes the optimal number of jobs that should be activated from each class so that the system is neither under-utilized nor saturated.

3.2 System Model

We here describe a queueing network model of a computer system. Figure 3.4 is a diagram of such a network which is widely cited in the literature.

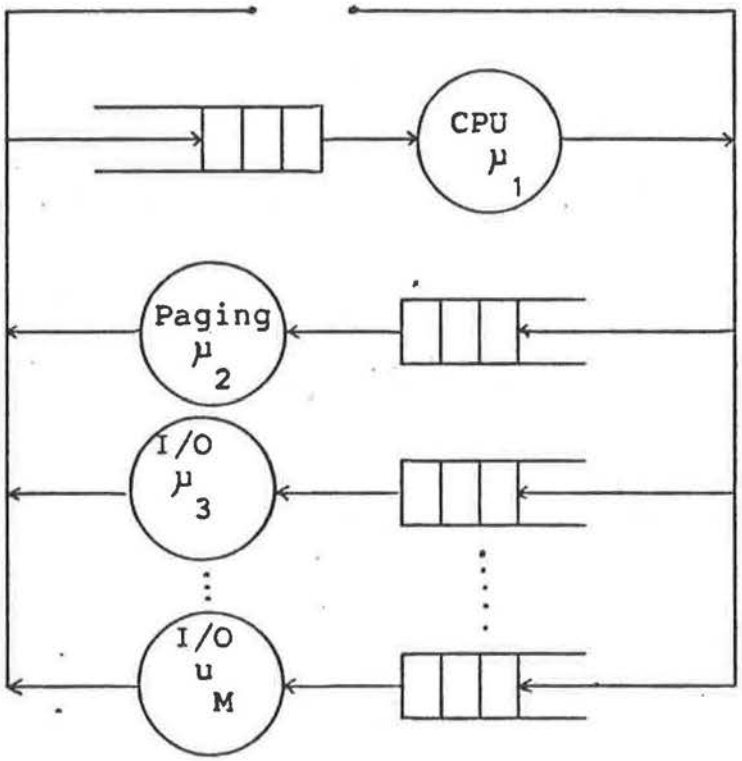

Figure 3.4 General Central Server Model

The network consists of M service centres -- a CPU, a paging device, and (M-2) I/O units. Each service centre consists of a server and an associated queue. Upon arrival, a job waits in the queue if the server is busy.

When a job completes its service at the paging device or any of the I/O units, it always rejoins the CPU queue. When a job finishes its service at the CPU, it either leaves the system or visits one of the other service centres. A new job always joins the CPU queue.

This model assumes that no job overlaps its requirement of different devices i.e., no job is being served by more than one service centre at the same time. This assumption may not hold in practice. However the error introduced is generally not significant [Buze78b].

We shall assume, for the time being, that all jobs are identical in their resource demand. Specifically, the branching frequencies from the CPU to the paging device or one of the I/O devices or out of the system is same for all jobs. Also the mean service rates at various centres are same for all jobs. We shall relax these assumptions in Chapter 4 where we analyze a multi-class model.

We shall use operational analysis as introduced by Denning and Buzen [DeBu78] to study such a system. The main reason for using operational analysis rather than classical queueing theory is that we do not need to determine the distribution of the system parameters required by classical queueing theory. The values of the system parameters can be directly measured from the system

or derived from the measured quantites. Moreover, the assumptions made to make the mathematics tractable can be verified by the analyst.

3.3 Saturation Estimation

Most load control mechanisms are based on some system saturation definition. Various system saturation definitions have been proposed ([Klei68],[Ferr78],[DeBu78]. Invariably the system is considered to be saturated at the point the response time starts to rise rapidly with an increase in some system load index. Under the assumption that all jobs are identical in their resource demands, the number of active jobs or the degree of multiprogramming can be considered to be a measure of system load. A typical response time curve against degree of multiprogramming is shown in Figure 3.3 which is shown again in Figure 3.5 for convenience.

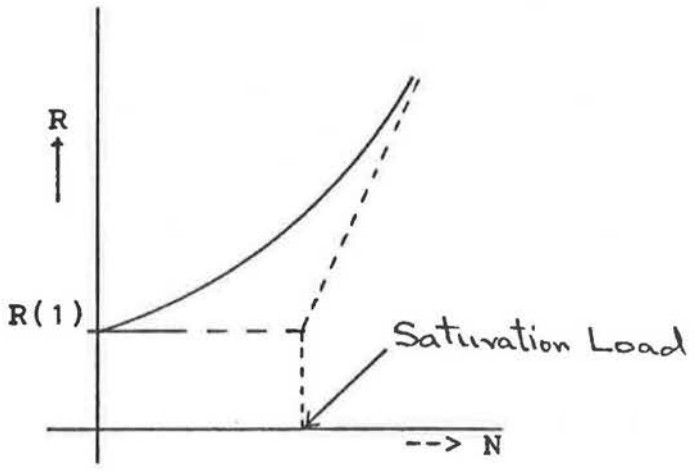

Figure 3.5 Saturation Point

Badel et al. [BaLP75] propose to use the "dilatation" function as a measure of system load. Badel and Leroudier [BaLe78] later used this function in various definitions of saturation critera. For Kleinrock [Klei68], a system is considered to be saturated at the intersection of the asymptote of the normalized mean response time curve vs the number of active terminals and the horizontal line corresponding to when there is only one job in the system (see Figure 3.5). If the system is not allowed to get saturated according to this definition, the mean response time of the active jobs does not exceed an acceptable level. However, it is implicitly assumed that the program population is homogeneous and the system is in a stationary state. In the following analysis, the system saturation is computed at the end of each small interval (usually a few seconds long) during which the stationary assumption is more reasonable. One may argue that this may lead to end effects (i.e., discontinuity at the initial and terminal point of the observation interval). The end-effects, however, do not affect the validity of operational laws since these laws can be shown to be internally consistent and valid for all initial and terminal states so long as the operational assumptions are satisfied [Buze78b]. As well, since we assume all jobs to have identical resource demands, the homogeneity condition is satisfied. We shall now find an expression for the system saturation point.

Let S_1, S_2, ..., S_M be the M service centres as shown in Figure 3.4. S_1 is the CPU, S_2 is the paging device and S_3, ..., S_M are the various I/O units.

The following are observed quantities from the system. They are mean values within an observation period and, as such, are functions of time which is omitted for clarity.

T : observation period

X_i : observed number of completions at centre S_i during T

B_i : the total amount of time during which the service centre S_i is busy during T

C_i : observed number of requests for centre S_i during T

q_i : request frequency, the fraction of jobs proceeding to service centre S_i after completing a service request at the CPU ($= c_i/X_1$), $i \neq 1$.

We now compute the following operational quantities.

Mean service rate of server $S_i = \mu_i = X_i/B_i$

$$\text{System throughput rate } T = (X_1 \cdot q_1) / T$$
$$= (X_1/B_1)(B_1/T) \cdot q_1$$
$$= \mu_1 \rho_1 q_1 \qquad (3.5)$$

$$\text{Utilization of server } S_i = \rho_i = B_i / T$$
$$= (B_i/X_i)(X_i/X_1)(X_1/B_1)(B_1/T)$$

Using the job flow balance assumption $X_i = C_i$ $i \neq 1$ (i.e., the number of requests for service at centre S_i during an interval is equal to the number of departures from the centre) we obtain:

$$\rho_i = \rho_1 \cdot (\mu_1/\mu_i) q_i; \quad i \neq 1 \qquad (3.6)$$

$$\Rightarrow \quad \sum_{i=1}^{M} \rho_i = \rho_1 + \sum_{i=2}^{M} \rho_1 (\mu_1/\mu_i) q_i$$

If there is one and only one job in the system it can be present at only one service centre. Therefore

$$\sum_{i=1}^{M} \rho_i = 1$$

Which implies that the CPU utilization with exactly one job in the system is given by:

$$\rho_1(1) = \left[\sum_{i=1}^{M} (\mu_1/\mu_i) q_i + 1 \right]^{-1} \qquad (3.7)$$

Using Little's Law, the mean response time of the system with N jobs is given by:

$$R(N) = N / T$$

Using (3.5)

$$R(N) = N/(\mu_1 q_1 \rho_1)$$

$$= N\, q_i/(\mu_1 \rho_i q_i) \quad i \neq 1 \qquad (3.8)$$

From (3.7)

$$R(1) = (1/\mu_1 q_1) \left[\sum_{i=1}^{M} (\mu_1/\mu_i)\, q_i + 1 \right] \qquad (3.9)$$

The equation of the asymptote (i.e., as N approaches infinity) is more difficult to derive. Let us first consider the simple case of a non-paging system. The asymptote occurs at the point when the utilization of a service centre (i* say) reaches unity (i.e., it becomes the first bottleneck of the system).

From Buzen's analysis [Buze71], i* is that service centre which has the highest utilization in the interval (note that i* may vary from interval to interval as the workload characteristics change). If the CPU is the bottleneck, the equation of the asymptote is simply:

$$R(N) = N / (\mu_1 q_i) \qquad (3.10)$$

Otherwise, using equation (3.8) and noting that μ_i as well as the ratio (q_i/q_1) remains unchanged as N increases, the equation of the asymptote is given by:

$$R(N) = N\, q_{i^*} / (\mu_{i^*} q_1) , \quad i^* \neq 1 \qquad (3.11)$$

For a paging system the eventual bottleneck as N approachs infinity must be the paging device. It need not however, be the first device to be saturated.

Case(i) The paging device is not the first to saturate.

In this case, as the system is saturated before the paging device is fully utilized, the asymptote should be computed based on the first device to reach saturation. Therefore equation (3.11) is still valid (see Figure 3.6)

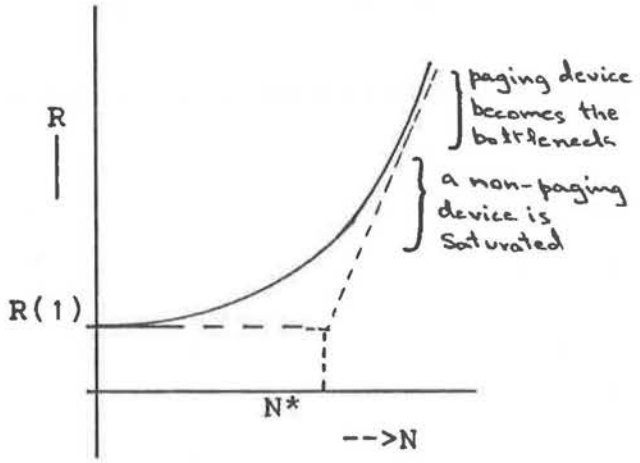

Figure 3.6 System Bottleneck

Case(ii) The paging device is first to saturate

The ratio q_{i^*}/q_1 will continue to increase as N increases and will approach infinity. A realistic approach

consistent with the one used in Case(i) is to use the values of q_i^*/q_1 corresponding to the point the paging device first gets fully utilized. However, this ratio is not easy to estimate. The observed values of q_i^*/q_1 can be used if the system is close to saturation (i.e., $N^* \simeq N$, see below). Otherwise the saturation load will be under-estimated. This is not a problem when the system is lightly loaded. As can be seen subsequently, when the system workload becomes heavy, the control policy will adjust itself and the observed ratio will eventually reach the desired value. Thus the saturation load N^* i.e., the point of intersection of equations (3.9) and (3.10) is given by :

$$N^* = 1 + \sum_{i=2}^{M} (\mu_1/\mu_i)\, q_i \qquad (3.12)$$

if the CPU is the bottleneck. Otherwise it is given by equation (3.13) as the point of intersection of (3.9) and (3.11):

$$N^* = (\mu_i^*/\mu_1 q_i^*) \left[\sum_{i=2}^{M} (\mu_1/\mu_i) q_i + 1 \right] \qquad (3.13)$$

Thus now we have equations (3.12) and (3.13) as estimates of system saturation point under the assumptions made earlier. Both equations (3.12) and (3.13) can also be derived using classical queueing theory with exponential distributions of service times.

Most proposed schemes assume a fixed saturation load. The Michigan Terminal System for example computes the values of five factors at fixed intervals. If one or more exceeds the corresponding predetermined static saturation value, the system is assumed to be saturated. For the 50% criterion, the saturation point corresponds to the load beyond which the utilization of the paging device will exceed 0.5+c, where c is some positive constant. The L=S criterion to a certain extent assumes the system to be saturated when the mean system life time is below the page swap time, which is fixed for a given paging device.

Chanson [Chan79] has shown that the saturation load is really a function of the characteristics of the current workload and cannot be very well represented by some constant measure. For the present model, these work load characteristics are q_i and μ_i, i= 1, 2, ..., M. Any model that does not take these into consideration will sometimes over-estimate and sometimes under-estimate the system saturation load. The fact that on the average the over-estimation is equal to the under-estimation provides no comfort when the goal is to optimize performance at all times.

Therefore the first criterion for load control is to keep the system from saturation. For a multiprogrammed paging computer system, the simplest way to accomplish this

is to keep the number of active jobs below N*, where N* is given by equations (3.12) and (3.13). Since the system throughput rate is a non-decreasing function of N before the system saturates [Buze71], activating N* jobs whenever possible will also maximize the system throughput.

Three cases have to be considered:

(i) the system is saturated (i.e., $N > N^*$)

(ii) the system is under-utilized (i.e., $N << N^*$)

(iii) the system is close to saturation but not yet saturated.

Only case(iii) is of interest, because if the system is under-utilized, it is unnecessary to apply any control measure. Each job will be activated as soon as it arrives, until the condition for case(iii) is reached. If the system load is properly controlled, the system will attain saturation infrequently and only for brief periods of time. When the system is saturated, one simple control measure is not to activate any more batch jobs until the system comes out of saturation. If the system reaches saturation state frequently and remains in this state for extended periods of time, then it is highly probable that the hardware is inadequate to handle the normal work load and the system

should be upgraded.

In the next section we consider a combined batch-interactive system with the assumption that the batch and terminal jobs are identical in their resource demands. In a typical environment the terminal jobs have relatively higher priority than the batch jobs. Therefore, their waiting times should be a function of their relative priority. Our objective is therefore, to maximize a weighted sum of the waiting times , without saturating the system.

3.4 Optimal Selection

The combined batch-interactive system under consideration is shown in Figure 3.7.

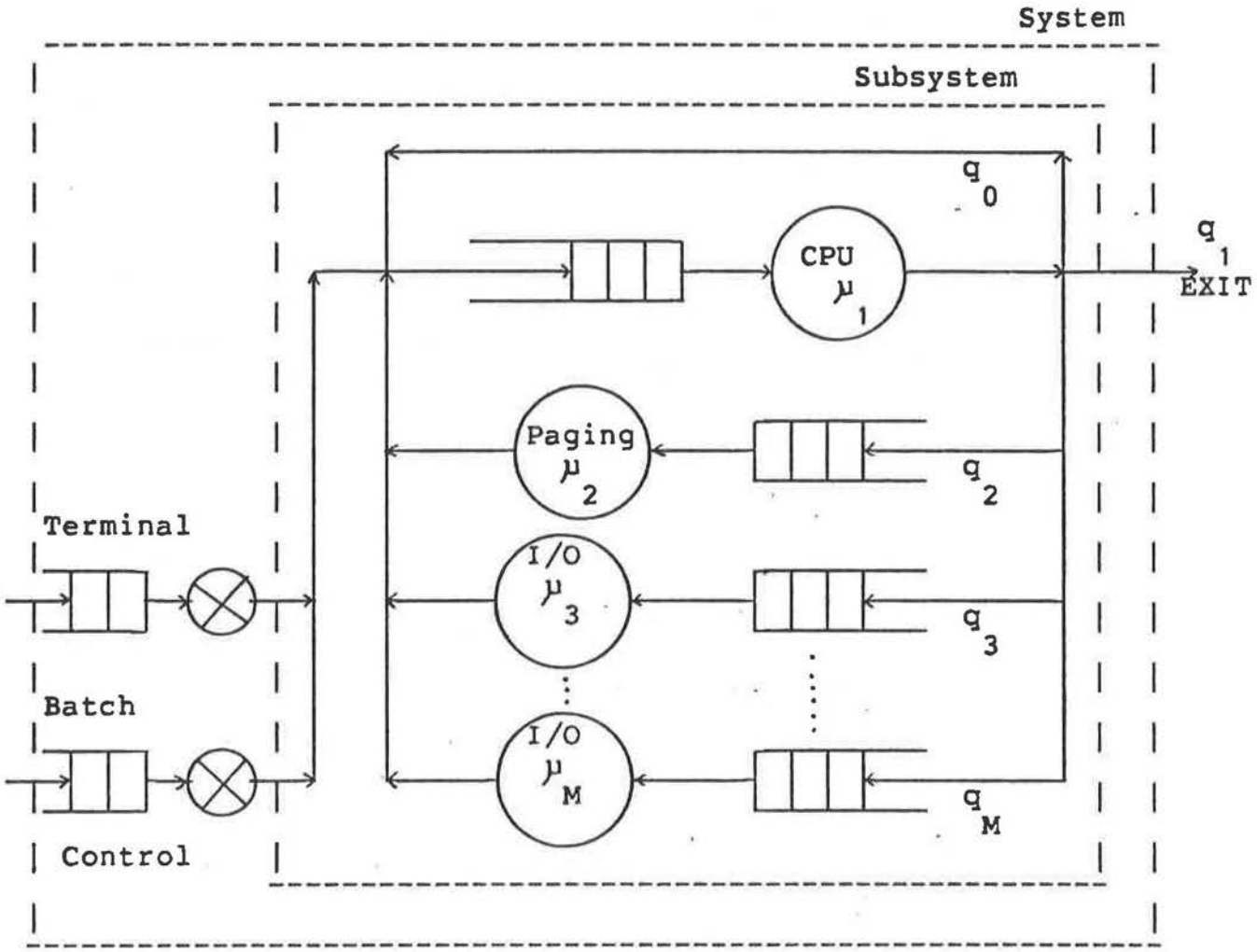

Figure 3.7 Load Control Model

Upon arrival into the system, batch and interactive jobs enter the control unit A and B, respectively. A control policy then decides whether to admit any job from the control units into the subsystem. Only those jobs which are present in the subsystem are allowed to compete for system resources. In the previous section we have derived the maximum number of jobs N* that should be allowed in the subsystem. Let:

N^* : be the maximum number of jobs that should be allowed in the subsystem (obtained in the previous section),

S_1 : be the mean total interactive jobs in the system (observed during the previous interval),

S_2 : be the mean total batch jobs in the system (observed during the previous interval),

N_1 : be the number of interactive jobs to be maintained in the subsystem during the next interval (to be calculated),

N_2 : be the number of batch jobs to be maintained in the subsystem during the next interval (to be calculated),

R : be the mean response time of the subsystem (not required in the final algorithm),

R_1 : be the mean response time of interactive jobs in the subsystem,

R_2 : be the mean response time of batch jobs in the subsystem.

In order to keep the subsystem from saturation it is required that

$$N_1 + N_2 \leq N^*. \tag{3.14}$$

We want to compute the optimal values for N_1 and N_2. Using Little's Law, the throughput rate for interactive jobs is:

$$T_1 = (N_1 / R_1). \tag{3.15}$$

Similarly the throughput rate for batch jobs is:

$$T_2 = (N_2 / R_2). \qquad (3.16)$$

Under the assumption that the terminal and batch jobs are identical in their resource demands, the values R_1 and R_2 will be the same and will not change as long as the equality is satisfied in condition (3.14), irrespective of the values of N_1 and N_2.

Under the job flow balance assumption [DeBu78] (which should be verified) interactive and batch jobs will be entering the subsystem at the rates T_1 and T_2 respectively. Therefore, on the average, an interactive job enters the system every $1/T_1$ sec. It follows that on the average, the waiting time in the control unit for interactive jobs (using Little's Law) is given by:

$$W_1 = (S_1 - N_1) / T_1 \qquad (3.17)$$

Similarly the waiting time in the batch control unit is given by:

$$W_2 = (S_2 - N_2) / T_2 \qquad (3.18)$$

In a real system interactive jobs are normally given higher priority compared to batch jobs. Let C_1 and C_2 be the weights associated with the interactive and the batch jobs respectively. Our objective is to minimize a weighted

sum of the waiting times in the control unit. Therefore the optimization problem becomes:

$$\begin{aligned} \text{Min } z \equiv \text{Min } & (C_1W_1 + C_2W_2) \\ \text{subject to } & N_1 + N_2 \le N^* \\ & 0 \le N_1 \le S_1 \\ & 0 \le N_2 \le S_2 \end{aligned} \tag{3.19}$$

Using (3.15), (3.16), (3.17) and (3.18), the above problem (3.19) can be shown to be equivalent to

$$\begin{aligned} \text{Min } & (C_1S_1/N_1 + C_2S_2/N_2) \\ \text{subject to } & N_1 + N_2 \le N^* \\ & 0 \le N_1 \le S_1 \\ & 0 \le N_2 \le S_2 \end{aligned} \tag{3.20}$$

For the time being we shall not consider the constraints $0 \le N_1 \le S_1$ and $0 \le N_2 \le S_2$ (i.e., we assume that there are enough jobs waiting in the interactive and batch control queues).

We shall use the Lagrange multiplier method to solve the problem (3.20) (without extra constraints). The Lagrangian equation of the problem is:

$$L(N,\lambda) = C_1S_1/N_1 + C_2S_2/N_2 + \lambda(N_1/N^* + N_2/N^* - 1) \qquad (3.21)$$

The optimal values of N_1 and N_2 from (3.21) are:

$$N_1^* = N^*\sqrt{C_1S_1} \;/\; (\sqrt{C_1S_1} + \sqrt{C_2S_2}) \qquad (3.32)$$

and

$$N_2^* = N^*\sqrt{C_2S_2} \;/\; (\sqrt{C_1S_1} + \sqrt{C_2S_2}) \qquad (3.23)$$

Let us look at the problem (3.20) geometrically with the constraints $0 \le N_1 \le S_1$ and $0 \le N_2 \le S_2$ active. Figure 3.9 is a diagram of the problem (3.20) showing the feasible region and the objective function for its various values.

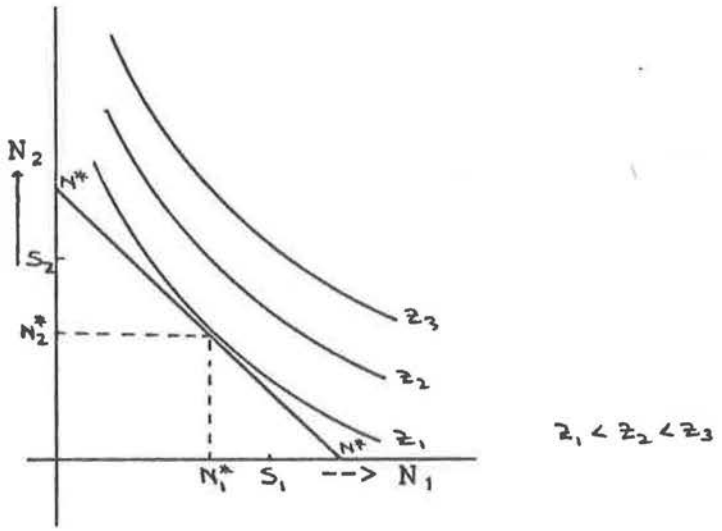

Figure 3.8 Active Constraints

From the above graph, in the absence of the constraints $0 \leq N_1 \leq S_1$ and $0 \leq N_2 \leq S_2$, N_1^* and N_2^* are the optimal values.

Case (i) $N_1^* > S_1$ and $N^*_2 < S_2$.

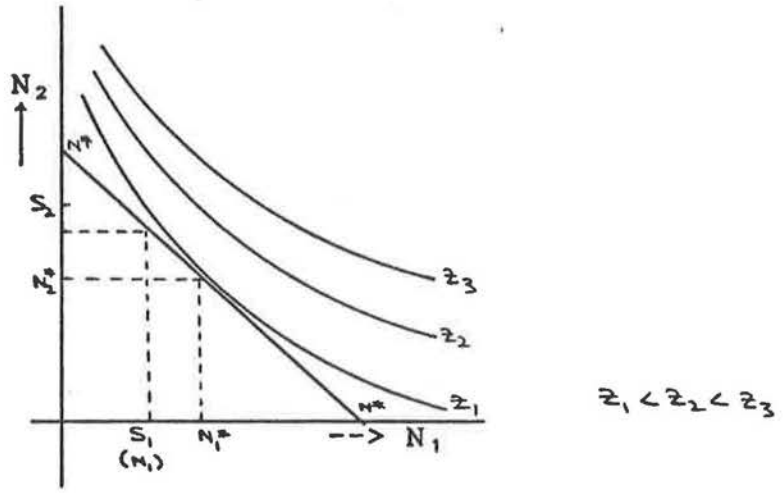

Figure 3.9 Inactive Constraint

Clearly the optimal values for N_1 is

$$N_1' = S_1$$

and $N_2' = N^* - N_1'$, which gives maximum value of the objective function satisfying the constraints.

Case (ii) $N_2^* > S_2$ and $N_1^* < S_1$.

As in Case(i) it can be seen that an optimum value for N_2 is

$$N_2' = S_2$$

and $N_1' = N^* - N_2'$, which gives maximum value of the objective function satisfying the constraints.

Case (iii) $N_1^* > S_1$ and $N_2^* > S_2$

It can be easily seen that $N_2' = S_2$ and $N_1' = S_1$ will give the optimal solution in tnis case.

The final control policy, which we shall call SELF (standing for Saturation Estimation and Load-control with Feed-back), works as follows. The values N_1^* and N_2^* are computed from the equations (3.22) and (3.23) or from one of the three cases as described above. During the observation period, whenever an interactive job leaves the subsystem, an interactive job will be injected into the subsystem provided the interactive control queue is non-empty. When a new interactive job arrives, it enters the subsystem immediately provided (i) there is no waiting interactive job, and (ii) the number of interactive jobs in the subsystem is less than N_1^*. Otherwise, it waits in the interactive control queue.

The batch control works in a similar fashion.

In the implementation of SELF the values of the parameters are estimated on the basis of their past values. In order to reduce the error in the estimation, we use a special technique from time-series analysis [Kend76], called exponential smoothing. This technique can be described as follows. Let P_i be the expected value of the parameter for the time interval [i, i+1]. Let X_i be the observed value of the parameter at the time t. P_i can be expressed as

$$P_i = (1-\beta)(X_i+\beta X_{i-1}+\beta^2 X_{i-2}+\ldots)$$

$$= (1-\beta) \sum_{j=0}^{\infty} \beta^j X_{i-j} \qquad (3.25)$$

where the exponential weight factor β is a constant between zero and one. Similarly

$$P_{i-1} = (1-\beta) \sum_{j=0}^{\infty} \beta^j X_{i-j-1}$$

$$\Rightarrow \quad P_i = (1-\beta)X_i + \beta P_{i-1} \qquad (3.26)$$

Now, let the error made at time (i-1) in predicting X_i be ϵ_i, then,

$$\epsilon_i = X_i - P_{i-1}. \qquad (3.27)$$

Substituting in equation (3.26)

$$P_i = X_i - \beta \epsilon_i$$
$$= P_{i-1} + (1-\beta)\epsilon_i \qquad (3.28)$$

The remaining problem is to find a proper value of β.

If the error ϵ_i is sufficiently small, equation (3.28) will be satisfied for almost any value of β, so that we can use the value of β from the previous interval. If ϵ_i is large, we recompute a value for β by minimizing the sum of squares of the errors given by:

$$\sum_{j=i}^{\infty} \{X_j - (1-\beta) \sum_{k=0}^{\infty} \beta^k X_{j-k-1}\}^2. \qquad (3.29)$$

In practice, the summation in equation(3.29) does not have to involve many terms (say K) before β^K $(0<\beta\leq 1)$ approaches zero. Moreover, β need not be very accurate, and standard techniques exist for its efficient computation.

From (3.28), it can be seen that not all previous values of X are needed to compute the expected value of X for the next interval. The future value can be computed from the previous predicted values and the present error. However, several immediate past values of X may be needed in the computation of β. The effect of β on the expected

error is discussed in Chapter 5.

So far we have assumed that all the jobs are statistically identical in their characteristics. In the next chapter we shall relax this condition and develop a control scheme that makes an optimal selection by taking into consideration the job characteristics along with their external priorities.

CHAPTER 4

MULTICLASS CONTROL

4.1 Introduction

In a typical general purpose computer system there exist various classes of jobs as far as their resource demand characteristics are concerned. In such an environment the natural approach for an analyst is to use a multiclass model of the system that would treat the different classes of jobs differently ([Rode79], [Bard79], [ReLa80]). The jobs within a class, however, are supposed to have identical resource demands. In this analysis, it is assumed that the jobs do not change class during their stay in the system. The number of job classes considered is finite. Moreover, it is assumed that when a job arrives at the system it is possible to classify it into one of the job classes. An example of a primitive method of job classification is to compute the job class based on card parameters for batch jobs (e.g., CPU time limit, user-given

job priority, user IDs etc.) and on the command type for terminal jobs (e.g., edit, compile, copy, etc.,). However, it is not assumed that the resource demands of a job in a particular class are known. The resource demands of various classes of jobs are continuously being measured, thereby preserving the dynamic nature of the control.

4.2 Multiclass Saturation Estimation

In a non-paging system with jobs that are identical in their resource demands, the branching frequencies, from one service centre to another, are independent of the number of jobs present in the system. They depend mainly on the job characteristics. On the other hand, in a multiclass environment, the branching frequencies are not only dependent upon the total number of jobs but also on the number of jobs present in each class (specifically, the ratio of the number of jobs in each class).

Thus load control in such a system consists of two mutually dependent problems as follows:

(a) compute an optimal ratio corresponding to the number of jobs that should be selected from each class,

(b) compute the saturation point of the system for

the ratio obtained in (a).

The optimality in (a) is subject to a constraint defined by (b) and the saturation point obtained in (b) must satisfy the ratio obtained in (a). Thus the two problems (a) and (b) are dependent upon each other and we must seek a simultaneous solution. In a number of studies, a solution of one of the problems has been obtained for a fixed solution of the other problem. For example, Schonbach [Scho80] solves the problem (a) for the case in which system saturation is already defined and fixed.

In the rest of this section we first derive the saturation load for a multiclass system and then describe the multiclass control procedure. The following notations are used:

K	:	total number of job classes,
M	:	total number of service centres,
$s(j)$	:	number of class j jobs in the subsystem
$\underline{N}$	:	the system state vector$(s_1, s_2, \ldots, s_K)$,
$n(i,j,\underline{N})$	:	number of class j jobs at centre i for a given system state $\underline{N}$,
$n_i(\underline{N})$	:	total number of jobs at centre i for a given system state $\underline{N}$,
$\lambda(j)$	:	throughput rate of class j jobs

$w(i,j)$: mean time a class j job spends at service centre i during its stay in the system (including queue wait and service times),

$R(j)$: mean time spent in the multiprogramming subsystem by class j jobs,

$d(i,j)$: mean total service demand of class j jobs at service centre i,

$S(j)$: mean total number of class j jobs in the system,

μ_i : mean service rate of service centre i,

$q(i,j)$: normalized frequency of requests for centre i by class j jobs (i.e., the ratio of class j jobs joining centre i after being serviced by the CPU to the total number of class j job completions at the CPU),

$q_i(\underline{N})$: normalized frequency of requests for centre i (i.e., the ratio of jobs joining centre i after being serviced by the CPU to the total number of completions at the CPU),

$C(j)$: weight for class j jobs.

It can be easily shown that $q_i(\underline{N})$ is given by :

$$q_i(\underline{N}) = \sum_{j=1}^{K} n(1,j,\underline{N})\, q(i,j,\underline{N}) \;/\; n_1(\underline{N}). \qquad (4.1)$$

Clearly $\sum_{i=1}^{M} q_i(\underline{N}) = 1$

$\therefore$ $\sum_{i=1}^{M} q(i,j,\underline{N}) = 1$ and $\sum_{j=1}^{K} n(1,j,\underline{N}) = n_1(\underline{N})$

There are several methods to compute $n(i,j,\underline{N})$ for a given system state vector $\underline{N}$ ([Buze73],[ReKo76],[ReLa80]). The drawback in most of these methods is that the overhead involved is quite high. Since the aim here is to develop an adaptive scheme, it is desirable to minimize the amount of computation involved. Reiser and Lavenberg's [ReLa80] mean value analysis for example, can be used to compute $n(i,j,\underline{N})$. The method assumes that the service time of each service centre is load-independent and has only one server. According to their study the mean waiting time, the mean throughput rate and the mean queue sizes for each class j at service centre i are recursively given by:

$$w(i,j) = d(i,j)\,(1 + n_i(\underline{N}-e(j))) \qquad (4.2)$$

$$\lambda(j) = s(j) \;/\; \sum_{i=1}^{M} w(i,j) \qquad (4.3)$$

$$n(i,j) = \lambda(j)\, w(i,j) \qquad (4.4)$$

where e(j) is the jth unit vector. Starting with the initial condition $n(i,j,\underline{N}) = 0$ and using equations (4.2), (4.3) and (4.4) one can then obtain $n(i,j,\underline{N})$. Their arguments are based on the intuition that upon arrival at a service centre, a job "sees" the system with itself removed (i.e., one job less in the system). Little's Law is then applied to the entire system and to each individual service centre. This algorithm requires $2KM - K$ additions and $2KM + K$ multiplications/divisions per recursive step. There are a total of $s_1.s_2...s_K$ such steps. In a system with a moderately large number of classes and a large number of jobs such a scheme might not be economically feasible. Reiser and Lavenberg have extended their scheme and have reduced it to the problem of solving a set of non-linear equations.

For each centre r they introduced a correction term $\epsilon(i,j,r,\underline{N})$ such that

$$n(i,j,\underline{N}-e(r)) = n(i,j,\underline{N}) - \epsilon(i,j,r,\underline{N}) \qquad (4.5)$$

with the assumption that

$$\epsilon(i,j,r,\underline{N}) = 0 \quad \text{for } i \neq r$$

i.e., only the class with the customer removed is affected.

They also assumed that $\epsilon(i,j,i,\underline{N})$ is given by

$$\epsilon(i,j,i,\underline{N}) = n_i'(s_j) - n_i'(s_j-1) \qquad (4.6)$$

where $n_i'(s_j)$ is the mean number of class j jobs at the service centre i with s_j class j customers. The modified parameters of the system are

$$d_i'(i,j) = d(i,j)\ n_i(\underline{N}) \ / \ n(i,j,\underline{N}) \qquad (4.7)$$

Using (4.5), the recursive equations (4.2), (4.3) and (4.5) can be reduced to the set of non-linear equations

$$w(i,j) = d(i,j)\ (1 + n_i - \sum_{r=1}^{K} \epsilon(r,j,i)) \qquad (4.8)$$

$$\lambda(j) = s(j) \ / \sum_{i=1}^{M} w(i,j) \qquad (4.9)$$

$$n(i,j) = \lambda(j)\ w(i,j) \qquad (4.10)$$

The equations (4.8), (4.9) and (4.10) along with (4.6) can be solved simultaneously to obtain $n(i,j,\underline{N})$. The number of operations required by this procedure is proportional to $M.|\underline{N}|$, which is considerably less than the original algorithm. In the implementation of the multiclass control we use this extended version of their scheme.

Once the $n(i,j,\underline{N}))$'s are known for a given $\underline{N}$ one can compute $q_i(\underline{N})$ from equation (4.1). Following the same arguments used in the case of a single class system, it can be shown that the saturation vector $\underline{N}^*$ of the system is given by the relation

$$|\underline{N}^*| = (\mu_i^*/\mu_1 q_i^*(\underline{N})) \left[1 + \sum_{i=2}^{M} \frac{\mu_1}{\mu_i} q_i(\underline{N})\right] \qquad (4.11)$$

where $|\underline{N}^*| = \sum_{j=1}^{K} s(j)^*$, $q_1^*(\underline{N})=1$ and i^* is the device with the highest utilization in the observation period.

Notice that $\underline{N}^*$ is no more a single number as in the case of a single class. Rather it is a vector and equation (4.11) alone is not sufficient to uniquely determine $\underline{N}^*$. This is why we need a second criterion for load control.

4.3 Multiclass Optimal Selection

The following different objectives are considered for optimal selection.

(a) Compute the ratio that maximizes the system throughput rate.

(b) Compute the ratio that takes into consideration job priorities (i.e., some weights are associated

with each class of jobs) and minimizes a weighted function of their waiting time.

Schonbach [Scho80] solved the problem using (a). One obvious problem with this approach (i.e., without considering the job priorities) is that it may give relatively higher priority to small jobs. In some cases large jobs may have to wait indefinitely. Furthermore, it usually leads to a dynamic programming problem which in turn requires high overhead. Therefore, we shall use criterion (b).

For criterion (b) the optimization problem becomes

$$\text{Min } z \equiv \text{Min } \{ \sum_{i=1}^{K} C_i W_i \}$$

$$\text{subject to } \sum_{i=1}^{K} s(i) \leq |N^*|$$

where W_j is the waiting time for class j jobs in the system.

$$W_j = R(j) + (S(j)-s(j)) / (s(j)/R(j)), \quad j=1,2,\ldots,K.$$

Following arguments similar to the one in Chapter 3 it can be shown that the solution of the problem is given by:

$$s(i)^* = \frac{|N^*| \sqrt{C(i)S(i)R(i)}}{\sum_{j=1}^{K} \sqrt{C(j)S(j)R(j)}}, \quad i = 1, 2, \ldots, K. \qquad (4.12)$$

Furthermore, it can be shown that only (K-1) out of the K relationships in (4.12) are linearly independent. In order for the system not to be saturated, the s(i)'s must satisfy (4.11). Therefore, there are K unknowns (the s(i)'s) in K non-linear independent equations. A unique solution therefore exists.

Note that $R(j) = \sum_{i=1}^{M} w(i,j)$ whose value is obtained in in the computation of $q_j(\underline{N})$ in equation (4.1).

The multiclass control procedure, which we shall call MULTI-SELF, can be summarized as follows:

Step 1. During the observation period T, collect the values of the parameters required for the computations (i.e., the branching frequencies to different service centres for different classes of jobs; the average service rates of different centres and the mean number of jobs in the system for each class).

Step 2. From the measured parameter values compute their expected values for the next interval using exponential smoothing.

Step 3. Solve the system of non-linear equations (4.8) through (4.12) simultaneously.

Step 4. For each class i, maintain the number of jobs in the subsystem to be s(i)* (if possible) during the next observation period.

The next chapter describes a simulation study of the performance of the two control schemes SELF and MULTI-SELF. Their performances are compared to that of other existing schemes.

CHAPTER 5

SIMULATION RESULTS AND ERROR ANALYSIS

5.1 Introduction

Throughout the development of the control schemes SELF and MULTI-SELF presented in the last two chapters, a number of assumptions have been made. These assumptions may not be satisfied in practice. It is therefore desirable to determine the accuracy of these models and to compare their performance to those of some of the previous models.

Because of the lack of resources it was not possible to implement the proposed schemes on a real system. Instead, a general purpose event-driven simulator for a central server model was developed. The schemes were then implemented on this simulator to control the flow of jobs through the simulated system. The workload that drives the simulator does not make the same assumptions that were made in the development of SELF and MULTI-SELF. For example,

the jobs are not identical, and their characteristics may change from time to time. Also it may not be possible to maintain the computed degree of multiprogramming during every observation interval and the job flow balance might not be satisfied during every observation interval, etc. The simulator is quite flexible. It can be run with any practical number of different classes of jobs and can handle any practical number of I/O devices. Also, it can use various distributions to generate the service times of different service centres.

The major components of the simulator are described in the next section. Results comparing the performance of the two schemes with some of the existing ones are also presented. A summary of the various assumptions made and their effects on the performance of the two schemes is given in the last section.

5.2 Description of the Simulator

The simulator is written in PL/1 and is implemented on an Amdhal 470 V/8 running under MTS. There are several components of the simulator. They will be examined after a discussion of the flow of jobs through the simulated system.

A sequence of job-processing step is described below.

Each of them corresponds to an event within the model. The simulated model is the same as that depicted earlier in Figure 3.7.

(1) Upon arrival, a job enters the control unit and joins the queue corresponding to its class. If the control queue corresponding to its class is empty and the current control policy allows more jobs of this class in the subsystem, the job moves to the CPU queue immediately. Otherwise, it waits for its turn in the control queue.

(2) A job may join the CPU queue from one of the several service centres or from the control queue. Depending on where it came from and the conditions under which a job enters the CPU queue the following activities will happen.

(a) If it enters from the control unit, it is assigned a full CPU quantum. Its next I/O time and type are generated. The time indicates the amount of CPU service required between two successive I/O operations. The type indicates whether the job will leave the system or join one of the I/O units. In the latter case it also indicates which of the I/O units it will join.

(b) If it enters after completion of its time quantum, it

is assigned a new full time quantum.

(c) If it enters after completion of an I/O request, its next I/O time and type are generated as in (a).

(d) If it enters after completion of a page fault, its remaining time quantum, the next I/O time and type will be used.

Depending upon the system to be simulated, jobs may enter the head or the tail of the CPU queue. In MTS, for example, a job joins the head of the CPU queue after a page fault or an I/O competion. It joins the tail of the queue on the first arrival or if its time quantum has expired.

(3) Departure from the CPU occurs on completion of one of the following events:

(a) the CPU time quantum expires; in this case, the remaining time for the next I/O event is computed, and the job is moved back to the CPU queue,

(b) an I/O request is made; in this case, the remaining CPU quantum time is computed and the job is moved to the requested I/O unit,

(c) a page fault occurs; in this case, the remaining CPU

quantum and I/O time for the currently running job are computed, the job is moved to the paging device and the next page fault time is generated.

(4) Upon arrival to the paging device, the job waits in the paging queue if the device is busy. Before the paging device starts processing the job, its service time is generated. Upon completion of the page service the job rejoins the CPU queue.

(5) Upon arrival at an I/O unit, the job waits in the I/O queue if the device is busy. Before the I/O device starts processing the job, its service time is generated. After completion of the I/O service the job rejoins the CPU queue.

(6) Upon completion of all the CPU and I/O demands, the job's statistical data are collected. The control procedure is then activated to check if another job can be injected into the subsystem.

The simulated system is initialized with the system and job parameters. The system parameters then generate a basic data stucture for the modelled system. Some of the system parameters are: the number of job classes, the number of I/O units, the control criterion, the memory

size, the life-time function parameters, etc. The job parameters for each job class are the exit probability, the I/O rates etc.

When a job is admitted into the subsystem, a job descriptor is created for it. The job descriptor identifies the job and its class. Data associated with the job's activity throughout its life span in the subsystem are collected and stored in the job descriptor.

The queues at the various service centres are maintained as a linked list with head and tail pointers. Each node of the list is a pointer to a job descriptor indicating the presence of the job at that particular position in the queue.

A separate procedure dynamically creates the I/O units at the beginning of a session. Upon each I/O request, another procedure links this request to the proper I/O unit. The request is then serviced.

The simulator can be run under different control schemes viz., 50%, L=S, Knee, SELF, MULTI-SELF and NO CONTROL. An event is scheduled every T units of time to collect data and activate the required control procedure. The control procedure then computes the number of jobs in each class that should be maintained in the next T units of

time. During the observation period the mean number of jobs in the subsystem is not allowed to exceed the computed control number.

Page faults are generated using the system life-time function as defined in [BeKu69] i.e.,

$$L = \frac{2b}{1 + (c/p)^2} \qquad (5.1)$$

where b and c are constants and p is the average number of pages allocated to each job. Each time a page fault occurs, the next page fault time is computed based on equation (5.1) and the number of jobs currently in the subsystem. A page fault is implemented as a system activity rather than a job activity. Therefore whenever the page fault time is reached, whichever job is present at the CPU at that time will experience the page fault.

In order to create exactly the same workload for various experiments with different control schemes, a pseudo-random number generator is used. Given the same initial seed, the same sequence of random numbers will be generated each time. To compare various control schemes, the simulator is run using the same random-number stream. Under various control schemes different job activities can take place in different order. Therefore, a common random-number stream can not be used for different jobs.

In order to resolve this problem, we use the fact that the order of job arrivals is independent of the control scheme. Thus each job is assigned a seed as soon as it arrives at the system. This seed is used to generate a pseudo random-number sequence for various job-related activities (such as, next I/O time and type, I/O service time etc.). Furthermore, because paging is a system activity and depends on the size of main memory, the number of jobs in the system etc., the page-fault time and page service time use independent random-number streams.

The simulator was validated in two different ways. (a) Selective dumps of all activities over several prespecified periods of time were taken and hand traced. Because there is a fairly large number of activities even in a small interval of time it was not possible to go over all the activities of one session. Therefore, (b) Little's Law was used to compute the global mean behaviour of the system as well as of the individual service centres; these were then verified against the observed ones. The observed values were within 0.05% of the computed values.

5.3 Simulation Results

In order to study the feasibility and behaviour of SELF and MULTI-SELF we first compare the performance of SELF with some other schemes, specifically the 50%, the

L=S, and the Knee criteria. Then SELF is compared with MULTI-SELF. The workload corresponding to the results in Table 5.1 through Table 5.13 are given in Table A.1 through Table A.9 in Appendix A. The system parameter values were selected to be similar to those of the IBM 370/168 system used by the UBC computing centre a few years ago. The workload parameter values were based on measured workloads on the 370/168 with perturbations introduced to test the stability of SELF and MULTI-SELF.

Because simulation runs are expensive, the runs were made as short as possible. The runs of 120 simulated seconds were made. During this period, approximately 200 jobs were processed. It is found that the mean response time stabilizes around 120 seconds. Table 5.1 shows a typical set of mean response times observed between 75 and 180 seconds at an interval length of 3 seconds.

SIMULATED TIME (SEC.)	TERMINAL	BATCH
75	0.3112	0.3744
90	0.3484	0.4420
105	0.3482	0.4632
120	0.3489	0.4627
135	0.3380	0.4630
150	0.3389	0.4630
165	0.3337	0.4396
180	0.3403	0.4448

Table 5.1 Mean Response Time

The performance of the 50% criterion and the L=S criterion depends upon certain parameter values which are functions of the system load. For example, in the L=S criterion, we must find a constant μ and use $L=\mu S$. The value of μ depends upon the job characteristics. From Table 5.2 we obtain 0.6 as the best value of μ for the workload under consideration.

	0.2	0.50	0.60	0.75
TERMINAL	9.2093	1.4124	1.7085	1.7247
BATCH	13.7833	4.1951	3.4939	4.4230
SYSTEM	10.3977	2.3303	2.3109	2.6465

Table 5.2 Mean Response Time (in sec.) For various values of μ in L=S criterion

Although the job characteristics change dynamically in the workloads that we shall be analysing, the value of μ is a constant in the L=S criterion. Therefore for every workload we first obtain a best value of μ and then use it.

Several simulation runs were made which show the superiority of SELF over 50% and L=S. Some of the results obtained are presented in Tables 5.3 through 5.5.

	SELF RESP. TIME (SEC)	50% RESP. TIME (SEC)	%IMP[1]. OVER 50%	L=S RESP. TIME (SEC)	%IMP OVER L=S
TERMINAL	0.5409	0.5531	2.20	0.5444	0.64
BATCH	0.8901	0.9600	7.28	0.9574	7.08
SYSTEM	0.6552	0.6785	3.43	0.6724	2.55

Table 5.3 Resp. Time and %Impr. for Workload in Table A.3

	SELF RESP. TIME (SEC)	50% RESP. TIME (SEC)	%IMP[1]. OVER 50%	L=S RESP. TIME (SEC)	%IMP OVER L=S
TERMINAL	0.6111	0.8262	26.03	0.8404	27.28
BATCH	1.5418	2.2729	32.16	2.0050	23.15
SYSTEM	0.9531	1.3586	29.84	1.2684	24.86

Table 5.4 Resp. Time and %Impr. for Workload in Table A.4

---

[1] % impr. = $\frac{\text{(50\%Resp. time - SELF Resp. time)}}{\text{50\% Resp. Time}} * 100$

	SELF RESP. TIME (SEC)	50% RESP. TIME (SEC)	%IMP. OVER 50%	L=S RESP. TIME (SEC)	%IMP. OVER L=S
TERMINAL	1.9096	2.7638	30.90	2.6061	26.73
BATCH	3.8750	4.9179	21.21	4.4218	12.37
SYSTEM	2.5218	3.4243	26.36	3.1707	20.47

Table 5.5 Resp. Time and %Impr. for Workload in Table A.5

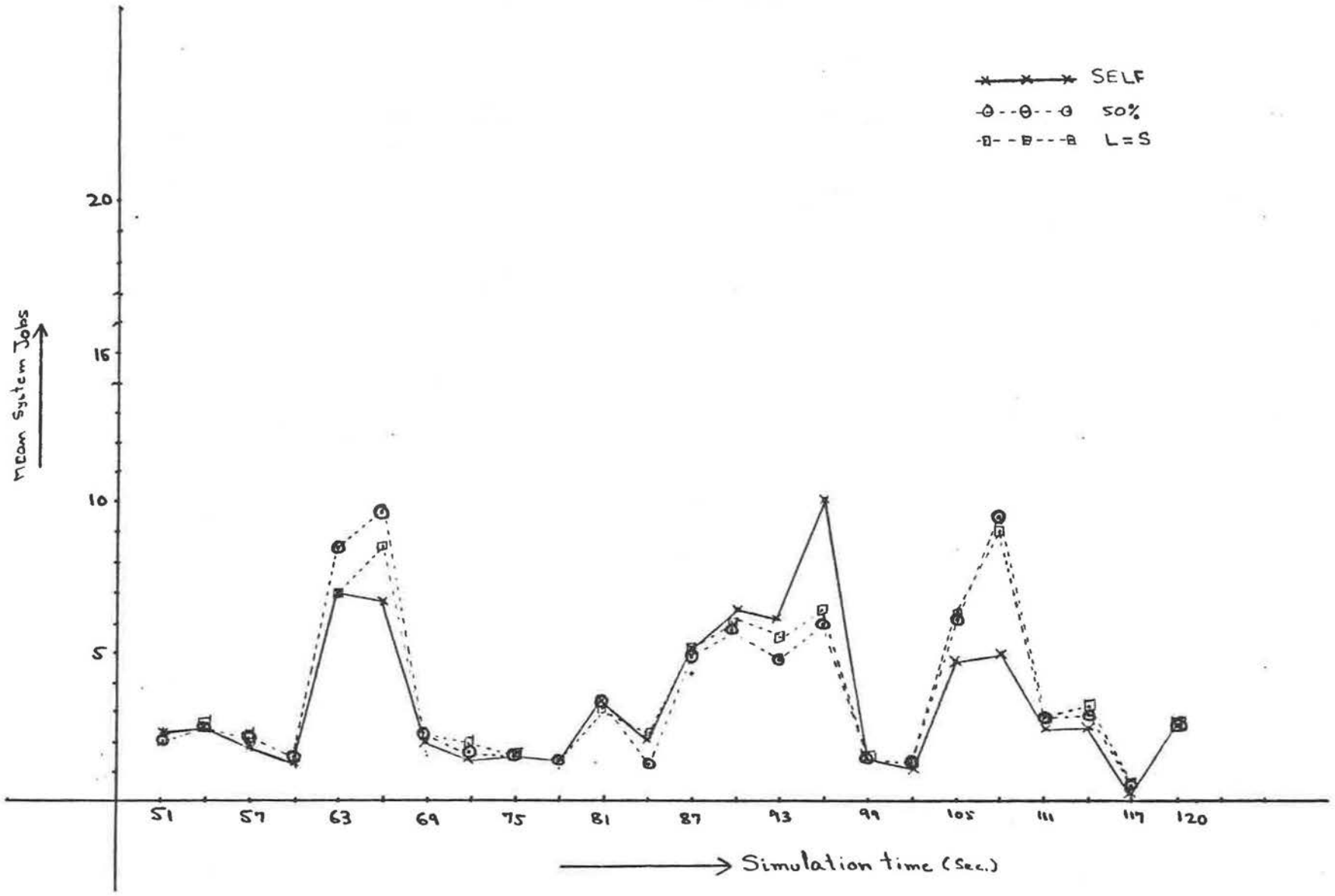

Figure 5·1 Mean System Jobs Under Light Load

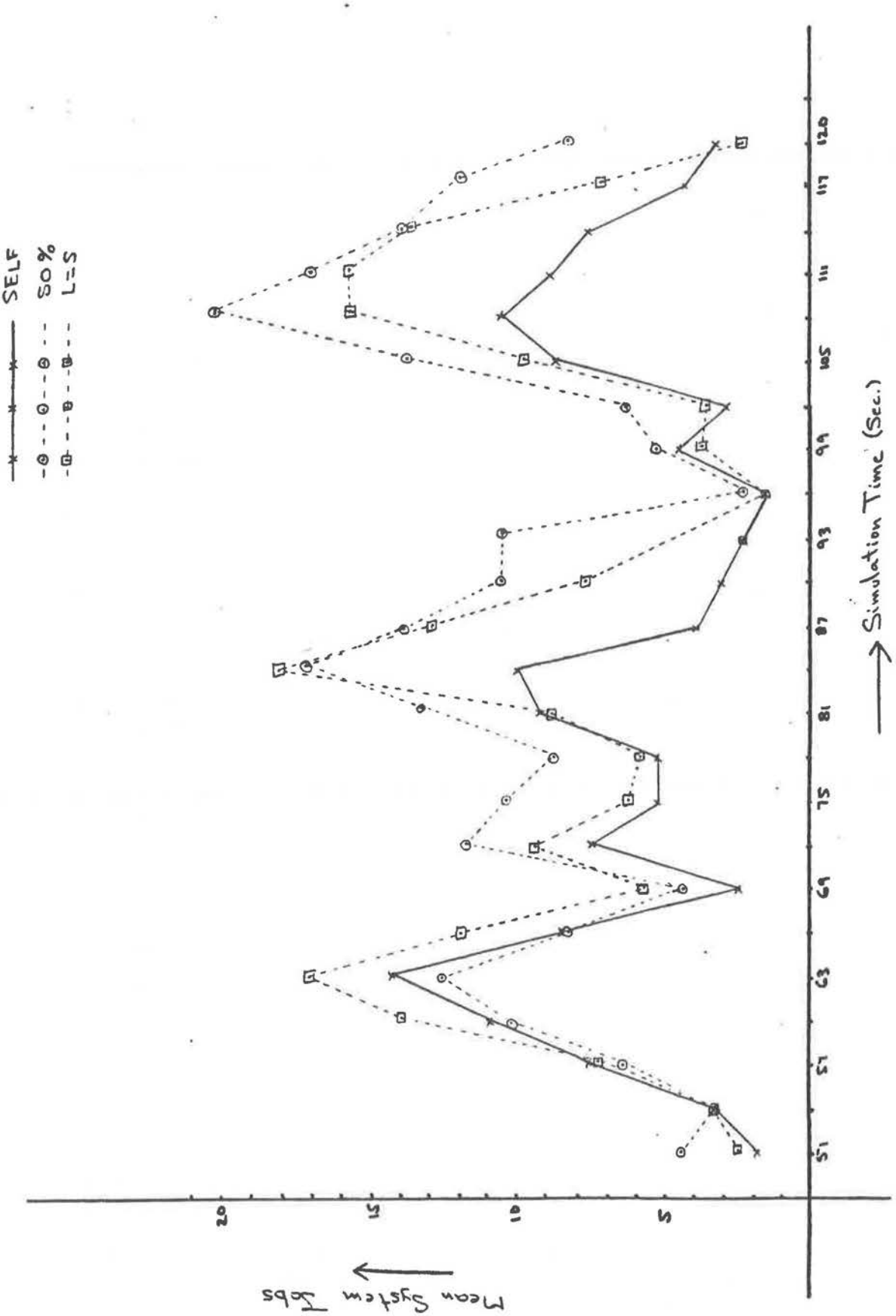

Figure 5.2 Mean System Jobs Under Heavy Load

The parameters for the workload corresponding to these results are given in Appendix A. In order to demonstrate the dynamic adaptability of SELF over the other schemes, an artificial variation in the workload is introduced. Figures 5.1 and 5.2 represent graphs of the mean number of jobs vs simulation time and correspond to the workload of Table 5.3 and 5.4 respectively. The workload corresponding to Figure 5.1 has a smaller variation compared to the workload of Figure 5.2. The relative improvement of SELF over 50% and L=S in Table 5.4 is also greater as compared to the one in Table 5.3. Thus it is apparent that the larger the workload variation, the more superior is the performance of SELF relative to the other two schemes. This demonstrates the robustness and adaptive nature of SELF under varying workload. Under light workload all the schemes give approximately the same results as no control is required (see for example Table 5.3).

Although the Knee criterion is better than the 50% and the L=S criteria, it is expensive to implement in practice. However, since the workload of the simulator is distribution-driven and the life-time function approximated by equation (5.1), it is possible to simulate the Knee criterion without excessive overhead. The Knee criterion requires each job to run at the knee of its life-time function, i.e., at the point where the curve of the life-time of a process vs its memory allocated has maximum

slope. It can be shown that, if the life-time function is simulated using equation (5.1), then this maximum slope is attained when p = 2c, which is independent of the parameter b. Therefore, if equation (5.1) is used to simulate the life-time, by suitably choosing the values of b and c one can create a worst case workload for the Knee criterion without significantly affecting the performance of the other criteria. After selecting a combination of parameters to favour the Knee criterion, the results shown in Table 5.6 were obtained.

	SELF RESP. TIME (SEC)	50% RESP. TIME (SEC)	%IMP. OVER 50%	L=S RESP. TIME (SEC)	%IMP. OVER L=S	KNEE RESP. TIME (SEC)	%IMP. OVER KNEE
TERMINAL	2.2405	3.2066	43.11	3.1424	40.25	2.8461	27.02
BATCH	4.2405	5.0555	18.29	4.6618	9.94	4.3399	2.34
SYSTEM	2.8448	3.7727	32.62	3.6112	26.94	3.3070	16.24

Table 5.6 Resp. Time and %Impr. of SELF, 50%, L=S and Knee

We observe that the knee criterion is better than the 50% and L=S criteria but not as good as SELF under the workload considered.

SELF allows one to adjust the quality of service given to the terminal and batch jobs. By choosing proper weights

the analyst can reduce the mean response time of the terminal jobs to almost the lower limit (i,e., when only terminal jobs are present in the subsystem), at the expense of the mean batch response time. Tables 5.7 and 5.8 show the mean response times of terminal and batch jobs for different values of the weight factors for two different workloads corresponding to those in Tables A.7 and A.8 respectively.

	C1/C2=1	2	3	4	5	20
TERMINAL	10.6451	7.4705	5.4945	4.8306	4.5491	2.9849
BATCH	20.1333	22.7354	22.6035	23.4670	24.8099	27.6247
SYSTEM	14.2816	13.0492	11.6829	11.4475	11.5916	11.4095

Table 5.7 Mean Resp. Time (in sec) Using SELF With Diff. Weight Ratios Under Heavy Workload

	C1/C2=1	2	3	4	5	20
TERMINAL	2.3774	2.2405	2.0437	1.9000	1.7773	1.7770
BATCH	3.8004	4.2405	4.1070	4.3152	4.4305	5.1705
SYSTEM	2.8128	2.8448	2.6620	2.6286	2.5777	2.7853

Table 5.8 Mean Resp. Time (in sec) Using SELF With Diff. Weight Ratios Under Light Workload

The results in Tables 5.7 and 5.8 show the effect of changing the weights under heavy and light loads respectively.

The overhead involved in the implementation of SELF consist of two different components.

(a) The overhead due to collecting the data during the observation intervals.

(b) The overhead due to the computation of the control number.

Overhead (a) depends upon the system configuration (e.g., the number of I/O units etc.,) and job characteristics (e.g., total CPU demand, number of I/O requests etc.). The overhead in (b) only depends upon the system configuration. The overhead (a) for the system and the workload considered in the above examples is estimated to be approximately 0.125% of CPU time on an Amdhal 470 V/8 system. The percentage is computed as follows:

$$\% \text{ CPU Time} = \frac{\text{Computation Time}}{\text{Interval Length}} * 100$$

The overhead in (b) is estimated to be approximately 0.04% of CPU time. Therefore, the total overhead for SELF is approximately 0.165%. This level of overhead is certainly acceptable.

We now compare SELF with MULTI-SELF. In the previous examples only two classes of jobs were considered. We use multi-class control to handle four different classes of jobs in our next examples. This small number is chosen in order to keep the simulation costs reasonable. MULTI-SELF can theoretically handle any number of classes. The jobs in the first two classes are short jobs with high priorities and can be considered as terminal jobs. The jobs in the other two classes are longer jobs with low priorities and can be considered as batch jobs.

The mean response times of the four different classes of jobs under SELF and MULTI-SELF are shown in Table 5.9.

CLASS	WEIGHT	SELF RESP. TIME (SEC)	MULTI SELF RESP. TIME (SEC)	%IMP OVER SELF
1	2.5	0.4329	0.3000	30.70
2	2.0	0.4483	0.3191	28.82
3	1.5	2.0155	1.9418	3.66
4	1.0	4.4868	4.2737	4.75

Table 5.9 Mean Response Times of Jobs Under SELF and MULTI-SELF With Static Beta

It may be observed that there is a considerable improvement in the response times of short jobs with high priorities, whereas only marginal improvement is observed for longer jobs with low priorities. This improvement is achieved at the expense of additional overhead. The total overhead of MULTI-SELF for this configuration of the system and for the selected workload is approximately 4.32% of the total CPU time. The corresponding overhead for SELF is approximately 0.165%.

In the implementation of SELF and MULTI-SELF in the above example, the values of β (in equation (3.25)) are computed only once for each parameter and then these constant values are used throughout the experiment. One can improve the performance of these schemes by dynamically computing the values of β at each interval using equation (3.29), thus reducing the error in the estimation of the values of the workload parameters.

CLASS	WEIGHT	SELF RESP. TIME (SEC)	MULTI SELF RESP. TIME (SEC)	%IMP OVER SELF
1	2.5	0.3564	0.2875	23.96
2	2.0	0.3057	0.2998	1.92
3	1.5	1.8350	1.8029	1.74
4	1.0	4.1917	4.1322	1.41

Table 5.10 Mean Response Times of Jobs Under SELF and MULTI-SELF With Dynamic Beta

Table 5.10 shows the mean response times of the four classes of jobs with dynamic computation of β. It is observed that MULTI-SELF exhibits an improvement in the response time over SELF. It is not as much as in the case of Table 5.9. Furthermore, by dynamically recomputing the values of β in SELF an improvement is observed over SELF with static β (see Table 5.11). In the case of MULTI-SELF only marginal improvement is achieved when β is dynamically recomputed (see Table 5.12).

CLASS	WEIGHT	STATIC BETA RESP. TIME (SEC)	DYM. BETA RESP. TIME (SEC)	%IMP. OVER STATIC BETA
1	2.5	0.4329	0.3564	21.44
2	2.0	0.4483	0.3057	46.64
3	1.5	2.0155	1.8350	09.83
4	1.0	4.4868	4.1917	07.04

Table 5.11 Mean Response Times of Jobs Under SELF with Static and Dynamic Beta

CLASS	WEIGHT	STATIC BETA RESP. TIME (SEC)	DYM. BETA RESP. TIME (SEC)	%IMP. OVER STATIC BETA
1	2.5	0.3000	0.2875	4.17
2	2.0	0.3191	0.2998	6.05
3	1.5	1.9418	1.8029	7.15
4	1.0	4.2737	4.1322	3.31

Table 5.12 Mean Response Times of Jobs Under MULTI-SELF Resp. With Static and Dynamic Beta

The overhead involved in the case of SELF with dynamic computation of the values of β is approximately 12.40% of

CPU time, where as in the case of MULTI-SELF it is approximately 45.69%. Therefore we can conclude that it is not worthwhile to dynamically compute the values of β, at least in the case of MULTI-SELF. It seems better to implement MULTI-SELF with a constant value of β rather than SELF with dynamic values of β.

The above observations were made on the basis of a few examples. This is due to the high cost of simulation. However, the workload was carefully selected to reflect the worst case for these schemes. It is expected that the performance of these schemes will vary with different workloads but their order of magnitude will not differ significantly.

5.4 Error analysis

In this section we outline some of the most important assumptions made in order to make the models mathematically tractable and the control schemes practically feasible. We also analyse the error introduced because of these assumptions.

Assumption 1. Identical jobs

SELF assumes that all the jobs are identical in their resource demands, whereas MULTI-SELF assumes that the jobs

within each class are identical. In an actual system, job characteristics may vary widely. The synthetic workload selected to drive the simulator does not make this assumption. Not only different jobs have different characteristics, but also their characteristics change from time to time. The extent of the improvement obtained by classifying jobs into four classes can be seen in Table 5.9. Schonbach [Scho80] suggested a way of reducing this error by creating an independent class for each job. This may solve the problem to a certain extent but the overhead involved will also increase considerably.

Assumption 2. Estimation of Parameters

Both the schemes estimate the values of parameters on the basis of their past values. In order to reduce the error in the estimation we have used the simplest method of exponential smoothing. This error can be further reduced by dynamically computing the values of β in the exponential smoothing. Although dynamic computation of β does not require more than a maximum of 10 previous values (i.e., an insignificant storage requirement) the computation overhead in quite high. Moreover, it is observed that the improvement achieved by using dynamic β is marginal in the case of MULTI-SELF. A compromise is to recompute the value of β after large intervals of time. Table 5.13 shows the percentage error involved in the prediction of one of the

system parameter without smoothing, with static smoothing and with dynamic smoothing.

	% ERROR	RESP. TIME (SEC)	%IMP.
NO SMOOTHING	50.5	6.2271	___
STATIC SMOOTHING	12.6	4.4868	38.78
DYNAMIC SMOOTHING	04.4	4.1917	48.55

Table 5.13 %Error and %Imp. in Resp. Time under Static, Dynamic and No Smoothing

The improvements in both cases are significant over no smoothing. But there is not much improvement of dynamic smoothing with respect to static smoothing.

Assumption 3. Constant Degree of Multiprogramming

The two schemes require the degree of multiprogramming to be maintained at the computed level. This condition may not be satisfied during every observation interval. For example, during certain intervals there could be very light load (i.e., very few jobs) at the beginning followed by a sudden burst of jobs. Under such circumstances, the number of jobs in the system will be initially below the computed

number and then, because of the control, it will never exceed the control number. As a result, the mean number of jobs in the subsystem after the time interval will be less than the control number. This problem can be solved to a certain extent by comparing the control number with the mean number of current jobs in the system rather than with the actual number of current jobs in the system. An improvement of approximately 12% in response time was observed when this modification was made.

Assumption 4. Job Flow Balance

Operational analysis requires the job flow to be balanced at every service centre in the system. It is found that in the simulated system this is satisfied almost 95% of the time. Whenever it is not satisfied (i.e., the number of arrivals at a centre during an interval is not equal to the number of departures) the error[2] is never more than 2%.

There are a few other factors that effect the validity of the results. The observation interval and the CPU time quantum length are set to 3.0 and 0.010 simulated seconds respectively. These are the values used by the Michigan Terminal System at UBC. However, there are standard

---

[2] $\%\text{Error} = \frac{|\text{no. arrival} - \text{no. departure}|}{\text{no. Departure}} * 100$

techniques to compute the value of the observation interval (see [BoJe76]).

CHAPTER 6

CONCLUSIONS AND EXTENSIONS

This thesis has demonstrated the versatility of using optimization techniques along with queueing theory to solve some of the decision-making problems in computer system performance evaluation. The emphasis has been to obtain solutions on the basis of mathematical modelling. Closed form solutions were obtained wherever possible to reduce the computational overhead and to attempt a theoretical explanation of empirical findings.

The first problem considered was to obtain an optimal design for the memory hierarchy of a multiprogramming system. To our knowledge, this is the first work that considers explicit queueing at some of the memory levels. A model for estimating the optimal capacities and speeds of the memory hierarchy has been developed. It was assumed that the technology cost function and the hit-ratio

function can be represented by power functions. This appears to be a rough but reasonable approximation. The quantities to be optimized (mean response time in this case) are expressed as a function of the desired parameters through the use of queueing models. Optimization techniques are then applied to derive the optimal values for the design parameters. Using the assumptions stated in Chapter 1, the design problem is shown to have a global optimal solution. It was observed that there is a considerable difference between the optimal design of uniprogrammed and multiprogrammed systems. Therefore, the results obtained for a uniprogrammed system cannot be adequately used in the multiprogrammed case. The empirical observation that there is a constant ratio between the optimal sizes of different levels of memory was mathematically verified. It was also inferred that once a system has achieved the optimal memory size, any extra budget should be used in the acquisition of faster rather than more memory.

The technique presented here can be extended and applied to several other related problems. A natural extension is to include the CPU cost and speed in the design problem. One can also attempt to find the optimal amount of information (e.g., page size) that should be moved from one level to another, upon each request. A similar problem has been attempted by Trivedi

et al. [TrWS80]. By considering a closed queueing network model and by maximizing the throughput rate they find the optimal CPU speed, device capacities and allocation of files among various secondary devices. Most of the previous work in optimal system design assumes identical jobs. This strong assumption needs to be relaxed.

The second problem considered was to control the flow of jobs through a system. Two control schemes, SELF and MULTI-SELF, were developed. Unlike most other work, the schemes are based on mathematical modelling and thus their optimality can be proven. Furthermore, by the use of operational analysis, the assumptions used in the model formulation are minimized and all the required parameters are observable.

Basically, these schemes consist of two steps. In the first step, the saturation point of the multiprogramming subsystem is estimated. Next, the optimal ratio of the number of jobs from each class that should be maintained in the multiprogramming subsystem is computed. The objective in this optimization problem is to minimize a weighted sum of the waiting times of jobs in the system without saturating the system.

In the development of SELF it was assumed that all the jobs have identical resource demands. However, the scheme

is adaptive and is capable of modifying itself when the workload varies. If there is a sudden burst of jobs during an observation interval and the subsystem gets saturated, then during the subsequent observation intervals, more jobs are prevented from entering the subsystem until the system comes out of saturation. On the other hand, if the calculated saturation point is smaller than the actual saturation point, more jobs are allowed to begin execution in the subsequent interval.

The identical job assumption of SELF is relaxed by using mean-value analysis of multi-class systems. The multi-class scheme MULTI-SELF can handle any practical number of different classes of jobs. Although an improvement was observed over SELF when using MULTI-SELF, the overhead involved in the implementation of MULTI-SELF are also higher than that of SELF. In the development of MULTI-SELF it was assumed that once a job arrives at the system it is possible to determine the class to which it belongs. It is also assumed that the jobs do not change their class during their stay in the system. An extension to this work would be to relax these assumptions.

In order to compare the performance of SELF and MULTI-SELF with some of the major control schemes a general purpose simulator of a central server model was developed and different control schemes were implemented on it. The

workload that drives the simulator did not make most of the assumptions required for the development of SELF and MULTI-SELF. It was found that the two schemes perform better than the existing ones over a variety of workloads. The superiority of the two schemes over the other schemes increases as the variation in the workload increases. This shows that the two schemes are more adaptable to changes in the workload. Furthermore, to improve the accuracy, an exponential smoothing technique is used in the estimation of system and job parameters required by the two schemes. Finally, we would like to implement these schemes on an actual system and verify their performance using real workloads.

REFERENCES

[ArGa73] Arora, S. R. and A. Gallo, "Optimization of Static Loading and Sizing of Multilevel Memory System". JACM, Vol. 20, No. 2, April 1973, (307-319).

[BaLe78] Bedel, M., and J. Leroudier, "Adaptive Multiprogramming Systems Can Exist". in Performance of Computer Installations, D. Ferrari (ed.), North-Holland Publ. Co., 1978(115-135).

[BGLP75] Badel, M., E. Gelenebe, J. Leroudier, and D. Potier, "Adaptive Optimization of a Time-Sharing System's Performance". Proc. IEEE, Vol. 63, No. 6, June 1975(958-965).

[Baer74] Baer, J. L., "On Program Placement in a Directly Executable Hierarchy of Memories". IEEE Trans. Computers, Vol. C-23, No. 8, Aug. 1974(838-849).

[Bard79] Bard, Y., "Some Extensions to Multiclass Queuing analysis". in Performance of Computer Systems, M. Arato, A. Butrimenko, and E. Gelenbe (ed.), North-Holland Publ. Co., 1979(51-62).

[BCMP75] Baskett, F., K. M. Chandy, R. R. Muntz, and F. G. Palacois, "Open Closed and Mixed Networks of Queues with Different Classes of Customers". JACM, Vol. 22, No. 2, April 1975(248-260).

[BeKu69] Belady, L. A., and C. J. Kuehner, "Dynamic Space Sharing in Computer System". Comm. ACM, Vol. 12, No. 5, May 1969(282-288).

[BeBW72] Bell, T. E., B. W. Boehm, and R. A. Watson, "Framework and Initial Phase for Computer Performance Improvement". FJCC, 1972(1141-1154).

[BoJe76] Box, G. E. P., and G. M. Jenkins, Time-Series Analysis: Forcasting and Control. Holden-Day, San Francisco, 1976.

[BrBC77] Brown, R. M., J. C. Browne, and K. M. Chandy, "Memory Management and Response Time". Comm. ACM, Vol. 20, No. 3, March 1977(154-165).

[BCBK75] Browne, J. C., K. M. Chandy, R. M. Brown, T. N. Keller, D. F. Twosley, and C. W. Dissly, "Hierarchical Techniques for the Development of Realistic Models of Complex Computer Systems". Proc. IEEE, Vol. 63, No. 6, June 1975(966-975).

[Buze71] Buzen, J. P., "Analysis of System Bottlenecks Using a Queuing Network Model". Proc. ACM-SIGOPS Workshop on System Performance Evaluation, April 1971(82-103).

[Buze73] Buzen, J. P., "Computational Algorithms for Closed Queuing Netwirks with Exponential Servers". Comm. ACM, Vol. 16, No. 9, Sep. 1973(527-531).

[BuSh74] Buzen, J. P. and A. W. C. Shum, "Optimal Load Balancing in Memory Hierarchies". Proc. Eight Ann. Princeton Conf. on Information Sci. March 1974(335-339).

[Buze76] Buzen, J. P., "Operational Analysis: The Key to the New Generation of Performance Prediction Tool". Proc. IEEE COMPCON, N.Y., 1976(166-171).

[BuPo77] Buzen, J. P., and D. Potier, "Accuracy of Exponential Assumption in Closed Queuing Models". Proc. Sigmetrics Conf. On Computer Perf., 1977(53-64).

[Buze78a] Buzen, J. P., "A Queuing Network Model of MVS". Computing Surveys, Vol. 10, No. 3, Sep. 1978(319-331).

[Buze78b] Buzen, J. P., "Operational Analysis: An Alternate to Stochastic Modeling and Prediction". Proc. Int. Conf. Perf. Comp. Installation, North-Holland Publ. Co., 1978(175-194).

[BuDe80] Buzen, J. P., and P. J. Denning, "Operational Treatment of Queue Distributions and Mean-value Analysis". Computer Performance, Vol.1, no.1, June 1980(6-15).

[ChHW75a] Chandy. K. M., U. Herzog, and L. Woo, "Approximate Analysis of General Queuing Networks". IBM J. Res. And Develop. Jan. 1975(43-49).

[ChHW75b] Chandy. K. M., U. Herzog, and L. Woo, "Parametric Analysis of Queuing Networks". IBM J. Res. And Develop. Jan. 1975(36-42).

[ChHT77] Chandy, K. M., J. H. Howard Jr., and D. F. Towsley, "Product Form and Local Balance in Queuing Networks". JACM, Vol. 24, No. 2, April 1977(250-263).

[ChHS77] Chandy, K. M., J. Hogarth, and C. H. Sauer, "Selecting Capacities in Queuing Network Models of Computer/Communication Systems". IEEE Trans. Software Engg., Vol. SE-3, No. 4, July 1977(290-295).

[ChSa78] Chandy, K. M., and C. H. Sauer, "Approximate Method for Analysing Queuing Network Models of Computing Systems". Computing Surveys, Vol. 10, No. 3, Sep. 1978(281-317).

[Chan79] Chanson, S. T., "Saturation Estimation in Interactive Computer Systems". Dept. of Comp. Sci., Univ. of British Columbia, TR 79-7, 1979.

[ChTr79] Private Communication with S. T. Chanson and K. S. Trivedi, Jan 1979.

[ChSi80a] Chanson, S. T. and P. S. Sinha, "Optimization of Memory Hierarchies in Multiprogrammed Computer Systems with Fixed Cost Constraint". IEEE Trans. Computers July 1980(611-618).

[ChSi80b] Chanson, S. T., and P. S. Sinha, "Optimal Load Control in Combined Batch-Interactive Computer System". Proc. Of 16th Conf. Of CPEUG, Florida, October 1980(207-213).

[Chow74] Chow, C.K., "On Optimization of Storage Hierarchies". IBM J. Res. Devolop., Vol. 18, May 1974(194-203).

[Chow76] Chow, C. K., "Determination of Cache's Capacities and Its Matching Storage Hierarchy". IEEE Trans. Computers, Vol. C-25, No. 2, Feb. 1976(157-164).

[CoDe73] Coffman, E. G., Jr., and P. J. Denning, Operating System Theory. Prentice-Hall, Englewood Cliffs, N.J.,1973.

[Cour77] Courtois, P. J., Decomposibility - Queuing and Computer System Applications. Academic Press, 1977.

[Denn68a] Denning, P. J., "Thrashing: Its Causes and Prevention". Proc. AFIPS, 33, (FJCC) 1968(915-922).

[Denn68b] Denning, P. J., "The Working Set Model for Program Behaviour". Comm. ACM, Vol. 15, No. 5, May 1968(323-333).

[Denn70] Denning, P. J., "Virtual Memory". Computing Surveys, Vol. 2, No. 3, Sept. 1970(153-189).

[Denn71] Denning, p. J., "Third Generation Computer System". Computing Surveys, Vol. 3 No. 4, Dec. 1971(175-216).

[DeGr75] Denning, P. J., and G. S. Graham, "Multiprogramming Memory Management", Proc. IEEE, Vol. 63, No. 6, June 1975(924-939).

[DKLP76] Denning, P. J., K. C. Khan, J. Leroudier, D. Potier, and R. Suri, "Optimal Multiprogramming". Acta Informatica, 7, 1976(197-216).

[DeKh76] Denning, P. J., and K. C. Khan, "An L=S Criterion for Optimal Multiprogramming". Proc. Int'l. Symp. on Computer Performance Modeling, Measurement and Evaluation, March 1976(219-229).

[DeBu78] Denning, P. J., and J. P. Buzen, "The Operational Analysis of Queuing Network Models". Computing Surveys, Vol. 10, No. 3, Sep. 1978(225-261).

[DuPZ67] Duffin, R. J., E. L. Peterson and C. Zener, Geometric Programming. John Weley and Sons Inc., N.Y., 1967.

[Ferr78] Ferrari, D., Computer Systems Performance Evaluation. Prentice-Hall, 1978.

[FoBr76] Foster, D., and J. C. Browne, "File Assignment in Memory Hierarchies". Proc. Modelling and Perf. Eval. Of Computer Systems, North-Holland, Oct. 1976(119-127).

[Grav67] Gaver, D. P., "Probability Models for Multiprogramming Computer System". JACM, Vol. 14,No. 3, July 1967(423-438).

[Gece74] Gecsei, J., "Determining Hit Ratio for Multilevel Hierarchies". IBM J. Res. and Dev., Vol. 18, No. 4, July 1974(316-327).

[GeLu74] Gecsei, J., and T. A. Lukes, "A Model for the Evaluation of Storage Hierarchies". IBM Sys. J., Vol. 13, No. 2, 1974(163-178).

[GeMu76] Gelenbe, E., and R. R. Muntz, "Probabilistic Models of Computer Systems --Part-I (Exact Results)". Acta Informatica 7, 1976(35-60).

[GoNe67] Gorden, W. J., and G. F. Newell, "Closed Queuing Systems With Exponential Servers". Oper. Res., 15, 1967(254-265).

[GrDe77] Graham, G. S., and P. J. Denning, "On the Relative Controllability of Memory Policies". Proc. Int'l. Symp. on Computer Performance Modeling, Measurement and Evaluation, Aug. 1977(411-428).

[Hadl72] Hadley, G., Nonlinear and Dynamic Programming. Addison-Wesley Pub. Company, 1972.

[HiMT79] Hine, J. H., I. Mitrani, and S. Tsur, "The Control of Response Time in Multi-class Systems by Memory Allocation". Comm. ACM, Vol. 22, No. 7, July 1979(415-424).

[Jack63] Jackson, J. R., "Job Shop-like Queuing System". Management Science, Vol. 10, No. 1, Oct. 1963(131-142).

[Kend76] Kendall, M., Time-Series. Griffin, London, 1976.

[Klei68] Kleinrock, L., "Certain Analytic Results for Time-shared Processors". Info. Processing, Proc. IFIP Congress 1968(838-845).

[Land76] Landwehr, C. E., "An Endogenous Priority Model for Load Control in a Combined Batch-Interactive Computer System". Proc. Int'l. Symp. on Computer Performance Modeling, Measurement and Evaluation, March. 1976(282-287).

[Lazo77] Lazowska, E. W., "The Use of Percentiles in Modeling CPU Service Time Distribution". in Computer Performance, K. M. Chandy and M. Reiser (ed.), Elsevier North-Holland Inc., new York, 1977(53-66).

[LePo76] Leroudier, J., and D. Potier, "Principles of Optimality for Multiprogramming". Proc. Int'l. Symp. on Computer Performance Modeling, Measurement and Evaluation, March 1976(211-218).

[LeSc71] Lewis, P. A. W., and G. S. Schedler, "A Cyclic-Queue Model of System Overhead in Multiprogrammed Computer systems". JACM, Vol. 18, 1971(199-220).

[LiMa72] Lin, Y. S., and R. L. Mattson, "Cost-Performance Evaluation of Memory Hierarchies". IEEE Trans. Magnetics, Vol. MAG-8, No. 3, Sep 1972(390-392).

[LiCh77] Lipsky, L., and J. D. Church, "Application of Queuing Network Model for a Computer System". ACM Computing Surveys, Vol. 9 No. 3, Sep. 1977(205-222).

[Litt61] Little, J. D. C., "Proof for the Queuing Formula L=λW". Operations Research, 9, 3, May 1961(383-387).

[Lo80] Lo, R., "The Application of Optimal Stochastic Control Theory to Adaptive Performance Control of Computer Systems". M.Sc. Thesis, Dept. of Comp. Sci., Univ. of British Columbia, 1980.

[MaSi75] MacDonald, J.E., and K.L. Sigworth, "Storage Hierarcy Optimization Procedure". IBM J. Res. Develop., Vol 19, March 1975(133-140).

[Matt71] Mattson, R.L., "Evaluation of Multilevel Memories". IEEE Trans. Magnetics, Vol. MAG-7, December 1971(814-819).

[Pugh71] Pugh, E. W., "Storage Hierarchies: Gaps, Cliffs, and Trends". IEEE Trans. Magnetics, Vol. MAG-7, No. 4, Dec. 1971(810-814).

[RaCh70] Ramamoorthy,C.V., and K.M. Chandy, "Optimization of Memory Hierarchy in Multiprogramming System". JACM, Vol.17, July 1970(426-445).

[Rege76] Rege, S. L., "Cost, Performance and Size Trade-Off for Different Levels in Memory Hierarchy". Computer, Vol. 9, No. 4, April 1976(43-51).

[Reis76] Reiser, M., "Interactive Modeling of Computer System". IBM Sys. J., Vol. 15, No. 4, 1976(309-327).

[ReKo76] Reiser, M., and H. Kobayashi, "On the Convolution Algorithm for Separable Queuing Networks". Proc. Int'l. Symp. Comp. Per. Modelling Measurement and Evaluation, 1976(109-117).

[Reis79] Reiser, M., "Mean Value Analysis of Queuing Network, A New Look at an Old Problem". in Performance of Computer System, M. Arato, A. Butrimenko, and E. Gelenbe (ed.), North-Holland Publ. Co., 1979(63-77).

[ReLa80] Reiser, M., and S. S. Lavenberg, "Mean-Value Analysis of Closed Multiclass Queuing Network". JACM, Vol. 27, No2, April 1980(313-322).

[Rode79] Rode, J. D., "Multiclass Operational Analysis of Queuing Network", in Performance of Computer Systems, M. Arato, A. Butrimenko, and E. Gelenbe (ed.), North-Holland Publ. Co., 1979(339-352).

[RoDu73] Rodriguez-Rosel, J., and J. P. Dupuy, "The Design, Implementation, and Evaluation of Working Set Dispatcher". Comm. ACM, Vol. 16, No. 4, April 1973(247-253).

[Scho80] Schonbach, A., "Macro-Scheduler for High Productivity". Ph.D. Thesis, Dept. of Comp. Sci., Univ. of Toronto, 1980.

[Spos75] Sposito, V. A., Linear and Nonlinear Programming. Iowa: Iowa State Univ. Press/ames, 1975.

[TrMa71] Traiger, I. L., and R. L. Mattson, "The Evaluation and Selection of Technologies for Computer Storage System". AIP Conference Proceedings, No. 5, Part I, Magnetism and Magnetic Materials, 1971.

[TrSe77] Tripathi, S. K., and K. C. Sevick, "The Influence of Multiprogramming Limit on Interactive Response Time in a Virtual Memory System". Proc. Sigmetrics Conf. On Computer Performance 1977(121-130).

[Triv78] Trivedi, K. S., "Analytic Modeling of Computer Systems". Computer, Oct. 1978(38-56).

[TrWa80] Trivedi, K. S., and R. A. Wagner, "A Decision Model for Closed Queuing Networks". IEEE Trans. Software Eng. SE-5, 4, July 1979(328-332).

[TrWS80] Trivedi, K. S., R. A. Wagner, and T. M. Sigmon, "Optimal Selection of CPU Speed, Device Capacities, and File Assignments". JACM Vol.27, No.3, July 1980(457-473).

[Welc78] Welch, T.A., "Memory Hierarchy Configuration Analysis". IEEE Trans. on Computers, Vol. C-27, No. 3, May 1978(408-413).

[Will73] Williams, J. G., "Asymetric Memory Hierarchies". CACM, Vol. 16, No. 4, April 1973(213-222).

Appendix A

SYSTEM / JOB CHARACTERISTICS	TERMINAL	BATCH
ARR. RATE(/SEC.)	3.0	1.25
DELTA ARR. RATE*(/SEC)	1.50	1.00
I/O REQU. RATE(/SEC.)	250	99
DELTA I/O REQU. RATE**	100	20
EXIT PROBABILITY	0.90	0.90
DELTA EXIT PROB.***	0.10	0.10
I/O SERVICE RATE(/SEC)	30	30
PAGE SERV. RATE(/SEC)	100	100
QUANTUM LENGTH(SEC)	0.01	0.01
B (EQU 5.1)	0.01	0.01
C (EQU 5.1)	120	120
MAIN MEMORY (PAGES)	500	500
OBSERVATION INTERVAL LENGTH (SEC)	3	3
DELTA ARR. PERIOD+(SEC)	12	12
DELTA CHARACTERISTIC PERIOD++ (SEC)	3	3

TABLE A.1 WORKLOAD FOR TABLE 5.1

SYSTEM / JOB CHARACTERISTICS	TERMINAL	BATCH
ARR. RATE(/SEC.)	5.0	3.00
DELTA ARR. RATE*(/SEC)	2.50	1.50
I/O REQU. RATE(/SEC.)	250	50
DELTA I/O REQU. RATE**	100	20
EXIT PROBABILITY	0.90	0.90
DELTA EXIT PROB.***	0.10	0.10
I/O SERVICE RATE(/SEC)	30	30
PAGE SERV. RATE(/SEC)	100	100
QUANTUM LENGTH(SEC)	0.01	0.01
B (EQU 5.1)	0.01	0.01
C (EQU 5.1)	80	80
MAIN MEMORY (PAGES)	500	500
OBSERVATION INTERVAL LENGTH (SEC)	3	3
DELTA ARR. PERIOD+(SEC)	12	12
DELTA CHARACTERISTIC PERIOD++ (SEC)	3	3

TABLE A.2 WORKLOAD FOR TABLE 5.2

* Variation in arrival rate.
** Variation in I/O request rate.
*** Variation in exit probability.
\+ Period of variation in exit probability.
++ Period of variation in job charateristics.

SYSTEM / JOB CHARACTERISTICS	TERMINAL	BATCH
ARR. RATE(/SEC.)	3.5	1.50
DELTA ARR. RATE*(/SEC)	0.50	0.00
I/O REQU. RATE(/SEC.)	250	50
DELTA I/O REQU. RATE**	50	10
EXIT PROBABILITY	0.90	0.90
I/O SERVICE RATE(/SEC)	30	30
PAGE SERV. RATE(/SEC)	100	100
QUANTUM LENGTH(SEC)	0.01	0.01
B (EQU 5.1)	0.01	0.01
C (EQU 5.1)	120	120
MAIN MEMORY (PAGES)	500	500
OBSERVATION INTERVAL LENGTH (SEC)	3	3
DELTA ARR. PERIOD+(SEC)	12	12
DELTA CHARACTERISTIC PERIOD++ (SEC)	3	3

TABLE A.3 WORKLOAD FOR TABLE 5.3

SYSTEM / JOB CHARACTERISTICS	TERMINAL	BATCH
ARR. RATE(/SEC.)	4.5	2.50
DELTA ARR. RATE*(/SEC)	1.50	0.50
I/O REQU. RATE(/SEC.)	250	50
DELTA I/O REQU. RATE**	50	10
EXIT PROBABILITY	0.87	0.87
DELTA EXIT PROB.***	0.05	0.05
I/O SERVICE RATE(/SEC)	30	30
PAGE SERV. RATE(/SEC)	100	100
QUANTUM LENGTH(SEC)	0.01	0.01
B (EQU 5.1)	0.01	0.01
C (EQU 5.1)	120	120
MAIN MEMORY (PAGES)	500	500
OBSERVATION INTERVAL LENGTH (SEC)	3	3
DELTA ARR. PERIOD+(SEC)	12	12
DELTA CHARARATERISTIC PERIOD++ (SEC)	3	3

TABLE A.4 WORKLOAD FOR TABLE 5.4

SYSTEM / JOB CHARACTERISTICS	TERMINAL	BATCH
ARR. RATE(/SEC.)	4.00	2.00
DELTA ARR. RATE*(/SEC)	1.50	0.50
I/O REQU. RATE(/SEC.)	250	50
DELTA I/O REQU. RATE**	50	10
EXIT PROBABILITY	0..87	0.87
DELTA EXIT PROB.***	0.05	0.05
I/O SERVICE RATE(/SEC)	30	30
PAGE SERV. RATE(/SEC)	100	100
QUANTUM LENGTH(SEC)	0.01	0.01
B (EQU 5.1)	0.01	0.01
C (EQU 5.1)	120	120
MAIN MEMORY (PAGES)	500	500
OBSERVATION INTERVAL LENGTH (SEC)	3	3
DELTA ARR. PERIOD*(SEC)	12	12
DELTA CHARACTERISTIC PERIOD++ (SEC)	3	3

TABLE A.5 WORKLOAD FOR TABLE 5.5

SYSTEM / JOB CHARACTERISTICS	TERMINAL	BATCH
ARR. RATE(/SEC.)	5.0	2.5
DELTA ARR. RATE*(/SEC)	2.50	1.50
I/O REQU. RATE(/SEC.)	250	50
DELTA I/O REQU. RATE**	100	20
EXIT PROBABILITY	0.90	0.90
DELTA EXIT PROB.***	0.10	0.10
I/O SERVICE RATE(/SEC)	30	30
PAGE SERV. RATE(/SEC)	100	100
QUANTUM LENGTH(SEC)	0.01	0.01
B (EQU 5.1)	0.01	0.01
C (EQU 5.1)	120	120
MAIN MEMORY (PAGES)	500	500
OBSERVATION INTERVAL LENGTH (SEC)	3	3
DELTA ARR. PERIOD*(SEC)	12	12
DELTA CHARACTERISTIC PERIOD++ (SEC)	3	3

TABLE A.6 WORKLOAD FOR TABLE 5.6

SYSTEM / JOB CHARACTERISTICS	TERMINAL	BATCH
ARR. RATE(/SEC.)	5.0	3.50
DELTA ARR. RATE*(/SEC)	2.50	1.50
I/O REQU. RATE(/SEC.)	250	50
DELTA I/O REQU. RATE**	100	20
EXIT PROBABILITY	0.90	0.90
DELTA EXIT PROB.***	0.10	0.10
I/O SERVICE RATE(/SEC)	30	30
PAGE SERV. RATE(/SEC)	100	100
QUANTUM LENGTH(SEC)	0.01	0.01
B (EQU 5.1)	0.01	0.01
C (EQU 5.1)	120	120
MAIN MEMORY (PAGES)	500	500
OBSERVATION INTERVAL LENGTH (SEC)	3	3
DELTA ARR. PERIOD+(SEC)	12	12
DELTA CHARACTERISTIC PERIOD++ (SEC)	3	3

TABLE A.7 WORKLOAD FOR TABLE 5.7

SYSTEM / JOB CHARACTERISTICS	TERMINAL	BATCH
ARR. RATE(/SEC.)	5.0	2.50
DELTA ARR. RATE*(/SEC)	2.50	1.50
I/O REQU. RATE(/SEC.)	250	50
DELTA I/O REQU. RATE**	100	20
EXIT PROBABILITY	0.90	0.90
DELTA EXIT PROB.***	0.10	0.10
I/O SERVICE RATE(/SEC)	30	30
PAGE SERV. RATE(/SEC)	100	100
QUANTUM LENGTH(SEC)	0.01	0.01
B (EQU 5.1)	0.01	0.01
C (EQU 5.1)	120	120
MAIN MEMORY (PAGES)	500	500
OBSERVATION INTERVAL LENGTH (SEC)	3	3
DELTA ARR. PERIOD+(SEC)	12	12
DELTA CHARACTERISTIC	3	3

TABLE A.8 WORKLOAD FOR TABLE 5.8

SYSTEM / JOB CHARACTERISTICS	CLASS 1	CLASS 2	CLASS 3	CLASS 4
ARR. RATE(/SEC.)	2.75	2.25	1.75	1.25
DELTA ARR. RATE*(/SEC)	0.50	0.50	0.50	0.50
I/O REQU. RATE(/SEC.)	250	200	75	50
DELTA I/O REQU. RATE**	50	50	10	10
EXIT PROBABILITY	0.78	0.80	0.90	0.92
DELTA EXIT PROB.***	0.05	0.05	0.05	0.05
I/O SERVICE RATE(/SEC)	30	30	30	30
PAGE SERV. RATE(/SEC)	100	100	100	100
QUANTUM LENGTH(SEC)	0.01	0.01	0.01	0.01
B (EQU 5.1)	0.01	0.01	0.01	0.01
C (EQU 5.1)	120	120	120	120
MAIN MEMORY (PAGES)	500	500	500	500
OBSERVATION INTERVAL LENGTH (SEC)	3	3	3	3
DELTA ARR. PERIOD (SEC)	12	12	12	12
DELTA CHARACTERISTIC PERIOD++ (SEC)	3	3	3	3

TABLE A.9 WORKLOAD FOR TABLE 5.9 AND TABLE 5.12