A Performance Evaluation of Several Priority Policies for Parallel Processing Systems

Randolph Nelson and Donald Towsley

COINS Technical Report 91-32
May 1991

A Performance Evaluation of Several Priority Policies for Parallel Processing Systems

Randolph Nelson
IBM Research Division
T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Donald Towsley *
Dept. of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

May 9, 1991

Abstract

In this paper we develop analytical models for a multiprocessor executing a stream consisting of K classes of fork-join jobs. Here a fork-join job consists of a random number of tasks that can be executed independently of each other. Several priority policies are analyzed: (a) a strict non-preemptive head of the line policy, (b) a preemptive policy that allows preemptions at the job level, (c) a preemptive policy that allows preemptions at the task level, and (d) a policy where the priority is a nondecreasing function of the number of tasks in the queue with preemptions at the job level. Using these models, we compare the mean job response time for the different classes under the different policies. We compare these policies to a system in which processors are partitioned so that classes are allocated only to certain processor groups. We show that, for the system considered, the task preemption policy has a uniformly better mean class response time and thus is preferable to a system with partitioned processors.

*The work of this author was partially supported by the Office of Naval Research under Contract N0014-87-K-0796 and the National Science Foundation under grant CCR-8500332.

1 Introduction

With the advent of shared memory multiprocessors[28] and programming languages that support parallel programming, (e.g., Concurrent Pascal [11], CSP[14], and Ada [29]) there is increasing interest in modeling the performance of parallel programs. In this paper, we evaluate the performance of a simple parallel program, a *fork-join* job, on a multiprocessor consisting of identical processors for a variety of priority scheduling policies. In our model, a fork-join job consists of a set of tasks which can be scheduled independently on any processor. All tasks in a given job arrive simultaneously to the system and the job is considered to be completed when its last task completes. We assume that tasks are coarse grained and thus have execution times that are large with respect to overheads associated with parallel execution. We assume that overheads of synchronization have been subsumed in task service requirements. These overheads include contention for shared resources (i.e., memory access or interconnection bandwidth) and costs associated with synchronizing the computation among the tasks [7, 10, 22]. Previous work of single class fork/join job models includes deterministic models that assume unbounded parallelism and include effects of the protocol used in synchronization [1, 7, 6, 9, 10] as well as models that concentrate on the effects that randomness in task execution time has in synchronization [2, 4, 3, 15, 18, 24, 23, 27, 32, 33].

The performance of parallel programs modeled as fork-join jobs is significantly affected by the choice of policy that is used to schedule tasks (see [20, 21, 30, 31, 35] for simulation studies). In this paper, we analyze the performance of three priority scheduling policies where jobs are given fixed priorities and a fourth policy where the priority of a job varies inversely with the number of its tasks remaining in the queue. The first three policies differ from each other according to what type of preemption (if any) is allowed. The primary contribution of the paper is the development of an analytic model for these scheduling policies. The analysis is carried out under the assumption of Poisson arrivals and task times that are independent and identically distributed exponential random variables (r.v.'s). The number of tasks in a fork-join job is a r.v. with a general distribution. Expressions for the mean fork-join job response times are obtained using the mean value analysis introduced by Cobham [5] for the M/G/1 priority queue.

Our assumption that tasks are independent and identically distributed are similar to assumptions made in [20, 21, 30, 35] and are motivated by the fact that most systems attempt to statistically partition work of a job evenly among its tasks. These studies also assume that a job consists of exactly one fork/join segment. Extensions to more complicated job structures can be made along the lines of [25], where a geometric sequence of fork/join segments is modeled or, in the case of a fixed number of fork/join segments, one could approximate response times by summing up the individual segment response times. Since we assume that tasks are coarse grained, there can be significant statistical variation in their execution times which has an effect on the response time of a job. This variation in task execution times arises from inherent randomness in task work requirements as well as from queueing delays that result when tasks compete for shared resources. In some applications, the variation of exponential service times might prove to be larger than that actually found. Our analysis for some of these cases corresponds to an upper bound on the expected performance. Since the analysis makes extensive use of the memoryless property of the exponential distribution, it cannot be *directly* applied to other service distributions. The analysis presented here, however, can be extended to task service times that have a phase distribution [26]. Such an extension would require augmenting the service time by a index indicating the stage of service achieved for each task. Using a phase distribution, one can match any finite number of moments of a given distribution while preserving the tractability of the memoryless property of exponentials. Our analysis can be thus viewed as a first step towards modeling systems with more general task service distributions. On the other hand, we should point out that extending our analysis so that tasks service requirements are class dependent, appears to be intractable. Currently we model such a case by adjusting the number of tasks in a job to match the total amount of work found in the job.

In our results, we compare the policies analyzed to that of the first come first serve (FCFS) policy [23]. We consider a system with 8 processors and workloads consisting of three job classes, each with its own priority. We compare the mean response time for each class under each of the priority policies and FCFS. We also consider a workload consisting of two job classes, edit jobs consisting of one task and batch jobs consisting of 16 tasks. For this workload, we compare the performance of each job class under the priority policies to that achieved in a system where processors are partitioned so that edit and batch jobs have their own sets of processors. Our results suggest that priority schemes have better performance characteristics with respect to the partitioned system.

The paper is organized as follows. A mathematical model of the system and descriptions of the policies that we study are found in Section 2. The analysis of the four scheduling policies is found in Section 3. Numerical results comparing the policies with each other and with first come first serve and processor sharing are found in section 4. Last, a summary of this work is found in section 5.

2 The Model

We consider a system, shown in Figure 1, consisting of c processors that serve K classes of *fork-join* jobs. A class k job consists of D_k tasks, $1 \leq k \leq K$ with probability distribution $\alpha_i^{(k)} = \Pr[D_k = i]$, $1 \leq i$ and mean $\overline{D}_k$. The service time of each task is an exponential r.v. with mean $1/\mu$. Jobs of class k arrive according to a Poisson arrival process with rate λ_k. We derive expressions for the mean response time, $\overline{T}_k$, of a class k job, $1 \leq k \leq K$ for a variety of scheduling policies.

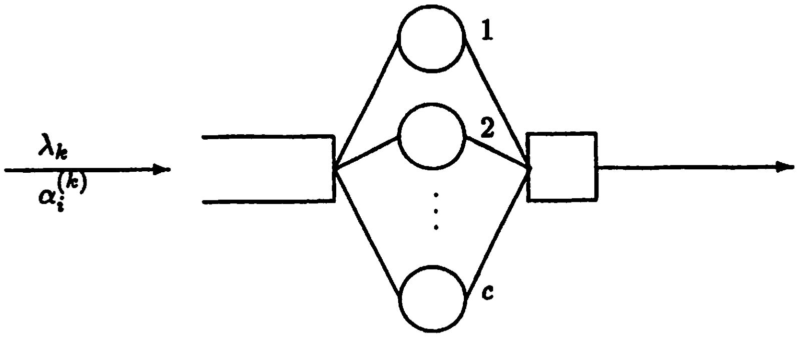

Figure 1. The Model

We are interested in scheduling policies that assign priorities to jobs. We use the convention that a job with priority i has higher priority than a job with priority $i+1$. We are interested in the following scheduling policies,

- *First Come First Serve (FCFS):* Tasks are scheduled in the order that they arrived. In the case of tasks from the same job, they are scheduled in random order.
- *Fixed Priority Policy with no Preemptions (FP):* Jobs from class k are assigned priority k. Jobs are served according to their priority. Jobs with the same priority are served in the order of their arrival. Once a job begins service, its tasks are scheduled in random order before those of any other job.
- *Fixed Priority Policy with Job Preemption (JP):* Jobs from class k are assigned priority k. Tasks are served according to the priority of the job to which they belong. Tasks with the same priority from different jobs are served in the order of their arrival. A task is not preempted once it begins service.
- *Fixed Priority Policy with Task Preemptions (TP):* Similar to the last policy except that tasks can be preempted by higher priority tasks.

- *Smallest Number of Tasks in Queue with Job Preemption (SQ):* Similar to the JP policy except that the priority is set to the number of tasks in the queue.

The behavior of FCFS is well understood and its analysis can be found in [23]. The remainder of the paper is devoted to obtaining the mean job response times under the other four policies.

3 Analysis

We begin this section with notation and definitions that will be common to the analyses of two or more of the scheduling policies. The remainder of the section is devoted to the analyses.

3.1 Preliminaries

Let N denote the total number of tasks of all priorities in the system at a random point in time. Because the service times of all tasks are independently and identically distributed exponential r.v.'s, the stationary distribution of N is the same under all of the policies. The statistics of N correspond to the statistics of the queue length of a $M^D/M/c$ queue where jobs arrive in batches at rate $\lambda = \sum_{k=1}^{K} \lambda_k$ and the batch size D has distribution $\alpha_i = \Pr[D = i] = \lambda^{-1} \sum_{k=1}^{K} \lambda_k \alpha_i^{(k)}$, $1 \le i$. We will describe shortly how these probabilities may be computed.

The time that a job spends in the system can be divided into two phases, a *wait phase* during which the job waits to be assigned its first processor, and a *service phase*, the time between the first assignment of a task from that job and the completion of the last task of that job. We denote these quantities as the r.v.'s W_k and X_k for class k jobs. Hence $\overline{T}_k$ can be written as

$$\overline{T}_k = \overline{W}_k + \overline{X}_k \tag{1}$$

where $\overline{W}_k$ and $\overline{X}_k$ depend on the scheduling policy.

We introduce the following random variables.

- Q_k - the number of class k jobs in the queue that have not received any service. According to Little's law, $\overline{Q}_k = \lambda_k \overline{W}_k$.
- R'_k - the number of tasks remaining in the queue at a random point in time for a class k job that is in its service phase.

In some cases, we are interested in the behavior of the system when we coalesce the first j classes of jobs. We define

- $\Lambda_j = \sum_{k=1}^{j} \lambda_k$ with $\Lambda_0 = 0$,
- $\sigma_j = \sum_{k=1}^{j} \lambda_k \overline{D}_k/(\mu c)$, the utilization of each processor due to jobs of class k, $1 \le k \le j$. For convenience, $\sigma_0 = 0$,
- D'_j - a r.v. denoting the number of tasks in a job randomly chosen from priorities $1, \cdots, j$ having distribution $\beta_i^{(j)} = \sum_{k=1}^{j} \lambda_k \alpha_i^{(k)}/\Lambda_j, i = 1, 2, \ldots$,
- B_j - a r.v. denoting the length of a busy period started by a single customer for a bulk arrival $M/M/1$ queue with arrival rate Λ_j, bulk size distribution $\beta_i^{(j)}$, and service rate μc, $B_0 = 0$,
- N_j - a r.v. denoting the number of tasks in a bulk arrival $M/M/c$ queue with arrival rate Λ_j, bulk size distribution $\beta_i^{(j)}$, and service rate μ having distribution $\pi_n^{(j)} = \Pr[N_j = n]$, $n = 0, 1, \cdots$.

The stationary probability distribution for N_j is given by (see [34]):

$$\pi_k^{(j)} = \frac{c\sigma_j}{\min(c,k)} \sum_{l=0}^{k-1} \pi_l^{(j)} \nu_{k-l}^{(j)}, \qquad k \ge 1, \tag{2}$$

where $\nu_k^{(j)} = \sum_{i=k}^{\infty} \beta_i^{(j)}/E[D'_j]$. The probabilities $\pi_i^{(j)}$, $i = 0, 1, \cdots, c-1$, are obtained from the first $c - 1$ equations of (2) along with the normalizing equation

$$(1 - \sigma_j) = \sum_{k=0}^{c-1} (1 - k/c)\pi_k^{(j)}.$$

The remaining probabilities can be computed directly from (2) in a recursive manner.

According to the definition of π_n given earlier,$\pi_n = \pi_n^{(K)}$, $n = 0, 1, \cdots$.

Our interest in B_j, a busy period corresponding to a bulk arrival $M/M/1$ queue can be explained as follows. Jobs incur queueing delays only when all c processors are busy. In this situation, tasks waiting in the queue are removed each time that one of the c processors completes. Thus, the departure of jobs from the queue corresponds to that of a single processor system where the service rate is μc. We will use the term *customer* when we discuss this bulk arrival $M/M/1$ queue. Here a customer is equivalent to a task in the real system. Last, in order to simplify the discussion, we will refer to a bulk arrival $M/M/1$ queue handling the bulks of size D'_k with arrival rate Λ_k and service rate μc as a *type-k $M/M/1$ queue*.

3.2 Fixed Priorities with no Preemptions

We are interested in obtaining $\overline{W}_k$ and $\overline{X}_k$. We focus on $\overline{W}_k$, the mean wait time for a randomly chosen class k job, J.

An argument similar to the one found in [16, pp. 119-121] for the $M/G/1$ priority queue yields the following set of equations for $\overline{W}_k$, $1 \le k \le K$,

$$\overline{W}_k = \frac{\Pr[N \ge c]}{\mu c} + \sum_{j=1}^{K} \frac{\overline{R'}_j}{\mu c} + \sum_{j=1}^{k} \frac{\overline{Q}_j \overline{D}_j}{\mu c} + \overline{W}_k \sum_{j=1}^{k-1} \frac{\lambda_j \overline{D}_j}{\mu c}, \quad 1 \le k \le K. \tag{3}$$

The first term is the mean delay incurred by J while waiting for the first processor to become available for any task. The second term is the mean delay due to the customer currently in service that still has tasks in the queue remaining to be served. The third term includes the delay due to all jobs of priority equal to or higher than the priority of J that are in the system but have not yet been scheduled. Last, the fourth term is the delay due to jobs of priority higher than J that arrive while J waits in the queue. Equation (3) can be solved to yield

$$\overline{W}_k = \frac{\Pr[N \ge c]/(\mu c) + \sum_{j=1}^{K} \overline{R'}_j/(\mu c)}{(1-\sigma_k)(1-\sigma_{k-1})}, \quad 1 \le k \le K. \tag{4}$$

It remains for us to calculate $\overline{R'}_k$ and $\overline{X}_k$.

Calculation of $\overline{R'}_k$: First we introduce the notion of a *partially scheduled job.*

Definition 1 *A job is a partially scheduled job if at the start of its service phase some of its tasks remain in the queue.*

Not all jobs are partially scheduled jobs, i.e., a job that obtains all of the processors that it requires at the beginning of its service phase is not a partially scheduled job. If we let γ_k denote the arrival rate of partially scheduled jobs, then according to the definition of such jobs, it can be expressed as

$$\gamma_k = \lambda_k \left[1 - \sum_{j=0}^{c-1} \pi_j \Pr[D_k \le c - j] - (1 - \sum_{j=0}^{c-1} \pi_j)\alpha_1^{(k)} \right]. \tag{5}$$

We define a binary valued r.v., I_k, to take on value one if at a random point in time there exists a partially scheduled class k job in the system that has at least one task remaining in the queue at that time. We note that $\overline{R'}_k$ takes value zero whenever no such class k job exists in the system at a random point in time. Here $\overline{R'}_k$ can be expressed as

$$\overline{R'}_k = \Pr[I_k = 1]E[R'_k | I_k = 1]. \qquad (6)$$

We define the r.v. R_k to denote the number of tasks that a randomly chosen partially scheduled class k job has in the queue immediately after it begins its service phase. The distribution of this r.v. is

$$\Pr[R_k = i] = \frac{\sum_{j=0}^{c-1} \pi_j \alpha_{i+c-j}^{(k)} + \Pr[N \geq c]\alpha_{i+1}^{(k)}}{1 - \sum_{j=0}^{c-1} \pi_j \Pr[D_k \leq c - j] - (1 - \sum_{j=0}^{c-1} \pi_j)\alpha_1^{(k)}}, \quad 1 \leq i; \ 1 \leq k \leq K.$$

Consider a partially scheduled class k job. The r.v. I_k takes value one so long as that job has one or more tasks remaining in the queue. The number of such tasks is described by a counting process where the times correspond to the time between successive schedulings of tasks belonging to this job. Hence these times are i.i.d. exponential r.v.'s with parameter μc. Hence the length of time that such a job has at least one task in the queue is the sum of R_k such exponential r.v.'s and has mean $\overline{R}_k/(\mu c)$. We can write the following expression for $\Pr[I_k = 1]$,

$$\Pr[I_k = 1] = \gamma_k \overline{R}_k/(\mu c).$$

When $I_k = 1$, the r.v. R'_k corresponds to the remaining number of tasks from a partially scheduled job that an arriving job finds in the queue. Application of the PASTA property [13] implies that this corresponds to the residual number of tasks for a partially scheduled job observed at a random point in time, given that one was present. From renewal theory,

$$\Pr[R'_k = n | I_k = 1] = \frac{1 - \Pr[R_k > n]}{\overline{R}_k}, \quad n = 1, 2, \cdots$$

which has mean

$$E[R'_k | I_k = 1] = \frac{\overline{R^2}_k}{2\overline{R}_k} + \frac{1}{2}.$$

Calculation of $\overline{X}_k$: The mean service time $\overline{X}_k$ is obtained by conditioning the mean service time on the number of tasks in the job and on the number of processors available to the job when it first begins execution. If we let $x_{i,n} = E[X_k | D_k = i, C = n]$ where C denotes the number of processors that a job acquires when it first starts service, then we have shown in [23] that these conditional expectations satisfy the following equations,

$$x_{i,n} = \begin{cases} H_n/\mu, & i = n; 1 \leq n \leq c, \\ H_c/\mu + (i - c)/(\mu c), & n = c; \ i > c, \\ 1/(\mu c) + n x_{i-1,n}/c + (c - n) x_{i,n+1}/c, & 1 \leq n < c; \ n < i, \\ 0, & \text{elsewhere} \end{cases}$$

where H_n denotes the harmonic series, $H_n \equiv \sum_{i=1}^{n} 1/i$.

We can obtain a closed form expression for $x_{i,n}$ by conditioning on the number of processors that are held by the job at the time that the last of its tasks are removed from the queue. Let $x_{i,n,k}$ denote the mean service time for a job consisting of i tasks given that it begins with n processors and, when the last task is scheduled to service, it holds $k \geq n$ processors. Then $x_{i,n,k}$ is expressed as

$$x_{i,n,k} = \frac{i-n}{c\mu} + \frac{H_k}{\mu},$$

and $x_{i,n}$ is

$$x_{i,n} = \sum_{k=n}^{\min(i,c)} q_{k,i,n} x_{i,n,k}$$

where $q_{k,i,n}$ is the probability that the job holds k processors immediately after the last task is scheduled given that it consisted initially of i tasks, of which n were initially scheduled to processors. These probabilities satisfy the following recurrence relation

$$q_{k,i,n} = \begin{cases} 1, & k = i = n, \ n = 1, 2, \cdots, \\ \frac{n}{c} q_{k,i-1,n} + \frac{c-n}{c} q_{k,i,n+1}, & i > n; \ k \geq n > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

We observe that the above recurrence relation is similar to that which describes the behavior of the normalization constant for a single chain closed product form queueing network with queue length independent service rates [19, section 3.5.1]. This suggests the following solution which satisfies the above relation

$$q_{k,i,n} = \prod_{r=0}^{k-n-1} \left(\frac{c-n-r}{c} \right) \sum_{s \in \mathcal{S}(k,i,n)} \prod_{r=0}^{k-n} \left[\frac{n+r}{c} \right]^{s_r},$$

where $s = (s_0, s_1, \cdots, s_{k-n})$ and $\mathcal{S}(k,i,n) = \{ s | \sum_{r=0}^{k-n} s_r = i - k + n \}$. This can be simplified via an application of a result by Harrison, [12, Theorem 1],

$$\begin{aligned} q_{k,i,n} &= \prod_{r=0}^{k-n-1} \left(\frac{c-n-r}{c} \right) \sum_{r=0}^{k-n} \left(\frac{n+r}{c} \right)^i \Big/ \prod_{r' \neq r} \left[\frac{n+r}{c} - \frac{n+r'}{c} \right], \\ &= \binom{c-n}{k-n} \sum_{r=0}^{k-n} (-1)^{k-n-r} \binom{k-n}{r} \left(\frac{n+r}{c} \right)^i. \end{aligned}$$

Removal of the conditioning on the job size and the number of processors initially available yields,

$$\overline{X}_k = \sum_{n=0}^{c-1} \pi_n \left(\sum_{i=1}^{c-n} \alpha_i^{(k)} x_{i,i} + \sum_{i=c-n+1}^{\infty} \alpha_i^{(k)} x_{i,c-n} \right) + (1 - \sum_{n=0}^{c-1} \pi_n) \sum_{i=1}^{\infty} \alpha_i^{(k)} x_{i,1}. \tag{8}$$

Substitution of (4) and (8) into (1) yields the mean job response time of a class k job, $1 \le k \le K$, under the nonpreemptive policy.

3.3 Fixed Priorities with Job Preemption

Again, we are interested in the mean response time of a randomly chosen class k jobs ($1 \le k \le K$), J, under a fixed priority scheduling policy where a job may be preempted by a higher priority job. Tasks, however, cannot be preempted. The analysis of this system differs from that of the system with no preemptions in several ways. First, because jobs can be preempted, a class k job need not wait for lower priority tasks in the queue. This results in the following set of equations,

$$\overline{W}_k = \frac{\Pr[N \ge c]}{\mu c} + \sum_{j=1}^{k} \frac{\overline{R'}_j}{\mu c} + \sum_{j=1}^{k} \frac{\overline{Q}_j \overline{D}_j}{\mu c} + \overline{W}_k \sum_{j=1}^{k-1} \frac{\lambda_j \overline{D}_j}{\mu c}, \quad 1 \le k \le K.$$

Because the second term depends on k (unlike equation (3)), we are unable to obtain a closed form for $\overline{W}_k$. Instead, we can transform the above equations into the following recurrence relations which can be solved numerically,

$$\overline{W}_1 = \frac{\Pr[N \ge c] + \overline{R'}_1}{\mu c(1 - \sigma_1)}, \tag{9}$$

$$\overline{W}_k = \frac{\Pr[N \ge c] + \sum_{j=1}^{k} \overline{R'}_j + \sum_{j=1}^{k-1} \lambda_j \overline{W}_j \overline{D}_j}{\mu c(1 - \sigma_k)}, \quad 1 < k \le K. \tag{10}$$

Second, the quantities $\overline{R'}_k$ and $\overline{X}_k$ are calculated differently as described below.

Calculation of $\overline{R'}_k$: The expression for $\overline{R'}_k$ in equation (6) remains valid for this scheduling policy. However, the probability of finding a partially scheduled class k job in the system that still has tasks in the queue differs from the system without job preemptions. This results from the fact that partially scheduled jobs can be preempted before all of their tasks are assigned to processors. The number of tasks that a partially scheduled job has in the queue remains a counting process that decreases by one. As for the FP policy, the time between changes in the number of tasks corresponds to the time between the scheduling of successive tasks. However, this time is no longer exponentially distributed since it includes the time required to schedule all higher priority tasks that may arrive during two successive task schedulings of the partially scheduled job. The time between successive schedulings is distributed as a *busy period initiated by a single customer* for a type-$(k-1)$ $M/M/1$ queue. Arrivals to this queue correspond to arrivals of tasks whose priorities are higher than k to the real system. Consequently, the probability $\Pr[I_k = 1]$ of finding a partially scheduled job with tasks still in the queue is

$$\Pr[I_k = 1] = \begin{cases} \gamma_1 \overline{R}_1/(\mu c), & k = 1, \\ \gamma_k \overline{R}_k \overline{B}_{k-1}, & k = 2, \cdots, K. \end{cases}$$

Here the quantity γ_k is given by equation (5). The mean busy period length, $\overline{B}_j$ of a type-j M/M/1 queue is

$$\overline{B}_j = \frac{1}{\mu c - \Lambda_j \overline{D'}_j}, \quad j = 1, 2, \cdots, K.$$

Calculation of $\overline{X}_k$: The mean service time $\overline{X}_k$ is obtained by conditioning the service time on the number of tasks in the job and the number of processors that it holds when it is first scheduled into service. Let $x_{i,n}^{(k)} = E[X_k | D_k = i, C_k = n]$. Consider the situation where a class k job holds n processors. One of the following three events will eventually occur,

1. The arrival of a higher priority job.
2. Completion of one of the n tasks that the job has in service.
3. Completion of some other task.

We note that if a higher priority job arrives, then no task from J will be scheduled until the end of the busy period initiated by that high priority job. The busy period is that of a type-$(k-1)$ $M/M/1$ queue.

One other definition is required before we present the equations for $x_{i,n}^{(k)}$. Consider the situation where J holds n processors and a job with higher priority arrives. This job will create a busy period consisting of it and other higher priority tasks that must terminate before the next task belonging to J can be scheduled. Let $L_{k-1}(n)$ be a r.v. denoting the number of processors that are relinquished by J during this busy period made up of tasks of priorities $1, \cdots, k-1$ given that J held n processors at the beginning of the busy period.

We now write the following equations that describe the behavior of $x_{i,n}^{(k)}$.

$$x_{n,n}^{(k)} = H_n/\mu, \quad 1 \le n;\ 1 \le k \le K, \tag{11}$$

$$
\begin{aligned}
x_{i,n}^{(k)} &= \frac{1}{\Lambda_{k-1}+\mu c} + \frac{n\mu x_{i-1,n}^{(k)}}{\Lambda_{k-1}+\mu c} + \frac{(c-n)\mu x_{i,n+1}^{(k)}}{\Lambda_{k-1}+\mu c} \\
&+ \frac{\Lambda_{k-1}}{\Lambda_{k-1}+\mu c}\left[\overline{D}'_{k-1}\overline{B}_{k-1} + \sum_{j=0}^{n-1}\Pr[L_k(n)=j]x_{i-j,n-j}^{(k)} + \Pr[L_{k-1}(n)=n](\overline{B}_{k-1}+x_{i,1}^{(k)})\right], \\
&\qquad 1 \le n \le c;\ n < i;\ 1 \le k \le K. \qquad (12)
\end{aligned}
$$

The first term of the r.h.s. in equation (12) is the mean time until the first event (departure, or arrival of a higher priority job) occurs. The second and third terms correspond to the events that a task belonging to J departs and a task belonging to some other job departs, respectively. The last term corresponds to an arrival of a higher priority job. In this case, there is a delay due to the busy period initiated by this high priority job followed by the time required to serve whatever tasks belonging to J that still remain in the system (some may have completed during the ensuing high priority busy period). If all tasks belonging to J that are in service at the start of the high priority busy period complete by its end, then J will incur an additional delay until the first of its remaining tasks is placed in service. This delay corresponds to a busy period in a type-$(k-1)$ $M/M/1$ queue that is initiated by a single task.

We now focus on the distribution of $L_k(n)$. Let U_k denote the number of jobs served in a busy period made up of tasks with priorities $1,\cdots,k$. Define $a_{i,j,n} = \Pr[L_k(n) = i | U_k = j]$. These conditional probabilities satisfy the following recurrence relation

$$
a_{i,j,n} = \begin{cases} \frac{c-i+n}{c}a_{i,j-1,n} + \frac{n-i+1}{c}a_{i-1,j-1,n}, & 0 \le i \le \min(j,n);\ 1 \le n \le c;\ 1 \le j, \\ 1, & i = j = 0,\ 1 \le n \le c, \\ 0, & \text{otherwise.} \end{cases}
$$

By noting the similarity between this recurrence relation and the one contained in equation (7), we can write the solution as,

$$
a_{i,j,n} = \binom{n}{i}\sum_{r=0}^{i}(-1)^{i-r}\binom{i}{r}\left(\frac{c-n+r}{c}\right)^j.
$$

Observe that these probabilities are independent of k. Removal of the conditioning yields

$$
\Pr[L_k(n) = i] = \sum_{j=\max(1,i)}^{\infty} \Pr[U_k = j]a_{i,j,n}, \quad k = 1,\cdots,K;\ i = 0,1,\cdots. \qquad (13)
$$

We now describe a procedure for numerically computing the distribution of U_k. Observe first that U_k corresponds to the number of customers served in a busy period for a type-k $M/M/1$ queue. In the following discussion $X^{(n)}$ denotes the n-th fold convolution of the r.v. X. We introduce three auxiliary r.v.'s

- A_k - the number of bulks that arrive in a type-k $M/M/1$ queue during a service time. This has the distribution $\Pr[A_k = n] = \frac{\mu c}{\Lambda_k+\mu}(\frac{\Lambda_k}{\Lambda_k+\mu})^n$, $n = 0, 1, \cdots$.
- V_k - the number of customers that arrive to a type-k $M/M/1$ queue during a single service time, $V_k = D'^{(A_k)}_k$.
- Z_k - the number of customers served during a busy period for a type-k $M/M/1$ queue initiated by the arrival of a single customer.

Now, U_k can be expressed as $U_k = 1 + Z_k^{(V_k)}$. The probability distributions for U_k, V_k, and Z_k are given by,

$$\Pr[V_k = 0] = \Pr[A_k = 0], \tag{14}$$

$$\Pr[V_k = v] = \sum_{n=1}^{v} \Pr[A_k = n] \Pr[D_k'^{(n)} = v], \; 1 \le v, \tag{15}$$

$$\Pr[Z_k = z] = \begin{cases} \Pr[V_k = 0], & z = 1, \\ \sum_{i=1}^{z-1} \Pr[V_k = i] \Pr[Z_k^{(i)} = z-1], & z = 2, 3, \cdots, \end{cases} \tag{16}$$

$$\Pr[U_k = u] = \sum_{i=1}^{u} \Pr[D_k' = i] \Pr[Z_k^{(i)} = u], \; u = 1, 2, \cdots. \tag{17}$$

3.4 Fixed Priorities with Task Preemption

Again, we are interested in the mean response time of J, a class k job ($1 \le k \le K$), under a fixed priority scheduling policy where a task may be preempted by a higher priority task. For this system we can write down the following set of equations for $\overline{W}_k$,

$$\begin{aligned} \overline{W}_k &= (E[N_k|N_k \ge c] - c + 1) \Pr[N_k \ge c]/(\mu c) \\ &\quad + \overline{W}_k \sum_{j=1}^{k-1} \lambda_j \overline{D}_j/(\mu c), \; 1 \le k \le K. \end{aligned}$$

The first term corresponds to the unfinished work in the queue at the time of the arrival of J, and the second term corresponds to the additional delay incurred by the arrival of higher priority jobs prior to the initiation of service of J. These equations have the solutions

$$\overline{W}_k = \frac{(E[N_k|N_k \ge c] - c + 1) \Pr[N_k \ge c]/(\mu c)}{(1 - \sigma_{k-1})}, \; 1 \le k \le K.$$

The mean response time, $\overline{T}_k$ is given as

$$\overline{T}_k = \overline{W}_k + \overline{X}_k, \quad 1 \le k \le K.$$

Note that X_k includes delays incurred due to the preemption of J by higher priority jobs after it begins service. The determination of $\overline{X}_k$ is accomplished by again conditioning X_k on appropriate events. In this case, we condition on the number of tasks in the marked job and the number of tasks of priority equal to or higher than k ahead of it in the system. Let this second r.v. be M_k. We define $x_{i,m}^{(k)} = E[X_k | D_k = i, M_k = m]$. These conditional expectations satisfy the following equations,

$$\begin{aligned} x_{i,m}^{(k)} = {} & \frac{1}{\Lambda_{k-1} + \min(m+i,c)\mu} + \frac{m\mu x^{(k)}{}_{i,m-1}}{\Lambda_{k-1} + \min(m+i,c)\mu} + \frac{\min(i, c-m)\mu x_{i-1,m}^{(k)}}{\Lambda_{k-1} + \min(m+i,c)\mu} \\ & + \frac{\Lambda_{k-1}}{\Lambda_{k-1} + \min(m+i,c)\mu} \left[\sum_{l=1}^{c-m-1} \beta_l^{(k-1)} x_{i,m+l}^{(k)} + \overline{B}_{k-1} \sum_{l=c-m}^{\infty} \beta_l^{(k-1)} (l+1-c+m) \right. \\ & \left. + x_{i,c-1}^{(k)} \left(1 - \sum_{l=1}^{c-m-1} \beta_l^{(k-1)}\right) \right] \end{aligned}$$

$$i \ge 1; \ 0 \le m < c, \tag{18}$$

and

$$x_{0,m}^{(k)} = x_{i,-1}^{(k)} = 0.$$

Consider the r.h.s. of equation (18). The first term is the mean delay until the first event (departure of a task of class $1, \cdots, k$ or arrival of a higher priority job). The second and third terms correspond, respectively, to the departure of a task belonging to J and to a job of higher priority or the same priority but older. Finally, the last term accounts for the arrival of a higher priority job. The first term in the numerator corresponds to the event where not all of the tasks belonging to J are preempted. The next two terms account for the case where all of the tasks belonging to J are removed from service. The second term corresponds to the delay until J has a task placed back into service, and the third term corresponds to the remaining service time from that point on.

These equations can be solved in a recursive fashion over the index i. For each i a set of c linear equations must be solved in order to obtain $x_{i,m}^{(k)}$. Once these conditional expectations have been obtained, the conditioning can be removed to yield

$$\overline{X}_k = \sum_{m=0}^{c-1} \pi_m^{(k)} \sum_{i=1}^{\infty} \alpha_i^{(k)} x_{i,m}^{(k)} + \sum_{m=c}^{\infty} \pi_m^{(k)} \sum_{i=1}^{\infty} \alpha_i^{(k)} x_{i,c-1}^{(k)}.$$

3.5 Smallest Number of Tasks in Queue with Job Preemption

Here, the job with the smallest number of tasks in the queue is scheduled for service. Preemptions are allowed at the job level, but not at the task level. As jobs from different classes receive the same service, provided they contain the same number of tasks, we need not distinguish between the job classes. Instead, we distinguish between jobs according to their size. Thus, we assign a job initially consisting of k tasks a priority of k. According to our model, such jobs arrive at rate $\lambda\alpha_k$. If we let W_k denote the time that a job consisting of k tasks must wait for service, then the mean wait time $\overline{W}_k$, $1 \leq k \leq K$, is the solution of equations (9) and (10) where $\lambda\alpha_l$ replaces λ_l and $\sigma_k = \sum_{j=1}^{k} j\lambda\alpha_j/(\mu c)$, $1 \leq k \leq K$. In addition, the definition of the r.v. R'_j differs slightly in this context. It is

- R'_j - the number of tasks in the queue at a random point in time belonging to a partially scheduled job *whose priority is j* at that time.

Here R'_j takes either value j or 0. The mean, $\overline{R'}_j$ is

$$\overline{R'}_j = \begin{cases} \gamma_1/(\mu c), & j = 1, \\ \gamma_j \overline{B}_{j-1}, & j = 2, \cdots, K \end{cases}$$

where γ_j denotes the arrival rate of partially scheduled jobs that will be assigned priority j at some point during its service phase. As before, B_j is the busy period of a type-j M/M/1 queue whose jobs are the same as those in the original system having j or fewer tasks. The quantity γ_j is expressed as

$$\gamma_j = \lambda \left[1 - \sum_{l=0}^{c-1} \pi_l \Pr[D \leq c - l + j] - (1 - \sum_{l=0}^{c} \pi_l) \Pr[D \leq j] \right], \quad 1 \leq j.$$

The mean response time, $\overline{T}_k$, is

$$\overline{T}_k = \overline{W}_k + \overline{X}_k$$

where $\overline{X}_k$ denotes the expected time required for a job consisting of k tasks to complete their service. This time is measured from the time the first task is scheduled onto a processor until the last task completes service.

The mean service time is obtained by further conditioning on the number of tasks that are left in the queue after the first task(s) is (are) allocated to processors and on the number of processors that it is given at that time. Let M and C be random variables that denote these quantities. We define $x_{i,n} = E[X_{i+n}|M = i, C = n]$. Using the same arguments that yielded equations (11) - (12), we obtain

$$\begin{aligned}
x_{0,n} &= H_n/\mu, \quad n = 1, \cdots, c, \\
x_{i,n} &= \frac{1}{\Lambda_{i-1} + \mu c} + \frac{n\mu x_{i-1,n}}{\Lambda_{i-1} + \mu c} + \frac{(c-n)\mu x_{i-1,n+1}}{\Lambda_{i-1} + \mu c} \\
&+ \frac{\Lambda_{i-1}\left[\overline{D}'_{i-1}\overline{B}_{i-1} + \sum_{l=0}^{n-1} \Pr[L_{i-1}(n) = l]x_{i,n-l} + \Pr[L_{i-1}(n) = n](\overline{B}_{i-1} + x_{i-1,1})\right]}{\Lambda_{i-1} + \mu c}, \\
& \qquad i = 1, 2, \cdots; \; n = 1, \cdots, c.
\end{aligned}$$

As the last equation contains $x_{i,n}$ in the r.h.s., it can be moved over to the l.h.s. to yield,

$$\begin{aligned}
x_{i,n} &= \frac{1 + n\mu x_{i-1,n} + (c-n)\mu x_{i-1,n+1}}{\Lambda_{i-1}(1 - \Pr[L_{i-1}(n) = 0]) + \mu c} \\
&+ \frac{\Lambda_{i-1}\left[\overline{D}'_{i-1}\overline{B}_{i-1} + \sum_{l=1}^{n-1} \Pr[L_{i-1}(n) = l]x_{i,n-l} + \Pr[L_{i-1}(n) = n](\overline{B}_{i-1} + x_{i-1,1})\right]}{\Lambda_{i-1}(1 - \Pr[L_{i-1}(n) = 0]) + \mu c}, \\
& \qquad i = 1, 2, \cdots; \; n = 1, \cdots, c. \qquad (19)
\end{aligned}$$

Here $\Lambda_i = \sum_{k=1}^{i} \lambda\alpha_k$ and $\beta_j^{(i)} \equiv \Pr[D'_i = j] = \alpha_j / \Pr[D \leq i]$. The r.v. $L_k(n)$ has the same definition as in Section 3.3 and its distribution can be calculated in the same way.

Last, the conditional mean service time, $\overline{X}_k$ is

$$\overline{X}_k = \begin{cases} x_{0,k} \sum_{l=0}^{c-k} \pi_l + \sum_{l=c-k+1}^{c-2} \pi_l x_{k+l-c,c-l} + \sum_{l=c-1}^{\infty} \pi_l x_{k-1,1}, & 1 \leq k \leq c, \\ \sum_{l=0}^{c-2} \pi_l x_{k+l-c,c-l} + \sum_{l=c-1}^{\infty} \pi_l x_{k-1,1}, & k > c. \end{cases}$$

Finally, if we are interested in the mean response time, $\overline{T}$, of a job, we have

$$\overline{T} = \sum_{k=1}^{\infty} \alpha_k \overline{T}_k.$$

4 Numerical Results

In this section we compare the performance of the four priority schemes analyzed in Section 3. For each of the four scheduling policies, we wrote a set of APL routines that solved the equations. For the Fixed Priority (FP) and Task Preemption (TP), our implementation yields an exact result. For the Job Preemption (JP) and Smallest number in Queue (SQ) schemes, our implementation yields approximate, but arbitrarily accurate, values. The approximate nature of our computations arises from the fact that the calculation of the values of $\Pr[L_k(n) = j]$ found in the service time expressions for these policies (equations (12) and (19)) depends upon the solution of a set of infinite recurrences (equations (14 - 17)). Our implementation sets an upper limit to the value of u in equation (17) and thus results in truncating the summation of equation (13). In order to achieve a required accuracy in calculating the mean service time for these two policies, we base the truncation on the relative difference between lower and upper bounds for the the service time. For a given u, the upper bound is obtained by replacing equation (13) by

$$\Pr[L_k(n) = i] = \begin{cases} 1 - \sum_{i=1}^{n} \sum_{j=1}^{u} \Pr[U_k = j] a_{i,j,n}, & k = 1, \cdots, K;\ i = 0, \\ \sum_{j=1}^{u} \Pr[U_k = j] a_{i,j,n}, & k = 1, \cdots, K;\ i = 1, 2, \cdots, n, \end{cases}$$

and the lower bound by

$$\Pr[L_k(n) = i] = \begin{cases} \sum_{j=\max(1,i)}^{u} \Pr[U_k = j] a_{i,j,n}, & k = 1, \cdots, K;\ i = 0, 1, \cdots, n-1, \\ 1 - \sum_{i=0}^{n-1} \sum_{j=\max(1,i)}^{u} \Pr[U_k = j] a_{i,j,n}, & k = 1, \cdots, K;\ i = n. \end{cases}$$

In our implementation, we choose the value of u so that the relative difference in service times between the upper and lower bounds was less than 2 percent, and used the value for the service time that is equidistant between the lower and upper bounds. This implies that our greatest error is on the order of 1 percent. Higher accuracy can be obtained at the cost of more computation time.

We ran many experiments to determine the performance of the priority schemes and have selected a set of results that has a practical motivation and offers representative examples of the type of behavior encountered. In the next subsection we graphically show how mean class response times vary with the priority policy. Following that we demonstrate the advantages of priority scheduling over processor partitioning.

4.1 Class Response Time Characteristics

In this subsection we present results for an 8-processor system having three priority classes. Class 1 models edit jobs and consists of 1 task. Classes 2 and 3 model medium and heavy batch jobs and consist of 8 and 16 tasks, respectively. For any given utilization, the relative class arrival rates are .7, .2 and .1, respectively. In Figures 2-4 we plot the mean class response times for FCFS, FP, JP, TP and SQ policies. The equations used for evaluating the FCFS system are found in [23]. These graphs are representative of the type of behavior we found for all other systems studied. We note that the definition of class for the SQ policy is different than that for other policies, since a job consisting of k tasks, will pass through classes $k-1, k-2, \ldots, 1$ before being completely executed. In general, letting T_k^P denote the r.v. for class k response time for priority policy P, the following relationships hold between class response times for the first and last class

$$T_1^{TP} \leq_{st} T_1^{JP} \leq_{st} T_1^{FP} \leq_{st} T_1^{FCFS}$$

and

$$T_K^{TP} \geq_{st} T_K^{JP} \geq_{st} T_K^{FP} \geq_{st} T_K^{FCFS}.$$

These relationships follow from simple path coupling arguments. To explain the first relationship, consider the tasks that can potentially delay a random class 1 arrival, J. Under TP, J is delayed only by other class 1 tasks that arrived before it. Under JP, J is delayed by these tasks as well as by tasks of all other job classes that are currently in service. Under FP, J is delayed by the same tasks as in JP to which are added further delays incurred when all the tasks of partially completed jobs in the queue are serviced. Finally, under FCFS, J is delayed by the same tasks as under FP but must also wait for all jobs that arrived earlier than it. A similar argument explains the second relationship.

In Figure 2, there is a inflection point in the mean response time curves of the FP and JP priority schemes. This occurs close to the arrival rate at which the queue becomes saturated. Such an inflection point also occurs in the M/G/1 HOL (Head of the Line) priority systems [5], and, in our case, is explained by the fact that the $Pr[N \geq c]$ term in equations (4) and (9-10) increases rapidly with the arrival rate until queue saturation, at which point it can no longer increase. Note that Figures 2 and 5 show that the SQ policy has a similar expected response time as the JP policy. This follows from the fact that class 1 jobs consist only of one task. For class 3 jobs, SQ has the lowest expected response time since the priority of a class 3 job increases as its tasks get executed.

In Figures 5-7 we plot the ratio of mean class response times for each priority class to that of FCFS. It is typical in real systems to attempt to provide more processing power to interactive jobs, at the cost of possibly delaying batch jobs. These graphs demonstrate the benefits of using priority schemes to do this. It is interesting to note that, for the workload we considered, the TP policy is not much better than the JP policy. The difference between these policies is dependent on the workload. For example, if a large fraction of the load comes from a low priority class consisting of a large number of tasks, then one would expect the relative difference between TP and JP to grow. If, as in our workload, the number of tasks is not very large with respect to the number of processors, then our results show that the benefits of allowing task preemption might not be worth the overheads associated with such preemption. The marginal improvement of TP over FP is also seen in Figure 8 where we plot the mean job response time. The SQ policy, as seen in this figure, is smallest for all utilizations.

4.2 Processor Partitioning

In this subsection we address the issue of whether it makes sense to run jobs having very different characteristics on the same parallel processing machine. We consider two types of jobs: edit jobs, which consist of 1 task, and batch jobs consisting of 16 tasks, and assume that there are 16 processors in the system. In [23] such a model was considered and the proposal was made to partition the processors into two groups, each being dedicated to one class of job. The rationale behind such a partitioning was to segregate the two classes of jobs so that each could be executed independently of the other. Since loads on real systems fluctuate with time, it was observed that an important component of such a partitioned system is to have a way to adjust the number of processors allocated to each job class as a function of their individual loads. The complexities of managing such a partition suggest that it might be profitable to seek a less complex approach leading to the same type of class response time management. In this section we investigate the use of priority scheduling for such a problem. We first start by restating some results from [23].

In Figure 9, we show the ratio of mean class response times when a partitioned system with κ processors is allocated to the edit jobs to that of a FCFS system. In this model the relative arrival rates of the two classes of jobs are .5 and .5. In this and subsequent figures we show a horizontal dashed line at the value 1. Curves above this line in Figure 9 indicate that the class response time for the partitioned system is greater than that of FCFS. It is interesting to note that both classes of jobs are hurt in the partitioned systems for $\kappa = 1$. For higher values of κ, however, edit jobs in the partitioned system receive preferential treatment at the expense of batch jobs. In going from $\kappa = 2$ to $\kappa = 3$ it is interesting to note that the response time of edit jobs is only marginally decreased but the increase in batch response time is substantial. This arises from the fact that the edit jobs cannot use the additional processor allocated to them and that the batch jobs need more processing power. This illustrates the importance of setting the partitioning correctly. A natural question arises: can a priority scheme do uniformly better than any partitioning scheme? The answer to this question, for the system we consider, is yes.

In Figures 10-12 we plot the ratio of mean class response times for the FP, JP and TP priority schemes to that of the partitioned system. A value below 1 here means that the priority scheme has a lower class response time than that of the partitioned system. Since the $\kappa = 1$ partition negatively influenced both classes of jobs, it is not surprising that all the priority policies have lower edit and batch response times than this partitioning. The behavior of the priority policies for the other two partitions is, however, quite revealing. The FP system has a lower edit response time, but a higher batch response time. The JP system has lower edit and batch response times, except for the $\kappa = 3$ batch response which is marginally larger than the partitioned system. The search for a uniformly better priority scheme, however, ends with the TP system which, for all values of κ, achieves better edit and batch response times.

One of the problems with a partitioned system is that work is not evenly divided among the processors. In particular, for such a system, there can be a idle processor in one partition when there are queued tasks in the other partition. In the priority systems we studied this can never happen and accounts for some of the performance improvements seen in our study. We generally expect that priority systems, and specifically the TP system, will have better performance characteristics than a partitioned system. Since a priority system is easier to implement and to control, this suggests that it is preferable to that of a statically partitioned system.

4.3 Variance of Parallelism

In this section we investigate the effects of varying the parallelism in the workload. Specifically, we consider a system with two classes of jobs, edit and batch, having relative arrival rates of .98 and .02, respectively. Edit jobs consist of one task and batch jobs have a mean of 8 tasks. We varied the distribution of the number of tasks composing a batch job to determine how the coefficient of variation of batch size influenced the mean response time. We ran four experiments with parameters given by the following table.

Number	Distribution	Coef. of Var.
1	8(1)	0
2	2(.5) 14(.5)	.5625
3	2(.6) 17(.4)	.8437
4	2(.75) 26(.25)	1.6875

To interpret the table, batch jobs in experiment 2 have 2 tasks with probability .5 and 8 tasks with probability .5. We first present the results for the fixed priority system. In all of the graphs we form the ratio of mean response times for systems 2-4 to that of the expected response time of system 1, which has a coefficient of variation equal to 0.

In Figures 13 and 14 we plot the ratio of mean response times for the edit and batch jobs, respectively, for the fixed priority case. The edit curves are seen to be increasing with the coefficient of variation. This follows from the fact that for fixed priorities, an edit job can be held up by a long batch job. For case 4 above, 25 percent of the time a batch job consists of 26 tasks which can delay an arriving edit job. The influence of the coefficient of variation of batch size on edit response time is somewhat intuitive. What is perhaps not so intuitive is the influence on batch response time. This is not strictly increasing with the coefficient of variation and, in particular, for up to moderate arrival rates, case 4 has the smallest mean response time. To explain this, consider a sufficiently small arrival rate so that queueing delays can be ignored. In this case the mean response time is simply the mean service time of a job. For batch jobs of size b this is given by H_b if $b \leq c$ and by $(b-c)/c + H_c$ if $b > c$. The mean response time is thus a convex combination of concave functions which decreases as the coefficient of variation of the distribution increases.

In Figures 15 and 16 we plot the corresponding graphs for the job preemption system (note the y-axis scale in Figure 15). Very small effects are seen for edit response times. This follows from the fact that long batch jobs do not hold up edit jobs. The batch response time ratios, in Figure 16, show similar qualitative behavior to that of the fixed priority case. We have not shown the figures for the task preemption case since they are similar to those for job preemption.

5 Summary

In this paper we have analyzed four different priority scheduling policies for fork/join jobs running on a parallel processing system. The number of tasks composing a job in our models is a class dependent random variable whose distribution can be selected to match a given workload. Graphs have been provided that show the relative benefits of the different policies. Our results demonstrate the benefits of using a priority scheduling policy to differentiate between the response times of different classes over using a FCFS policy. We have observed that for the workload we considered, the benefits of using a policy that allows task preemption over that of one using job preemption is not sufficient to warrant the probably overheads associated with preempting tasks. We have also observed that the policy that selects the job with the smallest number of tasks in the queue achieves the lowest mean response time.

In our study on the partitioned system we have shown that the TP policy has a uniformly better mean class response time than that found for any partitioning. This result is expected to generally hold and suggests that one should not consider statically partitioned systems.

Acknowledgments. We would like to thank the referees whose careful reading and comments of the first version of this manuscript helped us produce a significantly improved revision.

References

[1] T. Axelrod, "Effects of Synchronization Barriers on Multiprocessor Performance", *Parallel Computing*, 3,pp.129-140, 1986.

[2] F. Baccelli, A.M. Makowski, A. Shwartz, "The Fork-Join queue and related systems with synchronization constraints: Stochastic ordering and computable bounds",to appear in *Advances in Applied Probability.*

[3] F. Baccelli, W.A. Massey, D. Towsley, "Acyclic Fork-Join queueing networks", *J. Assoc. Comp. Mach.*, **36**,3, 615-642, July 1989.

[4] F. Baccelli and Z. Liu, "On the Execution of Parallel Programs on Multiprocessor Systems, A Queueing Theory Approach ", CNET Technical Report NT/PAA/ATR/SST/2238, August 1988. *to appear in J. Assoc. Comp. Mach.*

[5] A. Cobham, "Priority Assignment in Waiting Line Problems", *Operations Research*, **2**, 70-76, (1954).

[6] R. Cytron, "Useful Parallelism in a Multiprocessing Environment", *Proc. 1985 Parallel Processing Conf.*.

[7] M. Dubois, C. Scheurich and F. Briggs, "Synchronization, Coherence, and Event Ordering in Multiprocessors", *IEEE Computer*, 21,pp.9-21, February 1988.

[8] L. Flatto and S. Hahn, "Two parallel queues created by arrivals with two demands I", *SIAM J. Appl. Math.*, vol. 44, pp.1041-1053, Oct 1984.

[9] H Flatt and K. Kennedy, "Performance of Parallel Processors", *Parallel Computing*, 12,pp.1-20, 1989.

[10] A. Greenbaum, "Synchronization Costs on Multiprocessors", *Parallel Computing*, 10,pp.3-14, 1989.

[11] P. Brinch Hansen. "The programming language concurrent pascal", *IEEE Transaction on Software Engineering.*, **1**, 1975.

[12] P.G. Harrison. "On Normalizing Constants in Queueing Networks", *Operations Research*, **33**, 2, 464-468, 1985.

[13] D.P. Heyman, M.J. Sobel, *Stochastic Models in Operations Research*, vol. 1, McGraw Hill, 1982.

[14] C.A.R. Hoare. *Communicating Sequential Processes*, Prentice-Hall International, London, 1985.

[15] C. Kim, A.K. Agrawala, "Analysis of the Fork-Join Queue", *IEEE Trans. Computers, 38*, 2, pp. 250-255 (Feb. 1989).

[16] L. Kleinrock. *Queueing Systems, Volume II: Computer Applications*, John Wiley, 1976.

[17] Koenigsberg, E., "Comments on Normalizing Constants in Queueing Networks", *Operations Research*, **34**, 2, 330 (March-April 1986).

[18] P. Konstantopoulos and J. Walrand, "Stationarity and stability of Fork-Join Networks" To appear in the J.A.P., (1989).

[19] *Computer Performance Modeling Handbook*, Ed., S.S. Lavenberg, Academic Press, New York, 1983.

[20] S. Leutenegger and M. Vernon, "The Performance of Multiprogrammed Multiprocessor Scheduling Algorithms", *Proc. 1990 Conf. Measure. & Model. of Comp. Systems*, pp. 226-236, May 1990.

[21] S. Majumdar, D.L. Eager, R.B. Bunt, "Scheduling in Multiprogrammed Parallel Systems", *Proc. 1988 ACM Conf. Measurement and Modeling of Comp. Systems*, pp. 104-113, 1988.

[22] A.J. Musciano and T. L. Sterling, "Efficient Dynamic scheduling of medium-grained tasks for general purpose parallel processing," *Proc. Int. Conf. on Parallel Processing*, pp. 166-175, 1988.

[23] R. Nelson, D. Towsley, A.N. Tantawi. "Performance Analysis of Parallel Processing Systems", *IEEE Trans. Software Engng.*, 14, 4, 532-540, April 1988.

[24] R. Nelson, A.N. Tantawi, "Approximate analysis of Fork/Join synchronization in parallel queues", *IEEE Trans. Computers*, C-37, pp. 739-743, 1988.

[25] R. Nelson, "A Performance of a General Parallel Processing Model", *Performance Evaluation Review*, Vol 18, no 1, pp.13-26, 1990.

[26] M. Neuts, "Matrix Geometric Solutions in Stochastic Models", *The Johns Hopkins Press*, 1981.

[27] C.G. Rommel, D. Towsley, J.A. Stankovic, "Analysis of fork-join jobs using processor sharing," COINS Technical Report, TR 87-52, U. Massachusetts, 1987.

[28] Anita Osterhaug. *Guide to Parallel Programming.* Sequent Computer Systems, Inc, Beaverton, Oregon, 1986.

[29] I.C. Pyle. *The Ada Programming Language*, Prentice-Hall International, London, 1981.

[30] M. Seager and James Stichnoth, "Simulating the Scheduling of Parallel Supercomputer Applications," UCRL-102059, Lawrence Livermore National Laboratory, Livermore, CA 94551, Sept. 1989.

[31] K. Sevcik, "Characterizations of Parallelism in Applications and their use in Scheduling", *1989 ACM Sigmetrics Proceedings, Performance Evaluation Review* Vol. 17, No. 1, pp.171-180, May 1989.

[32] D. Towsley, S.P. Yu, "Bounds for Two Server Fork-Join Queueing Systems", COINS Tecnical Report, TR 87-123, U. Massachusetts, Nov. 1987.

[33] D. Towsley, C.G. Rommel, J.A. Stankovic, "The Performance of Processor Sharing for Scheduling Fork-Join Jobs in Multiprocessors", *High Performance Computer Systems*, ed. E. Gelenbe, North Holland, 145-156, 1988.

[34] D.D. Yao. "Some Results for the Queues $M^X/M/c$ and $GI^X/G/c$", *Oper. Res. Lett.*, 4, 79-83, July 1985.

[35] J. Zahorjan and C. McCann, "Processor Scheduling in Shared Memory Multiprocessors," *Performance Evaluation Review*, Vol 18, no 1, pp.214-225, 1990.

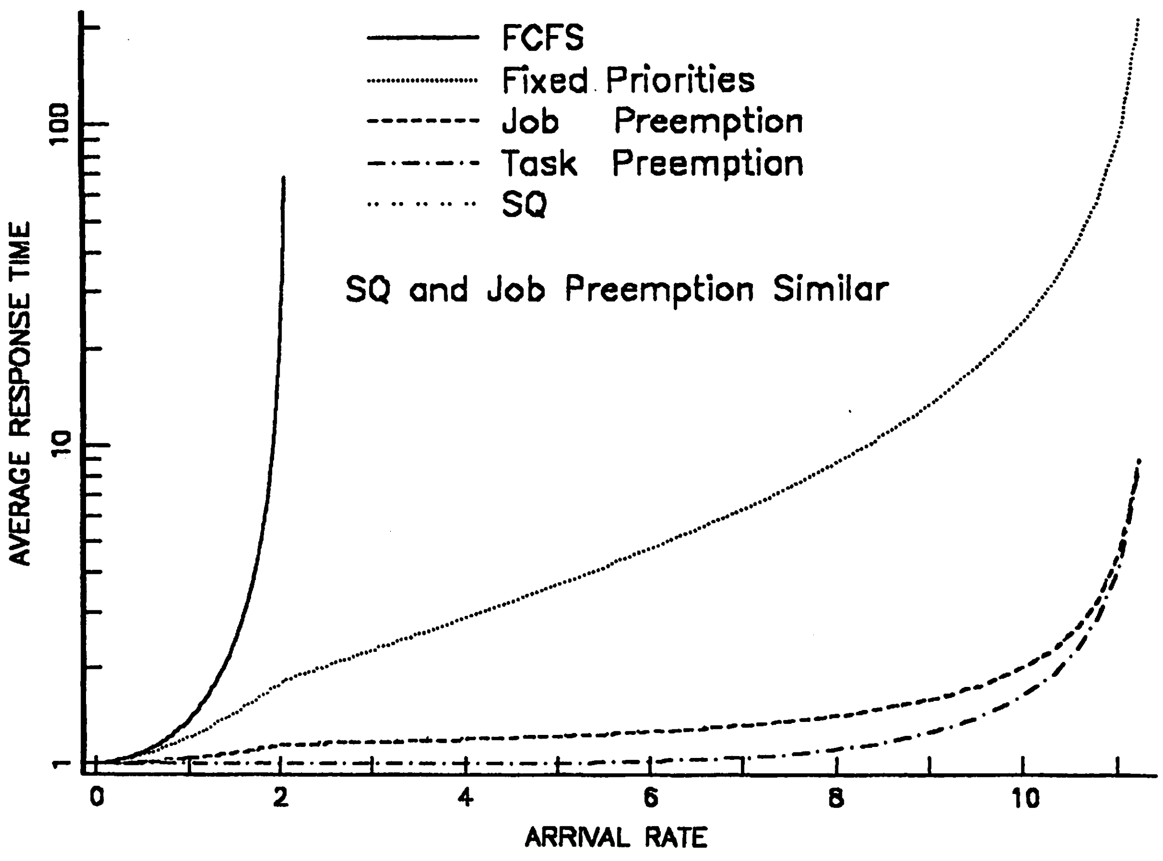

Figure 2

Class 1 Response Time, c = 8

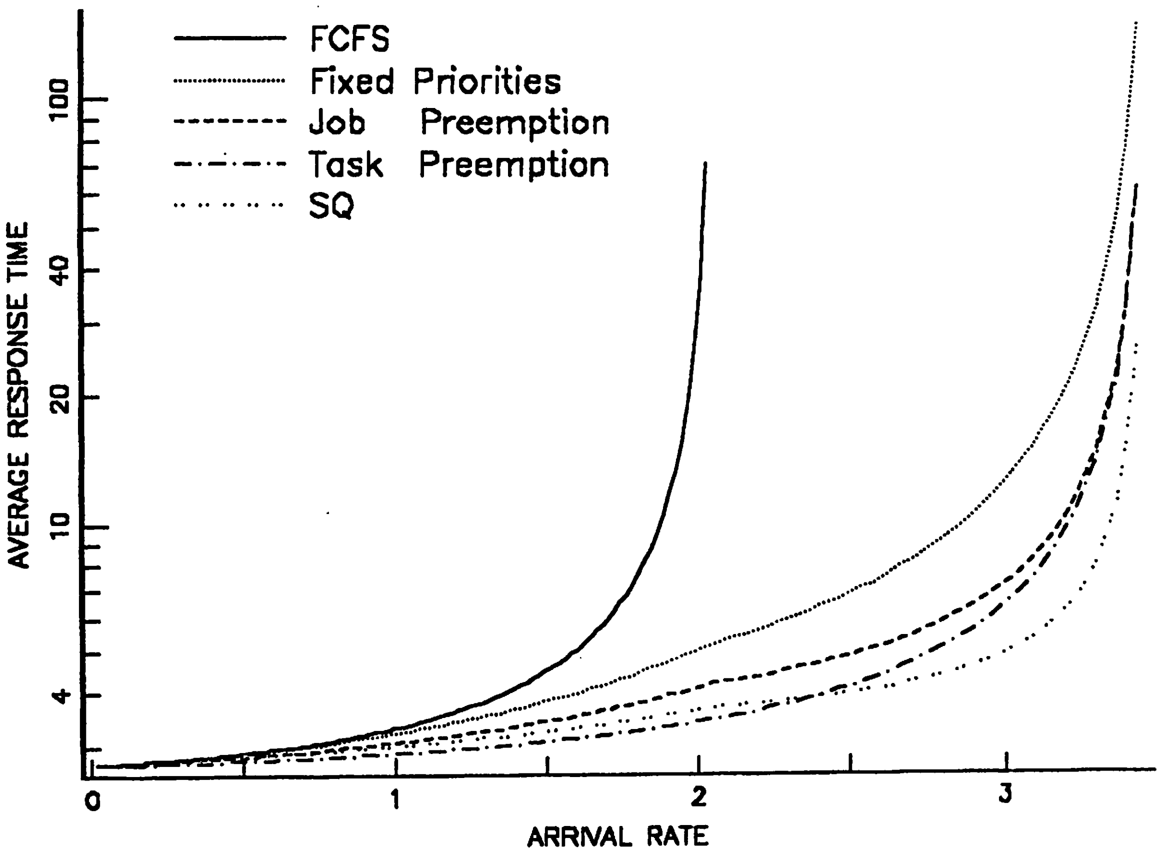

Figure 3

Class 2 Response Time, c = 8

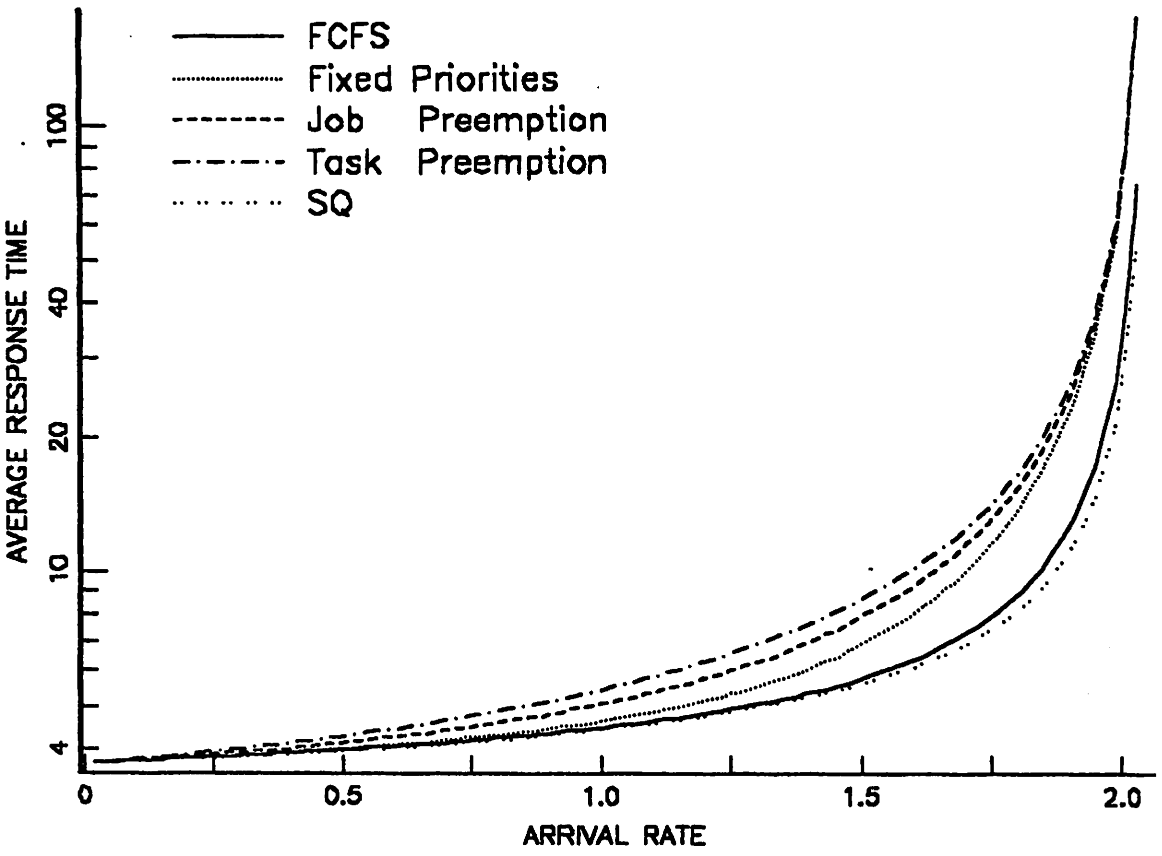

Figure 4

Class 3 Response Time, c = 8

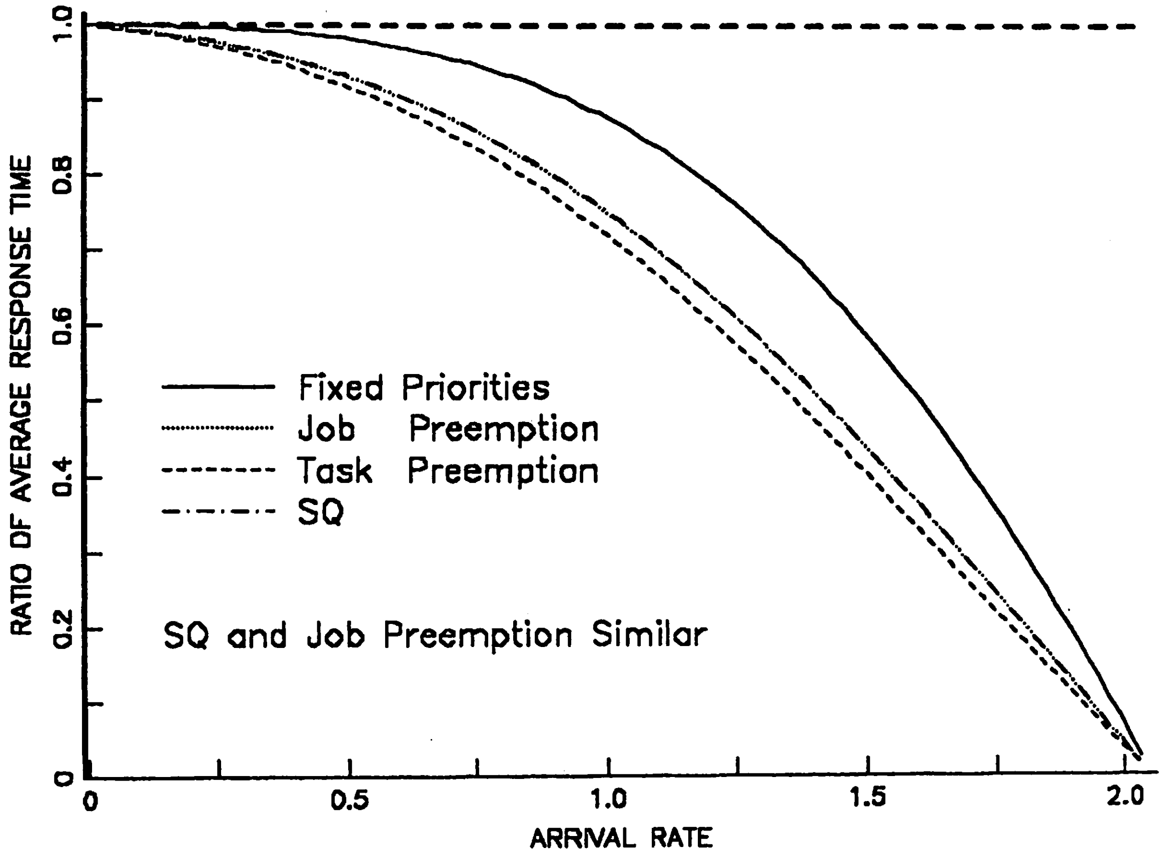

Figure 5

Ratio of Class 1 Response Time, c = 8

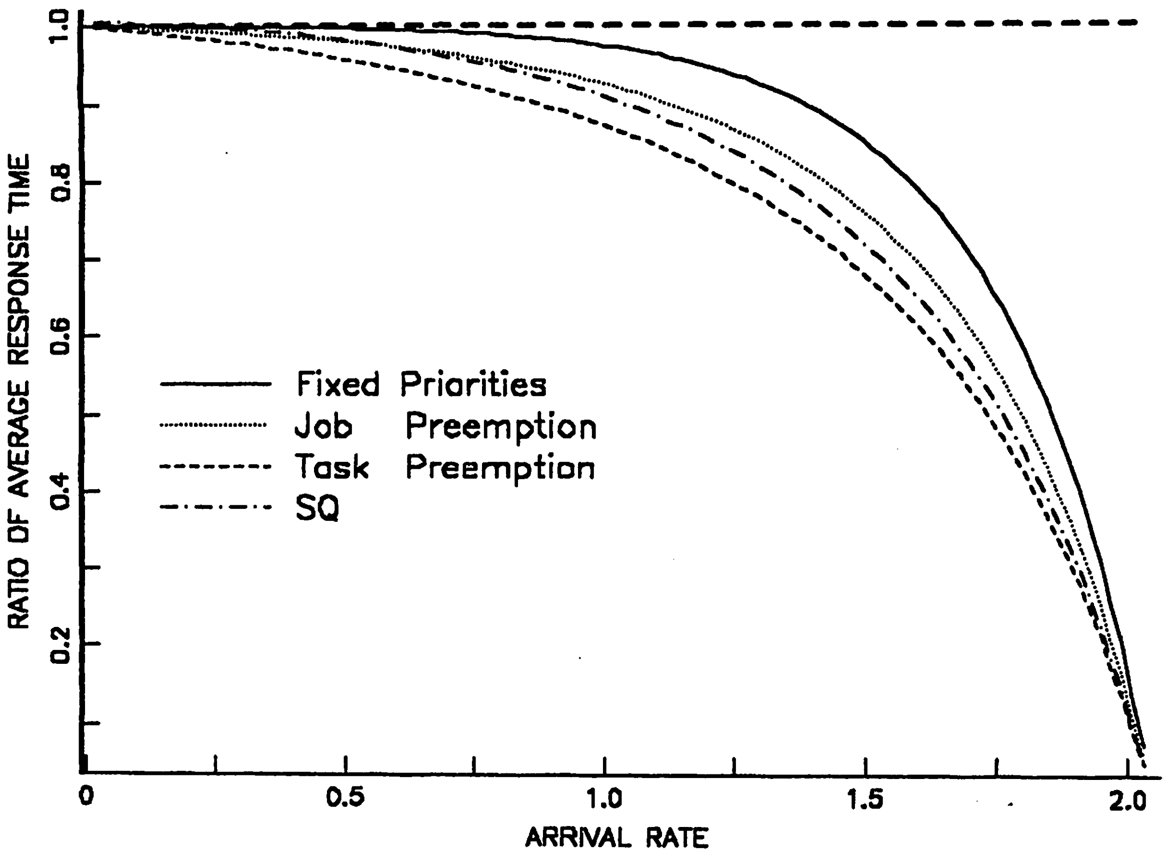

Figure 6

Ratio of Class 2 Response Time, c = 8

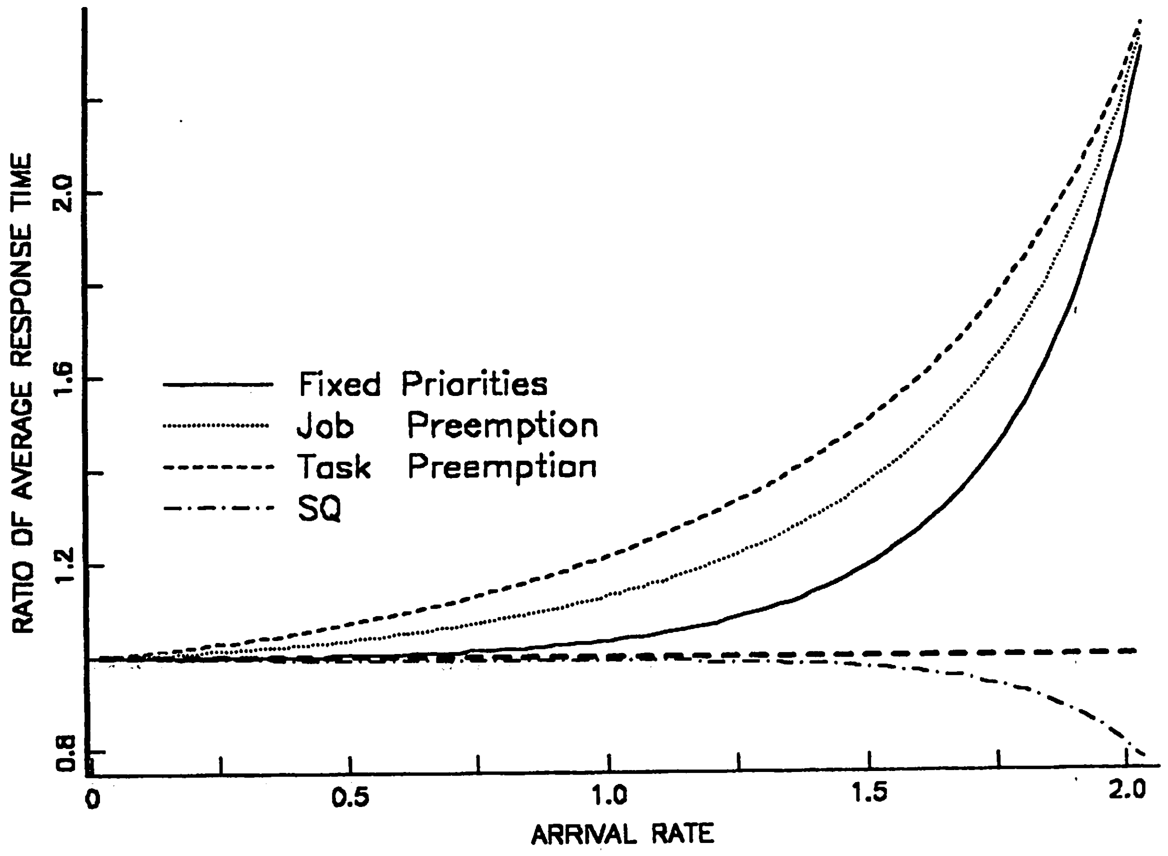

Figure 7

Ratio of Class 3 Response Time, c = 8

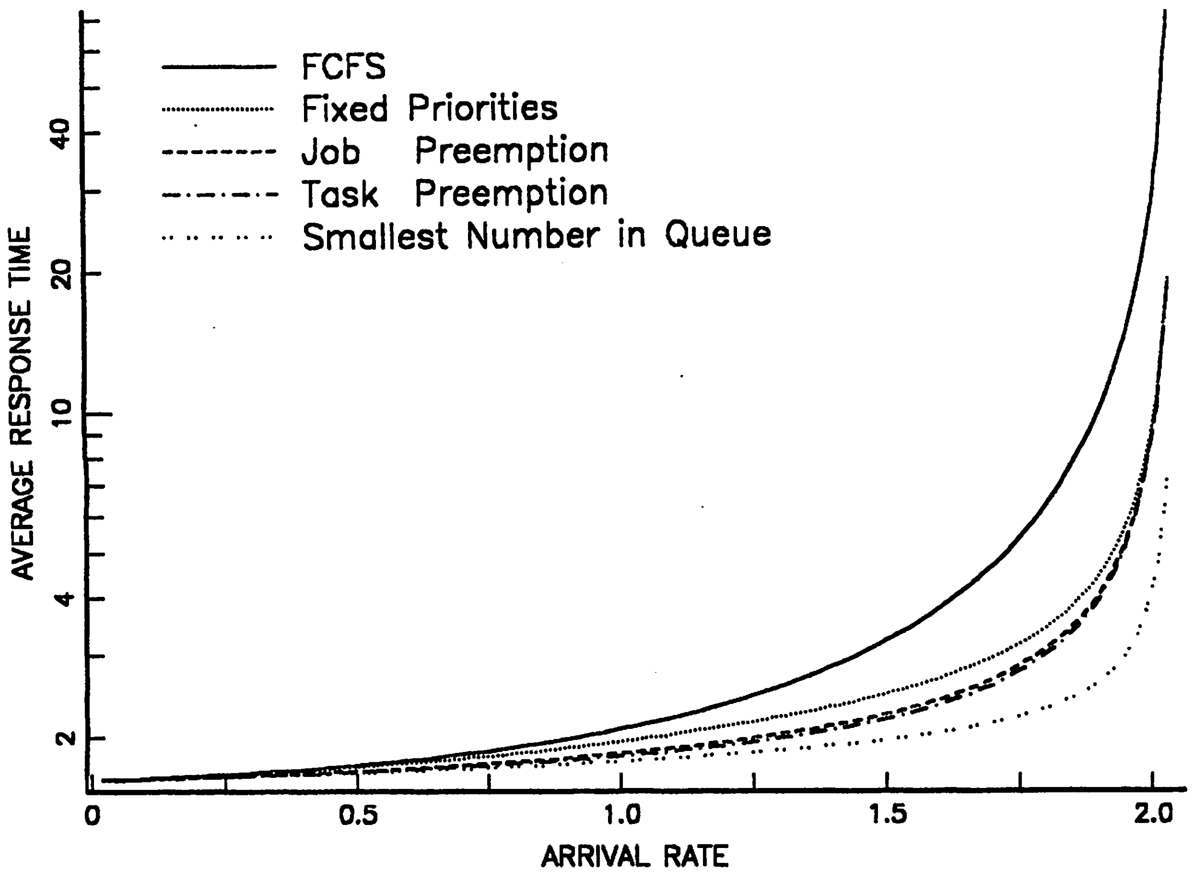

Figure 8

Average Job Response Time, $c = 8$

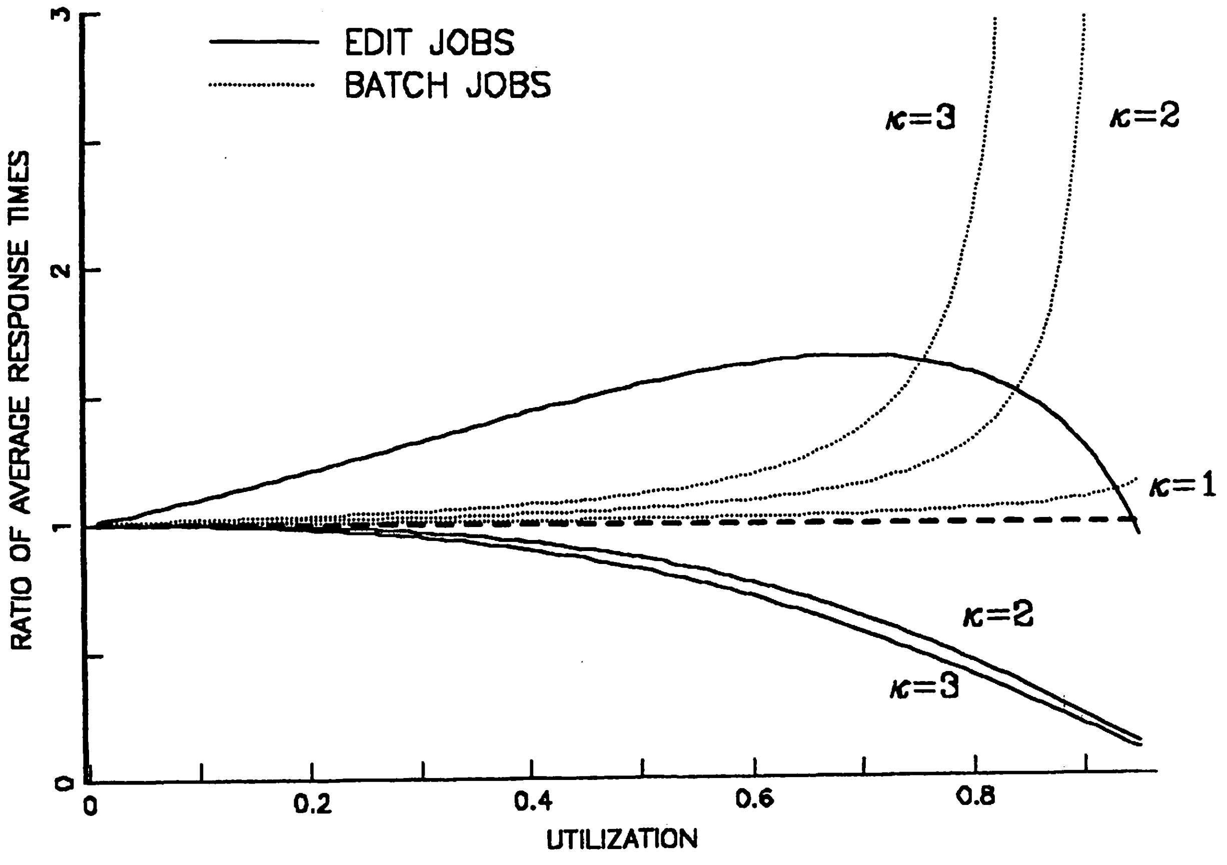

Figure 9

Ratio of Class Response Time

Partitioned to FCFS

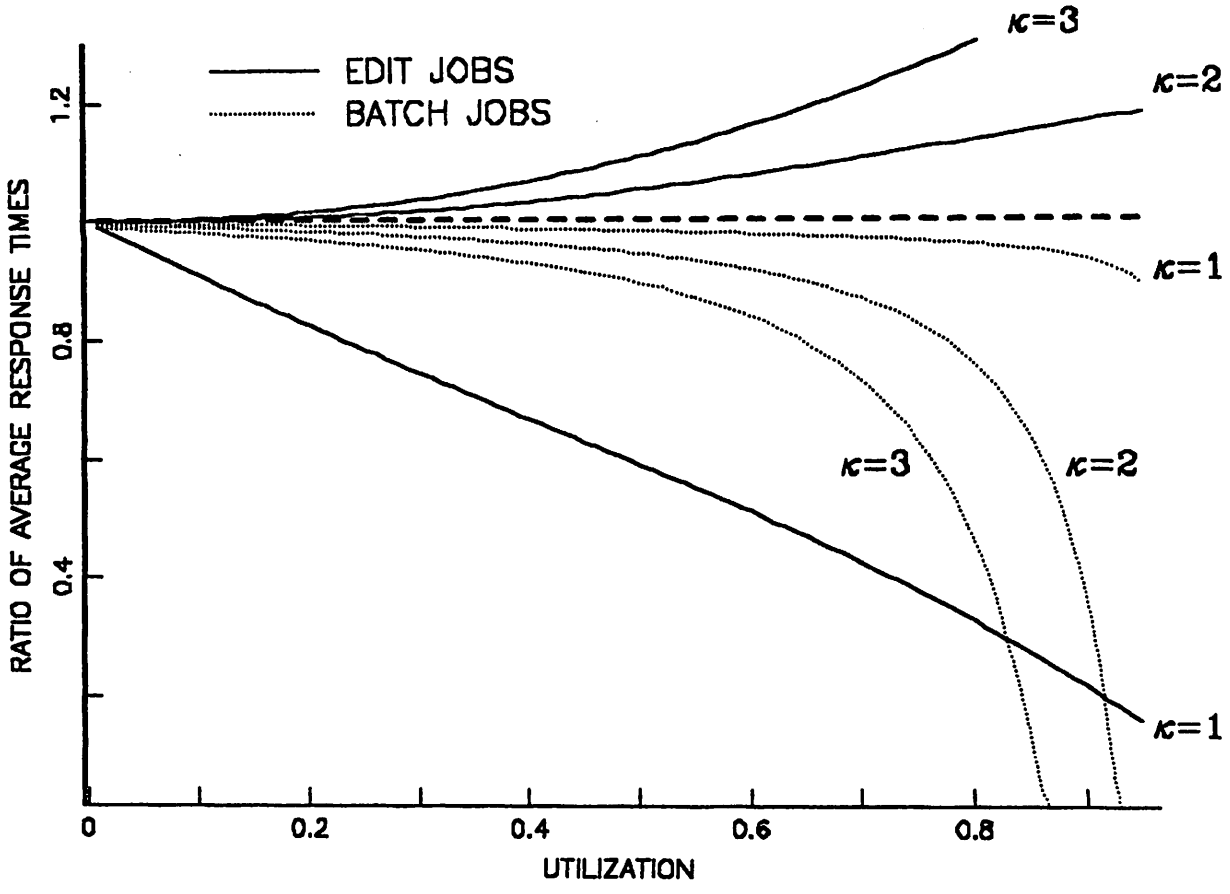

Figure 10

Ratio of Class Response Time

FP to Partitioned

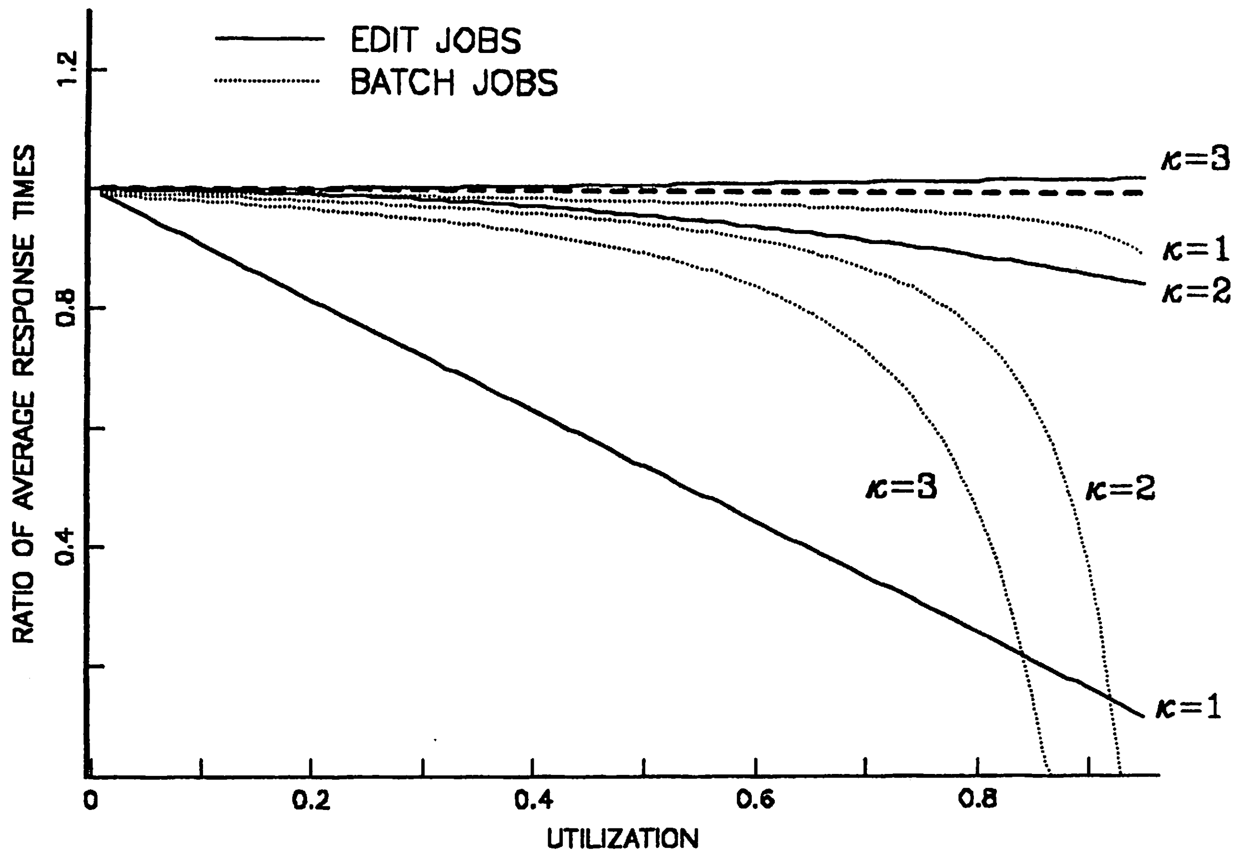

Figure 11

Ratio of Class Response Time

JP to Partitioned

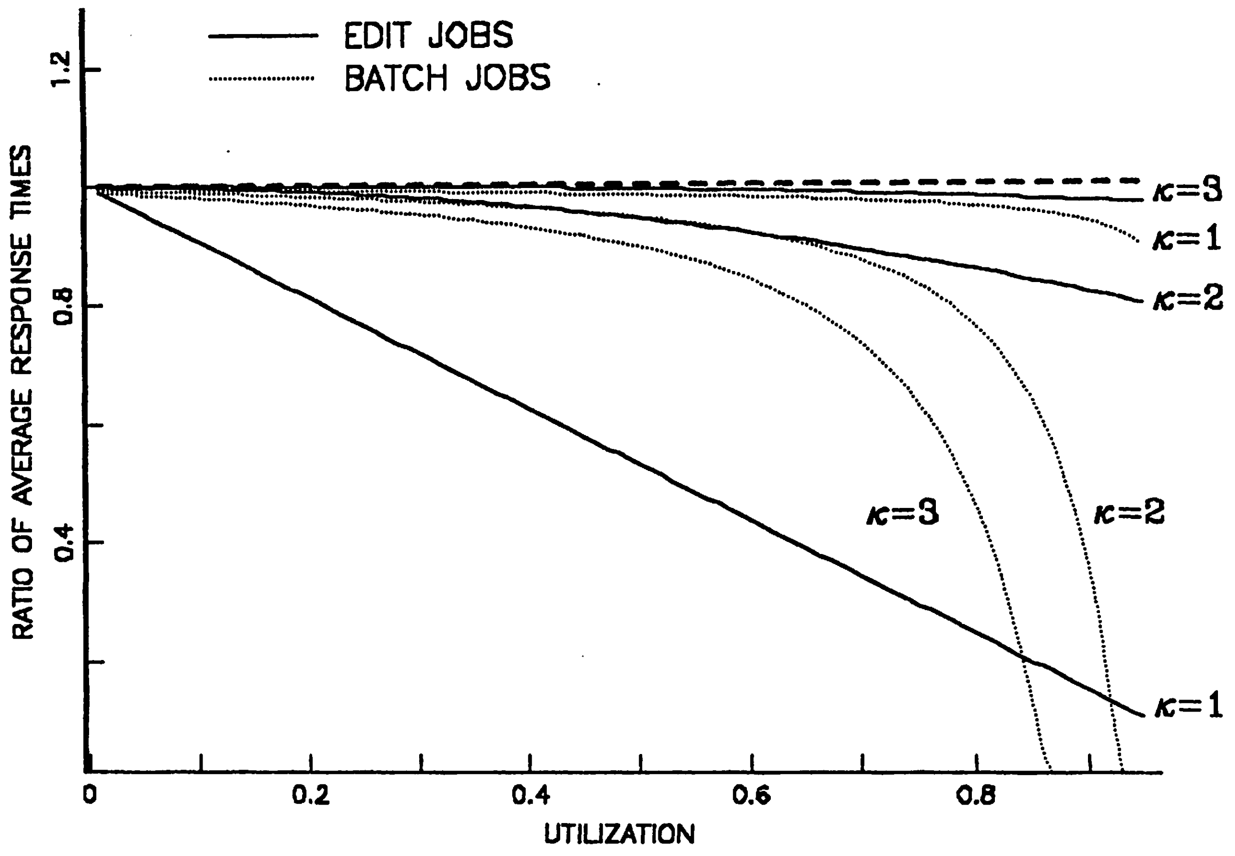

Figure 12

Ratio of Class Response Time

TP to Partitioned

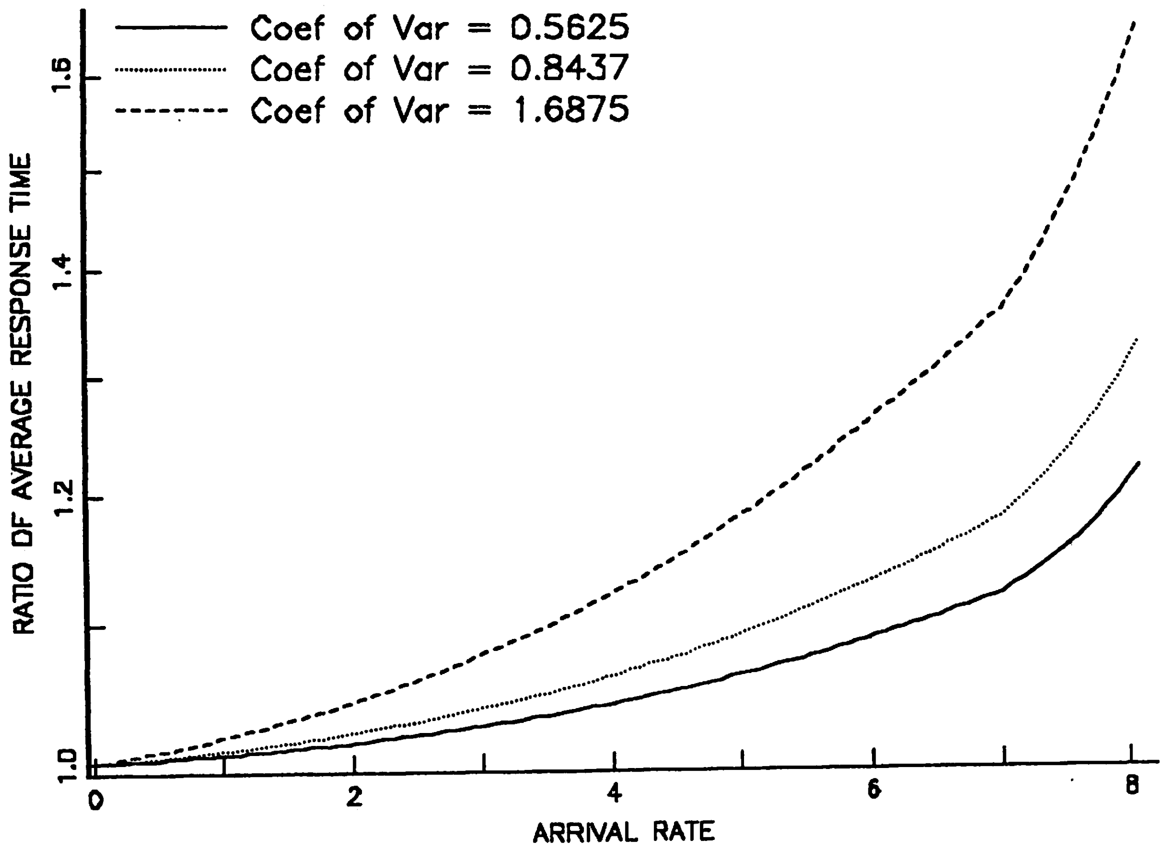

Figure 13

Ratio of FP Class 1 Response Time

Base System has Coefficient of Variation of Zero.

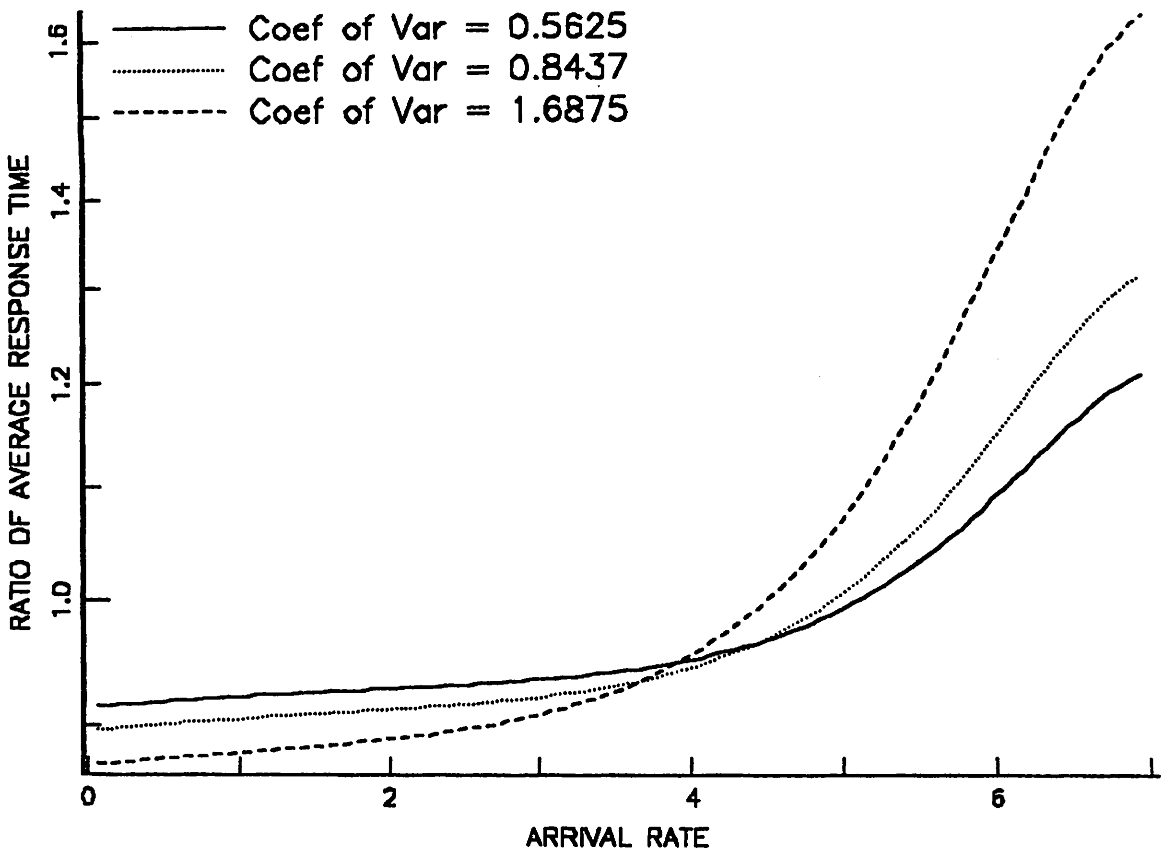

Figure 14

Ratio of FP Class 2 Response Time

Base System has Coefficient of Variation of Zero.

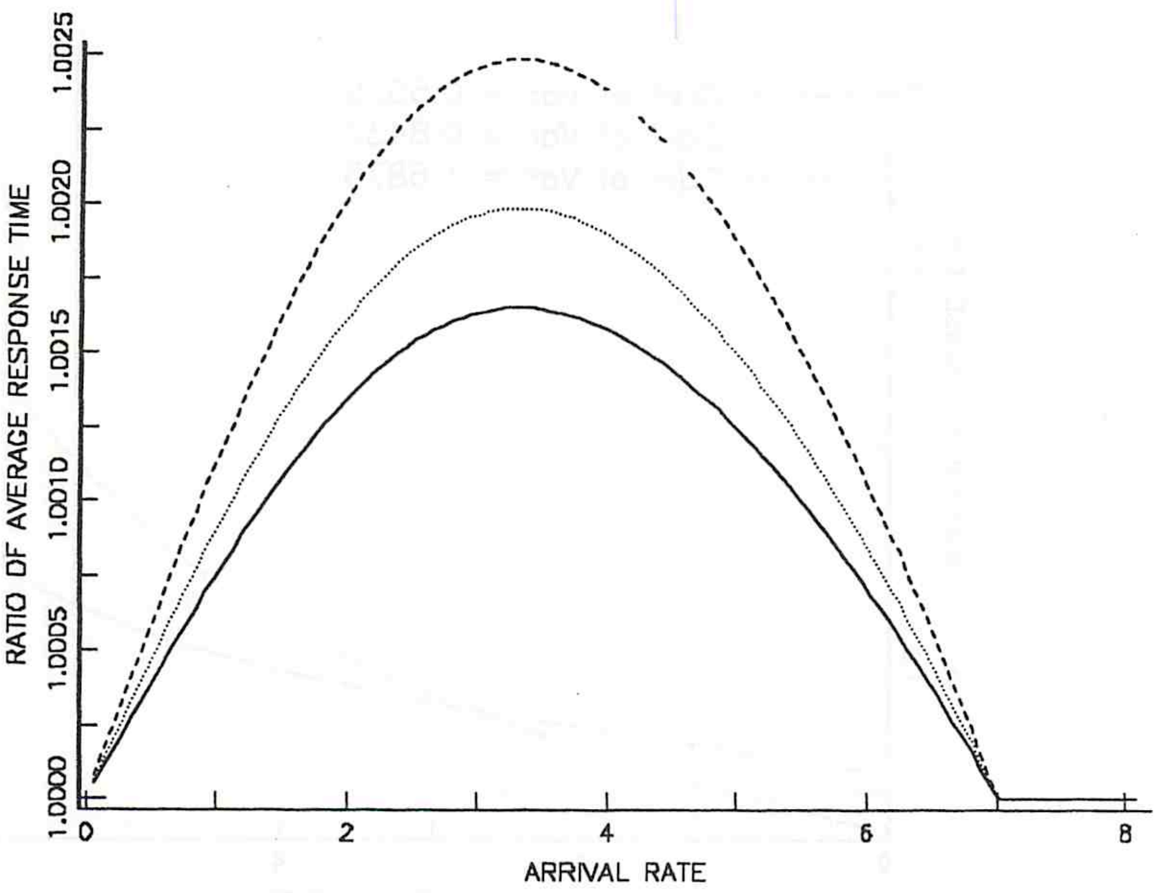

Figure 15

Ratio of JP Class 1 Response Time

Base System has Coefficient of Variation of Zero.

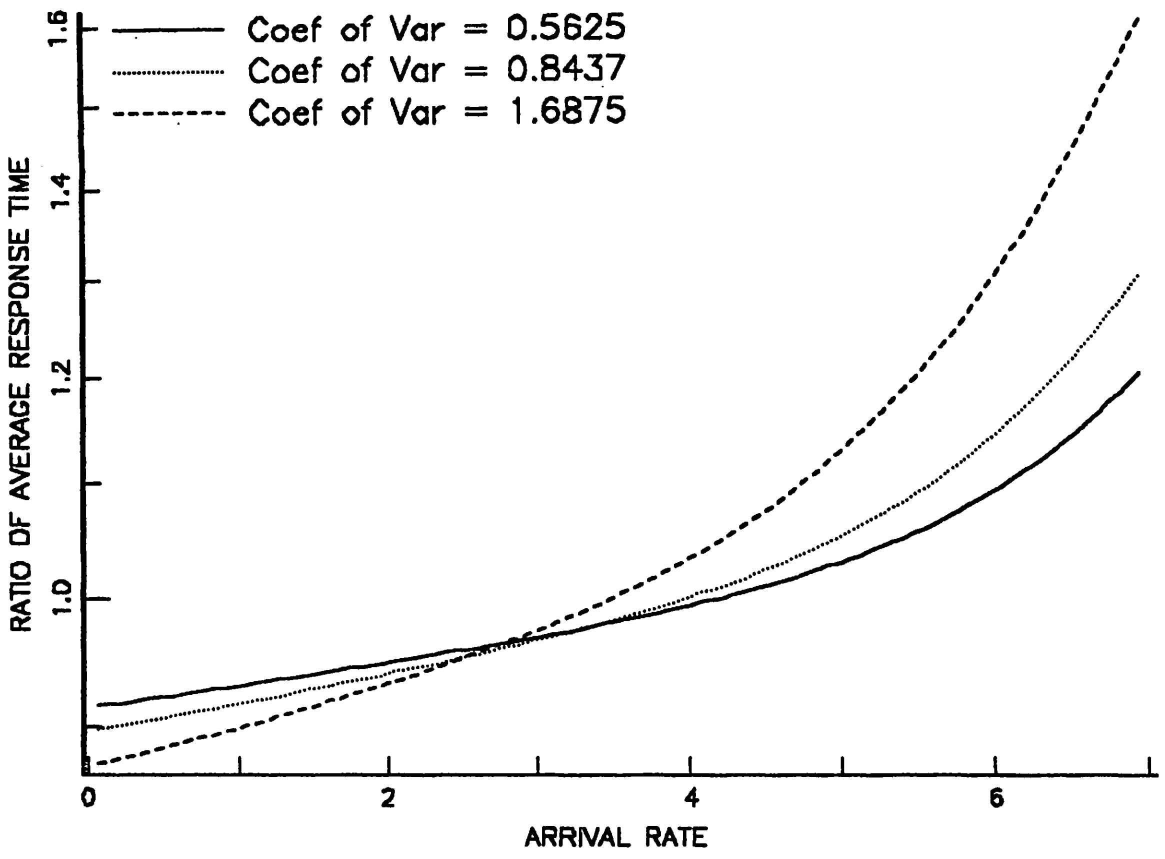

Figure 16

Ratio of JP Class 2 Response Time

Base System has Coefficient of Variation of Zero.