OPTIMAL MACRO-SCHEDULING

by

Samuel T. Chanson and Prem S. Sinha

TECHNICAL REPORT 81-11

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

August 1981.

Index Terms:

Load control, system saturation, degree of multiprogramming, operational analysis, queueing network models, workload, optimization theory.

Abstract

A multi-class macro-scheduler is described in this paper. The scheduler periodically determines the number of jobs from each class that should be activated to minimize a weighted sum of the mean system residence time without saturating the system. The computation is based on the estimated system workload in the next interval. Thus it is adaptive to workload variation. The service provided to each class (specifically, the mean response time) may be adjusted by changing the weight associated with the job class.

The scheme is based on mathematical modelling. The solution is obtained through the use of operational analysis method and optimization theory. Exponential smoothing technique is employed to reduce the error of estimating the value of the model parameters. From our simulation results, the scheme appears to be both stable and robust. Performance improvement over some of the existing schemes (the 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 used in the formulation and solution of the model are discussed.

1. Introduction

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 the multiprogramming set should be carefully controlled. This is the function of the 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.

The term macro-scheduling as used in this paper refers to the policy that determines which jobs may migrate from the eligible queues to the multiprogramming queues. The sequencing of jobs in the multiprogramming set to be allocated the CPU is often called micro-scheduling. The former scheduling policy has far greater effect on global system performance and is the subject of discussion in this

paper.

There are two parts to macro-scheduling which are mutually dependent on one another:

i) the determination of the optimal number of jobs in the multiprogramming set (optimal degree of multiprogramming) and,
ii) the composition of the jobs in the set to optimize some performance criteria.

It has been shown that for paged virtual memory systems, an optimal degree of multiprogramming exists which maximizes the system throughput rate (see for example [3]). This point is often reached just before the system saturates. Hence an integral function of many macro-schedulers is to estimate the system's saturation point [1,15].

Considerable amount of work has been done on macro-scheduling (see for example [1-13]). Most of these, however, deal with part i) above only. Typically they work by regulating the load to keep some measures related to program behaviour (usually the paging behaviour) to within some predetermined bounds. Generally the bounds are set according to some heuristics to hopefully allow the highest possible load without saturating the system. The 50%

criterion [4] for example, aims at maintaining the utilization of the paging device to around 0.5. The L=S criterion [13] proposes to keep the system life-time to approximately that of the page swap time. The Knee criterion ([4],[9]) suggests that the mean resident set of each process should be maintained at the value associated with the primary knee[1] of its life-time function, where life-time is defined to be the virtual time between two successive page faults [4].

Queueing theory is the most prevalent mathematical tool used in the analysis of computer systems. It basically gives steady-state results. Thus queueing theory may be useful in system design (see for example [25]) and other problems but is not directly suitable to the dynamic control of computer systems. As a result, most work in the past resorted to heuristic applications of control-theoretics. There are a few exceptions.

Chanson and Lo [16] for example, 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

---

[1] The point where the curve between the system life-time and the number of active jobs has maximum slope.

interarrival and service time distributions which may not be satisfied in practice.

Schonbach [17] too describes a macro scheduler based on mathematical modelling for high productivity. It is assumed that the "system balance" point is already specified. Here, system balance is a state in which the various processor utilizations are at some prespecified levels. The macro-scheduler then chooses, 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.

Most large scale virtual memory systems nowadays support both batch and interactive jobs. For such systems, one is interested not only to maximize the system throughput rate but also to guarantee good response times to the interactive users (possibly at the expense of the batch turnaround times). Landwehr [5] proposed a scheme to activate batch jobs based on the terminal load. The aim was to maintain good response to interactve requests by activating less batch jobs when the terminal load is heavy while ensuring a minimal level of batch throughput. There was, however, no attempt to prevent the system from becoming saturated or to optimize performance. Hine et al. [14] studied the problem from a slightly different viewpoint. Their goal was to provide different response

times to each class of jobs (batch and interactive) while maximizing the CPU utilization. They employed a mathematical model but optimization was achieved using an exhaustive search technique. A heuristic was also given which gives good but not necessarily optimal results.

In this paper, we describe an adaptive macro-scheduler which is based on the application of optimization theory on a multi-job-class model of multiprogrammed computer systems. The system model is solved using operational analysis [18]. The scheduler computes the optimal number of jobs from each class that should be activated to minimize a weighted sum of the mean system residence time (including the wait time in the eligible queues) without saturating the system. The weights can be adjusted to favour some classes of jobs (whose mean response times will decrease) at the expense of the jobs in the other classes. The scheme is applicable to pure interactive, combined batch/interactive systems as well as non-virtual memory systems. Its performance is compared to some existing schemes.

2. The Model and Notations

The model used is given in Figure 1 with the multiprogramming subsystem being represented by the popular central-server model [19].

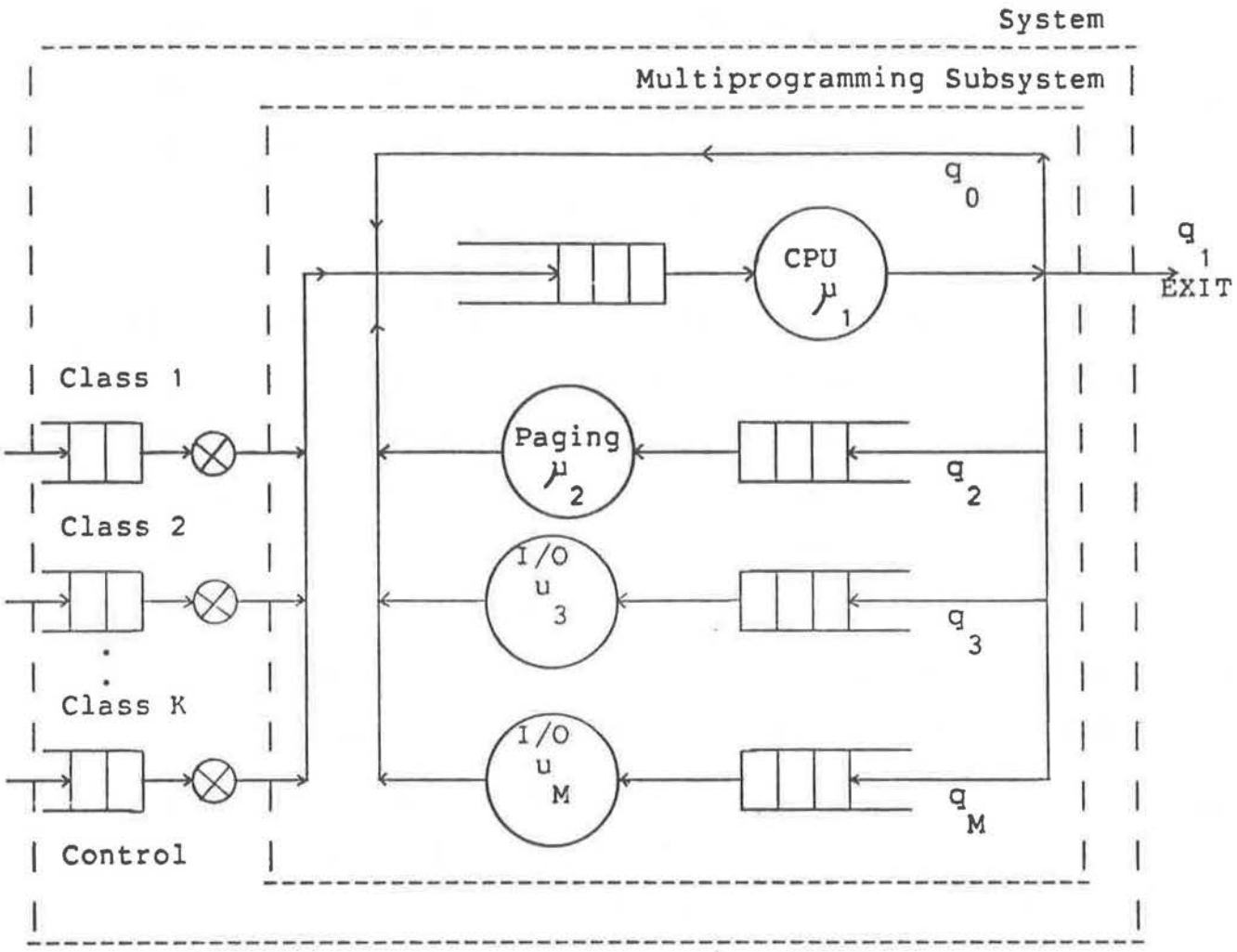

Figure 1. System Model

In this analysis, it is assumed that the jobs do not change class during their stay in the system. 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 the job 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. The error introduced by job classification is discussed in the next section.

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 multiprogramming subsystem,
s*(j)	:	optimal number of class j jobs in the multiprogramming subsystem (to be computed),
$\underline{N}$	:	the system state vector (s(1), s(2), ..., s(K)),
n(i,j,$\underline{N}$)	:	number of class j jobs at the centre i for a given system state $\underline{N}$,
$n_i(\underline{N})$	:	total number of jobs at the centre i for a given system state $\underline{N}$,

$\lambda(j)$: throughput rate of class j jobs from the multiprogramming subsystem,

$w(i,j)$: mean time a class j job spends at service centre i during its stay in the multiprogramming subsystem (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 the 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 easly shown that $q_i(\underline{N})$ is given by :

$$q_i(\underline{N}) = \sum_{j=1}^{K} n(1,j,\underline{N})\, q(i,j) \;/\; n_1(\underline{N}). \qquad (2.1)$$

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

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

We shall follow Reiser and Lavenberg's method (see [20] for details) to compute $n(i,j,\underline{N})$ by solving the following set of non-linear equations iteratively until the error is acceptably small:

$$w(i,j) = d(i,j)\,(1 + n_i - \sum_{r=1}^{K} \epsilon(r,j,i,\underline{N})) \qquad (2.2)$$

$$\lambda(j) = s(j) \;/\; \sum_{i=1}^{K} w(i,j) \qquad (2.3)$$

$$n(i,j) = \lambda(j)\, w(i,j) \qquad (2.4)$$

where $\epsilon(r,j,i,\underline{N})$ is a correction term given by

$$\epsilon(r,j,i,\underline{N}) = n'(i,j,\underline{N}) - n'(i,j,\underline{N}'), \qquad r = i$$
$$= 0, \qquad \text{otherwise.}$$

$n'(i,j,\underline{N})$ has the same definition as $n(i,j,\underline{N})$ except that the system has modified parameters for the traffic intensities (see equation (4.8) in [20]) and $\underline{N}'$ is the same as $\underline{N}$ except that the number of class j jobs in the system

is one less than that in $\underline{N}$.

Once the $n(i,j,\underline{N})$'s are known for a given $\underline{N}$, one can compute $q_i(\underline{N})$ from equation (2.1).

We first compute the saturation load of the system.

3. Estimation of System Saturation

Definitions of system saturation have been proposed [18,21]. Invariably the system is considered saturated at the point the response time vs system load curve starts to rise rapidly. Kleinrock [21], for example, using the number of active terminals as the load, defined the system's saturation point to correspond to the intersection of the mean normalized response time curve asymptote and the horizontal line corresponding to the minimum response time (i.e., when there is only one active terminal). (See Figure 2). If a system is not allowed to get saturated according to this definition, the mean response time of the active jobs will not exceed an acceptable level. However, the implicit assumption is that the program population considered is both homogeneous and stationary. Our approach is to compute the system saturation load at small intervals (such as a few seconds) during which the stationary assumption is justified. The homogeneous assumption is discussed below.

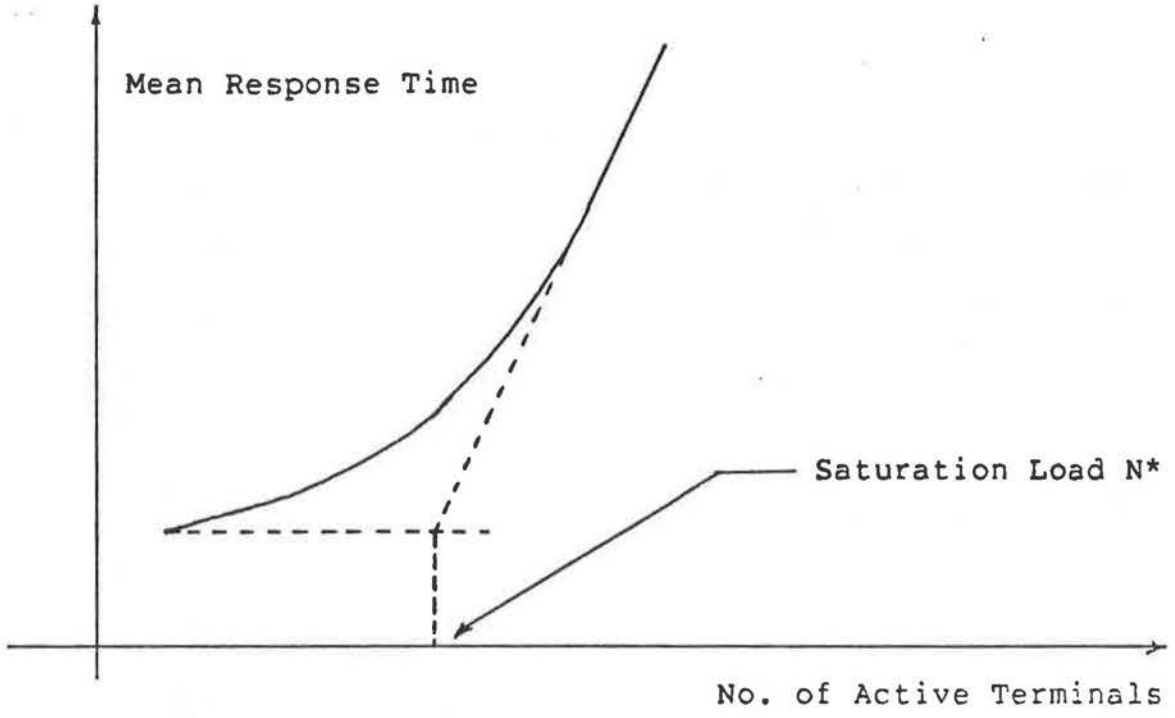

Figure 2. Mean Response Time vs. The Number of Active Terminals.

In a previous paper, we derived the saturation load of a uniclass model SELF (Saturation Estimation and Load-control with Feed-back) [1]. Following similar arguments (see Appendix A) it can be shown that the saturation vector $\underline{N}^*$ of the multi-class model is given by the relation

$$|\underline{N}^*| = (\mu_i^*/\mu_1 q_i^*(\underline{N})) \left[1 + \sum_{i=2}^{M} \frac{\mu_1}{\mu_i} q_i(\underline{N})\right] \qquad (3.1)$$

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 longer a single number as in the case of the single class model. Rather it is a vector and there can be several vectors satisfying condition (3.1). This is why we need to use optimization techniques to select the optimal value of $\underline{N}^*$.

4. Optimization

The next task is to compute the number of jobs in each class which optimizes some performance indices such that the sum total is less than or equal to $|\underline{N}^*|$. An obvious choice of performance index is the system throughput rate. However, this does not allow the inclusion of job priorities easily. Furthermore, it usually leads to a dynamic programming problem which in turn requires high computational overhead. Instead, we have chosen to minimize a weighted sum of the mean time spent by the jobs in the system (including the wait time in the eligible queues) subject to the constraint that the system is not saturated.

The mean system residence time of a class j job, W_j, is the sum of the times it spends in the eligible queues and the multiprogramming subsystem.

$$W_j = R(j) + (S(j)-s(j)) \,/\, (s(j)/R(j)), \quad j=1,2,\ldots,K. \quad (4.1)$$

Thus the optimization problem becomes

$$\text{Min } \left[\sum_{i=1}^{K} C(i)\ (W_i) \right]$$

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

Using (4.1) the above optimization problem can be shown to be equivalent to

$$\text{Min } \left[\sum_{i=1}^{K} C(i)\ S(i)\ R(i)/s(i) \right]$$

$$\text{subject to } \sum_{i=1}^{K} s(i) \leq |N^*| \qquad (4.2)$$

Given the C(i)'s and the estimated values of S(i)'s and R(i)'s, we wish to determine the optimal values of s(i)'s in the next interval. We shall use the Lagrange multiplier method to solve the problem (4.2). The lagrangian equation of the problem is:

$$L(\underline{N},\lambda) = \sum_{i=1}^{K} C(i)\ S(i)/s(i) + \lambda\left(\sum_{i=1}^{K} s(i)/|N^*| - 1 \right) \qquad (4.3)$$

The optimal values of s(i)'s are given by:

$$s(i)^* = \frac{|N^*|\ \sqrt{C(i)S(i)R(i)}}{\sum_{j=1}^{K} \sqrt{C(j)S(j)R(j)}}, \quad i = 1, 2, \ldots, K. \qquad (4.4)$$

It can be shown that only (K-1) out of K relations in

(4.4) are linearly independent. However, in order not to saturate the system, the s(i)'s must satisfy (3.1). Therefore, there are K unknowns (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 the computation of $q_i(\underline{N})$ in equation (2.2).

As in all feed-back schemes 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 technique from time-series analysis [23], called exponential smoothing. The technique is 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 time t. P_i can be expressed as

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

$$= (1-\beta) \sum_{j=0}^{\infty} \beta^j X_{i-j} \qquad (4.5)$$

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

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

$$\Rightarrow \quad P_i = (1-\beta)X_i + \beta P_{i-1} \qquad (4.6)$$

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

$$\epsilon_i = X_i - P_{i.1} \tag{4.7}$$

Substituting in equation (4.6)

$$\begin{aligned} P_i &= X_i - \beta\, \epsilon_i \\ &= P_{i.1} + (1-\beta)\epsilon_i \end{aligned} \tag{4.8}$$

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

If the error ϵ_i is sufficiently small, equation (4.8) 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 square of errors given by:

$$\sum_{j=i}^{-\infty} \{X_j - (1-\beta) \sum_{k=0}^{\infty} \beta^k X_{j.k.1}\}^2. \tag{4.9}$$

In practice, the summation in equation(4.9) does not have to involve many terms (say J) before β^J approaches zero (J was found to be around 3 and never greater than 10 in our experiments). Moreover, β need not be very accurate and standard techniques exist for its efficient computation.

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 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 (2.1) through (2.4), (3.1) and (4.4) 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.

5. Simulation Results

To compare the performance of the proposed macro-scheduler with some of the existing schemes (specifically the 50%, L=S and Knee criteria) and to see how it works when some of the assumptions used in its

derivation are not satisfied, a simulator was built. For example the jobs that drive the simulator 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. For details of the simulator see [24].

Our previous single-class (though batch and terminal jobs are treated differently, the model assumes their characteristics to be identical) model (SELF [1]) represents the worst case performance of MULTI-SELF. It can be considered as the case when job classification is completely random. We first compare SELF with the three above mentioned load control criteria and then SELF is compared with MULTI-SELF. The overhead involved and the error introduced by the assumptions used in the formulation and solution of the model are briefly discussed.

Since simulation runs are expensive, the runs were made as short as possible. Runs of 120 simulated seconds were made. It was found that the mean response time stabilized around 120 seconds and approximately 200 jobs were processed in that interval. A control decision is made every 3 seconds.

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=μS. The best value of μ depends upon the job characteristics. In our experiments, a best value of μ was obtained for each different workload that was run. Some of the results obtained are presented in Tables 5.1 through 5.3. The parameters for the workload corresponding to these results are given in Appendix B.

	SELF RESP. TIME (SEC)	50% RESP. TIME (SEC)	%IMP.[2] OVER 50%	L=S RESP. TIME (SEC)	%IMP OVER L=S
TERMINAL	0.5409	0.5531	2.25	0.5444	0.64
BATCH	0.8901	0.9600	7.85	0.9574	7.56
SYSTEM	0.6552	0.6785	3.58	0.6724	2.62

Table 5.1 Comparison of the performance of SELF, 50% and L=S criteria in terms of mean response time. The workload is defined in Table B.1

---

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

	SELF RESP. TIME (SEC)	50% RESP. TIME (SEC)	%IMP.[2] OVER 50%	L=S RESP. TIME (SEC)	%IMP OVER L=S
TERMINAL	0.6111	0.8262	35.19	0.8404	37.52
BATCH	1.5418	2.2729	47.41	2.0050	30.04
SYSTEM	0.9531	1.3586	42.54	1.2684	33.08

Table 5.2 Comparison of the performance of SELF, 50% and L=S criteria in terms of mean response time. The workload is defined in Table B.2.

	SELF RESP. TIME (SEC)	50% RESP. TIME (SEC)	%IMP.[2] OVER 50%	L=S RESP. TIME (SEC)	%IMP. OVER L=S
TERMINAL	1.9096	2.7638	44.73	2.6061	36.47
BATCH	3.8750	4.9179	26.91	4.4218	14.11
SYSTEM	2.5218	3.4243	35.78	3.1707	25.73

Table 5.3 Comparison of the performance of SELF, 50% and L=S criteria in terms of mean response time. The workload is defined in Table B.3

It is observed that the larger the workload variation, the better 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

5.1).

Although the Knee criterion is better than the 50% and the L=S criteria, it is expensive to be implemented in practice [9]. However, since the workload of the simulator is distribution-driven and the life-time function is given by equation (5.1) [22] (where b and c are constants and p is the average number of page allocated to each job), it is possible to simulate the Knee criterion without excessive overhead.

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

The Knee criterion requires each job to run at the knee of its life-time function, i.e., the point where the curve between the mean life-time of a process and its memory allocation 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 value of b and c one can create a best or 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 (5.4) 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 5.4 Comparison of the performance of SELF, 50%, L=S and Knee criteria in terms of mean response time. The workload is described in Table B.4.

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

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

(a) Overhead involved in collecting the data during the observation intervals.

(b) Overhead involved in the computation of the control number.

The overhead (a) depends upon the system configuration (e.g., number of I/O units etc.,) and job characteristics (e.g., total CPU requirement, number of I/O requests

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

We now compare SELF with MULTI-SELF. We use multi-class control to handle four different classes of jobs in our next examples. The small number is chosen in order to keep the simulation cost 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 is shown in Table 5.5.

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 5.5 Comparison of the performance of SELF and MULTI-SELF with static beta in terms of mean response time. The workload is defined in Table B.5.

It is 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 the selected workload is approximately 4.32% of CPU time compared to 0.165% for SELF.

In the implementation of SELF and MULTI-SELF in the above example, the values of β (in equation (4.5)) 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 (4.9), thus reducing the error in the estimation of the values of the workload parameters.

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 5.6 Mean response time of jobs under SELF with static and dynamic beta. The workload is defined in Table B.5.

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 5.7 Mean response times of jobs under MULTI-SELF with static and dynamic beta. The workload is defined in Table B.5.

From Tables 5.6 and 5.7, it is observed that dynamically varying β is more beneficial to SELF than MULTI-SELF. This is probably due to the fact that variation of job characteristics within a class is small compared to that of the total workload and far more predictable. Thus static β is often adequate in the multiclass case. As a result, the improvement in performance of MULTI-SELF over SELF in the case of dynamic β is small, averaging only 9% using workload B.5.

The overhead involved in the case of SELF with dynamic computation of the values of β is approximately 12.40% of CPU time whereas 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. In the case of SELF, it seems better to implement MULTI-SELF with constant value of β rather than SELF with dynamic values of β.

The above observations were made on the basis of a few experiments. This is due to the high cost of simulation. However, the workload was carefully selected to reflect the worst case for MULTI-SELF. It is expected that the performance of the scheme will vary with different workloads but the 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 scheme practically feasible. We also analyze 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 do different jobs have different characteristics, but their characteristics also change from time to time. The extent of the improvement obtained by classifying jobs into four classes can be seen in Table 5.5. Schonbach [17] 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 SELF and MULTI-SELF 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., insignificant storage requrement) the computational overhead is 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 5.8 shows the percentage error involved in the prediction of one of the system parameters 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 5.8 %Error and %Improvement in mean response time under static, dynamic and no smoothing

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

Assumption 3. Constant Degree of Multiprogramming

It may not be possible to maintain the degree of multiprogramming at the computed level. For example, during certain intervals the load could be very light (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 multiprogramming subsystem after the time interval will be less than the desired 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 the actual number of current jobs in the system. An improvement of approximately 12% in response time was observed by making this modification.

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 in an interval is not equal to the number of departures) the error[3] is never more than 2%. (3 seconds interval were used in our experiments).

Conclusion

We have presented a macro-scheduler which determines the number of jobs from each job class that should be activated in each interval to minimize a weighted sum of the mean system residence time without saturating the

---

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

system. The macro-scheduler is based on mathematical modelling and the solution is obtained through the use of operational analysis and optimization theory. Exponential smoothing technique is employed to reduce the error of estimating the values of the parameters. The analyst may determine the service given to each class of jobs by adjusting their associated weights. Our simulation results show the scheme to be robust and its performance is superior to some existing schemes. The overhead involved is acceptable if β is not varied dynamically.

Acknowledgement

This work was supported in part by the National Science and Engineering Research Council of Canada under grant A3554.

REFERENCES

[1] Chanson, S. and Sinha, P.,"Adaptive load control in batch-interactive computer systems", Proc. Of 16th Computer Performance Evaluation Users Group, Oct. 1980, 207-213.

[2] Badel, M., Gelenbe, E., Leroudier, J., Potier, D.,"Adaptive optimization of a time-sharing system's performance", Proc. IEEE, Vol.63, 1975, 958-965.

[3] Badel, M., Leroudier, J., "Adaptive multiprogramming systems can exist", Performance of Computer Installations, D. Ferrari (ed.), North-Holland, 1978, 115-135.

[4] Denning, P., Kahn, K., Leroudier, J., Potier, D., Suri, R., "Optimal multiprogramming", Acta Informatica, Vol.7, No.2, 1976, 197-216.

[5] Landwehr, C.,"An endogenous priority model for load control in a combined batch-interactive computer system", Proc. Int'l Symp. On Comp. Perf. Modelling, Meas. and Eval., March 1976, 282-293.

[6] Leroudier, J. and Potier, D.,"Principles of optimality for multiprogramming", Proc., Int'l Symp. On Comp. Perf. Modelling, Meas. and Eval., March 1976, 211-218.

[7] Gelenbe, E., Kurinckx, A., Mitrani, I., "The rate control policy for virtual memory management", Operating Systems: Theory and Practice, D. Lanciaux (ed.), North-Holland, 1979, 247-264.

[8] Gelenbe, E. and Kurinckx, A., "Random injection control of multiprogramming in virtual memory", IEEE Trans. On Software Engineering, Vol.4, 1978,2-17.

[9] Graham, G. S., and Denning, P. J., "On the relative controllability of memory policies", Proc. Int'l. Symp. on Computer Performance Modeling, Measurement and Evaluation, Aug. 1977, 411-428.

[10] Geck, A., "Performance improvement by feedback control of the operating system", Proc. Of the 4th Int'l Symp. On Modelling and Perf. Eval. Of Computer Systems, Vienna, Feb. 1979, 459-471.

[11] Brandwajn, H., and Hernandez, J., "A study of a mechanism for controlling multiprogrammed memory in an interactive system", Perf. Of Computer Installations, D. Ferrari (ed.), North-Holland, 1978, 487-500.

[12] Kritzinger, P., Krzesinski, A., Teunissen, P.,"Design of a control system for a timesharing computer system", Perf. Of Computer Installations, D. Ferrari (ed.), North-Holland, 1978, 103-114.

[13] Denning, P. and Kahn, K., "An L=S criterion for optimal multiprogramming", Proc. Int'l Symp. On Computer Perf. Modelling, Meas. and Eval., March 1976, 219-229.

[14] Hine, J., Mitrani, I., Tsur, S., "The control of response times in multi-class systems by memory allocation", Comm. ACM, Vol.22, No.7, July 1979, 415-423.

[15] Chanson, S. T., "Saturation estimation in interactive computer systems", Dept. of Comp. Sci., Univ. Of British

Columbia, TR 79-7, 1979.

[16] Chanson, S. T. and Lo, R., "The application of optimal stochastic control theory in computer load regulation", Technical report 81-5, Dept. Of Computer Science, University of British Columbia, June 1981.

[17] Schonbach, A, "macro-Scheduler for high productivity", Ph.D. Thesis, Dept. Of Comp. Sci., Univ. of Toronto, 1980.

[18] Denning, P. J., and Buzen, J. P., "The operational analysis of queuing network models", Computing Surveys, Vol. 10, No. 3, Sep. 1978, 225-261.

[19] Buzen, J. P., "Computational algorithms for closed queuing networks with exponential servers", Comm. ACM, Vol. 16, No. 9, Sep. 1973, 527-531.

[20] Reiser, M., and Lavenberg, S. S., "Mean-Value analysis of closed multiclass queuing network", JACM, Vol. 27, No. 2, April 1980, 313-322.

[21] Kleinrock, L., "Certain analytic results for time-shared processor", Info. Processing, Proc. IFIP Congress 1968, 838-845.

[22] Belady, L. A., and Kuehner, C. J., "Dynamic space sharing in computer system", Comm. ACM, Vol. 12, No. 5, May 1969, 282-288.

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

[24] Sinha, P. S., "Optimization techniques in computer system design and load control", Ph.D. Thesis, dept. of Comp. Sci., Univ. of British Columbia, 1981.

[25] Chanson, S. T. and Sinha, P. S., "Optimization of memory hierarchies in multiprogrammed computer systems with fixed cost constraint", IEEE Trans. on Computers, Vol. C-29, No. 7, July 1980, 611-618.

Appendix A

Derivation of the saturation load for a single class model

Let $S_1, S_2, \ldots, S_M$ be the M service centres as shown in Figure 2. S_1 is the CPU, S_2 is the paging device and $S_3, \ldots, S_M$ 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 $S_i = \mu_i = X_i/B_i$

System throughput rate $T = (X_1 \cdot q_1) / T$

$$= (X_1/B_1)(B_1/T) \cdot q_1$$

$$= \mu_1 \rho_1 q_1 \qquad (A.1)$$

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:

$$\rho_i = \rho_1 \cdot (\mu_1/\mu_i) q_i; \quad i \neq 1 \qquad (A.2)$$

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

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:

$$\rho_1(1) = \left[\sum_{i=1}^{M} (\mu_1/\mu_i)\, q_i + 1 \right]^{-1} \qquad (A.3)$$

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

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

Using (A.1)

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

$$= N\, q_i/(\mu_1 \rho_i q_i) \quad i \neq 1 \qquad (A.4)$$

From (A.3)

$$R(1) = (1/\mu_1 q_1) \left[\sum_{i=1}^{M} (\mu_1/\mu_i)\, q_i + 1 \right] \qquad (A.5)$$

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, 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:

$$R(N) = N \,/\, (\mu_1 q_i) \qquad (A.6)$$

Otherwise, using equation (A.4) 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:

$$R(N) = N\, q_i^* \,/\, (\mu_i^* q_1) \;, \quad i^* \neq 1 \qquad (A.7)$$

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 (A.7) is still valid (see Figure A.1)

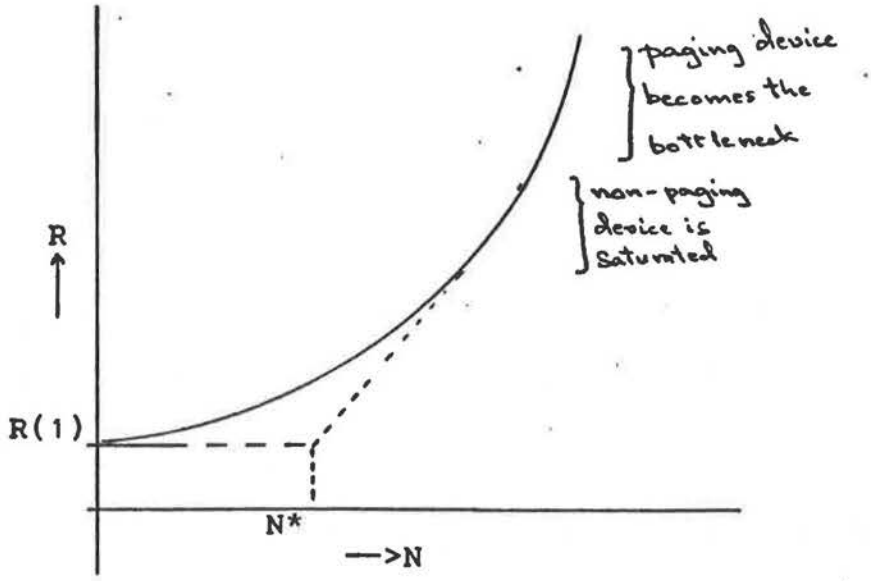

Figure A.1 System Bottleneck

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 (A.5) and (A.6) is given by :

$$N^* = 1 + \sum_{i=2}^{M} (\mu_1/\mu_i)\, q_i \qquad (A.8)$$

if the CPU is the bottleneck. Otherwise it is given by equation (A.9) as the point of intersection of (A.5) and (A.7):

$$N^* = (\mu_i^*/\mu_1 q_i^*) \left[\sum_{i=2}^{M} (\mu_1/\mu_i) q_i + 1 \right] \qquad (A.9)$$

Thus we have equations (A.8) and (A.9) as estimates of the system saturation point under the assumptions made earlier. Both equations (A.8) and (A.9) can also be derived using classical queueing theory with exponential distributions of service times.

Appendix B · Workload for the simulation runs

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 B.1 WORKLOAD FOR TABLE 5.1

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 CHARACTERISTIC PERIOD++ (SEC)	3	3

TABLE B.2 WORKLOAD FOR TABLE 5.2

* Variation in arrival rate.
** Variation in I/O request rate (in /sec).
*** Variation in exit probability.
\+ Perod of variation in arrival rate.
++ Period of variation in job characteristics.

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 B.3 WORKLOAD FOR TABLE 5.3

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 B.4 WORKLOAD FOR TABLE 5.4

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 B.4 WORKLOAD FOR TABLE 5.5 THROUGH TABLE 5.7