ISSUES IN MULTIPROGRAMMED MULTIPROCESSOR SCHEDULING

Scott T. Leutenegger

Computer Sciences Technical Report #954

August 1990

ISSUES IN MULTIPROGRAMMED MULTIPROCESSOR SCHEDULING

by

SCOTT T. LEUTENEGGER

A thesis submitted in partial fulfillment of the

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN-MADISON

1990

Abstract

Scheduling policies for general purpose multiprogrammed multiprocessors are not well understood. This thesis examines various policies to determine which characteristics of a scheduling policy are the most significant determinants of performance. In particular we consider three scheduling policy characteristics: allocation of processing power among competing jobs, support for inter-process synchronization, and preemption frequency. We find that allocation of processing power among competing jobs is at least as important as the other two scheduling policy characteristics.

We compare a more comprehensive set of policies than previous work, including four scheduling policies that have not previously been examined. We also compare the policies under workloads that may be more realistic than previous studies have used. Using these new workloads, we arrive at different conclusions than reported in earlier work. In particular, we find that the "smallest number of processes first" (SNPF) scheduling discipline performs poorly, even when the number of processes in a job is positively correlated with the total service demand of the job. We also find that policies that allocate an equal fraction of the processing power to each job in the system perform better than practical policies that allocate processing power unequally.

We find that allocation of processing power among competing jobs is at least as important as explicit support for spin-lock and barrier synchronization. (Minimizing spin-waiting is achieved by coscheduling processes within a job, or by using a thread management package that avoids preemption of processes that hold spinlocks.) We also find that allocation of processing power among competing jobs is a more important characteristic of a scheduling policy than preemption frequency for a wide range of preemption overhead values. Our studies are done by simulating abstract models of the system and the workloads.

Acknowledgements

I would like to thank Mary Vernon for being my advisor for the last 4 years. Through class work and three research projects, she has taught me the techniques and importance of performance modeling. The quality of this thesis was vastly improved by her many suggestions while conducting the research and preparing the thesis.

I would like to thank Josh Chover, Bart Miller, Jim Goodman, and Mark Hill for serving as committee members for my final defense. In addition I would like to thank Jim Goodman and Mark Hill for serving as readers, and providing numerous suggestions to improve the presentation of the material. I would also like to thank Miron Livny for numerous technical discussions, serving on my prelim committee, and providing the DeNet simulation environment used for conducting this research.

I would like to thank Lisa Leutenegger for discusing these ideas with me, proofreading this thesis, and offering many suggestions on how to enhance the presentation of the work. I would also like to think George Bier for many discussions about this work and in particular for pointing out that finding the optimal scheduling sequence for a multiprocessor is an NP-complete problem. I would also like to thank fellow students Lisa Leutenegger, Lester McCann, Donovan Schneider, Vikram Adve, Sarita Adve, George Bier, Amarnath Mukerhjee, and Shahram Ghandeharizadeh for discussing computer science as a whole, and providing companionship over the last 5 years. I would also like to thank Mark Friedman for providing the famous "Bubber" macros used in formatting this thesis.

On the non-technical front, I would like to thank my parents for instilling in me the importance of education. The most deserved thanks to anyone mentioned in this thesis go to my wife Lisa for her emotional support, patience, and love during my years in graduate school. Lisa, it is impossible to overstate your contributions towards this thesis. Last, but not least, I would like to thank our cats Toxie, Raid, Indie and Ashley for all the purrs and distractions they provided.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1. Thesis Research Overview

The goal of this thesis is to determine how to schedule competing jobs running on a general purpose multiprogrammed multiprocessor. In general, we aim to further our understanding of the issues involved, to identify the characteristics of a scheduling policy that are the most significant determinants of the policy's performance, and to identify the policies that are most promising for implementation. The environment we consider is a small (20 processor) tightly-coupled shared-memory multiprocessor. Examples include commercial products such as the Encore Multimax, the Sequent [LoTh 88], the DEC Firefly [TaSS 88], the BBN Butterfly, the Cray MPs, and the IBM 3090 MPs. We are interested in scheduling general workloads, allowing both serial and parallel jobs to share the processors. Our primary metric for comparing policies is mean job response time. We define a job to be a program composed of one (serial) or more (parallel) processes. We define job response time to be the time from when the job enters the system until its last process is completed. We assume the scheduling policy allocates processors to processes that are queued in a conceptually centralized ready queue.

There are three characteristics of a multiprocessor scheduling policy that may be important to the performance of the policy. These three characteristics are:

1) **Allocation of processing power among jobs.** How processing power is allocated among the jobs competing for service may have a significant impact on performance. For uniprocessor systems, it is well known that the preemptive smallest remaining processing time first (SRPT) policy results in the minimum mean job response time [Shra 68]. Furthermore, processor sharing (or round robin) scheduling performs well in the absence of a priori knowledge of job demands, and the performance is insensitive to the coefficient of variation of job demand. The good performance of the round robin policy results from giving each job in the system an equal fraction of processing power. Similarly, multi-level queues have been shown to perform well in the absence of a

priori job demand knowledge. We expect these general lessons to apply to the multiprocessor environment. However, the multiprocessor scheduling problem introduces two new considerations. First, the multiprocessor equivalent of the SRPT policy is not the optimal policy. Determining the optimal scheduling sequence for a multiprocessor is an NP-complete problem since the restricted problem of determining the optimal scheduling sequence for a multiprocessor with single tasking jobs is NP-complete [GaJo 79]. In a real system it is unlikely the scheduler would know job demands. Hence it is important to develop scheduling policies that perform well in the absence of a priori job demand knowledge. Second, processing power can be divided among competing jobs either temporally by time slicing, or spatially by partitioning the processors among the jobs.

2) **Support for inter-process synchronization.** Jobs that have inter-process synchronization can be adversely affected if the scheduling policy does not facilitate synchronization. In particular, processing power may be wasted if a policy allows processes to spin (busy-wait) on a processor while waiting for synchronization to occur when otherwise useful work could be done. We consider two approaches to facilitate inter-process synchronization: coscheduling and application level scheduling. Coscheduling facilitates inter-process synchronization by maximizing the number of processes from a job that are run at the same time. Application level scheduling facilitates inter-processes synchronization by making sure a job is always executing the processes most important for forward progress.

3) **Preemption Frequency.** Preemption of processes causes two types of overhead. First, there is the operating system context switch overhead at the time of the preemption. Second, there it the time to rebuild cache entries when a descheduled process is rescheduled on a processor. Even when the process is rescheduled on the same processor, a portion, possibly all, of the process' cache entries will have been removed by intervening processes. How frequently a policy preempts processes affects the response time of all jobs since the overhead for preemption directly translates into a slower execution rate.

Previous and concurrent work with one exception has proposed and evaluated specific policies that address either the second or third characteristic [Oust 82] [ZaLE 88] [ZaLE 89] [SeSt 89] [TuGu 89] [SqLa 90] . Our work seeks to determine the importance of the first characteristic, allocation of processing power per job, relative to the other two characteristics. In addition, we seek to directly compare the scheduling policies so we can determine which policies are good candidates for implementation. We use simulation to study twelve scheduling policies under a wide range of workloads. The study of these policies helps elucidate the importance of the three characteristics.

A previous study that looked only at the first characteristic concluded that allocating processing power to jobs with the smallest number of processes results in good performance even when there is no correlation between the number of processes per job and job demand [MaEB 88]. In addition, the study concluded that policies that allocate processing power on a round robin basis result in poor performance. We reexamine these results in our study. The questions pertaining to allocation of processing power we wish to address are:

- Does scheduling jobs with the smallest number of processes first result in good performance?
- Is it desirable to give all jobs in the system a fraction of the processing power as in the uniprocessor round robin policy?
- If each job is given a fraction of the processing power, is it important to give each job an equal share?
- What are the costs and benefits of temporally allocating the processing power versus spatially allocating the processing power?
- How important is allocation of processing power?

To evaluate the importance of a policy's support for inter-process synchronization we have run experiments assuming two types of inter-process synchronization: spin-lock and barrier synchronization. We consider both coscheduling and application level scheduling as techniques to support inter-process synchronization. By application level scheduling we mean there is two-level scheduling, a system level scheduler that allocates processors to jobs, and an application level scheduler responsible for deciding which processes of a job are run on the processors assigned to the job. By allowing the job to determine which processes are to be run we can assure that the processes executing are the most beneficial to forward progress. When a two level scheduler is used, threads instead of processes are actually scheduled. We will always refer to threads as processes to make the discussion simpler. More details on threads are found in section 1.5. Of all the policies studied only two require application level schedulers. For barrier synchronization, we consider both the absence and existence of two-level scheduling for the policies studied. The questions we wish to address are:

- In the absence of an application level scheduler, does coscheduling improve performance?
- Is coscheduling or application level scheduling a more effective approach for supporting inter-process synchronization?
- Does the combination of coscheduling and application level scheduling result in the best performance?
- How important is support for inter-process synchronization?

To evaluate the importance of preemption frequency we determine the likely overhead cost per preemption and then study the policies under a wide range of preemption overhead values. The overhead per preemption is equal to the operating system context switch overhead plus overhead for rebuilding process cache entries once a descheduled process is rescheduled. We have developed a simple analytical model to aid in estimating the overhead incurred in rebuilding process cache working sets. We wish to address the following questions:

- How large is the preemption overhead due to rebuilding process cache entries?
- How does preemption overhead affect the relative performance of the policies?
- How important is preemption frequency?

Multiprocessor scheduling policies can be divided into two classes: static and dynamic. In a static policy some number of processors are allocated to a job. Once allocated, the number does not change. If too many processors are allocated to a job the processors remain idle even if other jobs in the system could make us of them. In addition, if too few processors are allocated to a job, processing power is wasted if any of the other processors are idle. This wasted processing power results in poor overall performance. Dynamic policies allow the number of processors allocated to a job to vary during the job's execution. For example, FCFS is a dynamic policy since the number of processors allocated to a job changes as the processors become available. We consider only dynamic policies in this study since the static policies are not promising. We state other objections to the static policies in section 1.5. Concurrent work supports our decision to not consider static policies [ZaMc 90].

We study the twelve scheduling policies under six workload models with a variety of parameter settings. We choose these policies because they address one or more of the three characteristics above. We study three synchronization workload models: no synchronization, spin-lock synchronization, and barrier synchronization. We also consider two job demand models: uncorrelated and correlated. The uncorrelated model assumes no correlation between the number of processes per job and job demand, whereas the correlated workload model assumes there is a positive linear correlation between the number of processes per job and the job demand. We combine the three synchronization workload models with the two job demand workload models for a total of six workload models. We explore the implications of increasing job parallelism and also explore the effects of changing the distribution of the number of processes per job. From our studies we gain insight into the relative importance of the three scheduling policy characteristics listed above, and learn which policies are most promising for implementation.

1.2. Organization of thesis

In the remainder of this chapter we summarize the contributions of this thesis and discuss related work. Section 1.3 lists the contributions of this thesis, section 1.4 discusses relevant uniprocessor scheduling work, section 1.5 discusses previous multiprocessor scheduling work, and section 1.6 discusses recent (i.e. concurrent) multiprocessor scheduling research. Chapter 2 defines the twelve scheduling policies studied in this thesis. We include motivation as to how we would expect these policies to behave in regards to the three characteristics and why. Chapter 3 describes the models and parameters used in the study. We present the job structure model, the multiprocessor model, how the number of processes per job is distributed, how demand per job and per process is distributed, and what type of inter-processes synchronization is assumed. Chapter 4 presents the results of experiments assuming no inter-process synchronization. We demonstrate the importance of how processing power is divided among the jobs in the system. In chapter 5 we explore the importance of allocating processing power equally per job. In Chapter 6 we explore the importance of supporting inter-process synchronization. We present results from spin-lock synchronization and barrier synchronization. In chapter 7 we explore the importance of the preemption frequency. Chapter 8 contains our conclusions and plans for future work.

1.3. Contributions of this Thesis

It is well-known that a key characteristic of a good *uniprocessor* scheduling policy is that it rapidly grants some processing time to small jobs so that they finish quickly. One of the principal results of this thesis is that a key characteristic of a good *multiprocessor* scheduling policy is that it rapidly grants some processing power to small jobs so that they finish quickly.

The idea that rapidly granting processing power to small jobs is important in multiprocessors may seem obvious in retrospect, but this result has not been given much attention by all but one previous study of multiprocessor scheduling policies. Most studies have ignored this factor in favor of considering more complex multiprocessor interactions, such as supporting inter-process synchronization or minimizing overhead due to the rebuilding of process cache entries after switching processes. The one previous study that focused on the allocation of processing power was incomplete in regards to sensitivity studies and the policies included in the study. In addition, the study erroneously concluded that policies that give highest priority to jobs with the smallest number of processes result in good performance. No earlier study examined of all three characteristics of a multiprocessor scheduling policy enumerated in section 1.1.

Other more-specific contributions of this thesis, include:

- Scheduling policies that give highest priority to jobs with the smallest number of processes result in poor performance when the coefficient of variation of job demand is 3.0 or higher.
- In the absence of a priori job demand knowledge, policies that do not allocate a fraction of processing power to each job in the system result in poor performance.
- Policies that allocate processing power equally to all jobs result in better performance than policies that allocate processing power proportional to the number of processes per job. The difference in performance is magnified when job demand is correlated

with the number of processes per job.

- We present a new workload model that may be more realistic than models used in previous studies. We find that the qualitative performance of the policies is different for this new model than for a previous model.
- How processing power is allocated among the jobs is at least as important as support for inter-process synchronization.
- Spin-lock synchronization does not affect the relative performance of the scheduling policies unless the policies compared allocate processing power equally well.
- Barrier synchronization changes the relative performance of the policies only when there are more than two barriers per unit where a unit equals the duration of a quantum in a round robin policy.
- A high degree of coscheduling actually degrades performance when there is application level scheduling.
- For a wide range of likely process cache working set sizes, overhead due to rebuilding processor caches is likely to be less than 10%. As a result, scheduling policies that give each job in the system a fraction of the processing power by time slicing still perform better than polices that do not give a fraction of processing power to each job in the system. Thus, allocation of processing power among the jobs appears to be a more important characteristic of a scheduling policy than the preemption rate when the overhead per preemption is less than 10%.

1.4. Uniprocessor Scheduling Results

There has been much work done in studying uniprocessor scheduling policies. For a good overview of analytical models of uniprocessor scheduling we refer the reader to [Klei 76]. Figure 1, which is a reproduction of figure 1 in [MaEB 88], summarizes relevant uniprocessor scheduling results. The figure shows that as the coefficient of variation of service demand increases the mean job response time of First Come First Served (FCFS) and Shortest Job First (SJF) increases, Processor Sharing (PS) remains constant, and Shortest Remaining Processing Time (SRPT) decreases. The SJF policy is a non-preemptive scheduling policy that gives highest priority to the job with the smallest demand. The SRPT policy is a preemptive policy that gives highest priority to the job with the smallest remaining processing time.

From this figure we point out three key ideas. First, SRPT has the minimum response time, and improves as the coefficient of variation of job demand increases. In fact, SRPT has been shown to be the optimal uniprocessor scheduling policy [Shra 69]. The SRPT policy requires a priori knowledge of service demands and hence is not a practical policy to implement. Analysis of the policy does add intuition to our understanding of uniprocessor scheduling, and provides an optimal baseline to compare the other policies against. An

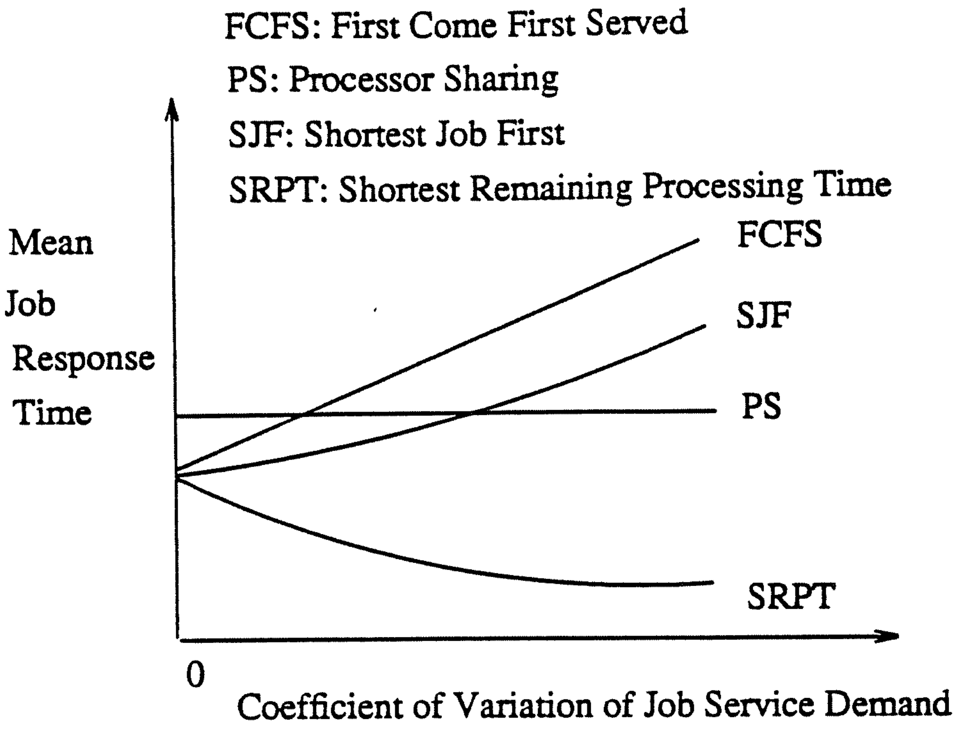


Figure 1.1: Uniprocessor Scheduling Poicies

optimal multiprocessor scheduling policy would be useful for the same reasons.

The second key idea is that FCFS and SJF perform poorly at a high variation in service demand. This is important since real uniprocessor systems have been observed to have a high variation in service demand [SaCh 81]. FCFS performs poorly because it makes no attempt to give small jobs good service. Although SJF does schedule the job with the current smallest demand, it is nonpreemptive. Once a job with a large service demand begins execution it can not be preempted, hence holding back jobs with smaller demands that may arrive after the large job begins service. We expect that multiprocessor scheduling policies that are nonpreemptive and/or do not take into consideration a job's demand will also perform poorly.

Third, PS is an attractive policy because of the following four properties [Klei 76]:

1) Mean job response time is insensitive to the coefficient of variation of job demand.

2) Job response time is linear to service demand.

3) Mean job response time is independent of the service time distribution.

4) The ratio of waiting time to service time is constant for all jobs. Hence, there is no way to "cheat" the system by breaking up a big job into several smaller jobs.

These properties can be attributed to the fact that each job in the system is allocated an equal share of processing power. The PS policy seems to be a good choice for uniprocessors. Since providing equal allocation per job for uniprocessors results in such good behavior, we conjecture that it may be desirable to allocate processing power equally per job in a multiprocessor.

1.5. Previous Multiprocessor Scheduling Research

In this section we present related research in multiprocessor scheduling conducted prior to the research presented in this thesis. Each of the studies addresses one of the three characteristics of a multiprocessor scheduling policy.

A comparison study of six policies was done by Majumdar, Eager and Bunt [MaEB 88] [Maju 88]. This is the only other work we know of that addresses the importance of how processing power is allocated to the jobs in the system. The six policies studied were: first come first served, preemptive and nonpreemptive smallest cumulative demand first, preemptive and nonpreemptive smallest number of processes first, and round robin process. Non-preemptive shortest demand first gives highest priority to jobs with the smallest unscheduled cumulative demand. Preemptive shortest demand first gives highest priority to jobs with the smallest remaining cumulative demand. Non-preemptive smallest number of processes first gives highest priority to jobs with the smallest number of unscheduled processes. Preemptive smallest number of processes first gives highest priority to jobs with the smallest number of processes including processes in service. Majumdar et. al. found that the smallest demand and smallest number of processes first policies performed better than round robin process, which in turn performed better than first come first served. They drew these conclusions for workloads where total job demand is independent of the number of processes per job, and also for workloads where the job demand is positively correlated with the number of processes per job.

We have confirmed that for their workloads the smallest demand first policy has a lower mean job response time than the other policies. Contrary to their results, we find that polices that give preference to jobs with the smallest number of processes perform poorly when job demand is independent of the number of processes per job. This discrepancy is the result of a subtle programming error in their workload generator [MaEa 89]. We also find that under what we hypothesize may be more realistic workload assumptions, the smallest number of processes first policies perform worse than the round robin policy even for the correlated workload. Their study also presented tables showing the frequency of preemptions for the policies, but did not consider what possible overheads per preemption might be, or the significance of the preemption frequency. This work was the starting point for our study. We include all of the policies they studied in our studies except the nonpreemptive smallest demand first policy.

Another area of related work that has been studied is how a multiprocessor scheduling policy should schedule processes so as to support inter-process synchronization. Two approaches have been proposed. An early approach is to maximize the number of processes from the same job running at the same time. The idea is called coscheduling and specific policies to achieve coscheduling were proposed by Ousterhout [Oust 82]. If all processes are not scheduled at the same time, a process requesting synchronization with a descheduled process must wait until the descheduled process is rescheduled. If all processes are scheduled at the same time this waiting time is eliminated. The policies as proposed by Ousterhout have a high degree of coscheduling. We modify the best policy proposed by Ousterhout, "undivided", and include it in our studies.

A more recent approach to support inter-process synchronization is to have a two-level scheduler. The system level scheduler is responsible for allocating processors to jobs while the application level scheduler is responsible for deciding which processes should be run on the processors currently allocated to the job. It was shown that by using threads parallel programs can quickly deschedule and reschedule processes with little overhead [Doep 87] [BeLL 88] [AnLL 89]. This ability to quickly change which process a job has running allows a job to ensure that it is currently running the most important processes.

When using a two-level scheduler a job is normally composed of "threads" [Doep 87] [BeLL 88] [AnLL 89]. Threads are also known as "light weight" processes. Threads are the execution of code, including a program counter and stack of activation records, but not the rest of the information normally associated with a process. All threads of a job share the same address space. Since threads have much less information associated with them, scheduling and descheduling of threads can be accomplished two orders of magnitude faster than scheduling and descheduling processes [AnLL 89]. For simplicity of discussion, we will refer to threads as processes for the remainder of the thesis. When we say an application level scheduler exists we mean the there is a two-level scheduler, and that processes (threads) can be scheduled and rescheduled by the job (application). We will assume no overhead for the switching of processes by the application level scheduler. Note that such a two-level scheduler requires significantly more effort to implement and maintain than a single level scheduler. In [ZaEL 88] and [ZaEL 89] Zahorjan et al. found that using application level scheduling can greatly reduce the amount of time wasted spinning in a multiprogrammed environment for barrier synchronization. Two of the policies included in our study assume this two-level scheduling idea.

A third area of less closely related work is how best to statically partion processors among jobs on a multiprocessor. In [EaLZ 89] and [Sevc 89] rules of thumb are suggested for how to determine the best allocation of processors to an arriving job for static or semi-static partitioning. One of the contributions in the paper by Eager et al. is the suggestion that average parallelism be used as a rule of thumb for deciding the number of processors that should be allocated to a job in static partitioning. Sevcik proposes using variance of parallelism, minimum parallelism, maximum parallelism, and system utilization in addition to average parallelism. The resultant response times are significantly better at high loads than when just using average parallelism.

Both of these approaches provide good rules of thumb and add insight to our understanding of static multiprocessor scheduling, but it appears that static policies are not suitable ways to schedule for a general purpose multiprogrammed multiprocessor. One objection is that processors sit idle when a job's parallelism shrinks below (or grows beyond) the number of processors allocated to the job. A second objection is that in order for the proposed policies to be implemented, we need to know the characterization parameters (average, maximum, minimum, and variance of parallelism). In a general purpose system where applications are being developed and tested, or execution time is data dependent, it is not reasonable to assume that this information is known.

1.6. Concurrent Multiprocessor Scheduling Research

In this section we discuss related multiprocessor research conducted concurrently with the research presented in this thesis. Once again, each study addresses one of the multiprocessor scheduling policy characteristics.

The issue of supporting inter-process synchronization has been further addressed by Seager and Stichnoth [SeSt 89]. In this paper they compare three different scheduling policies by simulating an 8 processor system. They assume the parallel jobs have frequent barrier synchronization. Upon reaching a barrier a process will spin for a certain amount of time. If all of the other processes from the job do not reach the barrier within that time, the spinning process blocks and relinquishes the processor. The process remains blocked until all of the job's processes reach the barrier. The policies they compare are called dog-eat-dog, family, and gang. Dog-eat-dog is simple round robin scheduling on a shared process queue. Family is a variation that achieves slightly better coscheduling than dog-eat-dog. Gang scheduling is a policy similar in spirit to Ousterhout's coscheduling policies. These policies do not assume the presence of a thread package to allow for changing processes as barrier points are reached. The lack of an application level scheduler coupled with the fact that the workloads studied assume frequent barriers results in a high degree of spinning and descheduling of processes. The study found that gang scheduling performs much better than the other two policies as a result of less spinning and blocking. The study further demonstrates the importance of coscheduling in an environment with frequent spinning barrier synchronization. We include a round robin policy (dog-eat-dog) in our studies. The other two policies are not included in our study because they are variations of the Ousterhout coscheduling policy that is included.

Another issue recently addressed is whether static or dynamic allocation results in better performance [ZaMc 90]. Zahorjan and McCann propose a dynamic partitioning policy that spatially partitions the processors among the jobs. The number of processors allocated to each job may change when a new job arrives to the system or the parallelism of the job changes. They compare their dynamic partitioning policy to two static partitioning policies. They found dynamic partitioning to be superior to static partitioning over a wide range of processor preemption overhead values. Their study confirms our intuition about the poor potential performance of static partitioning disciplines.

Another issue recently addressed is how the existence of processor caches may affect the performance of multiprocessor scheduling policies. The presence of processor caches may change the relative performance of scheduling policies due to different cache hit ratios for the different policies. When a process is preempted from a processor and then later rescheduled on another processor the process must rebuild its cache entries. Even if the process is rescheduled on the same processor, a portion of the process' entries may have been removed by intervening processes.

Tucker and Gupta proposed a dynamic partitioning policy to minimize cache miss penalty [TuGu 89]. Their policy allocates a subset of the processors to a job upon job arrival. The allocation changes as jobs enter and depart the system so that each job gets an equal share of the processors. They use application level scheduling to determine which of the job's processes are scheduled on the processors allocated to the job. By keeping processes on a processor for as long as possible they minimize cache misses needed to rebuild a process' cache working set upon the rescheduling of a process. They implemented and compared their dynamic-partitioning policy to a simple round robin process policy. Their experiments found the performance of the dynamic-partitioning to be superior to the round robin process policy for the workloads they studied. There are three reasons that may have caused their scheduling policy to perform better than the round robin process scheduling policy. Their policy gives an equal share of processing power to each job in the system, it minimizes cache misses by keeping processes on a processor as long as possible, and it minimizes spinning as a result of application level scheduling. A determination of how much each of these three factors contributes to the improvement in performance was not included. Note that the dynamic partitioning policy proposed by Zahorjan and McCann also minimizes cache misses.

The importance of processor caches has also been addressed by Squillante and Lazowska [SqLa 90]. They consider several policies that attempt to reschedule a process on the same processor from which the process was previously preempted. They found that when a processor needs a process, choosing the first process in the shared ready queue which last ran on that processor results in higher throughput than simply taking the first process from the queue. They found that at high system loads accompanied by large cache reload overhead, up to 60% improvements in throughput occur. At the operating points where this large improvement is seen, half of the time a process spends on a processor is spent rebuilding the process' cache entries. Squillante and Lazowska propose other slightly more sophisticated policies that exhibit slightly better performance. None of the policies proposed allocate an equal share of processing power to each job in the system. Instead, some policies allocate an equal share of processing power to each process. In some cases the policies favor specific processes.

Due to the complexity of modeling the parallelism and the synchronization in the system, analytical models to date have only had limited success. Most models to date have considered only FCFS or PS scheduling and ignored interprocess synchronization. Two recent papers have made progress in modeling more complex scheduling policies and including synchronization [Nels 90] [NeTo 90]. Nelson included barrier synchronization in a markov chain model of FCFS multiprocessor scheduling. Nelson and Towsley have considered less restrictive scheduling policies in the absence of inter-process synchronization. Due to the complexity of modeling more general systems, we decided to use simulation as a tool for our studies.

CHAPTER 2

Scheduling Policies

In this section we define the twelve scheduling policies studied, and consider the expected behavior of each. We include each of the policies in order to conduct a comprehensive study of general purpose multiprogrammed multiprocessor scheduling policies. The policies included in our studies are chosen either because they have been proposed as addressing one of the three characteristics of a multiprocessor scheduling policy enumerated in chapter 1, or because they add insight into understanding the importance of the three characteristics. Some of the policies studied address two or all three of the multiprocessor scheduling policy characteristics.

In section 2.1 we define the policies. In section 2.2 we discuss the optimal multiprocessor scheduling policy. In section 2.3 we explain why we chose to include each policy. In section 2.4 we discuss how we well we expect the policies to address the three characteristic of a scheduling policy.

2.1. Scheduling Policy Definitions

In this section we describe the policies studied. We first describe the policies proposed previously, then two policies recently proposed, and finally four policies we have proposed for study.

2.1.1. Previously Proposed Policies

- First Come First Served (FCFS) : When a job arrives, its processes are placed at the end of the shared process queue. When a processor becomes idle the process at the head of the queue is scheduled and run to completion.
- Smallest Number of Processes First (SNPF) and Preemptive Smallest Number of Processes First (PSNPF): For SNPF, highest priority is given to processes from jobs with the smallest number of unscheduled processes. Jobs of equal priority are scheduled FCFS. When a processor becomes idle, the first process from the queue is scheduled

and run to completion. For PSNPF highest priority is given to jobs with the smallest number of incomplete processes. An arriving job with a smaller number of processes than an executing job will preempt processes belonging to the scheduled job.

- Preemptive Smallest Cumulative Demand First (**PSCDF**) : Highest priority is given to jobs with the smallest remaining cumulative service demand. An arriving job with a smaller demand will preempt processes belonging to scheduled jobs with the largest demand.
- Coscheduling (**Cosched**) : There exists a linked list of processes. When a job arrives its processes are appended to the end of the list. When a process completes it is removed from the list. Scheduling is done by moving a window of length equal to the number of processors over the linked list. Each process in the window gets one quantum of service on a processor. At the end of the quantum, the window is moved down the linked list until the first slot of the window is over the first process of a job that was not completely coscheduled in the previous quantum. When a process within the window is not runnable (blocked for I/O), the window is extended by one process and the non-runnable process is not scheduled. The quantum ends for each processor at the same time. This policy is similar to Ousterhout's undivided policy. We use a linked list of processes to eliminate some of the problems associated with filling the holes in the array structure proposed by Ousterhout. Note, this policy requires special hardware to cause all processors to end their quanta simultaneously.
- Round Robin Process (**RRprocess**) : When a job arrives each of the processes are placed at the end of the shared process queue. A round robin scheduling policy is invoked on the process queue.

2.1.2. Recently Proposed Policies

- Equal Allocation Dynamic Partitioning (**EqualDP**) : This policy was proposed and implemented by Tucker and Gupta. Their goal was to minimize context switching so that less time is spent rebuilding processor caches. Each job is dynamically allocated an equal fraction of the processors, except that no job is allocated more processors than it has runnable processes. Thus, if a machine has 20 processors and three jobs with 4, 10 and 20 runnable processes each, the first job would be allocated 4 processors and the other two would be allocated 8 processors each. The dynamic acquiring and releasing of processors requires coordination between the system scheduler and the application processes, as described in [TuGu 89]. In their paper, Tucker and Gupta state that if the processes frequently reach states where they can safely suspend, then the actual number of processors a job is using will be very close to the allocated number. In our experiments, we assume the ideal case where the number of running processes in a job changes instantly whenever the allocations change. We also assume that when there are more jobs in the system than processors the extra jobs are held back in a "load queue". When a job leaves the system another job from the load queue is allowed into the system. We made this assumption in keeping with the idea that processes remain on a processor for as long as possible to minimize cache misses. As a result each job is always allocated at least one processor once it starts executing. If, instead, we assume all jobs in the system get one runnable process, the policy would be forced to time slice between the jobs. In this case many of the cache benefits of dynamic-partitioning would be lost.

- Unequal Allocation Dynamic Partitioning (**UnequalDP**) : This policy was proposed by Zahorjan and McCann [ZaMc 90]. To describe the policy we describe the actions taken on job arrival and process departure. On job arrival the following actions are taken: (1) If there are idle processors allocate them to the job. If the job needs fewer processors than are idle, just allocate as many as are needed. (2) If there are no idle processors and one or more jobs have 2 or more processors allocated, then take one of the processors away from one of the jobs with two or more processors. We assume the most recently arrived job with two or more processors allocated to it relinquishes one of its processors. (3) If no processors are idle and no job has two or more processors than the new job must wait until a scheduled job departs the system. When a processes finishes, the following actions are taken: (1) If the job has other processes still not scheduled, schedule one of them on the processor. (2) If the job has no unscheduled processes allocate the processor to the first job, based on arrival time, with unallocated processes.

2.1.3. New Policies

- Round Robin Job (**RRjob**) : Instead of a shared process queue there is a shared job queue. Each entry in the job queue has a queue holding its own processes. Scheduling is done round robin on the jobs. Each time a job comes to the front of a queue the job receives P quanta of size q, where P equals the number of processors in the system. If a job has fewer than P processes, each process gets a quantum of size $\frac{P}{N} \times q$, where N is the number of processes in the job's process queue. If a job has greater than P processes there are two choices. The first is to run P processes for one quantum each. Processes are chosen round robin from the job's process queue. The second choice is to give a quantum of size $\frac{P}{N} \times q$ to each process. This second choice has higher scheduling overhead. All of our studies assume P processes are scheduled for one quantum each.

- Foreground-Background First Come First Served (**FB-FCFS**) : We assume two queues, a high priority queue and a low priority queue. When a job arrives, its processes are initially placed in the high priority queue. When a process from the high priority queue is scheduled it executes for τ units. After τ units the process is moved to the low priority queue. Processes in the high priority queue have preemptive priority over processes in the low priority queue. Once a process from the low priority queue is scheduled it remains in service until it finishes or is preempted by a new arrival to the high priority queue. Processes from a queue are served FCFS.

- Foreground-Background Preemptive Smallest Number of Processes First **(FB-PSNPF)** : Same as FB-FCFS except processes of the same queue are scheduled PSNPF.

- Foreground-Background Round Robin Job **(FB-RRjob)** : Same of FB-FCFS except jobs of the same queue are scheduled RRjob. Once a process has received τ units in the high priority queue it is moved to the low priority queue. When a job comes to the head of a queue, the quantum size for the job is determined by the number of processes the job has in that queue. Note, another modification not considered is to have the job moved to the lower priority queue after the job has received some number τ units of service.

2.2. Optimal Discipline for Multiprocessor Scheduling

Unlike the uniprocessor case, PSCDF is not the optimal policy for multiprocessor scheduling. To see this consider the following simple counter example. A machine has three processors. There are two jobs. Job A has one process requiring 5 units and job B has 4 processes each requiring 1 unit. If jobs are scheduled PSCDF the mean response time is $\frac{(6 + 2)}{2}$.

time	Proc 1	Proc 2	Proc 3
1	B	B	B
2	B	A	
3		A	
4		A	
5		A	
6		A	

If we schedule jobs in the following order the mean response time is $\frac{(5 + 2)}{2}$.

time	Proc 1	Proc 2	Proc 3
1	A	B	B
2	A	B	B
3	A		
4	A		
5	A		

In [GaJo 79] a restricted problem of finding the optimal policy for a two processor system with single tasking jobs is shown to be NP complete. Hence, simulating the optimal scheduling policy is not practical since it can not be done in polynomial time.

2.3. Reasons for Inclusion in Study

Only a subset of the twelve policies have been studied before in a comparative study. We include the FCFS policy because it is a simple policy and it provides a baseline for comparison of other multiprocessor scheduling policies. We include the SNPF and PSNPF policies since they may approximate shortest job first scheduling if job demand is correlated with the number of processes per job. In the absence of job demand knowledge, there are two methods employed in uniprocessor scheduling provide good service to the short jobs. The first way is to give each job in the system a share of the processing power. By giving each job a share of the processing power, jobs with a small demand are not blocked in the queue behind jobs with a large demand. In the uniprocessor environment this is done temporally by round robin scheduling. We include five multiprocessor scheduling policies that use the approach of allocating a share of the processing power to each job in the system: Cosched, RRprocess, RRjob, EqualDP, and UnequalDP. The policies RRprocess, Cosched, and RRjob temporally divide processing power by time slicing. The policies EqualDP and UnequalDP spatially divide the processing power by allocating processors to jobs in the system. The second uniprocessor method to provide good service for short jobs is to have a multi-level queue where jobs decrease in priority as they acquire more processing time. We include the three foreground-background policies, FB-FCFS, FB-PSNPF, and FB-RRjob, to see if this adaptive approach improves mean response time for the multiprocessor environment. We include the policy PSCDF since shortest remaining processing time first is optimal in the uniprocessor environment, and although not optimal for the multiprocessor case, we would expect the policy to perform well.

In addition to the insight gained about division of processing power, we include some of the policies because they address the issue of support for inter-process synchronization. The policy Cosched was proposed to facilitate message passing. The policies EqualDP and UnequalDP include two level schedulers which have been shown to improve performance in the presence of synchronization.

We also include some of the polices because they were designed to have a low preemption frequency. The policies EqualDP and UnequalDP minimize cache misses by not time slicing.

2.4. Expected Behavior of the Policies

In this section we consider whether and how each of the twelve policies addresses each of the three characteristics of a scheduling policy enumerated in chapter 1. We first consider the division of processing power among the jobs in the system. From uniprocessor results we would expect good performance for multiprocessor scheduling policies to result from either giving preference to jobs that have a small demand, or, in the absence of knowledge of job demand, giving a share of the processing power to each job in the system. In addition, we would expect policies that give a fraction of processing power to each job to not be adversely affected by a high variation in job demand. Conversely, we would expect FCFS to perform poorly when there is a large coefficient of variation of job demand just as FCFS performs poorly in the uniprocessor case. The policies SNPF and PSNPF are not as easy to predict. In earlier work the policies were shown to exhibit good performance. We would expect the policies might perform well when job demand is correlated with the number of processes per job, but we would not expect the policies to perform well when job demand is not correlated with the number of processes per job. Like FCFS these two policies only allocate processing power to a subset of the jobs in the system, hence they may not perform well when there is a high coefficient of variation of job demand. The policies RRprocess, RRjob, Cosched, EqualDP, and UnequalDP all give some fraction of processing power to each job in the system under most conditions, hence we would expect these policies to perform well. The policies RRjob and EqualDP give an equal share of processing power to each job whereas the other three do not. We wish to determine whether this equal allocation results in better performance. The policies FB-FCFS and FB-PSNPF only schedule a subset of the jobs, but small jobs do not suffer since jobs are moved to the lower priority queue after τ units of execution. We would expect FB-FCFS and FB-PSNPF to perform well as long as τ is chosen appropriately. Whether the performance is comparable to the policies that allocate a fraction of processing power to each job needs to be determined. The policy FB-RRjob moves large jobs to a lower priority in addition to giving each job within a queue an equal share of the processing power. How much the foreground-background scheduling improves RRjob needs to be determined.

The second important characteristic of a multiprogrammed multiprocessor scheduling policy is how well the policy supports inter-process synchronization. We consider both spin-lock and barrier synchronization. As stated in chapter 1, we consider two approaches for supporting synchronization: coscheduling and application level scheduling. Of the policies studied, only EqualDP and UnequalDP assume application level scheduling. None of the other policies assume application level scheduling, but we consider the effect of including application level scheduling for RRjob and Cosched when there is barrier synchronization.

For spin-lock synchronization, spinning occurs when a processes requests a lock and the lock is not available. The lock may not be available for two reasons, there is a currently scheduled processes holding the lock, or there is a descheduled process holding the lock. We would expect the latter case to cause the processes waiting for the lock to spin for greater periods of time. All policies are susceptible to spinning due to the lock being held by a currently scheduled process. The policies FCFS and SNPF are immune from the latter case since they do not deschedule processes. Hence we would not expect FCFS and SNPF to be as adversely affected by spin-lock synchronization as policies that do descheduled processes. If we assume application level scheduling, processes holding locks are not descheduled. Hence, we would expect EqualDP and UnequalDP to perform very well in the presence of spin-lock synchronization. All other policies may deschedule processes and hence may deschedule processes holding a lock. The amount of time a process holding a lock is descheduled affects the performance of the policy. RRprocess and RRjob may deschedule processes holding locks, but a process holding a lock will be rescheduled as soon as the process comes to the front of the queue again. PSNPF and PSCDF on the other hand may deschedule processes holding a lock for long periods of time. Although the policy Cosched does deschedule processes, it usually deschedules all processes from the same job at the same time so that if a process holding a lock is descheduled, it is likely that the lock will not be requested until the processes holding the lock is rescheduled. We would expect as a result that Cosched will not be as adversely affected by spin-lock synchronization as RRprocess and RRjob.

For barrier synchronization spinning occurs when processes reach the barrier. Once a process reaches a barrier it spins until it is either descheduled or all other processes from the job reach the barrier. Once all processes from the job reach the barrier, all processes can proceed. If we assume each process requires about the same amount of time to reach the barrier, we would expect that having all processes from a job scheduled at the same time would result in less spinning. As a result we would expect Cosched to be less affected by barrier synchronization than the other policies. Policies such as PSCDF, FCFS, SNPF, PSNPF, FB-FCFS, and FB-PSNPF that may schedule only a fraction of a job's processes will likely result in poor performance since upon reaching a barrier processes will spin until all other processes from the job get scheduled and reach the barrier. In addition, FCFS and SNPF will deadlock if a job has more processes than there are processors, and the PSNPF policy will never complete jobs that have more processes than there are processors. If there is application level scheduling, processes that reach a barrier can be swapped with other processes from the job that have not yet reached the barrier. As a result, we would expect barrier synchronization to not affect performance much in the presence of an application level scheduler.

The third important characteristic of a scheduling policy is the preemption frequency. Preemptions cause overhead due to the context switch and the rebuilding of process cache entries after the process is rescheduled. The policies FCFS and SNPF never preempt processes once scheduled. The policies PSCDF, PSNPF, EqualDP and UnequalDP preempt processes only when higher priority jobs arrive to the system resulting in a low preemption overhead. The polices RRprocess, Cosched, and RRjob all time slice, hence we would expect these three policies will pay a performance penalty for these extra preemptions. The foreground-background policies will preempt a process at most once more than their non-foreground-background equivalents.

We present a summary of whether the twelve policies address the three scheduling policy characteristics in table 2.1. Table 2.2 contains a concise lest of the policies and their acronyms.

Table 2.1: Summary of the Expected Behavior of the Policies

Policy	Good Service for Short Jobs	Spin-lock Synchronization	Barrier Synchronization	Preemption Overhead
FCFS	no	yes	no	yes
PSCDF	yes	no	no	yes
SNPF	yes	yes	no	yes
PSNPF	yes	no	no	yes
RRprocess	yes	somewhat	somewhat	no
Cosched	yes	yes	yes	no
RRjob	yes	somewhat	somewhat	no
EqualDP	yes	yes	yes	yes
UnequalDP	yes	yes	yes	yes
FB-FCFS	yes	no	no	yes
FB-PSNPF	yes	no	no	yes
FB-RRjob	yes	somewhat	somewhat	no

Table 2.2: Scheduling Policy Acronyms

Acronym	Policy
FCFS	First Come First Served
SNPF	Smallest Number of Processes First
PSNPF	Preemptive Smallest Number of Processes First
PSCDF	Preemptive Smallest Cumulative Demand First
RRprocess	Round Robin Process
RRjob	Round Robin Job
Cosched	Coscheduling
EqualDP	Equal Dynamic Partitioning
UnequalDP	Unequal Dynamic Partitioning
FB-FCFS	Foreground Background FCFS
FB-PSNPF	Foreground Background PSNPF
FB-RRjob	Foreground Background RRjob

CHAPTER 3

Multiprocessor and Workload Models

In this section we describe our models of the multiprocessor and the parallel jobs that compete for the processors. Our goal is to keep the models simple, yet still capture the essence of the system. There are two reasons for keeping models simple. One is that model solutions can be obtained in less time. The second, and more important reason, is that keeping the models simple makes it easier to interpret the results. In section 3.1 we present the job structure model. In section 3.2 we present two multiprocessor models. In section 3.3 we describe the job characteristics, including distribution of number of processes per job, and distribution of job and process demand. In section 3.4 we describe the different types of inter-process synchronization. In section 3.5 we conclude with a summary of model parameters.

3.1. Job Structure

Each job in all of our workload models has the simple structure shown in figure 3.1. That is, the job forks into some number of processes. The processes are either assumed to be independent or to have some sort of synchronization behavior as explained in section 3.4. Once all of the processes complete, the job is finished. Similar structures have been studied in [NeTT 87] [MaEB 88] [ToRS 90]. This simple model captures the essence of the system, allowing us to gain initial insight and understanding of the issues pertaining to multiprogrammed multiprocessor scheduling.

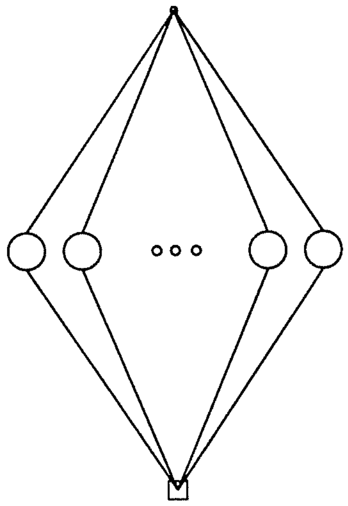


Figure 3.1: Job Structure

3.2. Multiprocessor Model We have studied two queueing models of the multiprocessor. The two differ only in that one is an open system and the other is a closed system. The open system model is show in figure 3.2. The servers represent the processors. We assume there are 20 processors. Jobs arrive to the system according to a Poisson process. Upon entering the system, jobs fork into some number of processes and enter a shared ready queue. The processes then are served by the processors according to the scheduling policy. Processes may be preempted, in which case they return to the shared ready queue. All processes from a job must complete before the job departs the system. In the closed system model there is a fixed population with a think time instead of a Poisson arrival process. Figure 3.3 shows the closed system model.

3.3. Job Characteristics

In this section we describe the job characteristics. In section 3.3.1 we discuss the distribution of the number of processes per job. In section 3.3.2 we discuss the distribution of job demand. In section 3.3.3 we discuss how the job demand is divided between the job's processes.

3.3.1. Distribution of Number of Processes Per Job

In this section we describe how the number of processes per job is determined. We have included two models of the number of processes per job. The first model we call the *hyperexponential workload* and the second the *geometric-bounded workload.* The hyperexponential

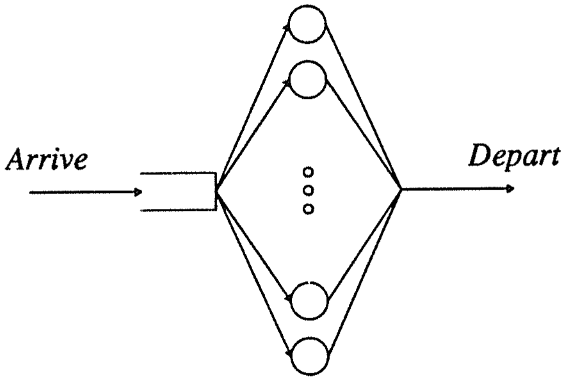


Figure 3.2: Open System Model

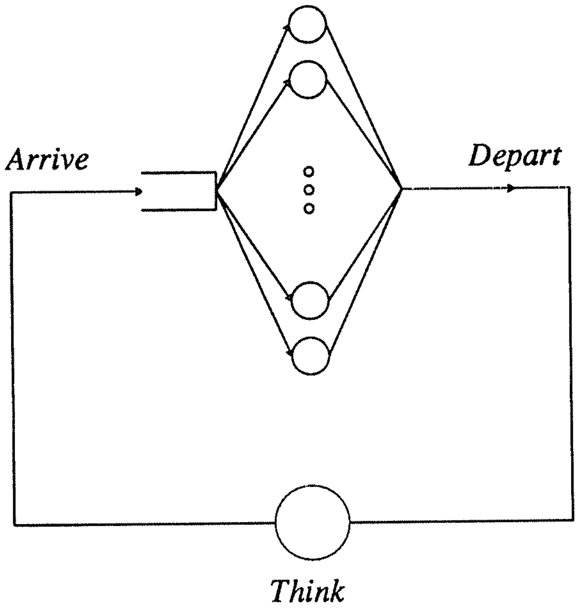


Figure 3.3: Closed System Model

workload was used in [MaEB 88]. We include the hyperexponential workload so we can compare our results with the results presented in that paper. A premise of our work is that the hyperexponential workload may not accurately model workloads of a real system, and that the geometric-bounded workload may be a more realistic representation of actual system workloads. We will emphasize the geometric-bounded workload. We explain our reasons for chosing the geometric-bounded instead of the hyperexponential workload below.

Hyperexponential workload: The number of processes is drawn from a two stage hyperexponential distribution, with 95% of the jobs having the number of processes drawn from an exponential distribution with the small mean and 5% of the jobs having the number of processes drawn from an exponential distribution with the large mean. The input parameters are mean number of processes ($\bar{n}$) and coefficient of variation of number of processes (C_n). Under these assumptions, a distribution with mean 4.0 and coefficient of variation of 5.0 results in 95% of the samples drawn from an exponential distribution with a mean of 0.82 and 5% of the samples drawn from an exponential distribution with mean 64.4. The number of processes is set equal to the ceiling of the number returned from the hyperexponential distribution.

The above model can be used to generate workloads with values for C_n greater than or equal to 1.0. One parameter setting that is used extensively in earlier work is $C_n = 5.0$ and $\bar{n} = 4.0$. Figure 3.4 shows the probability mass function for these parameters. We see that 67% of the jobs are sequential. In addition there is a very long thin tail which is not seen on the graph. Specifically, 4.6% of the jobs have more than 6 processes, 2.0% have more than 60 processes, and 1.0% have more than 100 processes. In a real system with 20 processors it is unlikely that many jobs would have more than 20 processes. We hypothesize that a real workload might have a larger fraction of jobs with small amounts of parallelism, jobs with parallelism equal to the number of processors, and far fewer jobs with parallelism much greater than the number of processors. When C_n equals 1, we feel the distribution of number of processes may still be unrealistic. There would be a reasonable percentage of jobs with small degrees of parallelism, but there would be few jobs with high degrees of parallelism. We thus define the following new model of the number of processes per job.

Geometric-bounded workload: The maximum number of processes per job is equal to the number of processors. Input parameters are the probability that a job has a number of processes equal to the number of processors (P_p), and the mean number of processes for all other jobs ($\bar{n}$). With probability ($1 - P_p$) the job has a number of processes chosen from a truncated geometric distribution with mean $\bar{n}$. The number of processes is set equal to the number of processors if the random number chosen from the geometric distribution exceeds the number of processors. (For a geometric distribution with a mean of 4.0 only 0.3% of the samples exceed 20.) We designate this workload as "bounded" since there is a bound as to the maximum number of processes a job may have. Figure 3.5 shows the probability mass function for the geometric-bounded workload with parameters $P_p = 0.05$ and $\bar{n}$ equal to 4.0 for the remaining ($1 - P_p$) jobs. The size of the spike at 20 can be changed by adjusting P_p and the distribution of the jobs with a smaller number of processes can be changed by adjusting $\bar{n}$.

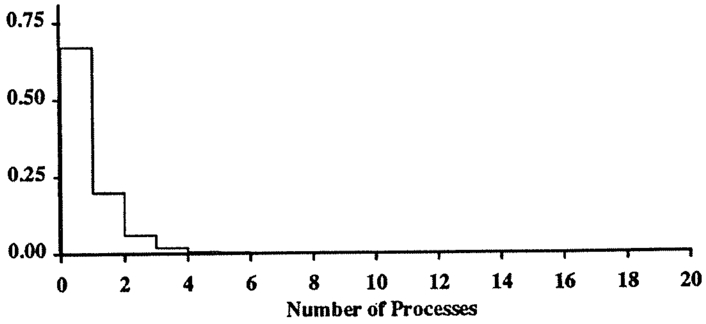


Figure 3.4: Hyperexponential PMF

3.3.2. Distribution of Job Demand

In this section we describe how job demand is determined. We assume the two workloads defined in [MaEB 88]:

Uncorrelated job demands: There is no correlation between the number of processes and the total demand of the job. The input parameters are mean job demand ($\bar{d}$) and the coefficient of variation for the demand (C_d). The job demand is determined from a two stage hyperexponential distribution with 95% of the jobs having the demand chosen from the exponential distribution with the small mean. The input C_d determines the means for the two stages of the distribution.

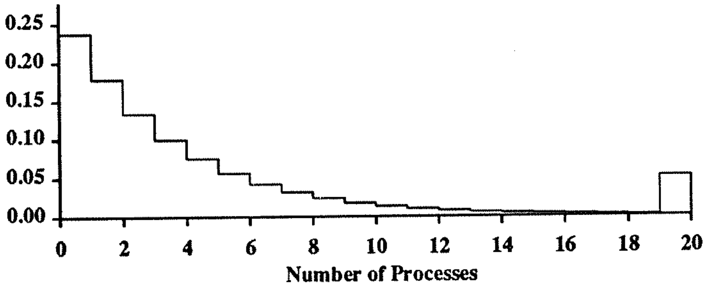


Figure 3.5: Geometric-bounded PMF

Correlated job demands: Mean job demand is linearly correlated with the number of processes per job. Thus, jobs with a large number of processes are likely to have a large demand, whereas jobs with a small number of processes are likely to have a small demand. (Such a linear correlation is partially justified by the arguments presented in [Gust88].) The input parameters are a demand variation parameter (C_v), and a scalar (t). Let the number of processes computed for a job be represented by N. The demand for the job is obtained from a hyperexponential distribution with the mean set equal to ($N \times t$) and coefficient of variation equal to C_v. Note that the input C_v is not equal to the output coefficient of variation of demand, C_d, due to the linear dependence.

3.3.3. Division of Job Demand Amongst Processes

In this section we describe how job demand is divided among the job's processes. We divide job demand among the job's processes equally. Let D equal the job demand and N equal the number of processes a job has. Then process demand simply equals $\frac{D}{N}$.

We also divided demand unequally as in [MeEB 88]. Assume a job has N processes. Generate N, u_1 - u_N, random numbers distributed uniformly between 0 and 1.0. Demand for a particular process, d_j, is:

$$d_j = \frac{D \times u_j}{\sum_{i=1}^{n} u_j}$$

As also observed in [Maju 88], we found that the results for both methods of job demand division were within a few percent of being the same. As a result we have decided to report only results for the equal job division.

3.4. Inter Process Synchronization

In this section we describe the three models of inter process synchronization used in our study. The first is to assume no synchronization. The other two synchronization models we studied are spin-lock synchronization and barrier synchronization. The actions taken upon reaching a synchronization point differ depending on the presence or absence of an application level scheduler. In sections 3.4.1 and 3.4.2 we describe the spin-lock and barrier synchronization models.

3.4.1. Spin-lock Synchronization

Our model is essentially the same as in [ZaLE 89]. We assume there is one shared lock for each job, and all processes within a job contend for the lock. We only consider a single lock per job to keep the model simple. A process requesting a lock that is currently held by another process spins (busy waits) until the lock becomes available. When a lock is released, the next process to acquire the lock is chosen randomly from all running processes waiting for the lock. One common example of spin-lock synchronization in a real system would be a lock needed to gain access to a shared queue. We assume that the lock holding time is deterministic and small, $\frac{1}{100}$ of a quantum. The time between requests to the lock is exponentially distributed. The mean inter-request time is set according to the number of processes in the job, so that the total lock demand per job remains fixed. Total lock demand is defined as (process lock demand) × (number of processes), where process lock demand is the percent of time each process would use the lock if there were no competition. For example, if job lock demand is 100% and the number of processes is 10, then the process lock demand is 10% and the inter-request time is $\frac{9}{100}$ of a quantum. Each time a process releases the lock a new interrequest time is taken from the exponential distribution with this mean. The process does not request the lock again if the time to the next request plus the lock holding time is greater than the remaining service time of the process. If there is no application level scheduler, then a preemptive policy may deschedule a process holding a lock. If there is an application level scheduler, then a process holding a lock is never descheduled unless no other process from the same job is scheduled. In this case, the process holding the lock will be the first process from the job to be rescheduled in the future.

3.4.2. Barrier Synchronization

All processes of a job must reach a synchronization point in their code before any proceed. This point in the code is called a barrier. Once all of the processes in a job have reached the barrier all the processes can proceed.

The actions taken upon reaching a barrier differ depending on whether or not there exists an application level scheduler. If there is not an application level scheduler, then upon reaching a barrier the processes will spin (busy wait) until all the other processes from the job reach the barrier. If there is an application level scheduler, then if upon reaching a barrier the process is not the last to reach the barrier and there exists another process from the same job that is currently descheduled and could be doing productive work, then the first process is descheduled and the later is scheduled in its place. If there are not any descheduled processes from the same job that could be doing productive work, then the process spins until either all other processes from the job reach the barrier, or the process is descheduled by the system level scheduler.

During the execution of a job some number of barriers are achieved. We consider three options for the distribution of the time between barriers.

1) The time between barriers for each process is independent and exponentially distributed.

2) The time between barriers is deterministic.

3) The time between barriers is a set amount of time, t, plus or minus 0.1 × t. In other words, the time between barriers for each process is uniformly distributed between $(t - \frac{t}{10})$ and $(t + \frac{t}{10})$.

The first option results in a large variation in the amount of time for each processes to reach a barrier. Real applications are not likely to have this much variation in the time between barriers. The second option is not realistic since the amount of time for each process to achieve a barrier might vary, either due to differences in number of instructions or due to different execution rates due to cache effects. We hypothesize that the third option is the most realistic and use it in our simulation studies. Note that the third option is the same inter-request time distribution as found in [ZaEL 88] [ZaEL 89].

3.5. Summary of Model Parameters

Table 3.1 summarizes the model parameters.

Table 3.1: Summary of Model Parameters

Parameter	Definition
P_p	Percent of jobs that have number of processes set to 20
$\bar{n}$	Mean number of processes per job
C_n	Coefficient of variation of the number of processes per job
$\bar{d}$	Mean job demand
C_d	Coefficient of variation of the job demand
t	Scalar for the correlated workload
C_v	Input coefficient of variation of job demand for Input the correlated workload

CHAPTER 4

No-Synchronization Results

In this chapter we examine the issue of how a scheduling policy should allocate processing power to the jobs competing for the processors so as to achieve the best performance. There are two basic choices, allocate the processors to a subset of the jobs based on some priority scheme, or give each competing job some fraction of the processing power. If the later is chosen the processing power can either be divided spatially by dynamically partitioning the processors among the jobs as in EqualDP and UnequalDP, or temporally by giving each processes from each job time slices as done in RRprocess, Cosched, and RRjob.

We surmise that allocation of processing power per job is a very important characteristic of a scheduling policy. We study this issue experimentally with workloads in which all processes execute independently. That is, there is no inter-process synchronization. We also do not include any preemption overhead. We omit synchronization and preemption overhead in these initial experiments in order to isolate the impact of processing power allocation among the jobs. We then study the impact of synchronization and preemption overhead in chapters 6 and 7 respectively.

All results in this chapter assume an open system. We choose to use the open system since the same fixed set of input parameter values results in the same system utilization for all policies. This allows us to more easily compare the results from our parametric studies. We will consider the closed system in chapter 5.

We first study nine policies: FCFS, SNPF, PSNPF, RRprocess, RRjob, Cosched, PSCDF, EqualDP, and UnequalDP, for both the uncorrelated and correlated workloads. We look initially at one particular setting of all the input parameter values and then conduct a sensitivity study of each input parameter. From the results of these experiments we gain insight into the impact of how a scheduling policy divides processing power among the jobs in the system. We then consider whether the addition of foreground-background scheduling can benefit the policies by studying the three foreground-background polices FB-FCFS, FB-PSNPF, and FB-RRjob.

Section 4.1 presents and corrects results from previous work. In section 4.2 we consider the impact of workload choice. In section 4.3 we present results for two baseline cases of the geometric-bounded workload. In sections 4.4 we explore the sensitivity of the baseline results to variations in model parameters for both the uncorrelated and correlated workloads. Section 4.4.1 examines the effect of varying the coefficient of variation of job demand, C_d. Section 4.4.2 examines the effect of varying the mean job demand, $\bar{d}$ for the uncorrelated workload and t for the correlated workload. Section 4.4.3 examines the effect of varying input parameter P_p. Section 4.4.4 examines the effect of varying input parameter $\bar{n}$. Section 4.5 examines the effectiveness of foreground-background scheduling. Section 4.6 contains the conclusions from this chapter.

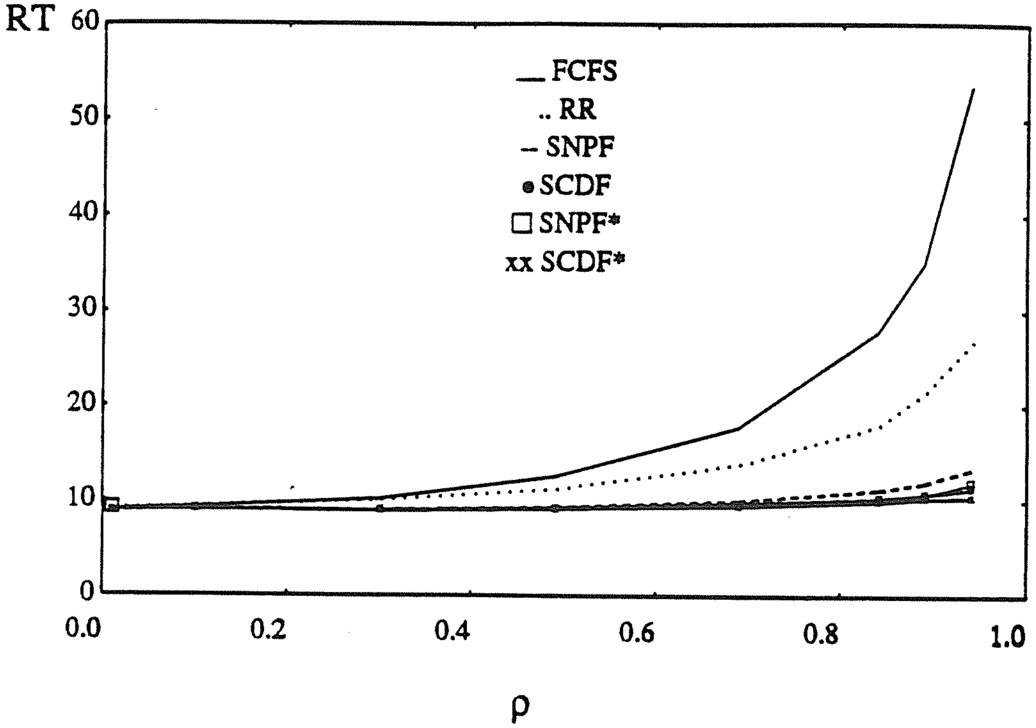
All times are normalized to the length of one quantum in the RRprocess scheduling policy. We include all input parameter values and simulation measured parameter values with each figure. Unless otherwise noted, the figures plot mean response time versus system utilization. The curves are listed in the legend in order of decreasing mean response time. Policies with nearly identical performance are drawn as one curve with multiple labels.

All models have been simulated using the DeNet simulation language [Livn 88]. All results have confidence intervals of 10% or less (usually less than 5%) at a 90 percent confidence level. Confidence intervals are calculated using batch means [Koba 78] with 20 batches per simulation run. Unless otherwise noted, uncorrelated workload experiments have a batch size of 5,000 samples and most correlated workload experiments have a batch size of 7,500 samples.

4.1. Correction to Previous Work

In this section we reexamine and correct results from a previous study. We consider only the uncorrelated workload since our results for the correlated workload qualitatively agree with the earlier study. Figure 4.1 plots mean response time versus system utilization for the uncorrelated hyperexponential workload. This figure is a reproduction of figure 4.3a from [MaEB 88]. We see that FCFS and RRprocess perform poorly as system utilization increases, whereas the SNPF policies perform nearly as well as the SCDF policies. In their study, Majumdar et. al. set a quantum equal to $\frac{1}{10}$ units, so a value of 19.9 for $\bar{d}$ corresponds to 199 quanta.

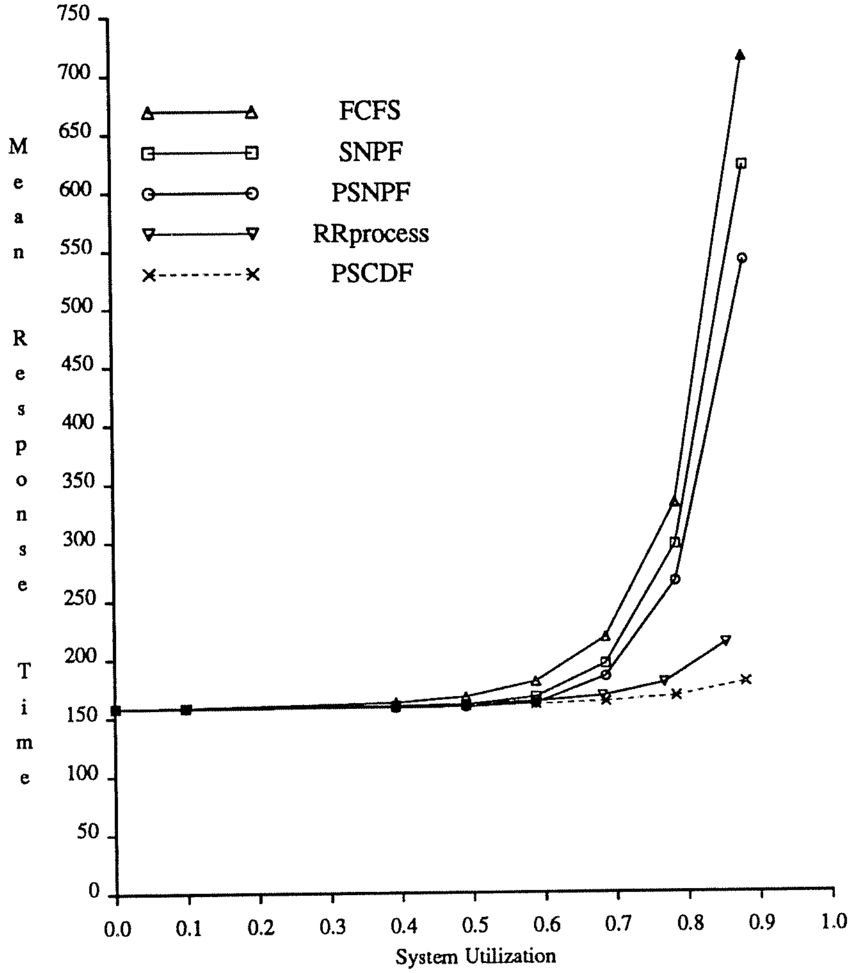
In figure 4.2 we present our results for the uncorrelated hyperexponential workload. We chose input parameter values so that our measured parameter values would be similar to those in figure 4.1. Note that the measured parameter values differ only slightly, and should not be the cause of qualitatively different results. However, we find a marked qualitative difference in our results. Specifically, RRprocess performs well, whereas the SNPF



	$\bar{n}$	C_n	C_d	$\bar{d}$
Output	4.2	4.0	199	4.97

Figure 4.1: Majumdar et. al. Hyperexponential Uncorrelated

policies perform poorly as system utilization increases. The difference in our results and Majumdar et. al.'s results is due to a subtle programming error in Majumdar's workload A generator. This error resulted in a positive correlation between number of processes and job demand [MaEa 89], [MaEB 88], although the workload is supposed to have no correlation in these parameters.



	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	4.0	5.0	200.0	5.0
Output	4.33	4.50	200.0	4.97

Figure 4.2: Correct Hyperexponential Uncorrelated Workload

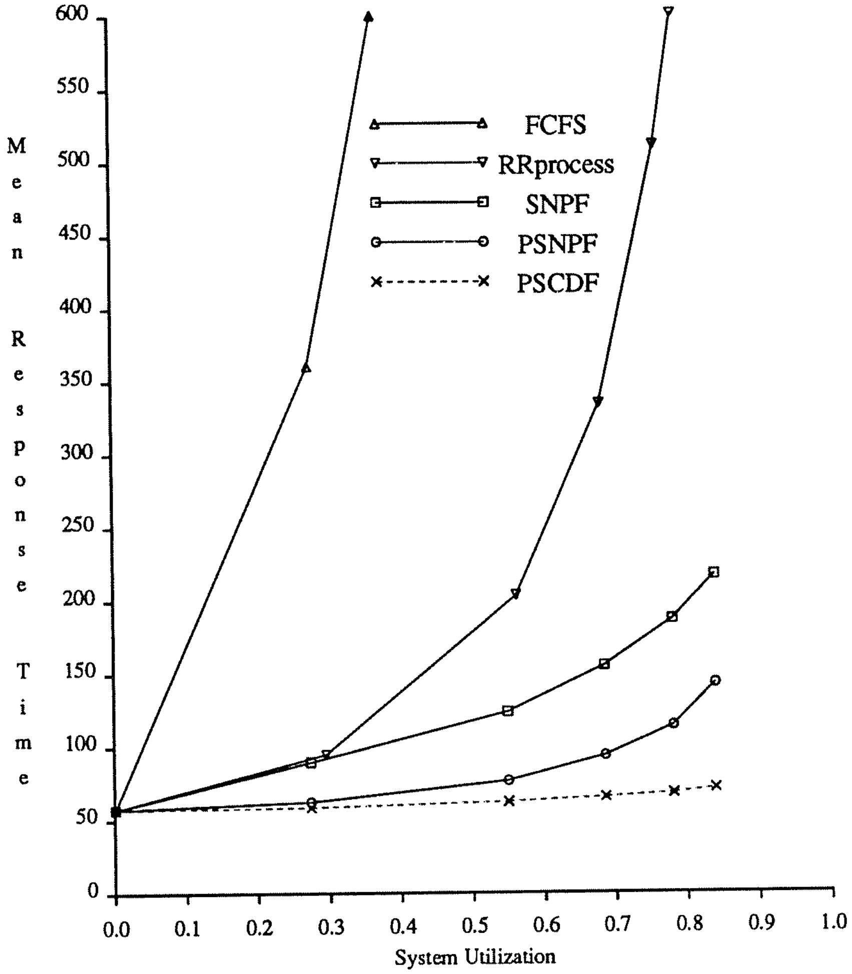
4.2. Impact of Workload Choice

In this section we explore how the choice of workload impacts the performance of the policies. Specifically, we present results from the correlated hyperexponential workload and then compare these results to results obtained from the correlated geometric-bounded workload. We consider only the correlated workload since there is little difference between the hyperexponential and geometric-bounded workloads for the uncorrelated case. After presenting results from these two workloads, we present results from a third workload, the hypergeometric-unbounded workload, to aid in explaining the differences observed between the hyperexponential and geometric-bounded workloads. As explained in chapter 3, we believe the geometric-bounded workload may more accurately represent typical multiprocessor workloads then the hyperexponential workload.

Figure 4.3 plots mean response time versus system utilization for the correlated hyperexponential workload. Our results for the hyperexponential workload agree qualitatively with those in [MaEB 88]. RRprocess performs significantly worse than SNPF, PSNPF, and PSCDF. In RRprocess each process in the system gets an equal share of the processing power, hence jobs with a large number of processes get a larger fraction of processing power than jobs with a small number of processes. This results in RRprocess giving more processing power to jobs that are likely to have a large demand since job demand is correlated with the number of processes. This is exactly the opposite of the goal of uniprocessor shortest time to completion first. As a result, the poor performance of RRprocess is not surprising. SNPF and PSNPF perform comparatively well since they give preference to jobs with a small number of processes. Hence, they give preference to jobs that are likely to have a small job demand. Note the preemptive version (PSNPF) performs considerably better than the non-preemptive version (SNPF). At a utilization of 69% the mean response time of SNPF is 57% larger than the mean response time of PSNPF. This is due to jobs with a large number of processes being scheduled at periods of transient low utilization. Once scheduled, these large jobs are not preempted and slow down small jobs that arrive during the large jobs execution. The policy PSNPF is not prone to this problem since a smaller job preempts scheduled larger jobs. The PSCDF policy performs best, but is not practical to implement since it requires a priori job demand knowledge.

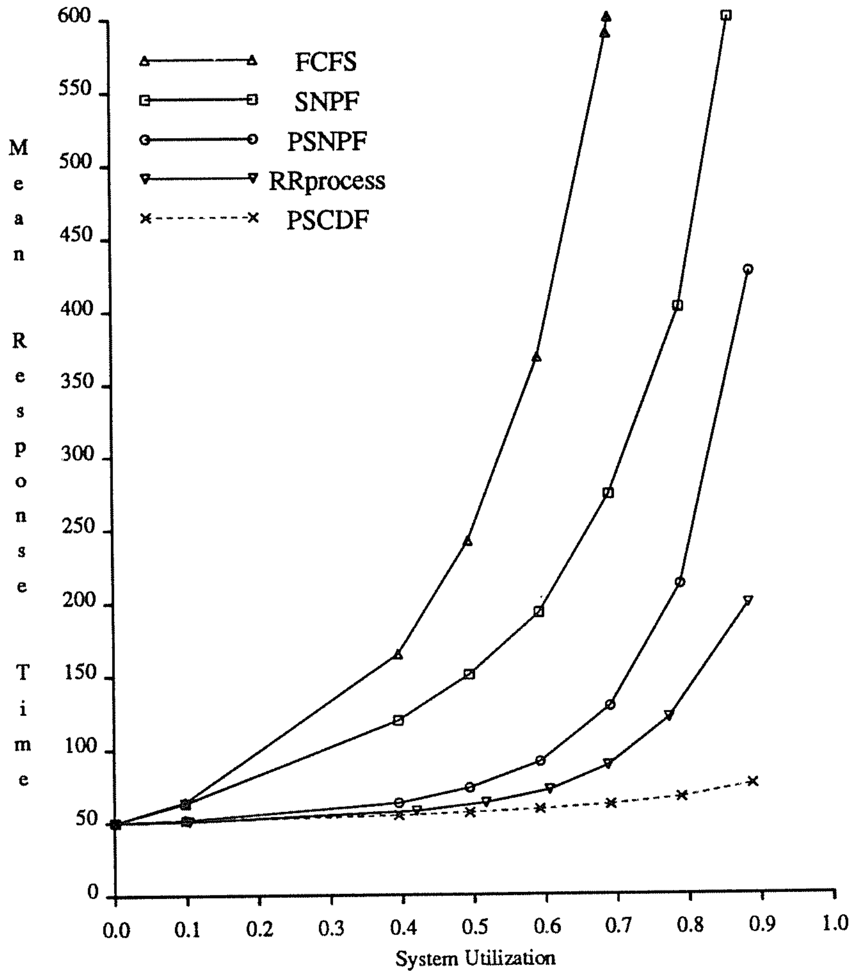
Figure 4.4 plots mean response time versus system utilization for the correlated geometric-bounded workload. The input parameters were chosen to correspond with figure 4.3. The parameter P_p equals 0.05 since 5% of the jobs in the hyperexponential workload have the number of processes drawn from the stage of the hyperexponential distribution with the large mean. Figures 3.4 and 3.5 show the probability mass functions for these hyperexponential and geometric-bounded workloads respectively.

RRprocess performs better than SNPF and as well as PSNPF for the geometric-bounded workload. Considering the distribution of the number of processes resulting from the hyperexponential workload helps to explain why RRprocess performs better for the geometric-bounded workload than for the hyperexponential workload. A C_n of 5, with 95% of the samples drawn from the first stage of the distributions results in the first stage of the distribution having a mean of 0.82, and the second stage of the distribution having a mean of 64. This results in 67% of the jobs being sequential, over 2% with more than 60 processes, and over 1% with more than 100 processes (See figure 3.4). Jobs with a small number of processes receive a smaller fraction of processing power relative to the



	$\bar{n}$	C_n	$\bar{d}$	C_d	C_v	t
Input	4.0	5.0			3.0	50.0
Output	4.55	4.32	219.9	12.70		

Figure 4.3: Hyperexponential Correlated Workload



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	C_v	t
Input	0.05	4.0				5.0	50.0
Output		4.79	1.00	237.5	6.97		

Figure 4.4: Geometric-Bounded Correlated Workload, $P_p = 0.05$

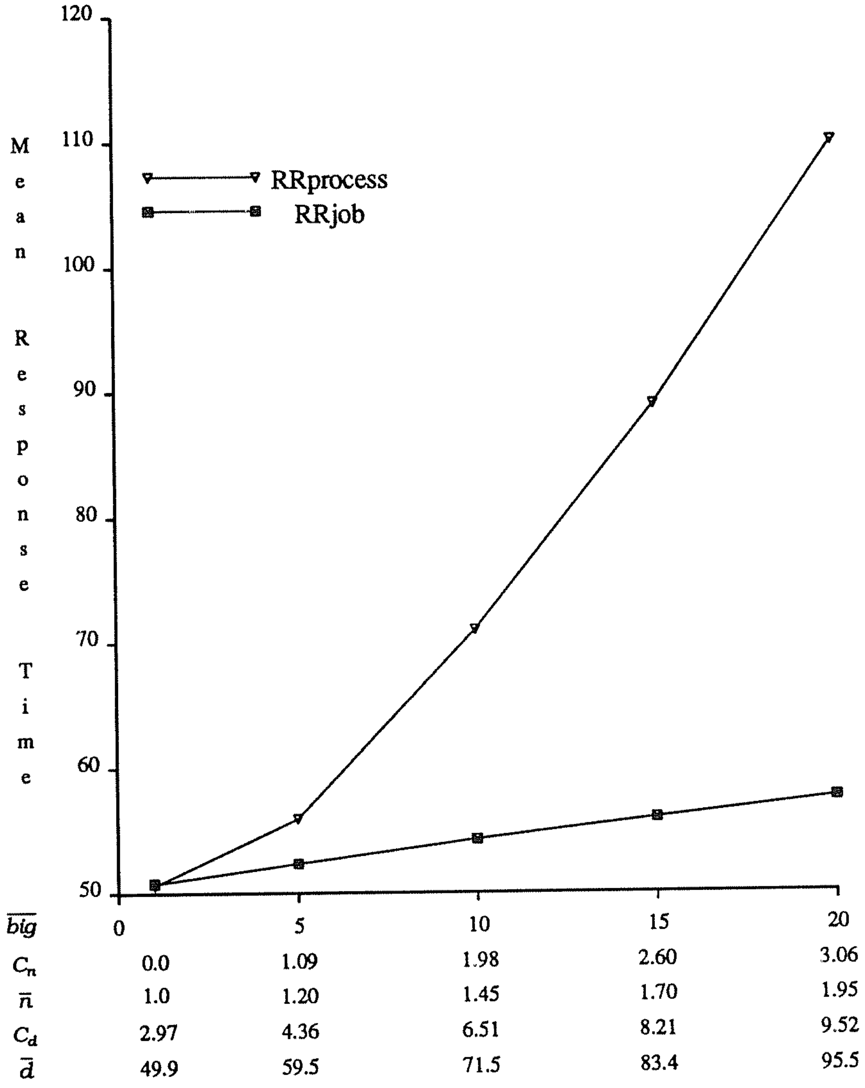
processing power allocated to jobs with a large number of processes. The geometric-bounded workload has much less variation in the number of processes. The measured C_n for the geometric-bounded workload equals 1.0, whereas the measure C_n for the hyperexponential workload equals 4.32. As a result, jobs with a small number of processes receive a larger fraction of processing power for the geometric-bounded workload than for the hyperexponential workload. Note that the measured C_d of figure 4.4 is smaller than the measured C_d in figure 4.3. We show in section 4.4.1 that RRprocess is resilient to variation in job demand, hence the difference between figures 4.4 and 4.3 can be primarily attributed to the difference in C_n.

To further demonstrate how C_n affects the performance of RRprocess, we consider the correlated hypergeometric-unbounded workload. For this workload $(1.0 - P_p)$ of the jobs have the number of processes drawn from a geometric distribution with mean $\bar{n}$, and P_p of the jobs have the number of processes drawn from a geometric distribution with mean $\overline{big}$. We denote the distribution as *unbounded* since we do not truncate the distribution. (In actuality we truncate the distribution at 500 when we generate the geometric random variable.)

In figure 4.5 the coefficient of variation of the number of processes is varied by setting $\bar{n}$ equal to 1, and varying $\overline{big}$. We consider the policy RRprocess since this policy was affected most by the change from the hyperexponential workload to the geometric-bounded, and we include the policy RRjob to illustrate one difference between policies which allocate processing power proportional to the number of processes per job and policies that allocate processing power equally per job. The parameter $\overline{big}$ is varied from 1 to 20. As $\overline{big}$ increases, the measured values of C_n, $\bar{n}$, C_d, and $\bar{d}$ increase. The measured values of these parameters are included on the x-axis. As $\overline{big}$ is increased the mean job response time of RRprocess rises dramatically while the mean response time of RRjob only increases slightly. At $\overline{big} = 20.0$ RRprocess has a mean job response time 91% larger than RRjob. Hence, an increase in C_n coupled with correlated job demands causes policies that allocate equally per process (RRprocess) to degrade, whereas policies that allocate processing power equally per job (RRjob) are not as affected. As stated above, this degradation is due to RRprocess allocating more processing power to the jobs most likely to have large job demands. Note that the measured C_d increases as $\overline{big}$ increases. We show in section 4.4.1 that these two policies are insensitive to variation of job demand, hence the degradation of RRprocess can be primarily attributed to the difference in C_n.

4.3. Baseline Comparison of the Policies

In this section we present results for two baseline cases. We present results for the geometric-bounded workload since we hypothesize that it more accurately models typical systems than the hyperexponential workload. We denote these experiments "baseline" since they provide a reference point for the remaining experiments in this thesis. We include one baseline each for the uncorrelated and correlated workloads. The parameter values for the uncorrelated baseline experiment are: $P_p = 0.4$, $\bar{n} = 4.0$, $C_d = 5.0$, $\bar{d} = 200$. The parameter values for the uncorrelated baseline experiment are: $P_p = 0.4$, $\bar{n} = 4.0$, $C_v = 5.0$, $t = 20$. We set P_p equal to 0.4 to represent a substantial portion of highly parallel jobs. We set $\bar{n}$ equal to 4.0 since many of todays parallel programs often run with only modest parallelism, since increases in parallelism offer little speedup gain. We set C_d and C_v equal to 5 since it is seems likely that multiprocessors will have a C_d of at least 5, given that uniprocessor workloads tend to have a C_d of around 10 [SaCh 81].

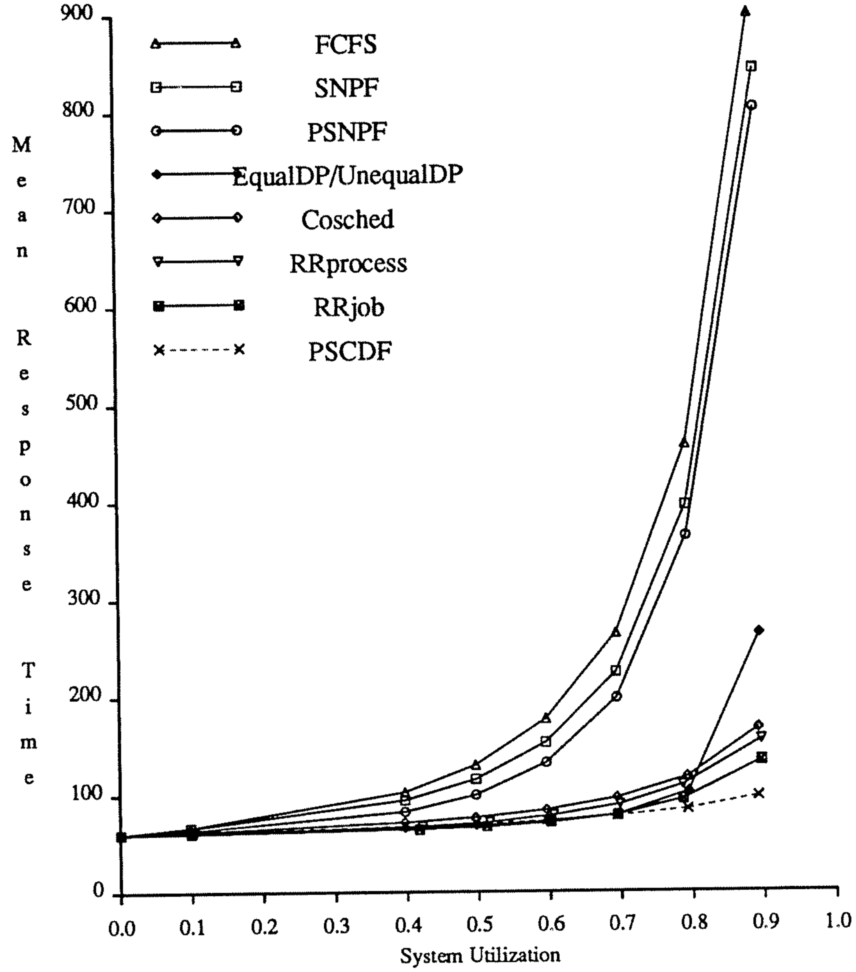


$\overline{big}$	0	5	10	15	20
C_n	0.0	1.09	1.98	2.60	3.06
$\bar{n}$	1.0	1.20	1.45	1.70	1.95
C_d	2.97	4.36	6.51	8.21	9.52
$\bar{d}$	49.9	59.5	71.5	83.4	95.5

	P_p	$\bar{n}$	$\overline{big}$	C_v	t
Input	0.05	1.0	vary	3.0	50.0

Figure 4.5: Effect of Coefficient of Variation of the Number of Processes

Figure 4.6 plots mean response time versus utilization for the uncorrelated workload. The policies FCFS, SNPF, and PSNPF perform significantly worse than the other policies as system utilization increases. The policy PSCDF performs best, followed by RRjob, RRprocess, and Cosched. The policies EqualDP and UnequalDP perform as well as RRjob up to a utilization of 80%. At a utilization of 90% the performance of the two polices degrades.

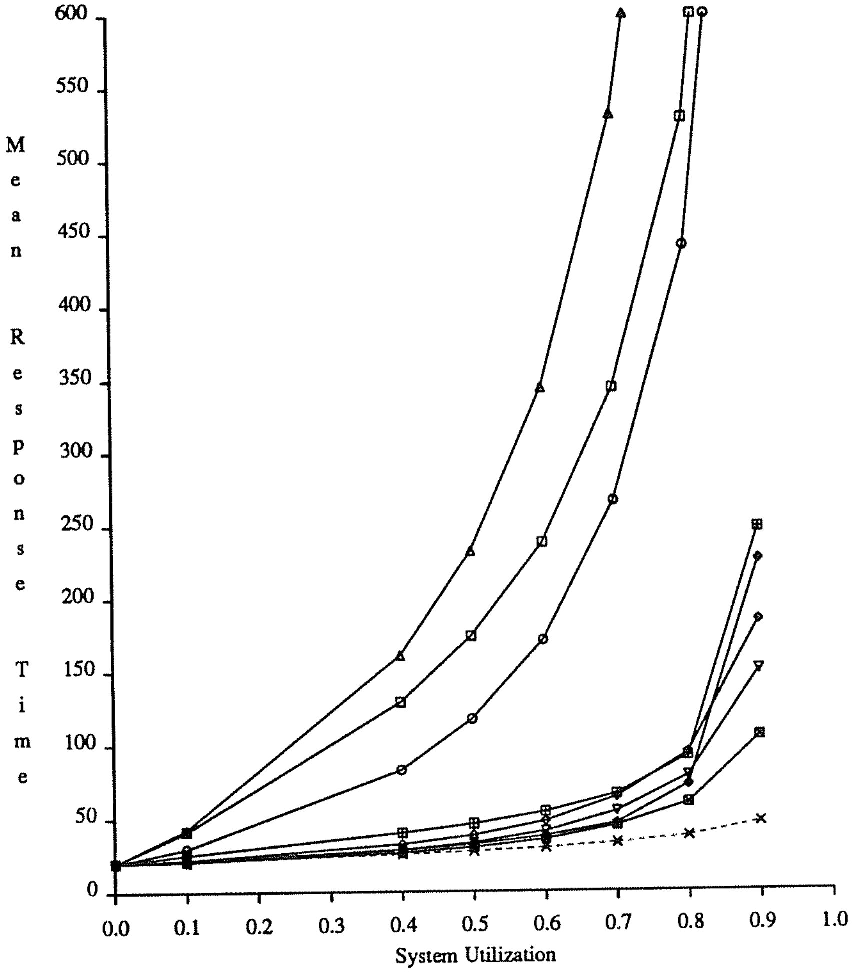


	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		200.0	5.0
Output		10.36	0.79	199.3	4.97

Figure 4.6: Baseline Uncorrelated Workload

This is due to our assumption that these policies only allow as many jobs into the system as there are processors (i.e., 20 in our experiments). When there are more jobs in the system than processors the excess jobs are held in a load queue, not receiving service (see section 2.1.2). At a utilization of 90% the number of jobs in the system often exceeds the number of processors. As a result, the dynamic partitioning policies exhibit FCFS behavior at high loads. This does not imply these policies are poor policies, but it does imply that a load queue is a bad design choice. A multi-level queue as used in Unix would help alleviate the problem. Note that in [ZaMc 90] they do propose using a multi-level queue.

Figure 4.7 plots mean response time versus system utilization for the correlated workload. There are two qualitative differences from the uncorrelated workload. The first is that the SNPF policies perform better relative to the other policies for the correlated workload than the uncorrelated workload. The second difference is that there is a larger difference in



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	C_v	t
Input	0.4	4.0				5.0	20.0
Output		10.36	0.79	210.0	6.48		

Figure 4.7: Baseline Correlated Workload

the mean response times of UnequalDP, Cosched, RRprocess, EqualDP, RRjob, and PSCDF.

That the SNPF policies still perform worse than the other policies may seem surprising since scheduling (P)SNPF for the correlated workload gives jobs that are likely to have a smaller demand higher priority. The reason the policies so still do not perform as well as the other policies is because the priority is based on imperfect job demand knowledge. While scheduling PSNPF results in running jobs that are likely to have a small demand, jobs with a small number of processes may have a large demand. When this happens, PSNPF will schedule jobs that actually have a large demand in preference to jobs with a smaller demand. Note that the SNPF policies perform worse in figure 4.7 than in figure 4.4 This is because the parameter P_p equals 0.4 in figure 4.7 and 0.05 in figure 4.4. We will consider the effect of P_p in section 4.4.3.

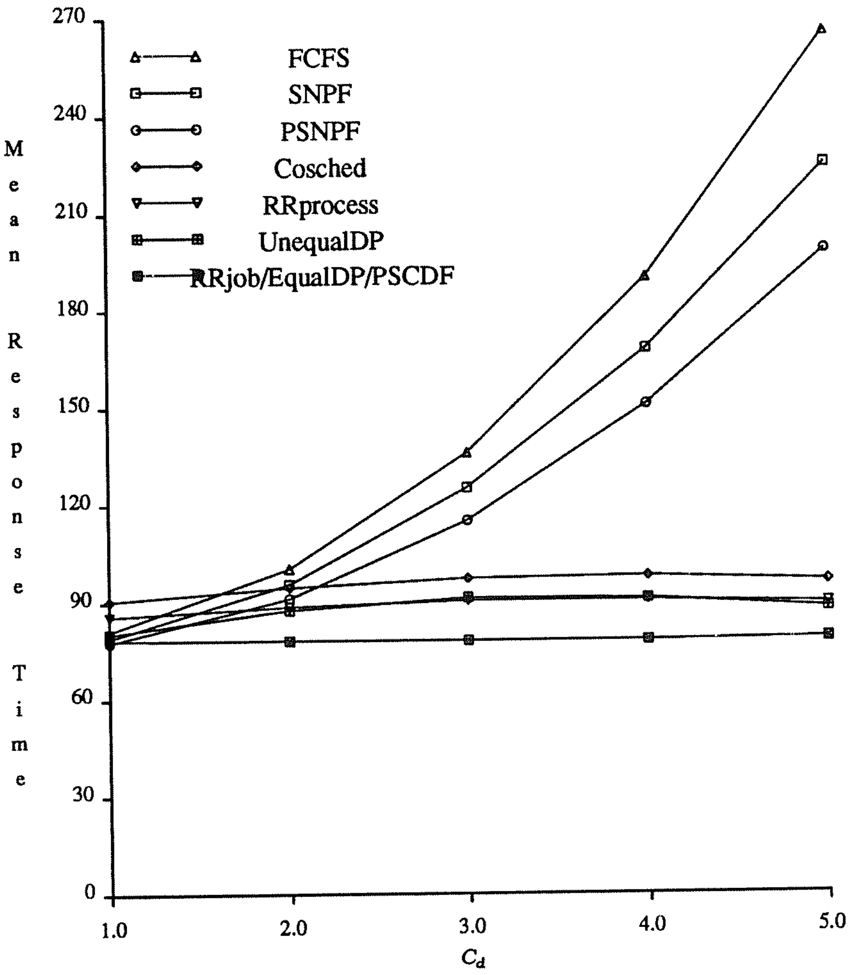
4.4. Sensitivity to Input Parameters

In this section we explore the effect of varying system parameters for the baseline models. In section 4.4.1 we consider the sensitivity to coefficient of variation of job demand C_d by varying the parameters C_d and C_v for the uncorrelated and correlated workloads respectively. In section 4.4.2 we consider the mean job demand $\bar{d}$ by varying input parameters $\bar{d}$ and t for the uncorrelated and correlated workloads respectively. In section 4.4.3 we consider the effect of varying P_p. In section 4.4.4 we consider the effect of varying $\bar{n}$.

4.4.1. Sensitivity to Variation of Job Demand

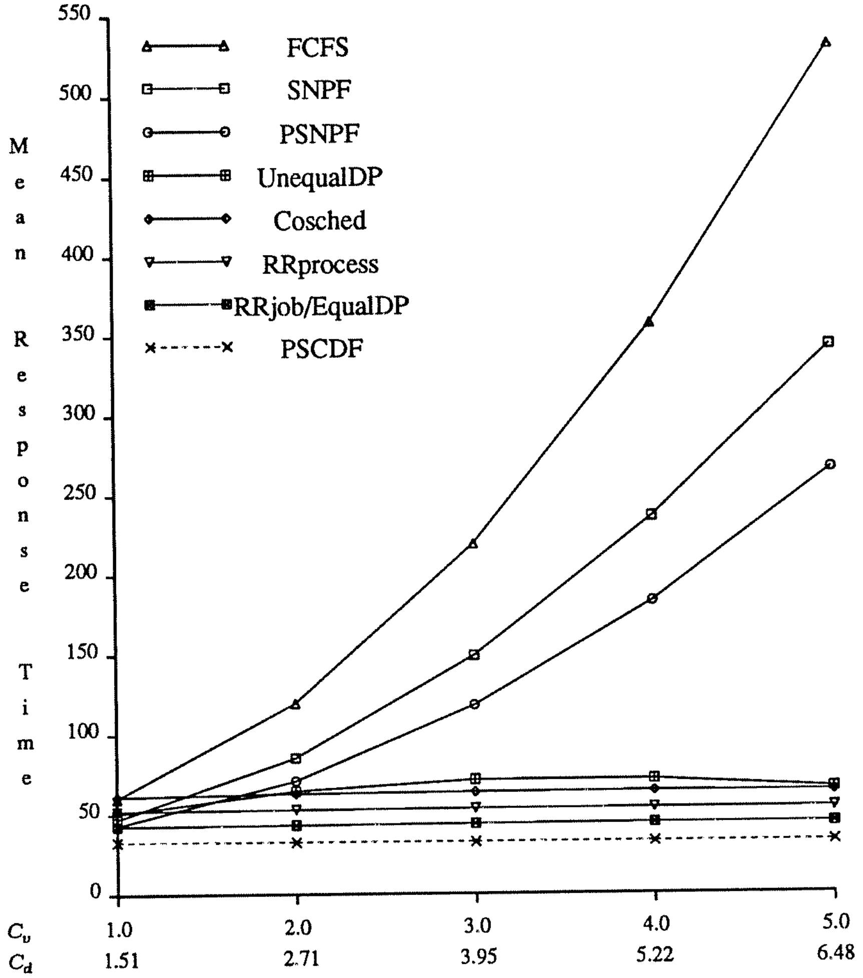
In the baseline workloads C_d and C_v were 5.0. In this section we investigate how sensitive the policies are to changes in the input parameters C_d and C_v. Figure 4.8 plots mean response time versus C_d for the uncorrelated baseline workload. Figures 4.9 and 4.10 plot mean response time versus C_v for the correlated for P_p = 0.4 and 0.05 respectively. The measured C_d is different for the correlated workload and is included on the x-axis. Parameters C_d and C_v are varied from 1.0 to 5.0. Utilization is set at 70%.

Consistent with uniprocessor scheduling results, FCFS is very sensitive to C_d and C_v. As expected, SNPF and PSNPF are also sensitive to C_d and C_v. At C_d and C_v = 1.0, FCFS, SNPF, and PSNPF perform somewhat better than RRprocess and almost as well as RRjob. However, as C_d and C_v increase, the performance of FCFS, SNPF, and PSNPF degrade dramatically. All other polices studied are insensitive to C_d and C_v. In typical uniprocessor workloads C_d has been shown to be on the order of 10 or more [SaCh81], [Klei 76 page 176]. If multiprocessor workloads are similar, FCFS, SNPF, and PSNPF are not viable policies. When the coefficient of variation is high, these three policies often result in jobs with short demands not receiving any service until jobs with large demands finish executing. The RRprocess, Cosched, RRjob, EqualDP, and UnequalDP policies perform better because each job in the system receives some fraction of the processing power. The policy PSCDF is



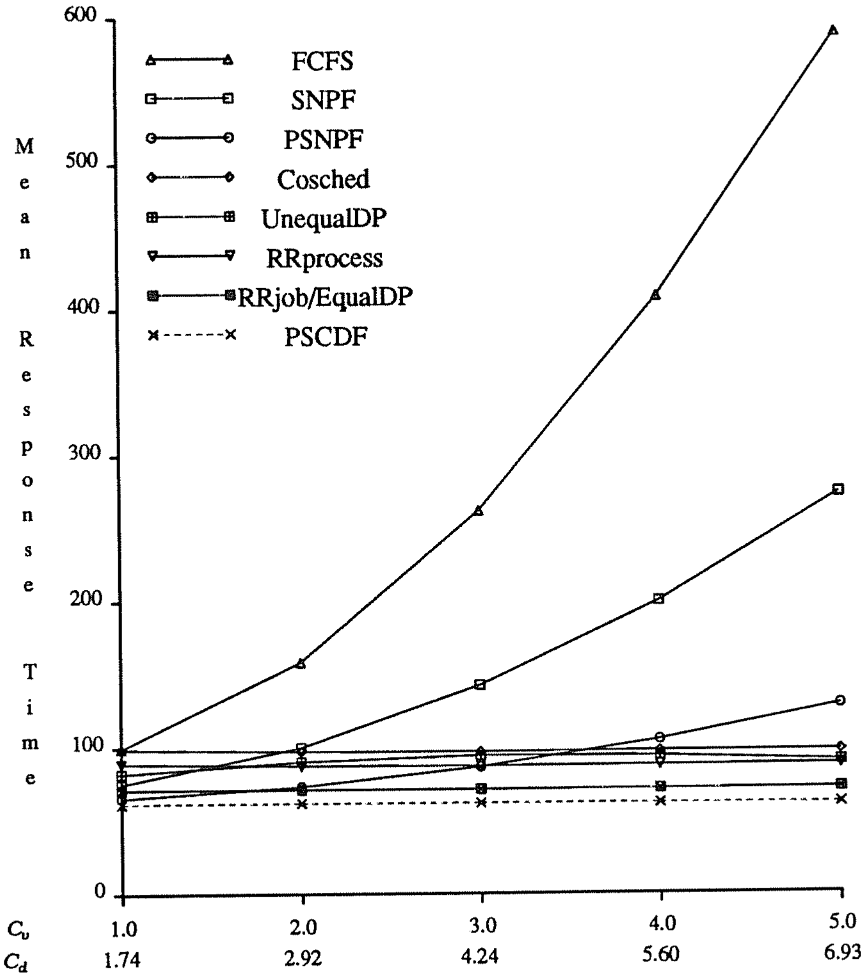
	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		200.0	vary
Output		10.36	0.79	200.0	

Figure 4.8: Effect of C_d, Uncorrelated Workload



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_v	t
Input	0.4	4.0			vary	20.0
Output		10.36	0.79	210.0		

Figure 4.9: Effect of C_v, Correlated Workload



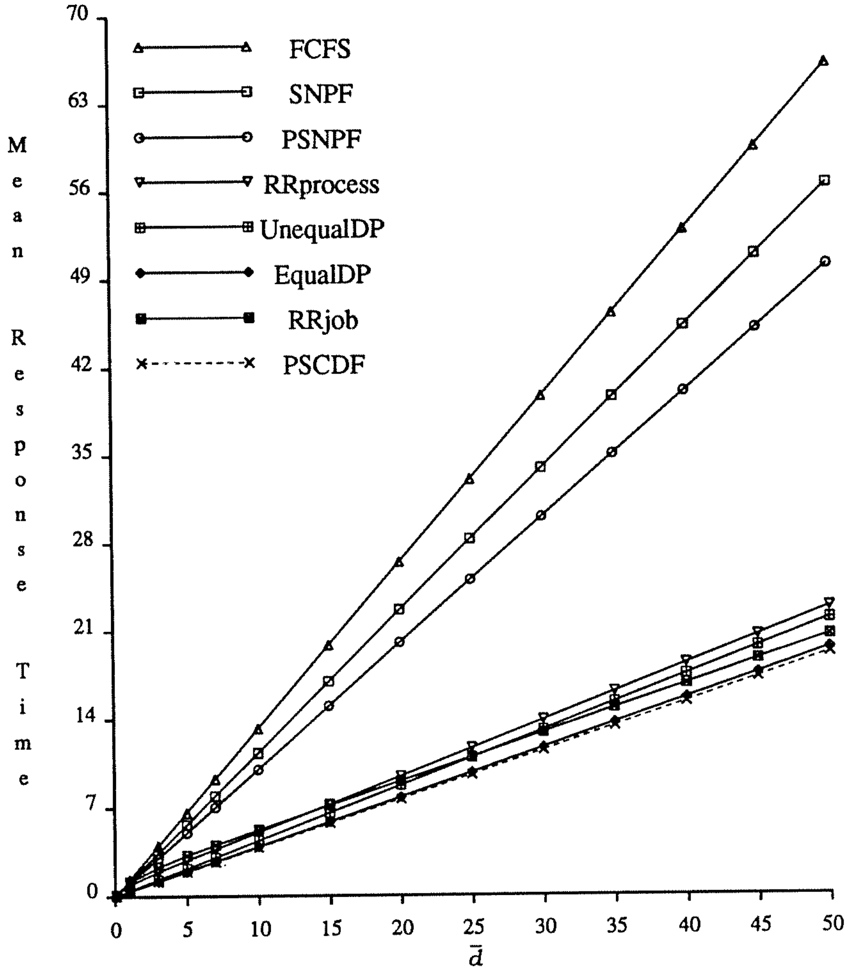
	P_p	$\bar{n}$	C_n	$\bar{d}$	C_v	t
Input	0.05	4.0			vary	50.0
Output		4.79	1.00	238.7		

Figure 4.10: Effect of C_v, P_p = 0.05, Correlated Workload

also resilient to variation in job demand even though at high utilizations the policy only allocates processing power to a subset of the jobs in the system. PSCDF performs well because it uses job demand information to schedule jobs with the smallest demand first. Note that PSNPF and SNPF perform better for P_p = 0.05 as shown in figure 4.10. We will further explore the effect of P_p in section 4.4.3.

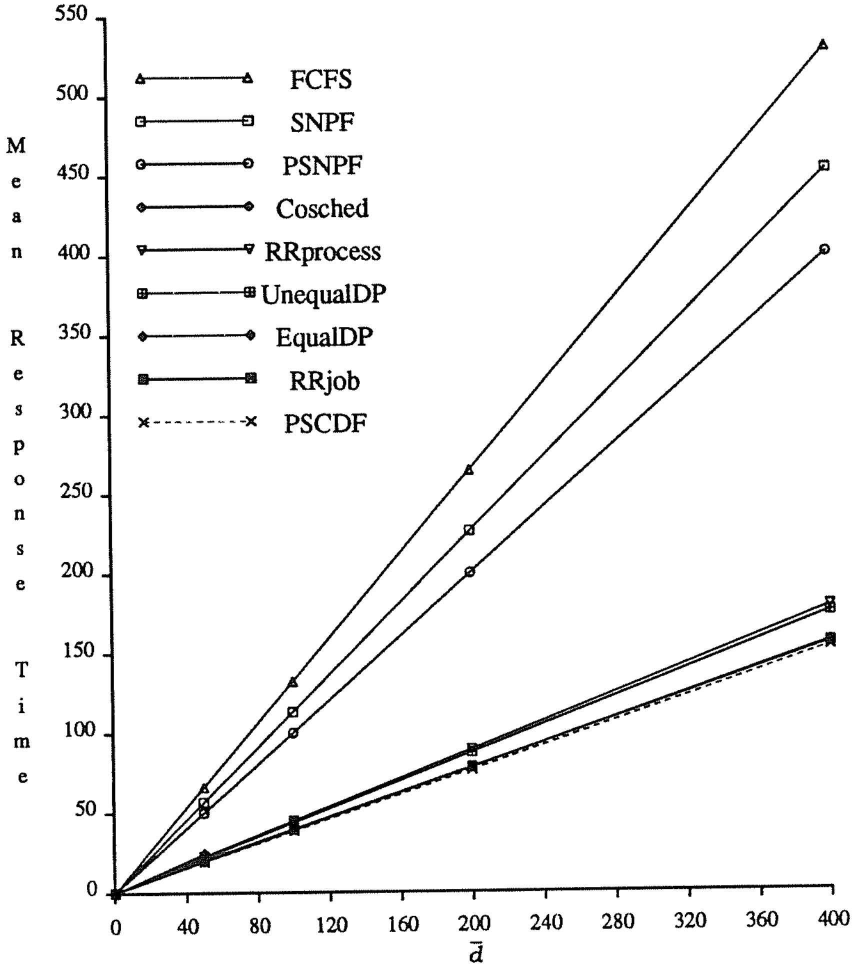
4.4.2. Sensitivity to Mean Job Demand

In this section we investigate how sensitive the policies are to changes in mean job demand. Figures 4.11a and 4.11b plot mean response time versus $\bar{d}$ for the uncorrelated geometric-bounded workload. Both figures present the same results, but figure 4.11a only plots $\bar{d}$ from 0 to 50 units to show more detail for the lower values. From figure 4.11a, we



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		vary	5.0
Output		10.37	0.79		4.96

Figure 4.11a: Effect of $\bar{d}$, Uncorrelated Workload



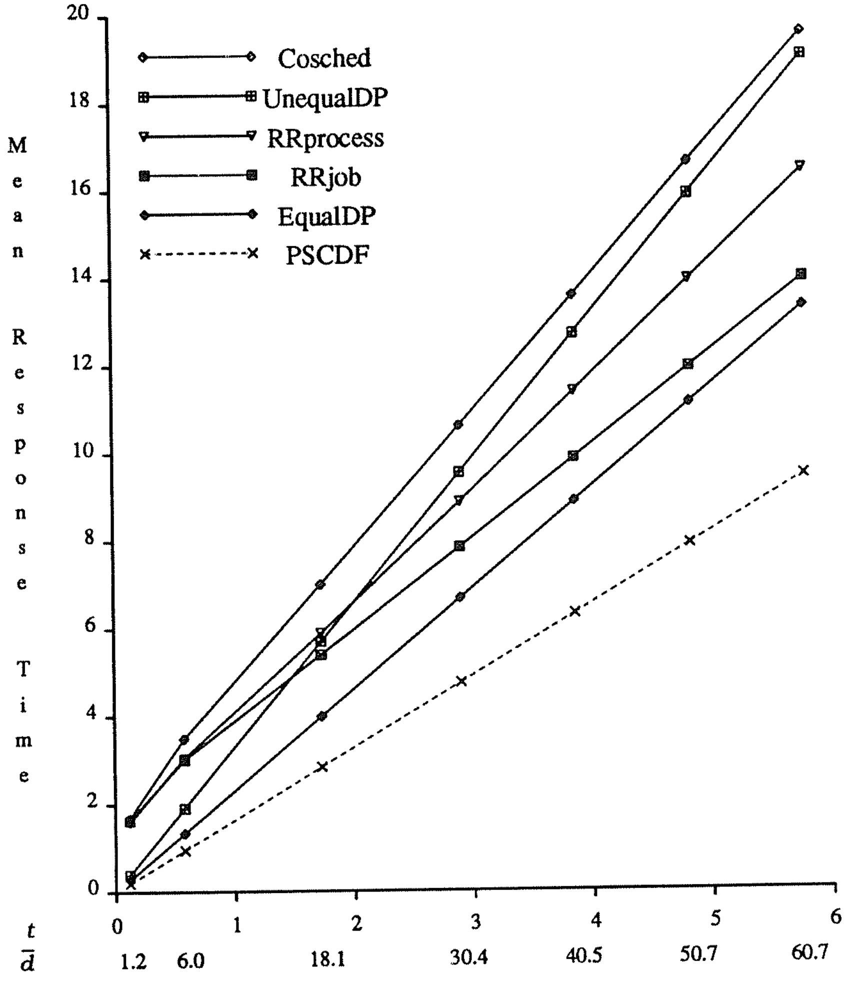
	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		vary	5.0
Output		10.37	0.79		4.96

Figure 4.11b: Effect of $\bar{d}$, Uncorrelated Workload

see that the relative performance of the policy changes as $\bar{d}$ changes. For values of $\bar{d}$ less than 3 units PSNPF performs as well as RRjob and RRprocess. For values of $\bar{d}$ less than 15 RRprocess performs better than RRjob. For values of $\bar{d}$ less than 25 UnequalDP performs better than RRjob. Once $\bar{d}$ is greater than 25 the relative performance of the policies does not change. From figure 4.11b we see that the performance of RRjob, EqualDP, and PSCDF are similar, and the performance of RRprocess and Cosched are similar. The reason that the relative performance of the policies changes for small values of $\bar{d}$ is because RRprocess, and RRjob are sensitive to the magnitude of the quantum relative to the mean job demand. Since the quantum is the unit time measure in our studies, a decrease in job demand implies the quantum size is increasing relative to mean job demand. As the relative quantum size increases, RRprocess and RRjob perform worse. RRjob has a larger average quantum than RRprocess so RRprocess performs better than RRjob at low job demands.

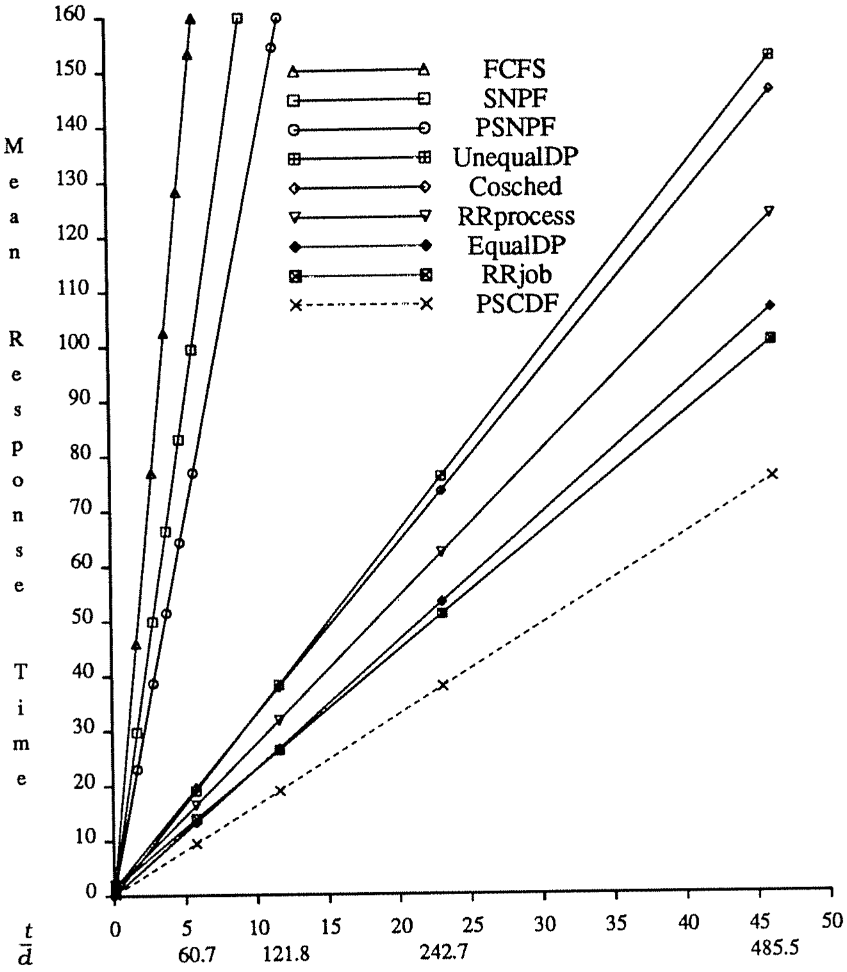
Figures 4.12a and 4.12b plot response time versus the scalar t for the correlated geometric-bounded workload. The two curves are for the same experiment, but figure 4.12a only plots t from 0.12 units to 5.78 units to show more detail for the lower values. The curves for FCFS, SNPF, and PSNPF are omited from figure 4.12a to show more detail for the other policies. As the scalar t increases the mean job demand also increases. We include the measured $\bar{d}$ on the x-axis. From figure 4.12a we see that the policies EqualDP and UnequalDP have a lower mean response time than RRjob, RRprocess, and Cosched when t is small. As t increases, the relative performance of RRjob, RRprocess, and Cosched improve. The mean response time for RRjob is 5% larger than the response time of EqualDP when t equals 5.78. When t equals 11.6 the response time of EqualDP is 1% larger than the response time of RRjob.

From this section we conclude that with the exception of EqualDP and RRjob the value of job demand does not affect the qualitative results if the value is in excess of 50 units. The mean job response time of RRjob is 1% - 5% larger than the mean job response time of EqualDP for demands in the range of 200 units to 50 units. We would expect demand in real systems to be on the order of 50. We base this on the uniprocessor rule of thumb that quantum length is chosen so that approximately 70% of the jobs finish within the first quantum. For our models, 60% of the processes complete in the first quantum if mean job demand is 50. To see this consider the uncorrelated workload. A C_d of 5.0 and $\bar{d}$ of 50.0 units results in the means of the two stages of the hyperexponential distribution being equal to 10.2 units and 805 units. Since 95% of the samples are drawn from the stage with the small mean, 95% of the jobs have the job demand drawn from an exponential distribution with mean 10.2. When using an exponential distribution 63% of the samples are less than the mean of the distribution. As a result, 60% (0.63×0.95) of the jobs have a job demand less than or equal to 10.2 units. Mean job parallelism for most workloads in this study is 10.3, which implies about 60% of the processes will finish within the first quantum. We will continue using a value of 200 units for all simulations. Note the policies RRprocess, and RRjob would not perform as well relative to EqualDP if job demand were in the range of 50 or less. If mean job demand is 3 units or less PSNPF may perform better than RRjob, RRprocess and Cosched.



	P_p	$\bar{n}$	C_n	C_d	C_v	t
Input	0.4	4.0			5.0	vary
Output		10.36	0.79	6.48		

Figure 4.12a: Effect of $\bar{d}$, Correlated Workload



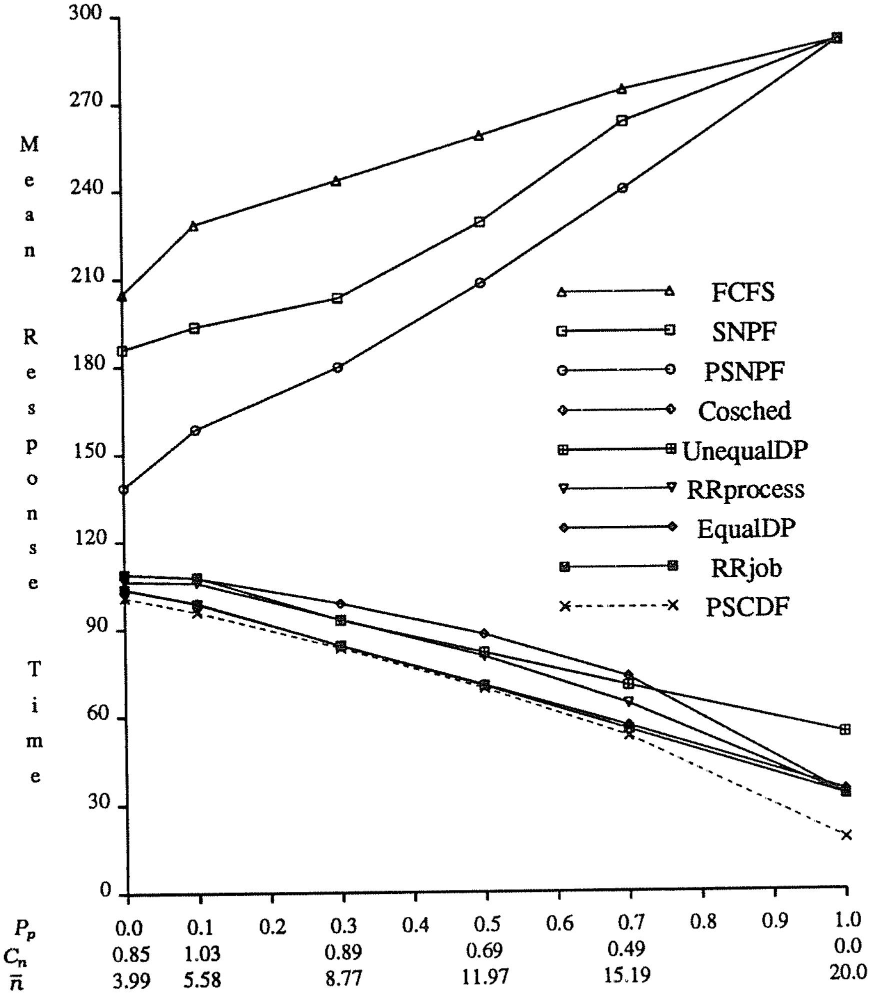
	P_p	$\bar{n}$	C_n	C_d	C_v	t
Input	0.4	4.0			5.0	vary
Output		10.36	0.79	6.48		

Figure 4.12b: Effect of $\bar{d}$, Correlated Workload

4.4.3. Effect of Varying Parameter P_p

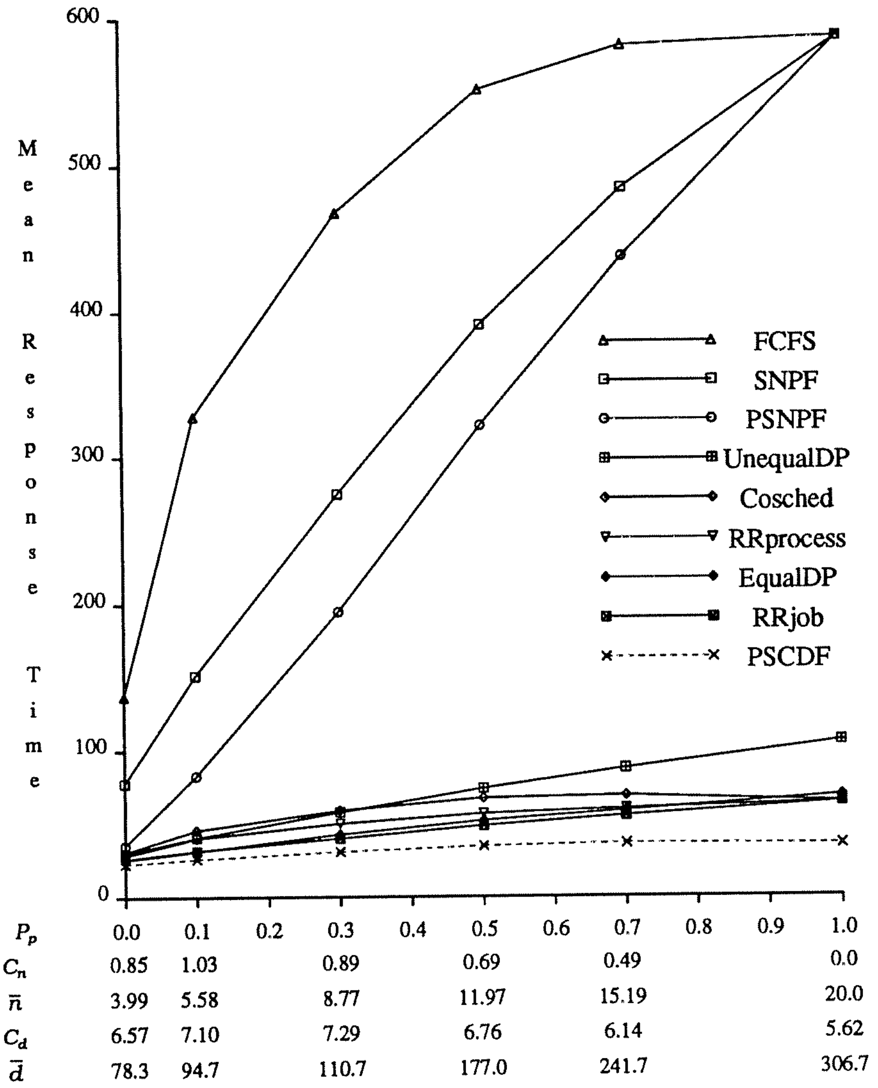
We expect in the future that application programs and system tools will exhibit higher degrees of parallelism, thus it is important to consider the effect of mean job parallelism on the relative performance of the scheduling policies. In this section we investigate how sensitive the policies are to the parameter P_p. By varying P_p we vary the percentage of jobs that have 20 processes, hence we vary the mean job parallelism. A second way to vary mean job parallelism is to vary the parameter $\bar{n}$. We investigate the effect of varying $\bar{n}$ in the next section.

Figures 4.13 and 4.14 demonstrate the sensitivity of the policies to P_p for the uncorrelated and correlated geometric-bounded workloads respectively. Parameter P_p is varied while all other input parameters are held constant. Utilization is set at 70%. Note that for the uncorrelated case the measured parameters C_n and $\bar{n}$ vary with P_p, and for the correlated



	P_p	$\bar{n}$	$\bar{d}$	C_d
Input	vary	4.0	200.0	5.0
Output			198.9	5.0

Figure 4.13: Effect of P_p, Uncorrelated Workload



	P_p	$\bar{n}$	C_v	$\bar{t}$
Input	vary	4.0	5.0	20.0

Figure 4.14: Effect of P_p, Correlated Workload

case the measured parameters C_n, $\bar{n}$, C_d, and $\bar{d}$ vary with P_p. We include the value of these parameters on the x-axis of the graphs.

For the uncorrelated workload, figure 4.13, the FCFS, SNPF, and PSNPF scheduling policies degrade as P_p increases, whereas all the other policies improve. The increased parallelism causes a decrease in the number of jobs that can run in parallel by FCFS, SNPF, and PSNPF. As a result, the probability of servicing jobs with small demands decreases. Note that at P_p = 1.0, SNPF and PSNPF scheduling is identical to FCFS. The other policies benefit from the increase in parallelism since job demand is held constant and all jobs receive some fraction of the processing power.

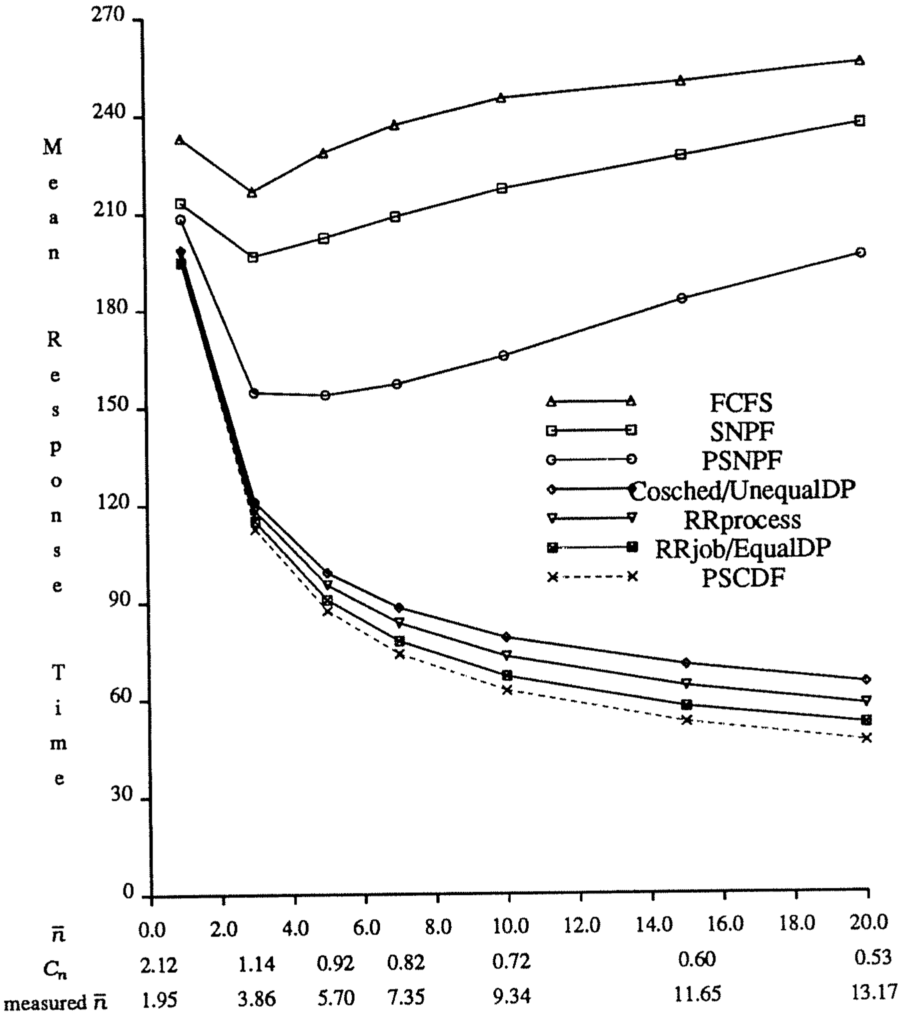
For the correlated workload, figure 4.14, FCFS, SNPF, and PSNPF degrade substantially as P_p increases, whereas all the other policies degrade slightly. As P_p increases, the mean number of processes, and hence the mean job demand increases. Even though the parallelism and job demand increase proportionally, the increase in parallelism does not negate the increase in demand since the increased parallelism is not useful when the number of processes in the system exceeds the number of processors. The increase in job demand causes the slight increase for Cosched, RRprocess, RRjob, EqualDP, and PSCDF. The policies FCFS, SNPF, and PSNPF degrade considerably more since fewer jobs can run in parallel. In addition, by increasing P_p more jobs have the same number of processes (observe C_n decreasing), hence the policies SNPF and PSNPF are not as capable of discriminating which jobs have the smallest demand.

4.4.4. Effect of Varying Parameter $\bar{n}$

The second way to vary mean job parallelism in the geometric-bounded workload is to vary the parameter $\bar{n}$. In this section we investigate the sensitivity of the policies to the parameter $\bar{n}$. In figures 4.15 and 4.16 we vary the input parameter $\bar{n}$ while holding all other parameters constant for the uncorrelated and correlated geometric-bounded workloads respectively. Utilization set at 70% and P_p is set to 0.05. We choose P_p equal to 0.05 for this experiment to emphasize the effect of $\bar{n}$. The parameter P_p is equal to 0.4 for all other experiments. The measured values of C_n and $\bar{n}$ vary with P_p for the uncorrelated workload, and the measured values of C_n, $\bar{n}$, C_d, and $\bar{d}$ vary with P_p for the correlated workload. We include the value of these parameters on the x-axis of the graphs. At $\bar{n}$ equal to 20.0 the measured $\bar{n}$ does not equal 20.0. This is due to the fact we truncate the number of processes per job to a maximum of 20 as described in section 3.3.

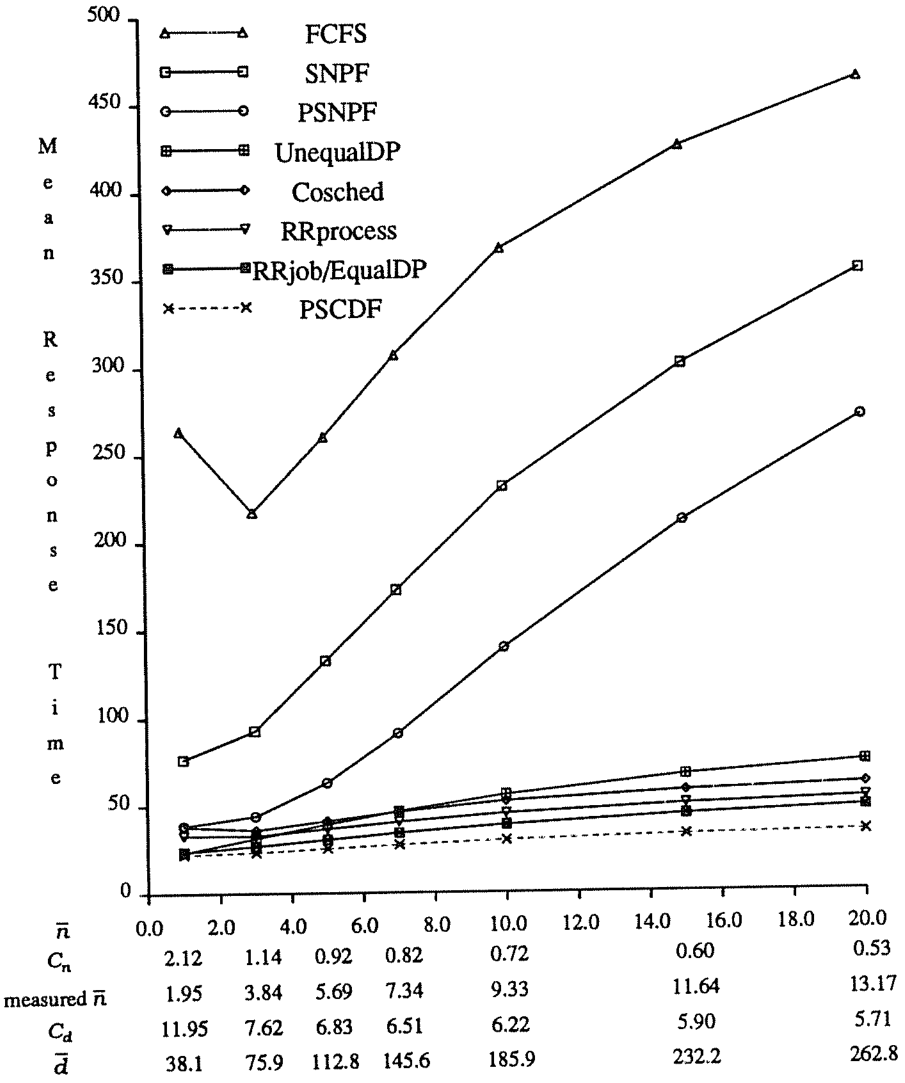
For the uncorrelated workload, figure 4.15, we see that as $\bar{n}$ increases, FCFS, SNPF, and PSNPF initially improve. As $\bar{n}$ increases further, the advantages of decreasing a job's service time due to the additional parallelism is eclipsed by the degradation due to not being able to run as many jobs at the same time. All of the other policies benefit from the increase in $\bar{n}$.

For the correlated workload, figure 4.16, increasing $\bar{n}$ also increases the mean job demand due to the linear correlation. As a result, as $\bar{n}$ increases the mean response time for all the policies increases. Once again FCFS, SNPF, and PSNPF especially suffer since they can not run as many jobs in parallel as $\bar{n}$ increases. Also, as $\bar{n}$ increases more jobs



	P_p	$\bar{n}$	$\bar{d}$	C_d
Input	0.05	vary	200.0	5.0
Output			199.3	4.97

Figure 4.15: Effect of $\bar{n}$, Uncorrelated Workload



	P_p	$\bar{n}$	C_v	t
Input	0.05	vary	5.0	20.0

Figure 4.16: Effect of $\bar{n}$, Correlated Workload

have the same mean demand resulting in SNPF and PSNPF not being as capable of discriminating short jobs. Note that FCFS actually experiences a decrease in mean response time when $\bar{n}$ is raised form 1.0 to 3.0. This is because the improvement from the drop in variation of job demand, note C_d on the x-axis, affects FCFS more then the drop in the number of jobs simultaneously running.

4.5. Effect of Foreground - Background Scheduling

The previous sections have shown that each job in the system must receive some fraction of processing power in order to have good performance. The reason performance suffers for FCFS, SNPF, and PSNPF is that sometimes jobs with a large demand are scheduled while other jobs that have a smaller demand receive no service. We have seen that allocating a fraction of processing power to each job in the system can alleviate this problem. Another approach to giving small jobs good service is to have jobs drop in priority after receiving some amount of service. Such a multi-level queue is used in many current uniprocessor scheduling policies. In this section we consider a two level priority queue. When jobs enter the system they are placed in the high priority queue. After some amount of service, τ, processes drop to the lower priority. The high priority queue is given preemptive priority over the low priority queue. Since we consider only two levels we refer to this type of scheduling as foreground-background scheduling. We consider only the policies FCFS, PSNPF, and RRjob. We assume that behavior for SNPF will be similar to PSNPF and behavior of the other policies will be similar to RRjob.

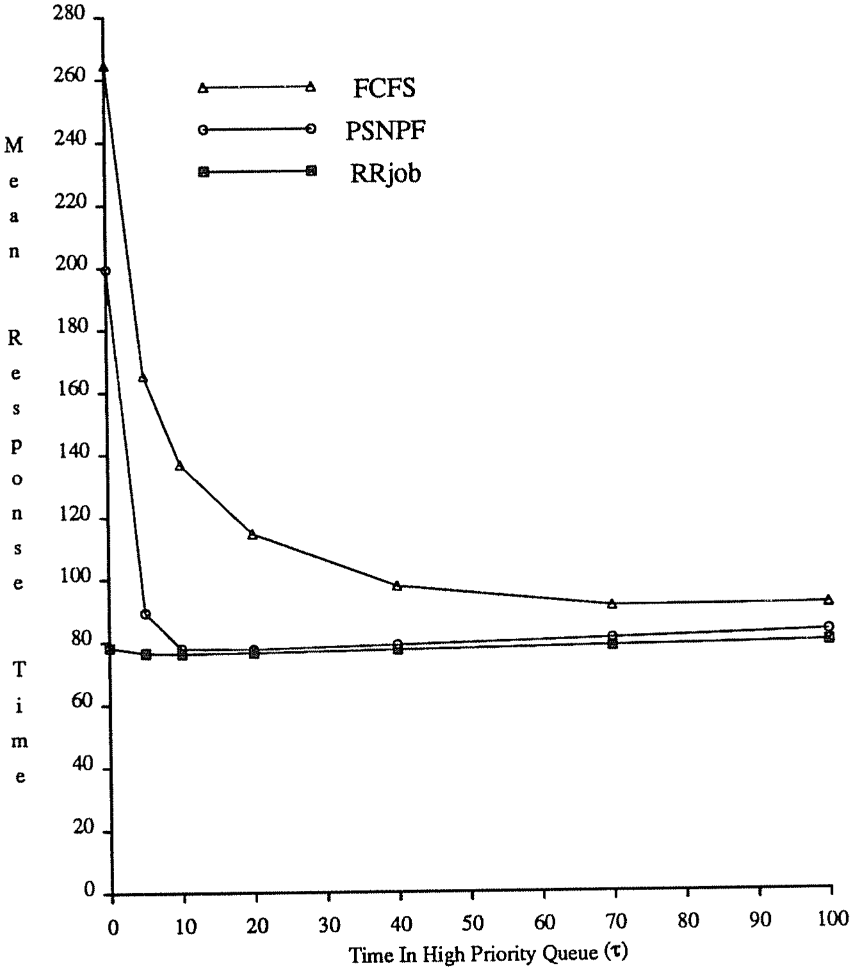
We experiment by varying the amount of time before a process is moved to the lower priority queue, τ, while holding all other parameters constant. Figures 4.17a and 4.17b plot mean response time versus time in the high priority queue before being lowered to the low priority queue for the uncorrelated geometric-bounded workload. System utilization is fixed at 70%. The two figures are the same except that figure 4.17a only plots up to a value of 100 units on the x-axis to show more detail for small values. We see in figure 4.17a that PSNPF quickly performs as well as RRjob. Performance improves because jobs with large demands do not compete for processing power until periods of low utilization. Note that RRjob does not derive much benefit from the foreground - background scheduling. This is because RRjob was not hindered much by large jobs without the two level queue. On the other hand, FCFS and PSNPF were hindered by the large jobs, so a large improvement in performance is gained from scheduling foreground - background. Figure 4.17b shows that as τ is increased, performance reverts back to just the single level case. When τ equals 10000 units the performance is the same as a single queue.

Figure 4.18 plots mean response time versus time in high priority queue for the correlated workload. We see that once again the foreground - background scheduling improves performance of FCFS and PSNPF, but does not help RRjob much.

It appears that PSNPF may be a good policy if a multi-level queue is used. One problem with such a multi-level queue is that performance will suffer if τ is not chosen correctly.

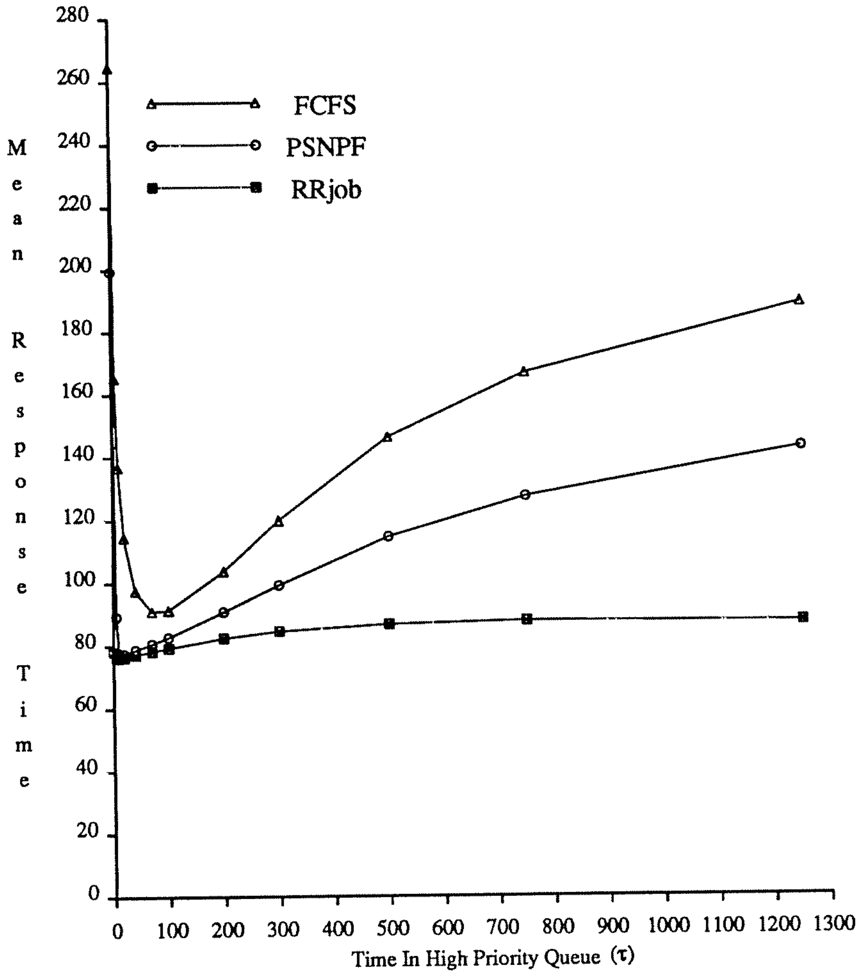
4.6. Conclusion

The main conclusion of this chapter is that it is important for a multiprocessor scheduling policy to give small jobs rapid service. We have considered three ways to



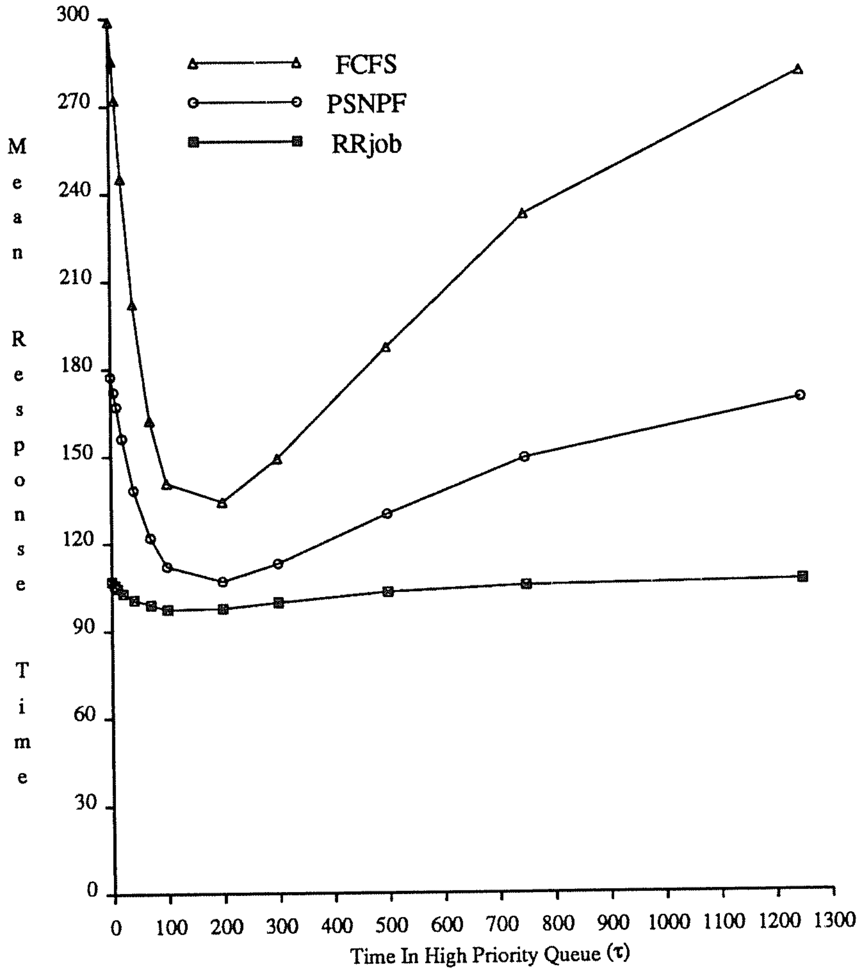
	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		200.0	5.0
Output		10.37	0.79	197.5	4.96

Figure 4.17a: Foreground - Background Scheduling
Uncorrelated Workload



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		200.0	5.0
Output		10.37	0.79	197.5	4.96

Figure 4.17b: Foreground - Background Scheduling
Uncorrelated Workload



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	C_v	t
Input	0.4	4.0				2.0	50.0
Output		10.36	0.79	522.5	2.70		

Figure 4.18: Foreground - Background Scheduling
Correlated Workload

achieve this goal:

1) Give highest priority to short jobs.

2) Give each job in the system a fraction of the processing power.

3) Use multi-level scheduling.

The first approach has been shown to perform best if we have perfect job demand knowledge and preemptive scheduling as in PSCDF. If job demand knowledge is imperfect, as in SNPF and PSNPF for the correlated workload, performance suffers if the coefficient of variation of job demand is 2 or higher. The second approach, giving a fraction of processing power to each job in the system, appears to work well. This is the approach used by Cosched, RRprocess, UnequalDP, EqualDP, and RRjob. These policies are insensitive to variations in job demand. The third approach appears to benefit the policies FCFS and PSNPF but not RRjob. This is because RRjob already gives the small jobs rapid service. If the amount of time spent in the high priority queue is tuned appropriately, it appears that the simple policies may be viable.

Other more specific conclusions from this chapter are:

- The policies FCFS, SNPF, and PSNPF perform as well as the other policies when the coefficient of variation of job demand is less than 2. Real systems would most likely exhibit much more variation in job demand, resulting in the policies FCFS, SNPF, and PSNPF not being feasible policies.

- Increases in mean job parallelism cause the policies FCFS, SNPF, and PSNPF to perform poorly for both the uncorrelated and correlated workloads.
- The policies EqualDP and UnequalDP do not give a fraction of processing power to each job in the system when the number of jobs in the system exceeds the number of processors. This results in poor performance at high utilizations.
- The policy PSCDF performed best for all workloads studied. Unfortunately the policy is not practical to implement since a real system would not have complete job demand knowledge.
- Policies that allocate processing power proportional to the number of processes per job (RRprocess and Cosched) perform worse for the correlated workload than policies that allocate processing power equally per job (RRjob and EqualDP).

We have shown that RRjob and EqualDP perform the best of all the practical polices studied. We attribute this superior performance to the fact that these two policies allocate processing power more equally per job that the others. In the next chapter we consider the importance of this equal allocation per job.

CHAPTER 5

Importance of Equal Allocation Per Job

In the previous chapter we have shown that in the absence of perfect job demand knowledge or a properly tuned multi-level queue, the most promising policies continuously allocate some fraction of processing power to each job in the system. We also demonstrated that the policies RRjob and EqualDP performed better than all other policies except PSCDF. We attribute the superior performance of RRjob and EqualDP to the equal allocation of processing power per job. In this chapter we highlight and further investigate the importance of allocating processing power equally per job. We consider only the policies RRprocess, Cosched, RRjob, EqualDP, and UnequalDP. We do not consider FCFS, SNPF, and PSNPF since we have shown that the performance of these three policies is inferior to the others. We do not consider PSCDF since in a real system we usually do not know job service demands.

The policies RRjob and EqualDP allocate processing power equally to all jobs that can make use of the available parallelism, while the other policies do not. In actuality, RRjob and EqualDP do not truly provide equal allocation per job. For example, consider a 20 processor system with two jobs, one with 1 process the other with 20 processes. If a policy provided true equal allocation per job, then both jobs would get one processor each while the other 18 processors sit idle. We will use a looser definition of equal allocation, where each job gets an equal share of processing power, but any additional capacity not being used is divided equally between all jobs that have enough parallelism to use it. RRjob provides equal allocation per job temporally by giving jobs with a small number of processes a longer time slice. EqualDP provides equal allocation per job spatially by partitioning the processors equally among the jobs. In addition to the unequal allocation due to jobs that can not make use of the additional parallelism, EqualDP divides processing power somewhat less equally than RRjob since the number of jobs may not evenly divide the number of processors. For example, a 20 processor system with 13 jobs having at least 2 processes results in 7 jobs receiving 2 processors and 6 jobs receiving one processor.

We investigate the importance of equal allocation by comparing the other four policies to RRjob for both the uncorrelated and correlated workloads. When comparing each policy to RRjob, we use the ratio of mean response time for the other policy over the mean response time for RRjob as our metric of comparison. We refer to this metric as the response time ratio. We plot the response time ratio for the policies Cosched, RRprocess, UnequalDP, and EqualDP for the uncorrelated and correlated baseline workloads, and for the sensitivity to P_p and $\bar{n}$ experiments presented in chapter 4.

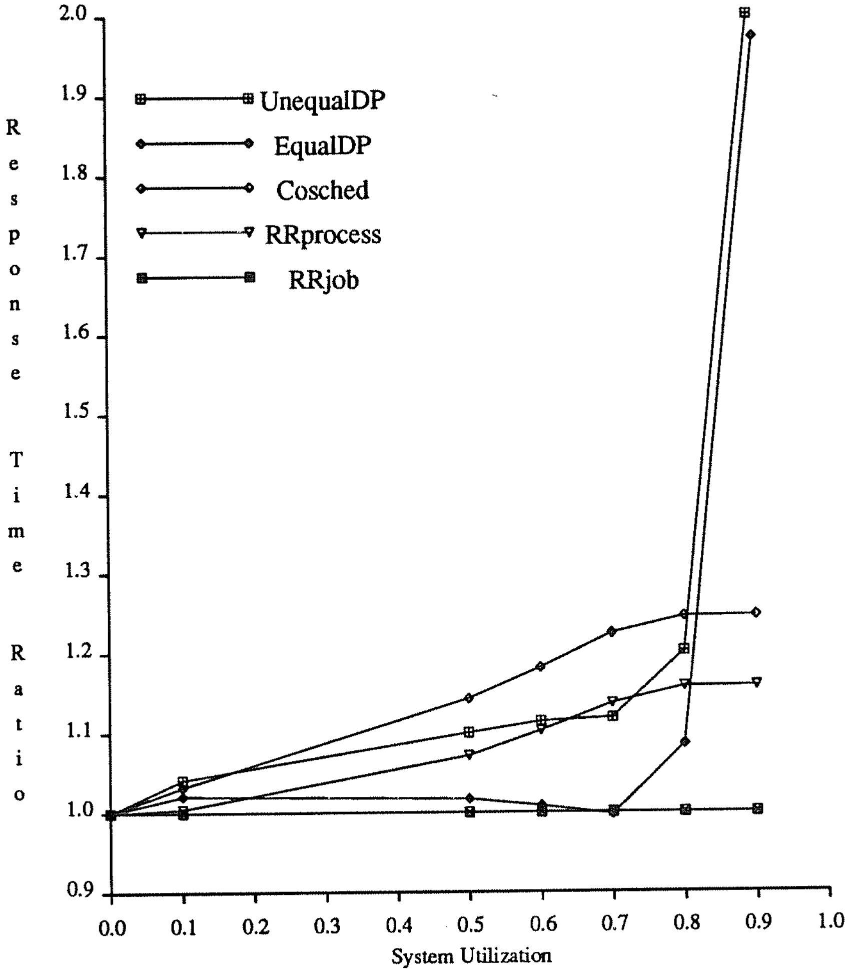
At high utilizations there are often more jobs than processors in the system. As a result, the mean response times of EqualDP and UnequalDP increases at high utilizations due to the load queue assumptions. To remove this side effect, we consider a closed system model with only 20 jobs so there are never more jobs than processors in the system. We include a section highlighting the difference between the open and closed system models to aid in understanding the closed system results.

Section 5.1 presents response time ratios for the open system model with an uncorrelated geometric-bounded workload. Section 5.2 presents response time ratios for the open system model with a correlated geometric-bounded workload. Section 5.3 presents mean response time results for the closed system model and highlights differences between the open and closed models. Section 5.4 presents response time ratios for the closed system model with an uncorrelated geometric-bounded workload. Section 5.5 presents response time ratios for the closed system model with a correlated geometric-bounded workload.

5.1. Open System Uncorrelated Geometric-bounded Workload

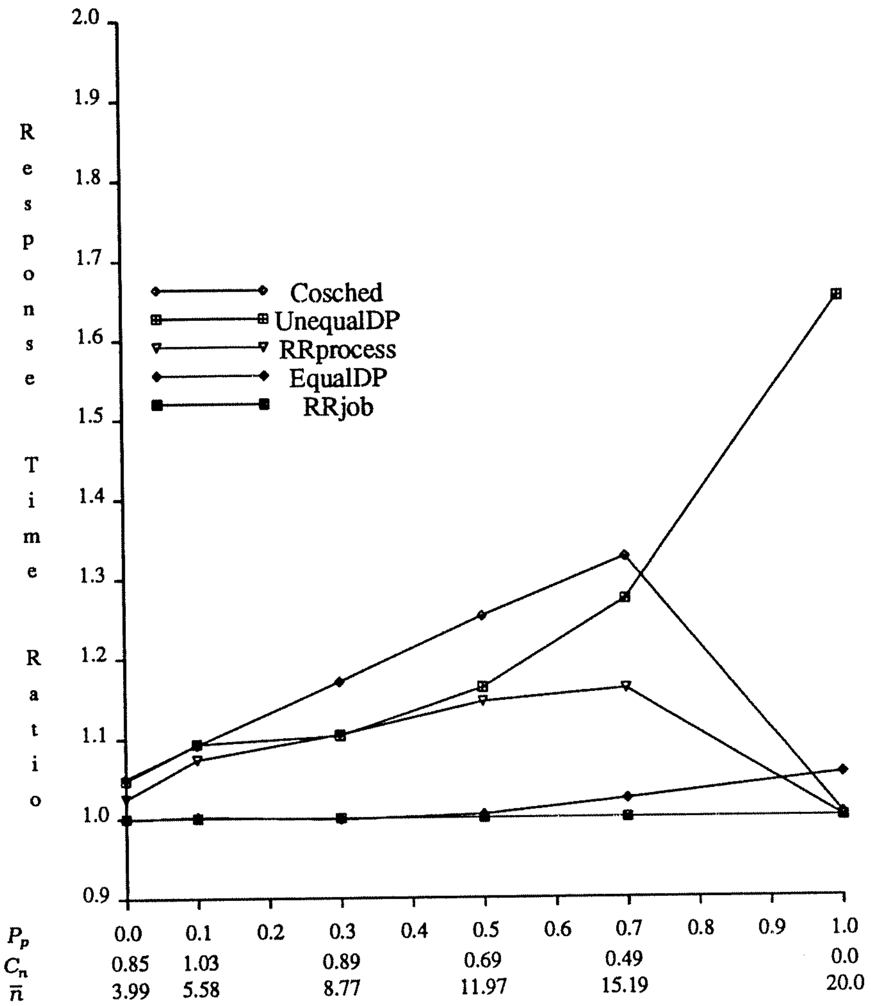
In this section we consider an open system with the uncorrelated geometric-bounded workload. Figure 5.1 plots the response time ratio versus system utilization for the baseline uncorrelated workload. As system utilization increases, the policies that do not provide equal allocation of processing power per job perform worse relative to RRjob. At 70% utilization Cosched, RRprocess, and UnequalDP perform 22.5%, 13.7%, and 11.2% worse respectively than RRjob. The polices EqualDP and UnequalDP have mean response times twice as high at a utilization of 90% due to the load queue.

Figure 5.2 explores the effect of varying mean job parallelism by varying the parameter P_p. System utilization is set at 70%. The response time ratio versus P_p is plotted. As P_p increases, the output parameters C_n and $\bar{n}$ also change. We include these two measured output parameters on the x-axis. The response time ratios of Cosched and RRprocess initially increase as P_p is increased. At P_p = 1.0 the mean job response time of RRprocess, Cosched and RRjob are identical. This is because equal allocation per process is equivalent to equal allocation per job when all jobs have the same number of processes. As P_p increases, the response time ratio of the UnequalDP policy increases. At P_p = 1.0 UnequalDP has a mean response time that is 65% greater than the mean response time of



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		200.0	5.0
Output		10.36	0.79	199.3	4.97

Figure 5.1: Response Time Ratios, Baseline Uncorrelated

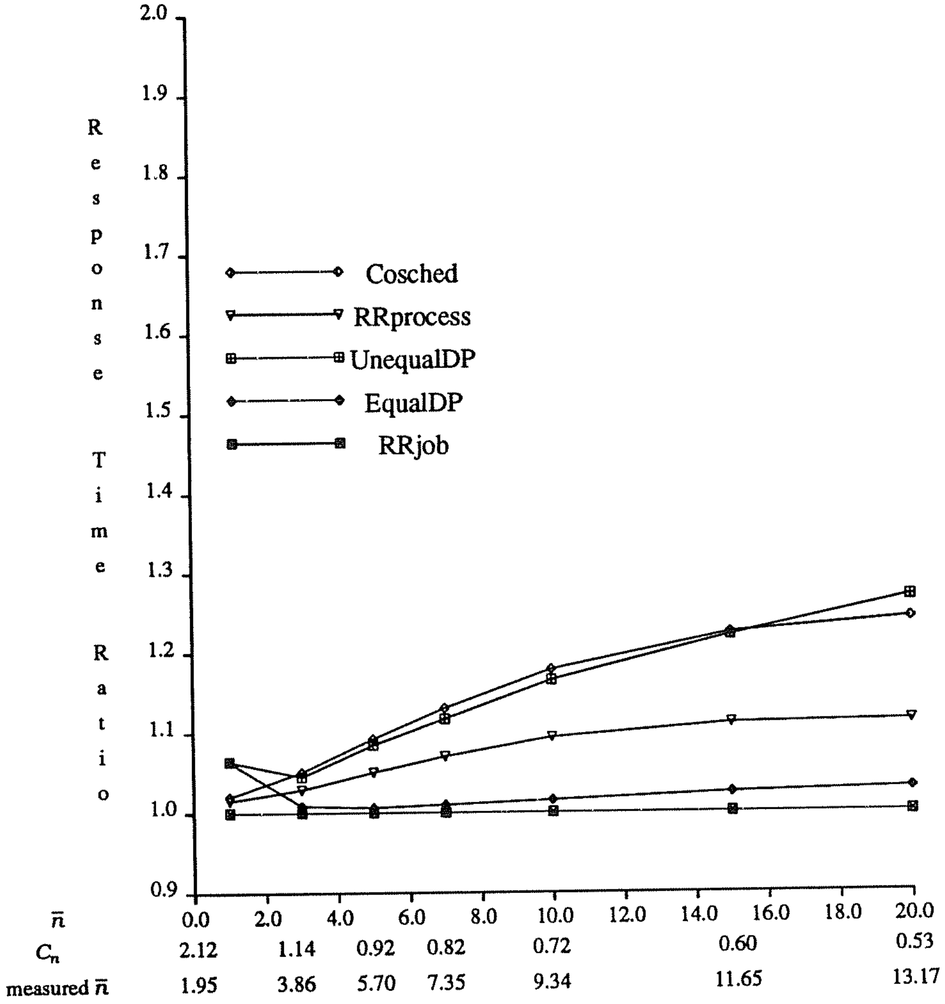


	P_p	$\bar{n}$	$\bar{d}$	C_d
Input	vary	4.0	200.0	5.0
Output			198.9	5.0

Figure 5.2: Response Time Ratios, P_p Varied, Uncorrelated

RRjob. This is because scheduling UnequalDP results in the first job getting all processors in the system less one for each other job in the system. This implies that Equal allocation per job is more critical as job parallelism increases. The policies EqualDP and RRjob are not identical since neither are providing truly equal allocation, but the difference in response times is always less than 10%. This 10% differences is within confidence intervals of the simulations.

In figure 5.3 we vary mean job parallelism by varying the input parameter $\bar{n}$. Figure 5.3 plots the response time ratio versus $\bar{n}$ with utilization set to 70% and P_p set to 0.05. Measured output parameters C_n and $\bar{n}$ are included on the x-axis. The response time ratios of RRprocess and Cosched become larger as $\bar{n}$ increases. When input parameter $\bar{n}$ = 20,



	P_p	$\bar{n}$	$\bar{d}$	C_d
Input	0.05	vary	200.0	5.0
Output			199.3	4.97

Figure 5.3: Response Time Ratios, $\bar{n}$ Varied, Uncorrelated

UnequalDP, Cosched, and RRprocess have mean response times 27%, 24%, and 11.3% higher than RRjob. We see again that as the mean parallelism increases equal allocation per job becomes more important. Note that at $\bar{n}$ = 1.0 the mean response times of UnequalDP and EqualDP are 6.3% higher than RRjob. This is due to the load queue, as will be demonstrated in section 5.3.

5.2. Open System Correlated Geometric-bounded Workload

In this section we consider the importance of equal allocation per job for the correlated geometric-bounded workload. The qualitative behavior is the same as for the uncorrelated geometric-bounded workload, but the magnitude of the differences between the policies is larger. Thus, equal allocation per job is even more important when a job's demand is positively correlated with the number of processes per job.

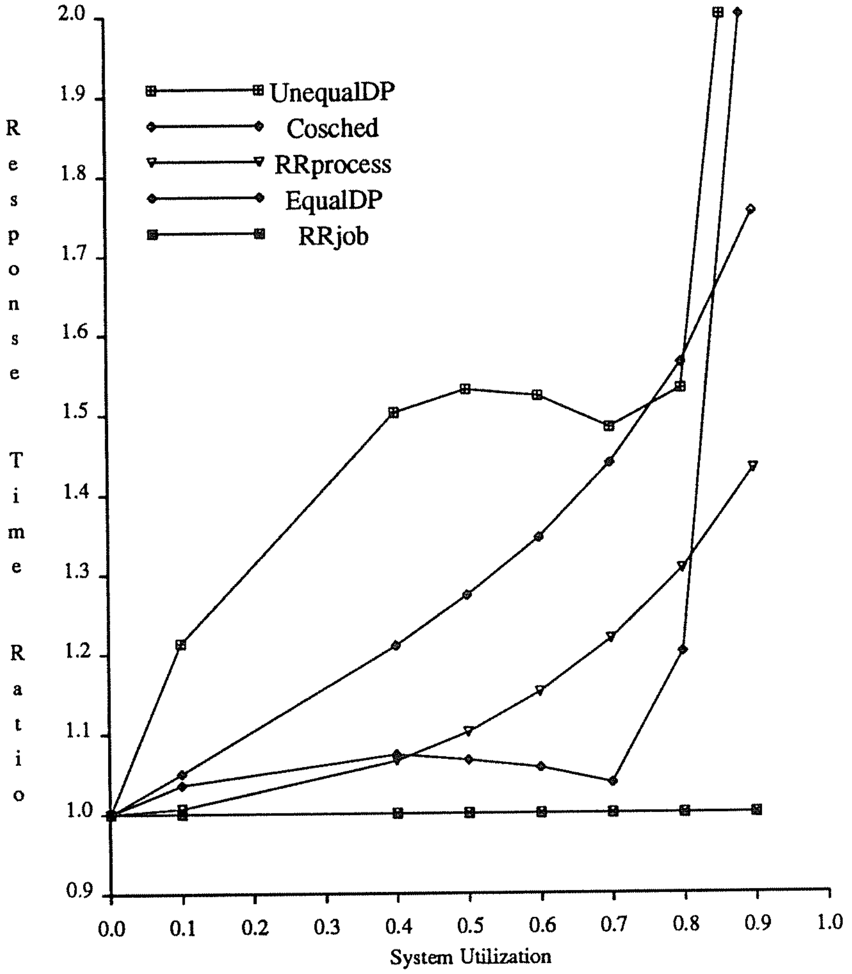
Figure 5.4 plots the response time ratio versus system utilization. At 70% utilization UnequalDP, Cosched, and RRprocess have mean job response time that are 48%, 44% and 22% larger respectively than RRjob. Once again at high utilizations the load queue assumption make UnequalDP and EqualDP much worse. The ratios of response times are considerably larger for the correlated workload than for the uncorrelated.

In figure 5.5 we vary mean job parallelism by varying the input parameter P_p. Figure 5.5 plots the response time ratio versus P_p with utilization set at 70%. The qualitative results are similar to the uncorrelated workload results, but the magnitude of the difference between policies is much larger. Once again we see that allocating processing power equally per job becomes more critical as mean job parallelism increases.

Figure 5.6 plots the response time ratio versus $\bar{n}$ for a utilization of 70%. As $\bar{n}$ increases RRprocess and Cosched improve while UnequalDP degrades. Unlike figure 5.3, RRprocess and Cosched improve relative to RRjob as $\bar{n}$ increases. This behavior can be attributed to the fact that as $\bar{n}$ increases, C_n decreases as can be seen on the x-axis. As we saw in chapter 4, Cosched and RRprocess are adversely affected by an increase in variation in the number of processes per job when job demand is correlated with the number of processes per job, hence we would expect a drop in the response time ratio as C_n decreases.

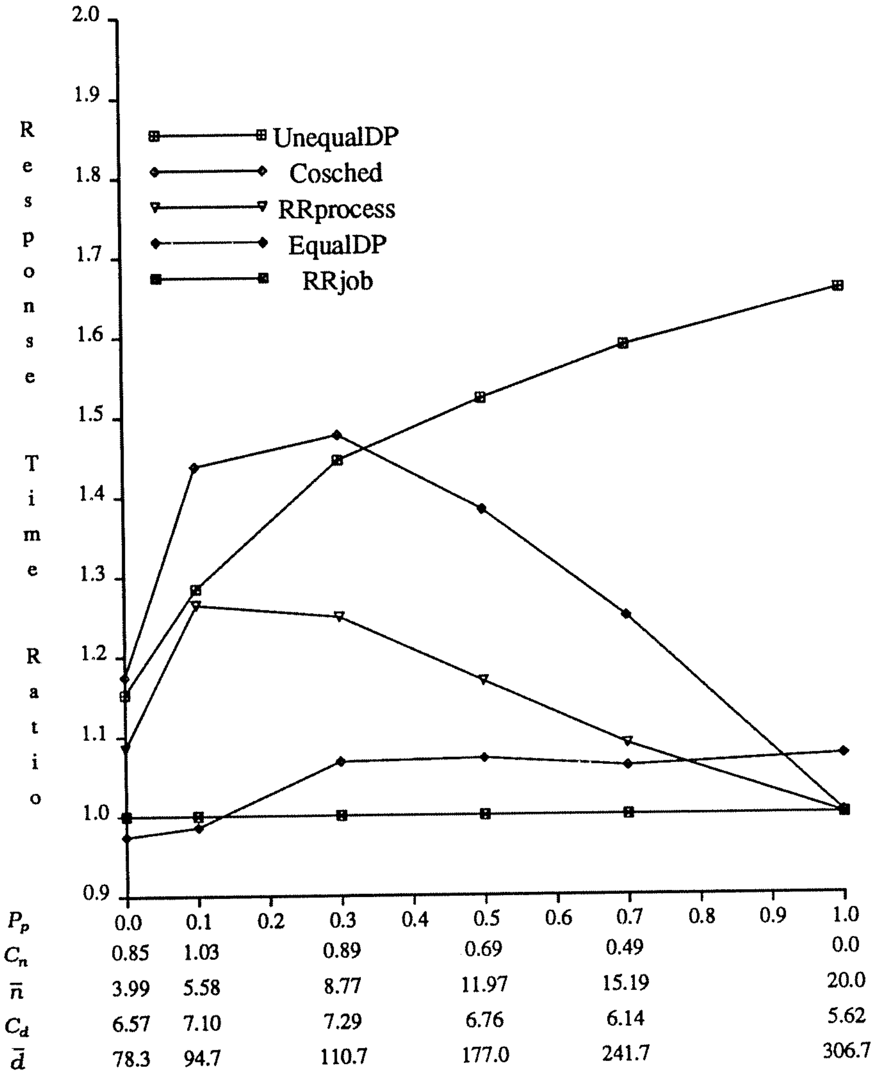
5.3. Closed System Model

The load queue assumption for EqualDP and UnequalDP causes these policies to perform poorly at high loads. As a result we are unable to compare the performance of these policies to other policies at high system utilizations. To enable a fair comparison between these policies it is necessary to remove the need for the load queue. We eliminate this need by using a closed system with a population of 20 jobs. In sections 5.4 and 5.5 we show response time ratios for the 5 policies studied in this chapter with the closed system. This allows us to better measure the importance of equal allocation per job at high utilizations.



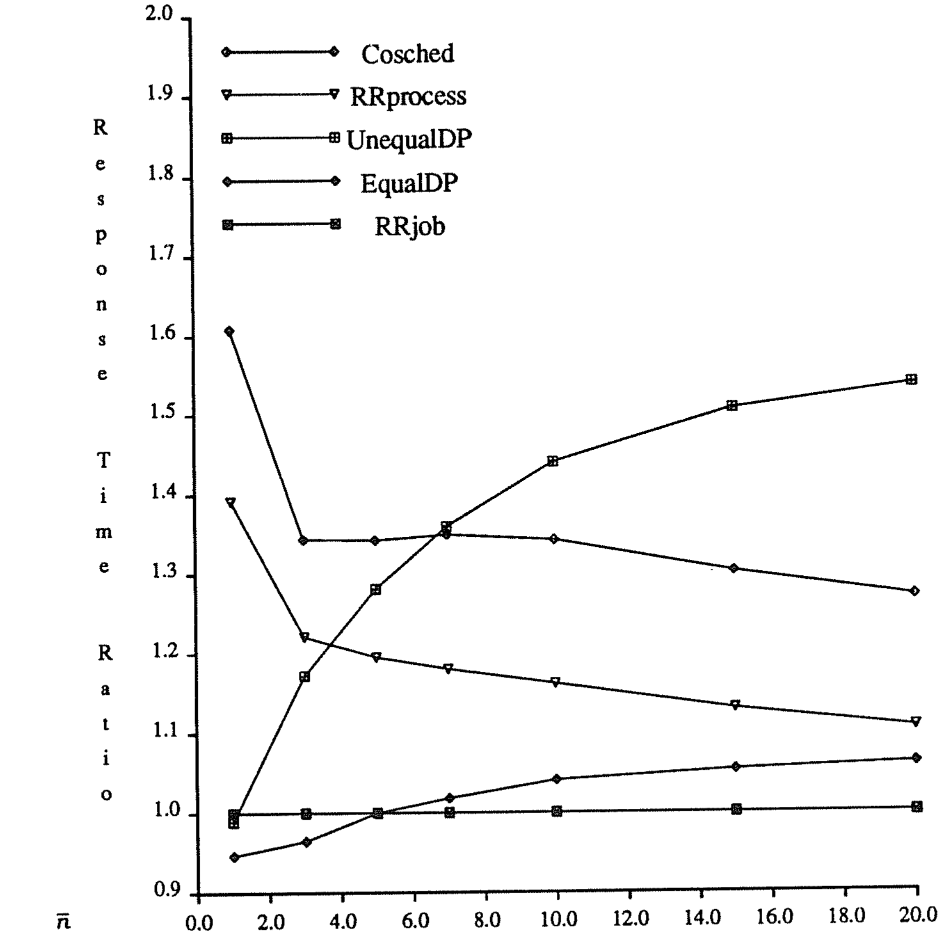
	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	C_v	t
Input	0.4	4.0				5.0	2.0
Output		10.36	0.79	210.0	6.48		

Figure 5.4: Response Time Ratios, Baseline Correlated



	P_p	$\bar{n}$	C_v	t
Input	vary	4.0	5.0	2.0

Figure 5.5: Response Time Ratios, P_p Varied, Correlated



C_n	2.12	1.14	0.92	0.82	0.72	0.60	0.53
measured $\bar{n}$	1.95	3.84	5.69	7.34	9.33	11.64	13.17
C_d	11.95	7.62	6.83	6.51	6.22	5.90	5.71
$\bar{d}$	38.1	75.9	112.8	145.6	185.9	232.2	262.8

	P_p	$\bar{n}$	C_v	t
Input	0.05	vary	5.0	2.0

Figure 5.6: Response Time Ratios, $\bar{n}$ Varied, Correlated

For all closed system results there are 20 jobs in the system. Think time is varied to change the load on the system. Unlike the open model where a given arrival rate results in all policies having the same utilization, a given think time does not result in all policies having the same utilization. This is because policies that have smaller mean response times service jobs faster, which results in increased throughput and higher utilizations. Throughputs can be derived from Little's results.

Before we present the response time ratio results for the closed system we will compare the behavior of the policies for the closed system to that for the open system. We present these results to show that the qualitative results do not change as we move from an open system to a closed system.

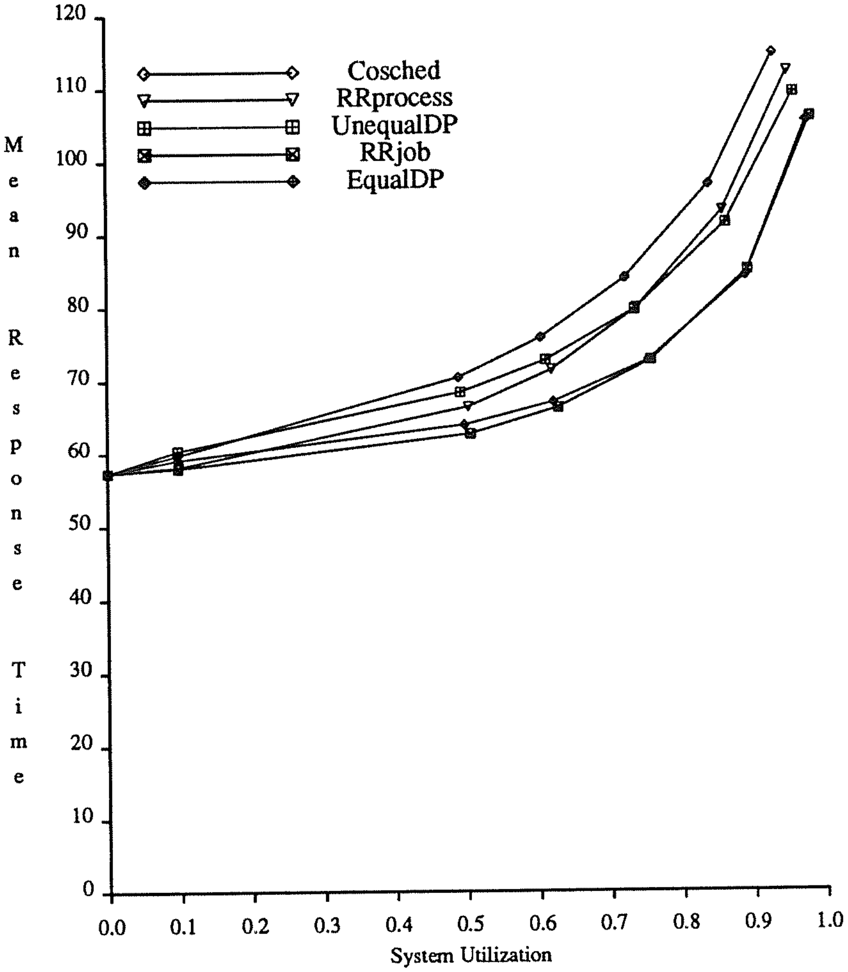
Figure 5.7 plots mean response time versus system utilization for the baseline uncorrelated geometric-bounded workload presented in section 4.3. There are seven points on the curve plotted for each of the policies. These seven points correspond to think times of 50000, 1900, 320, 240, 180, 130, and 90. At a think time of 90, the resultant utilizations range from 93% for Cosched to 98% for RRjob. In the closed system with 20 jobs, at most 20 jobs can ever be queued for service. As a result the policies EqualDP and UnequalDP never place jobs in a load queue. Hence EqualDP and UnequalDP do not degrade at high utilization due to a load queue as they did in the open system. Note that the same ordering of policy performance appears in the closed system as in the open system (see figure 4.6). The performance difference between the policies is smaller because there are never more than 20 jobs in the system at once.

Figure 5.8 plots mean response time versus system utilization for the baseline correlated geometric-bounded workload presented in section 4.3. The seven points plotted for each policy are for think times of 400000, 2500, 380, 230, 180, and 130. At a think time of 130, the resultant utilizations range from 92% for UnequalDP to 99.9% for RRjob. As in the uncorrelated workload, Once again, the same ordering of policies appears in the closed system as in the open system (see figure 4.7).

The main conclusion of this section is that the qualitative ordering of the policies remains the same for the closed system model as for the open system model.

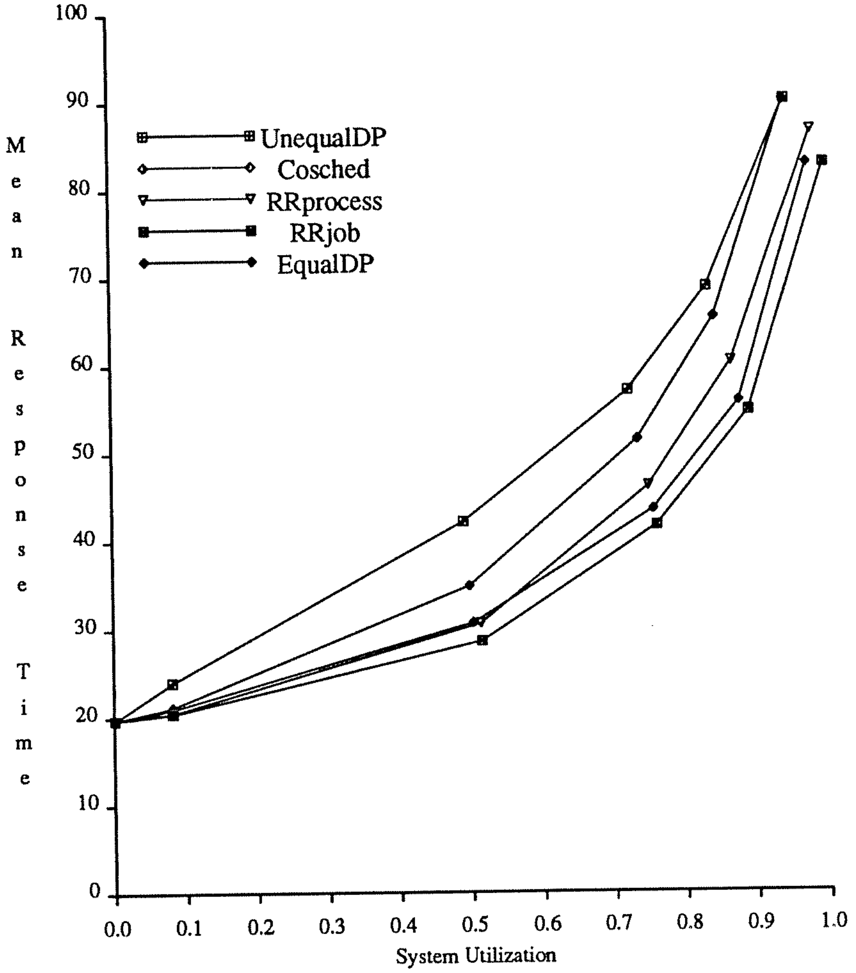
5.4. Closed System Uncorrelated Geometric-bounded Workload

The load queue effects colored the results for EqualDP and UnequalDP in sections 5.1 and 5.2. In this section we eliminate the problem by assuming a closed system with 20 jobs. The magnitude of the differences of response times are not as large in the closed system as in the open system since there can never be more than 20 jobs competing for service at once. Figures 5.9 - 5.11 correspond to the closed system versions of the experiments presented in section 5.1. Figure 5.9 plots the response time ratio versus the RRjob system utilization. We use the measured utilization of the RRjob policy for the x-axis coordinates. For figure 5.9 the corresponding think times are 1900, 320, 240, 180, 130, and 90 units. Figures 5.10 and 5.11 show the effect of increasing mean job parallelism by increasing P_p and $\bar{n}$ respectively. Think time is set equal to 180 units. As the mean job parallelism increases, response time decreases causing system utilization to increase. We draw the same conclusions from these three graphs as we drew for the open system, policies that provide equal allocation per job result in better performance than policies that do not



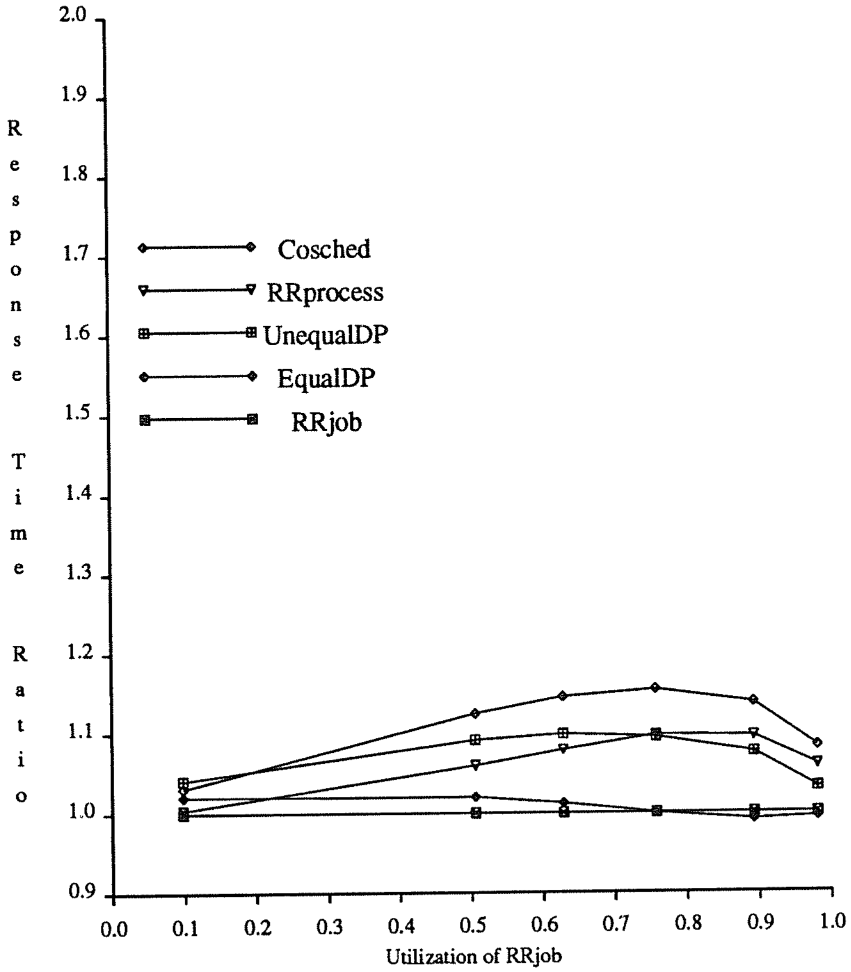
	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0	200.0	5.0	
Output		10.44	0.79	192.4	5.05

Figure 5.7: Response Times, Closed System Uncorrelated



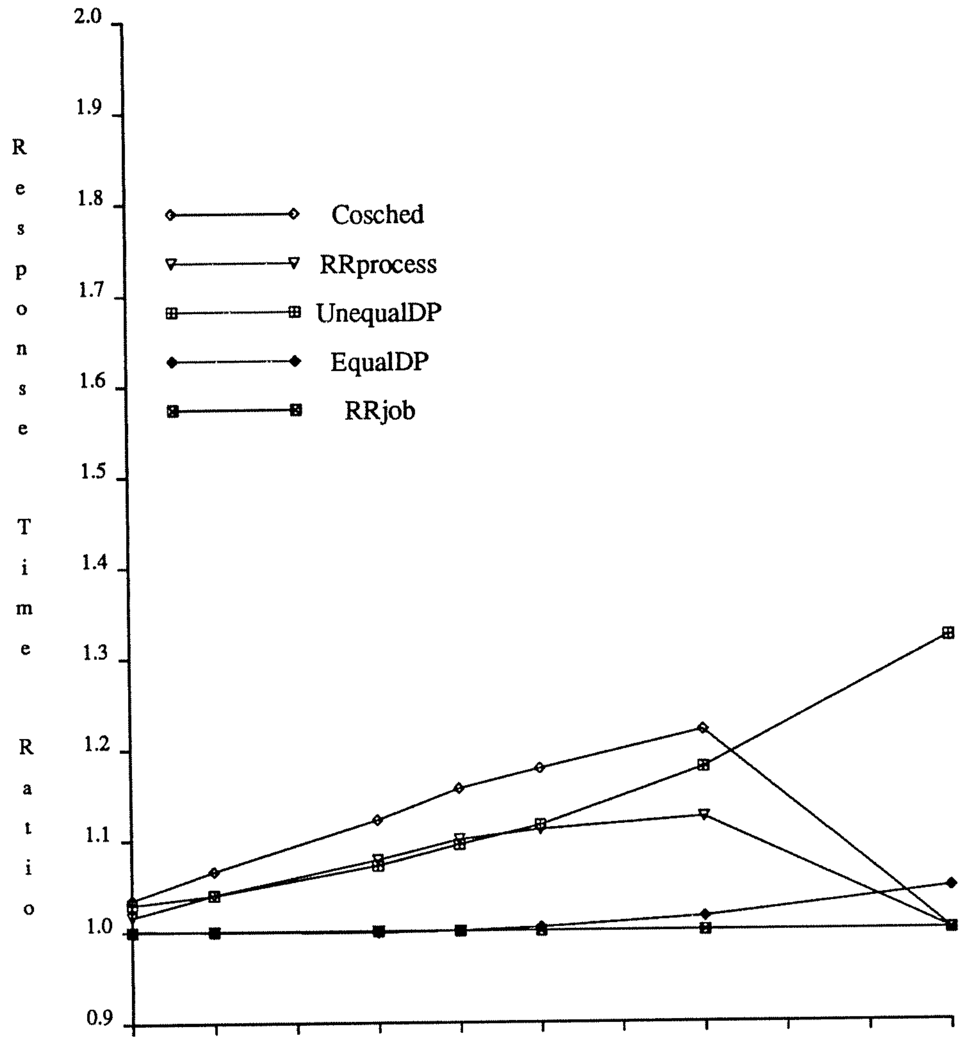
	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	C_v	t
Input	0.4	4.0				5.0	20.0
Output		10.42	0.79	207.8	6.42		

Figure 5.8: Response Times, Closed System Correlated



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		200.0	5.0
Output		10.44	0.79	192.8	5.06

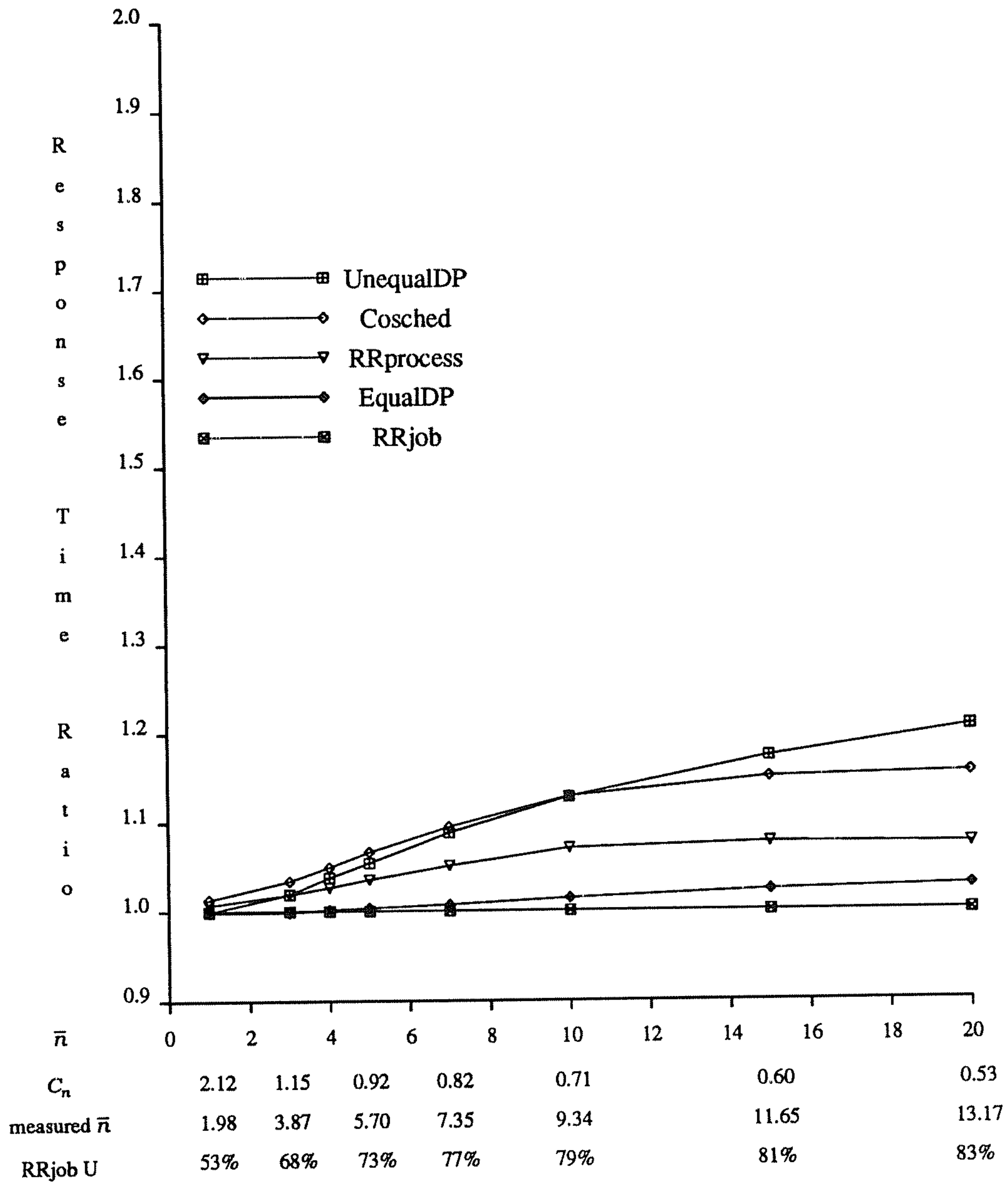
Figure 5.9: Response Time Ratios, Closed System, Uncorrelated



P_p	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
C_n	0.85	1.03		0.89	0.79	0.70		0.50			0.00
$\bar{n}$	3.99	5.59		8.79	10.37	11.95		15.17			20.0
RRjob U	71%	73%		76%	77%	79%		82%			88%

	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	vary	4.0		200.0	5.0
Output				197.5	4.96

Figure 5.10: Response Time Ratios, P_p Varied
Closed System Uncorrelated



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.05	vary		200.0	5.0
Output			192.8	5.06	

Figure 5.11: Response Time Ratios, $\bar{n}$ Varied
Closed System Uncorrelated

provide equal allocation per job, and that equal allocation becomes more important as the mean job parallelism increases. Note that in figure 5.11 the mean response time for RRjob, EqualDP and UnequalDP at $\bar{n} = 1.0$ are equal now that there is no load queue effect.

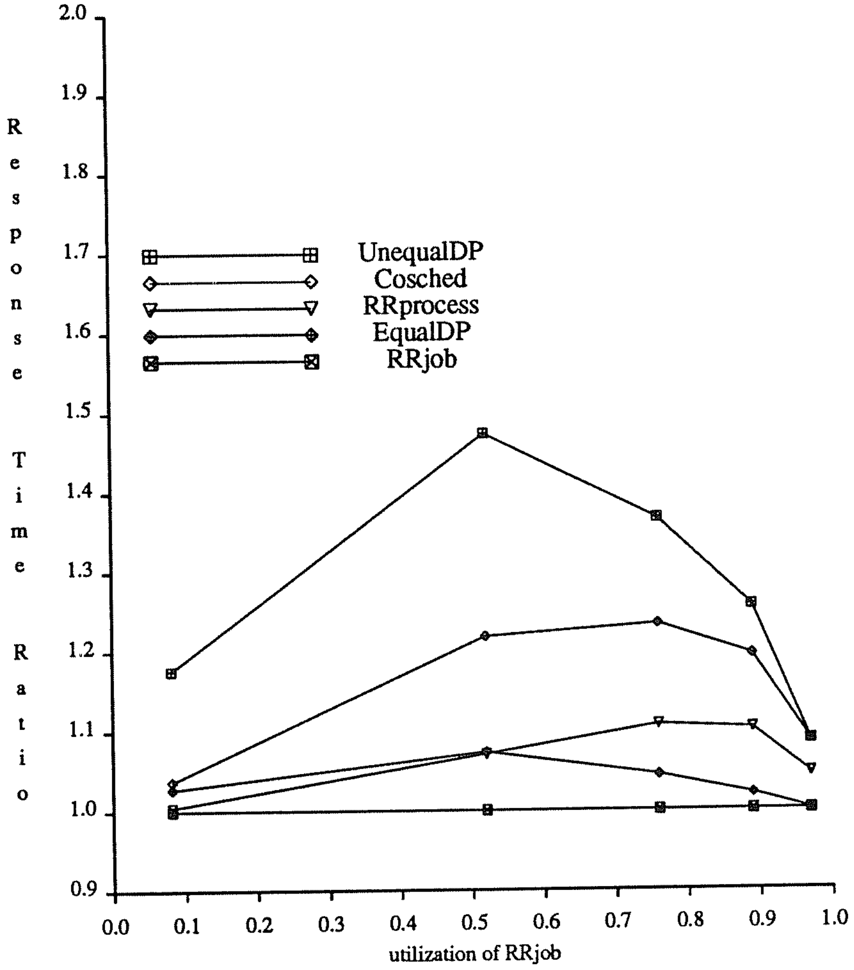
5.5. Closed System Correlated Geometric-bounded Workload

As in the section 5.4 we consider the closed system for the correlated geometric-bounded workload to remove any effects of the load queue assumption. We only present the graph for the changing RRjob utilization. Figure 5.12 plots the ratio versus RRjob utilization. Think times for the figure are 400000, 2500, 380, 230, 180, and 130 units. We now see that at high utilizations EqualDP performs as well as RRjob.

5.6. Conclusion

We find that policies that provide equal allocation per job result in lower mean response times than polices that do not provide equal allocation of processing power per job. In particular:

- For the uncorrelated workload, policies that do not provide equal allocation per job result in mean job response times that are 10% - 20% greater and up to 65% greater for extreme parameter values than the mean response time of policies which do provide equal allocation of processing power per job.
- For the correlated workload, policies that do not provide equal allocation per job often result in mean job response times that are 15% - 50% greater than the mean response times of policies that do provide equal allocation per job.
- Providing equal allocation per job becomes more important as mean job parallelism increases.
- Providing equal allocation of processing power per job is less important for the closed system than the open, but mean job response times of policies that do not provide equal allocation are still often 10% - 20% higher than polices that do provide equal allocation per job.



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	C_v	t
Input	0.4	4.0				5.0	20.0
Output		10.42	0.79	207.8	6.42		

Figure 5.12: Response Time Ratios, Closed System Correlated

CHAPTER 6

Importance of Supporting Inter-process Synchronization

In this chapter we investigate the importance of supporting inter-process synchronization. We consider both coscheduling and application level scheduling as approaches to facilitating inter-process synchronization. Coscheduling means that when a process is scheduled, all other processes from the same job will most likely be scheduled at the same time. Application level scheduling implies the existence of a two level scheduler, a system level scheduler for allocating processors to jobs, and an application level scheduler that determines which processes of a job are to run on the processors currently allocated to the job. A more detailed description is given in chapter 1.

We concentrate only on the policies RRjob, EqualDP, and Cosched. The policies EqualDP and Cosched explicitly support inter-process synchronization whereas RRjob does not. EqualDP provides support through application level scheduling. Cosched provides support through coscheduling. We do not consider UnequalDP because we expect inter-process synchronization to affect UnequalDP the same way it affects EqualDP. We do not consider RRprocess since it does not offer better inter-process synchronization support than RRjob, and in general it has worse performance than RRjob. We do not consider the policies FCFS, SNPF, and PSNPF because they exhibit poor performance under a variety of simple workloads, as shown in chapter 4, and are thus not very promising policies.

Of the policies considered, EqualDP and RRjob have been shown in the previous two chapters to perform better than Cosched since Cosched does not provide equal allocation per job. By including synchronization when comparing these policies we can determine whether the support for inter-process synchronization changes the relative ordering of the policies. In particular, we wish to determine whether the benefits of coscheduling can overcome the benefits derived from equal allocation per job. Our approach is to study the performance of the policies in the presence of spin-lock and barrier synchronization and draw conclusions about the importance of support for inter-process synchronization from the results of our experiments.

All experiments in this chapter, other than the first one, assume a closed system with 40 jobs. With the inclusion of interprocess synchronization, the simulation time required for the open system is too prohibitive, often up to eight days per data point. The closed system allows us to decrease simulation time by 60%. All results in this chapter have confidence intervals of 10% or less (usually less than 5%) at the 90 percent confidence level. Confidence intervals are again calculated using batch means with 20 batches per simulation run. Other than the first experiment, all uncorrelated workload experiments in this chapter have 2,000 samples per batch, and all correlated workload experiments have 3,000 samples per batch.

Section 6.1 presents the results for spin lock synchronization. Section 6.2 presents the results for barrier synchronization. Section 6.3 contains the conclusions of this chapter.

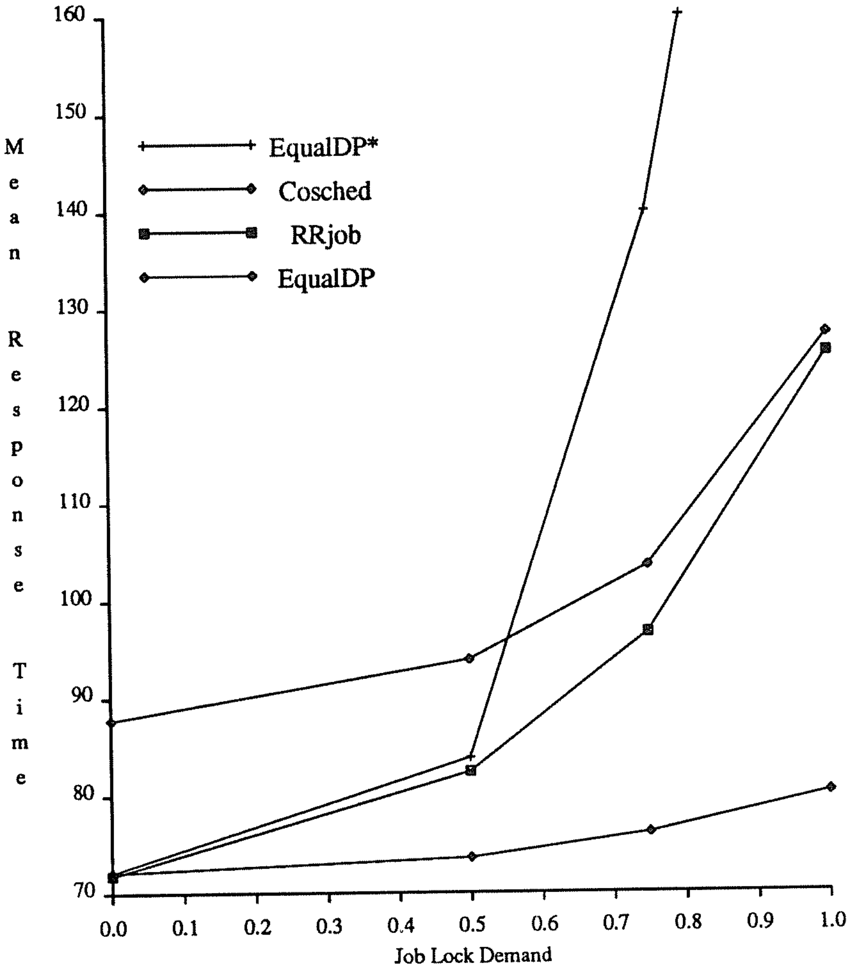
6.1. Spin Lock Synchronization

In this section we explore the effect of spin-lock synchronization on the relative performance of the policies. We assume processes from the same job compete for a single shared lock. The workload model is described in section 3.5.1. Processes spin when they request a lock that is held by another process. There are two types of spinning overhead that occur. The first is spinning while the requesting process waits for a process that is currently scheduled and holding the lock to relinquish the lock. This type of spinning occurs in all policies. However, a higher degree of coscheduling leads to higher competition and thus more spinning. The second type of spinning overhead occurs when a process requests a lock held by a process that is descheduled. In this case the requesting process must wait (spin) for the processes holding the lock to be rescheduled and release the lock. RRjob is prone to this problem since it preempts processes. Coscheduling and application level scheduling attempt to eliminate the second type of spinning. While Cosched does preempt processes, all other processes from the same job are almost always descheduled at the same time. As a result there are few points in time where only a fraction of the job's processes are scheduled, hence if a processes holding a lock is descheduled it is less likely to cause spinning than in RRjob. EqualDP uses application level scheduling to eliminate the second type of spinning. The application level scheduler makes sure that a process holding a lock is never descheduled while a process requesting the lock is spinning. The policy RRjob might use application level scheduling to avoid the second type of spinning, but it is not included in the experiments we perform in this section.

In all of the figures presented in this section we plot mean response time versus job lock demand. Lock holding time is deterministic and equals 0.01 units where one unit is a quantum in the RRprocess policy. We define job lock demand as (process lock demand) × (number of processes), where process lock demand is the percent of time each process would use the lock if it never has to spin. For example, if job lock demand is 100% and the number of processes is 10, then the process lock demand is 10%, hence the inter-request time is 0.09 units. For a job with 20 processes running by itself on the machine, a job lock demand of 100% results in the lock being utilized 84.6 percent of the time. A job lock demand of 100% is more contention for a lock than is reasonable for a real parallel program. A job lock demand of below 50% is a more likely operating range.

Figure 6.1 plots mean response time versus job lock demand for the uncorrelated exponential-bounded workload assuming an open system. The exponential-bounded workload is nearly identical to the geometric-bounded workload. The difference is that the truncated geometric distribution is replaced by a truncated exponential distribution and the ceiling of the sample is used. The graph is from earlier work before the geometric workload was defined. The no-synchronization utilization is set at 70%. We define no-synchronization utilization as the utilization of the system for RRjob assuming no-synchronization. The utilization of the system increases as job lock demand increases due to the extra work introduced by the spinning.

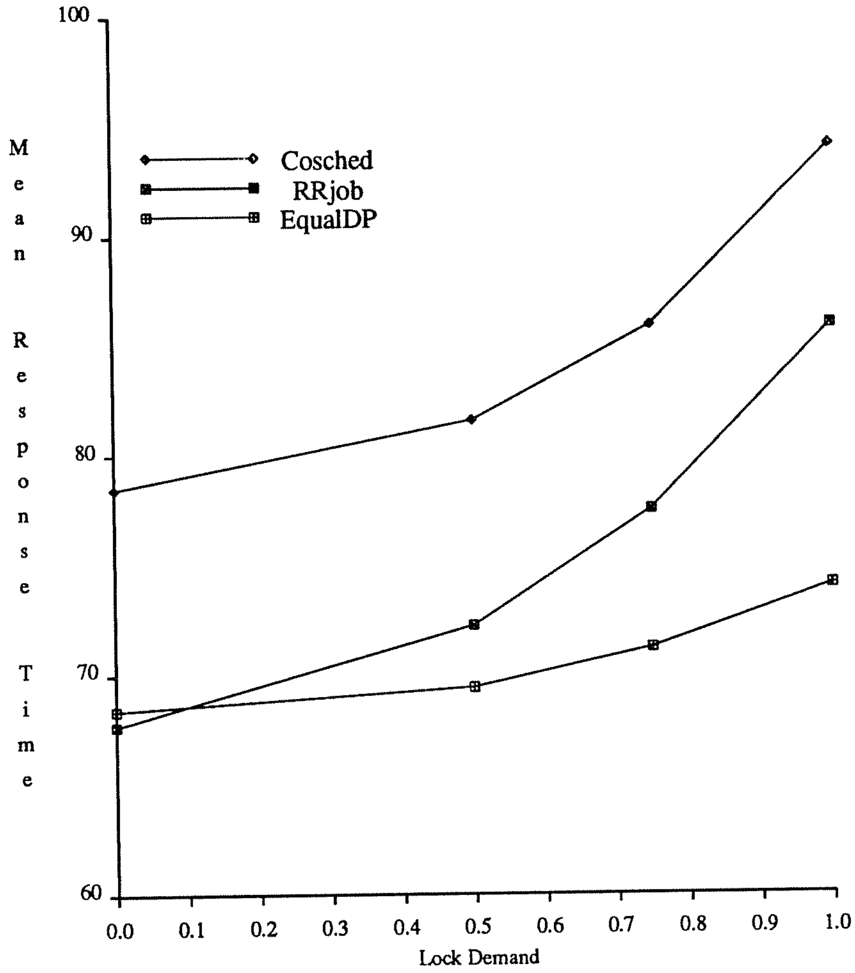
At zero job lock demand Cosched has a larger mean response time because it does not provide equal allocation per job. As job lock demand increases the mean response time of all policies increases. Policies which provide better support for spin-lock synchronization are less affected. The mean response times of Cosched, RRjob, and EqualDP are 75%, 45%, and 11.3% larger respectively at a job lock demand of 100% than for zero job lock demand. Even though RRjob is more affected by the spin-lock synchronization than Cosched, it still performs better because it provides equal allocation per job. In the likely ranges of less than 50% job lock demand RRjob performs considerably better than Cosched. RRjob does not explicitly support inter-process synchronization, but it does exhibit a reasonable degree of coscheduling. The mean response time of the EqualDP policy increases only slightly as job lock demand increases because the policy provides good support for spin-lock synchronization. EqualDP performs best of all the policies because it provides the best support for inter-process synchronization in addition to providing equal allocation per job. Cosched does perform better than RRprocess (not shown), but not until job lock demand exceeds 70%. We include a variation of EqualDP in this experiment called EqualDP*, which is the same as EqualDP except that it does not assume application level scheduling. Processes within a job a placed in a queue randomly and then serviced FCFS. As a result, processes holding locks can be descheduled for long periods of time. This accounts for the poor performance of EqualDP* at high lock demands. At a lock demand of 85%, the mean response time for EqualDP* is 2192. Note that another alternative to an application level scheduler for EqualDP not considered in this thesis is to time slice the processors allocated to a job between the job's processes. In this case processes holding a lock could not be descheduled for long periods of time, and an application level scheduler is not needed.



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	U^*
Input	0.4	4.0		200.0	5.0	70%
Output		10.68	0.76	197.5	4.97	

Figure 6.1: Spin-lock Synchronization
Uncorrelated Exponential-bounded

Figure 6.2 plots response time versus job lock demand for the uncorrelated geometric-bounded workload assuming a closed system. Think time is set to 550 units resulting in RRjob having a no-synchronization utilization of 62%. The workloads for figures 6.1 and 6.2 are essentially the same, the two differences are that the system is closed, and the no-synchronization utilization is lower. Once again EqualDP performs best, followed by RRjob

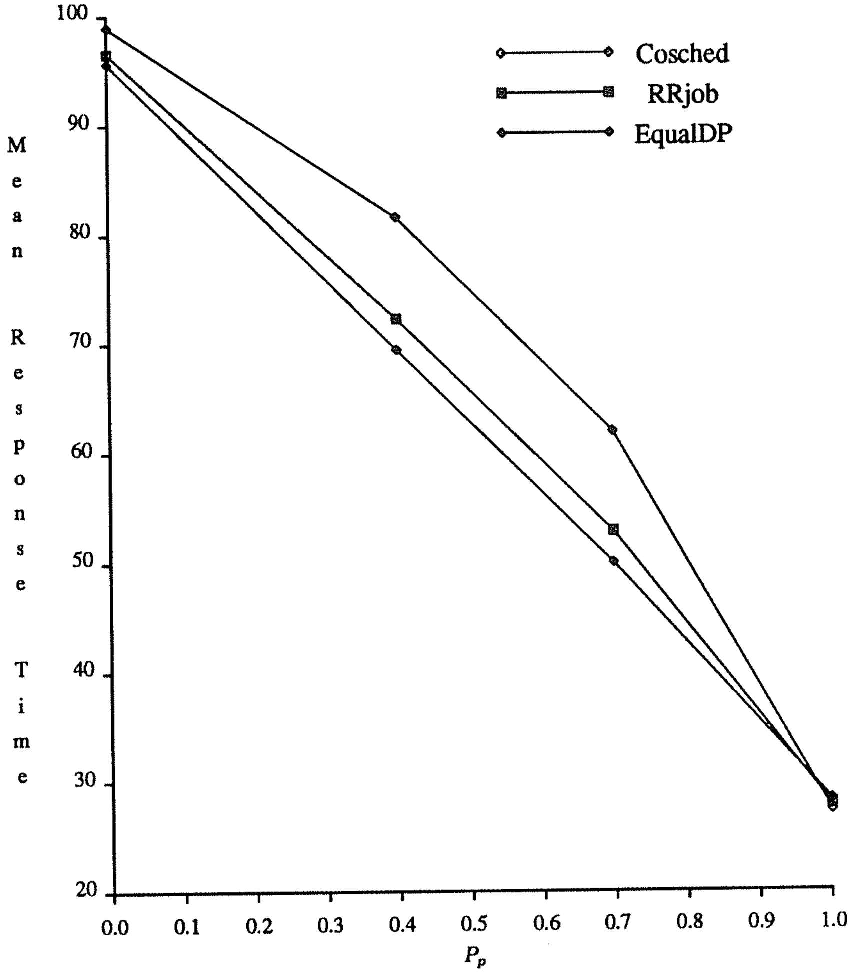


	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	U^*
Input	0.4	4.0		200.0	5.0	62%
Output		10.44	0.79	192.4	5.05	

Figure 6.2: Spin-lock Synchronization, Closed System Uncorrelated

and then Cosched. The differences in the performance of the policies are not as significant. This is because the lower utilization and closed system cause the queue lengths to be shorter so that the impact of descheduling processing holding a lock is not as significant.

In figure 6.3 parameter P_p is varied while all other parameters are held constant for the uncorrelated geometric-bounded workload assuming a closed system. Think time is set to 550 units as in figure 6.2. Job lock demand is equal set equal to 50%. The maximum



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	U^*
Input	varied	4.0		200.0	5.0	60%
Output				192.4	5.05	

Figure 6.3: Spin-lock Synchronization, P_p Varied
Closed System, Uncorrelated

difference between the policies is seen for P_p in the 40% - 70% range. When we varied P_p without synchronization, figure 4.13, we saw the same behavior. This indicates that the way processing power is divided among jobs may be a more important characteristic of a scheduling policy than the ability to facilitate spin-lock synchronization.

The qualitative behavior of the policies for the correlated workload (graphs not shown) is the same as the qualitative behavior for the uncorrelated workload shown in figures 6.1

and 6.2. We conclude from this section that the ability of a policy to support spin-lock synchronization does not appear to be as important as providing equal allocation per job as long as a policy provides a reasonable degree of coscheduling (such as RRjob). If a process holding a lock can be descheduled for long periods of time as in EqualDP*, then the performance can suffer dramatically.

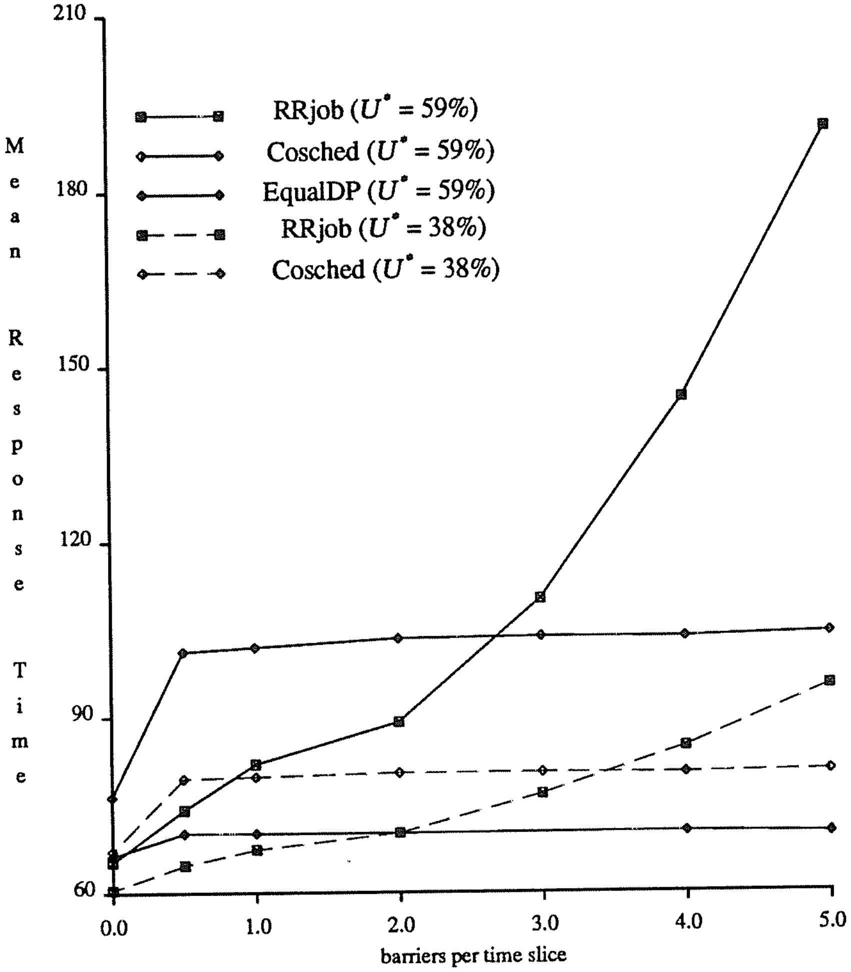
6.2. Barrier Synchronization

In this section we explore the effect of barrier synchronization on the relative performance of the scheduling policies. The barrier synchronization workload is described in section 3.4.2. Our goal is to determine the importance of a scheduling policy's ability to support barrier synchronization. A secondary goal is to determine how much of a performance improvement results from application level scheduling versus coscheduling. We first explore the impact of barrier synchronization on the relative performance of the polices RRjob, Cosched and EqualDP. We then investigate how including application level scheduling for RRjob and Cosched affect the relative performance of the policies.

6.2.1. Impact of Barrier Synchronization

In this section we compare the policies RRjob, Cosched, and EqualDP in the presence of barrier synchronization. In the previous section we have shown that the Cosched policy results in worse performance than RRjob even in the presence of spin-lock synchronization. In this section we wish to see whether Cosched's support of barrier synchronization can offset the performance loss due to unequal allocation.

Figure 6.4 plots mean response time versus barriers per time slice for the uncorrelated geometric-bounded workload assuming a closed system model. The solid lines are for a no-synchronization utilization of 59%, whereas the dashed lines are for a no-synchronization utilization of 38%, these utilizations correspond to think times of 600 and 1000 units respectively. EqualDP is only included for the 59% no-synchronization utilization. The RRjob policy has a lower mean response time than Cosched at 0 barriers per time slice, but crosses over at around 2.7 (3.5) barriers per time slice for the 59% (38%) no-synchronization utilization workload. This demonstrates that support of barrier synchronization is important if there is frequent barrier synchronization. When the frequency of barrier synchronization is in a more likely range, 1 barrier per time slice or less, Cosched performs worse than RRjob due to the unequal allocation per job. The policy EqualDP performs significantly better than RRjob and Cosched. For a no-synchronization utilization of 59% with 5 barriers per quantum the mean response time of RRjob and Cosched are 172% and 49% larger respectively than the mean response time of EqualDP. RRjob has a mean response time 17% larger than EqualDP when there is 1 barrier per quantum, and 6% larger when there are 0.5 barriers per quantum.



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		200.0	5.0
Output		10.45	0.79	197.4	5.03

Figure 6.4: Barrier Synchronization, Closed System, Uncorrelated

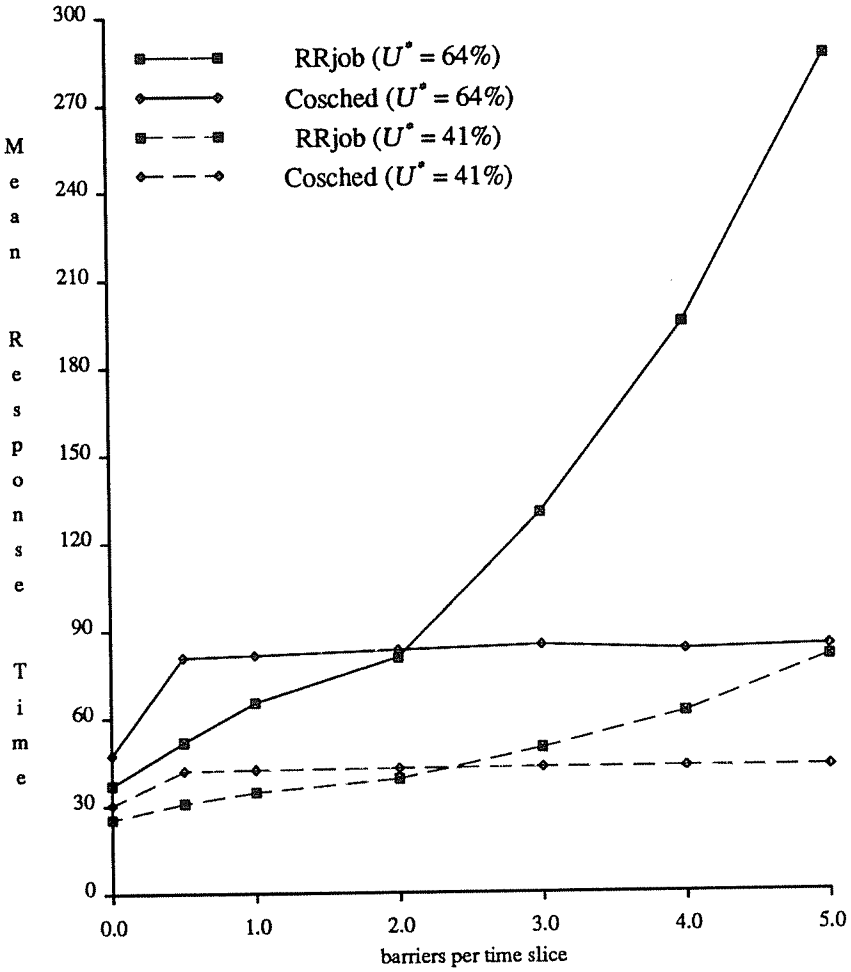
We find that supporting barrier synchronization is less important as offered utilization decreases. We see that the difference between Cosched and RRjob is smaller for a no-synchronization utilization of 39% than for 59%. For a no-synchronization utilization of 59% and 5 barriers per quantum, RRjob is 83% worse than Cosched whereas for a no-synchronization utilization of 39% RRjob is 18% worse than Cosched. The reason is that at lower utilizations processes are descheduled less often.

The increase in response time for all the policies is caused by processes spinning upon reaching the barrier. We now offer explanations for why the response time curve for Cosched remains flat, and why the response time curves for RRjob keep increasing. First we explain why the Cosched curves becomes flat. When a process reaches a barrier, all the other process in the job are currently scheduled unless the process belongs to a partially scheduled job at the end of a scheduling window. For sake of the argument, let us assume perfect coscheduling, hence when a processes reaches a barrier all other processes from the job are currently scheduled. Since our workload states all processes reach the barrier at the same time ± 10% of the inter-request time, the ratio of spinning to useful work remains constant as the inter-request time is decreased. Hence the flat line.

For RRjob it is not true that when a processes reaches a barrier all the other processes will be currently scheduled. To explain why RRjob keeps degrading as the amount of barrier synchronization increases let us illustrate by an example. Assume a system with 4 processors and 2 jobs, X and Y, with 4 processes each. Let us denote the processes X_1 ... X_4, and Y_1 ... Y_4. Assume job X currently has all four processes scheduled on the processors. Assume job Y has just reached a barrier and barrier inter-request time is orders of magnitude smaller than the quantum length. Assume the four processors context switch exactly 0.25 quantums apart, hence process Y_1 is scheduled at time t, Y_2 is scheduled at time $t + 0.25$, Y_3 at time $t + 0.5$, and Y_4 at $t + 0.75$. Since Y_1 reaches the barrier almost immediately, it spins until Y_4 is scheduled 0.75 units later. Once Y_4 is scheduled the 4 processes can do useful work, but Y_1 is descheduled at time $t + 1$. Hence, every time a process gets scheduled it only executes productively 25% of the time. In a system with 20 processors and jobs with 20 processes the problem is further exasperated. The amount of spinning can be more or less than our example depending on when the processors context switch. The example considers the case when the inter-request time is vary small relative to the quantum size, but the argument extends to less frequent synchronization. Thus, as the inter-request time decreases, the ratio of spinning time to useful work increases. At very frequent synchronization, such as in our example, but not shown in our results, we would expect the mean job response time of RRjob to plateau.

From figure 6.4 we also note that Cosched experiences a larger initial rise in mean response time than RRjob. For a think time of 600 units, the mean job response time of Cosched increases by 33%, whereas the mean response time of RRjob increases by only 13% when the amount of barrier synchronization is increased from no synchronization to 0.5 barriers per quantum. This is because the higher degree of scheduling causes more spinning near the time that all processes reach the barrier, whereas RRjob deschedules processes after reaching the barrier due to quantum expiration.

Figures 6.5 plots mean response time versus barriers per quantum for the correlated geometric-bounded workload assuming a closed system. The solid lines are for a no-



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	C_v	t
Input	0.4	4.0				5.0	2.0
Output		104468	0.79	205.1	6.46		

Figure 6.5: Barrier Synchronization, Closed System, Correlated

synchronization utilization of 64%, whereas the dashed lines are for a no-synchronization utilization of 41%. The corresponding think times are 600 and 1000 units. Qualitatively the behavior is similar to the uncorrelated workload.

We conclude from these results that coscheduling is an important characteristic of a scheduling policy in the presence of frequent barrier synchronization. Coscheduling is not as important as processing power allocation when barrier synchronization occurs less frequently than twice per quantum. We would anticipate that many parallel programs would synchronize less frequently than once per quantum, hence it appears that division of processing power per job is at least as important as the ability to support barrier synchronization.

6.2.2. Inclusion of Application Level Scheduling

In section 6.2.1 we showed that coscheduling can result in significantly better performance than RRjob if there is frequent barrier synchronization and no application level scheduling. In this section we add application level scheduling to the policies RRjob and Cosched. In the presence of application level scheduling three actions may be taken when a process reaches a barrier:

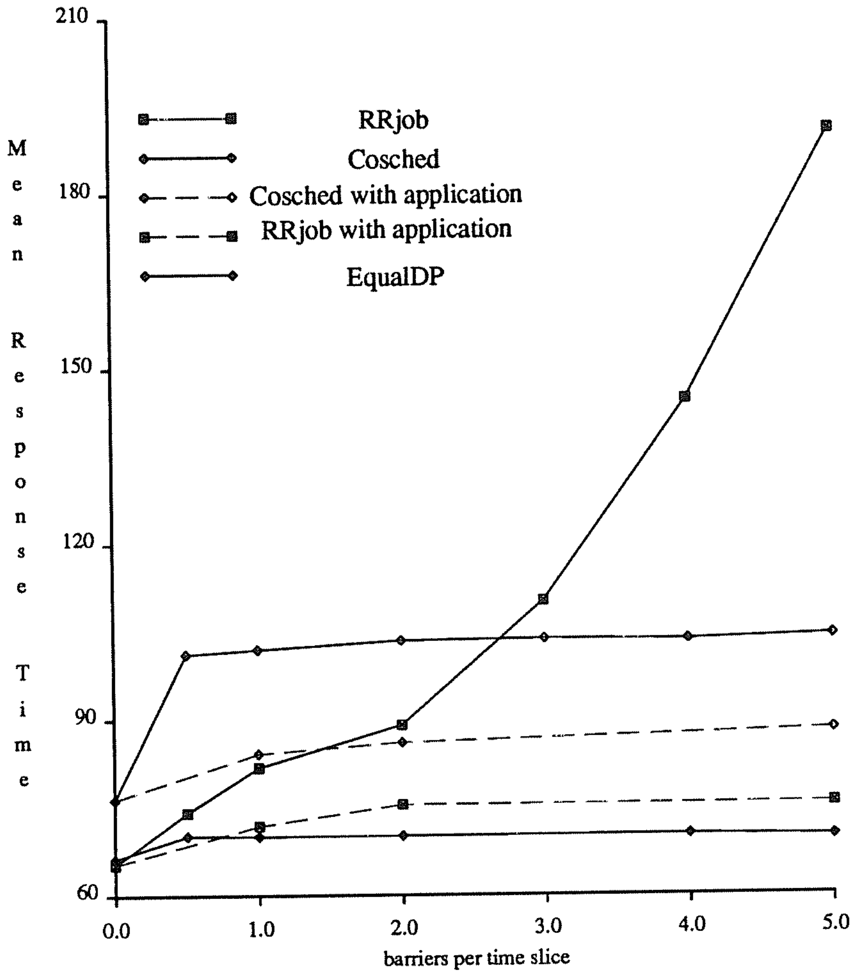
1) The process is the last to reach the barrier. In this case all other processes that may be spinning start productive work again.

2) The process is not the last to reach the barrier and the job has other processes currently not executing that could be doing productive work. In this case we assume the process is descheduled and another process from the same job which has productive work to do is scheduled in its place.

3) The process is not the last to reach the barrier and the job has no other processes not executing that could be doing productive work. In this case the process spins until either the quantum expires or the barrier is reached by all other processes.

Figure 6.6 plots mean response time versus barriers per time slice for the uncorrelated geometric-bounded workload assuming a closed system. Think time is set to 600 resulting in a no-synchronization utilization of 59%. The solid lines are the same as in figure 6.4, that is RRjob and Cosched do not assume application level scheduling, whereas the dashed lines assume the addition of application level scheduling.

Adding application level scheduling to RRjob and Cosched results in improved performance. RRjob experiences a larger benefit from the addition of application level scheduling than does Cosched. When application level scheduling is included, Cosched now performs worse than RRjob. This is because both polices provide support for synchronization and RRjob also provides equal allocation per job whereas Cosched does not. Note that RRjob performs worse than EqualDP even though both policies have application level scheduling and provide equal allocation per job. This is because RRjob has a higher degree of coscheduling. As a result, more processes are scheduled at the time a barrier is achieved resulting in more spinning. If there is a low degree of coscheduling, such as in EqualDP, a process will reach the barrier and then be swapped with a process that has not yet reached the barrier. If there is a high degree of coscheduling there will not be any descheduled processes to swap with, hence processes that are in the spinning state are forced to be scheduled. Cosched experiences a larger percent increase in response time than RRjob



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	U^*
Input	0.4	4.0		200.0	5.0	59%
Output		10.45	0.79	197.4	5.03	

Figure 6.6: Barrier Synchronization With Application Scheduling
Closed System, Uncorrelated

when the number of barriers per quantum is raised from zero to five barriers per quantum. This is because Cosched has a higher degree of coscheduling than RRjob. When all policies include application level scheduling EqualDP performs best, followed by RRjob and the Cosched.

The qualitative behavior of the policies for the correlated workload is the same as for the uncorrelated workload. We conclude from this section that application level scheduling improves performance and that scheduling policies which provide a high degree of coscheduling result in worse performance if the policies provide application level scheduling.

6.3. Conclusion

We have consider both spin-lock and barrier synchronization for the RRjob, Cosched and EqualDP polices. Are main conclusion from the chapter are:

- In general, for reasonable frequencies of synchronization, how well a policy supports synchronization does not have as significant an impact on the relative performance of the policies as does the allocation of processing power among the jobs. This leads us to conclude that allocation of processing power is at least as important as support for inter-process synchronization.
- At very frequent barrier synchronization, EqualDP performs best, followed by Cosched and then RRjob. At likely frequencies of barrier synchronization, EqualDP performs best followed by RRjob, and then Cosched.
- If we assume all policies use application level scheduling, a higher degree of coscheduling actually hurts performance when there is barrier synchronization, hence EqualDP performs best followed by RRjob and then Cosched.

CHAPTER 7

Importance of Preemption Frequency

In this chapter we investigate the effect of preemption frequency on the relative performance of the scheduling policies. The policies RRprocess, RRjob, and Cosched all periodically preempt processes. As a result, we would expect that the cost for these preemptions may cause the policies to perform worse than the other policies that only preempt processes when new jobs arrive to the system.

Preemption results in two types of overhead. The first is operating system overhead for the context switch. The second is overhead for rebuilding process cache working sets when processes are rescheduled. When a process is rescheduled it is often rescheduled on a processor different from the one on which it last ran. The process must then rebuild the cache working set. Even if rescheduled on the same processor, some portion of the process' working set entries are removed by intervening processes. Earlier work has implied that the cost for rebuilding cache entries may be more important than other characteristics of a scheduling policy. Tucker and Gupta state that the performance improvement seen for their policy is most likely attributable to higher cache hit rates for their policy than in the round robin process policy [TuGu 89]. Their policy also provides equal allocation per job and better support for inter-process synchronization. They do not provide any evidence as to how much each of the three improvements is responsible for the resultant improvement in performance.

This chapter has two goals. The first goal is to determine likely costs per preemption. In section 7.1 we estimate the overhead per preemption through the use of a simple model. The second goal of this chapter is to determine the importance of preemption frequency. Our approach is to study the policies including preemption overhead for a wide range of preemption overheads. In doing so we also determine the range of preemption overheads for which the policies RRprocess, Cosched, and RRjob are feasible. Section 7.2 contains the results from this study. Section 7.3 contains the conclusions from this chapter.

7.1. Preemption Overhead Estimation

In this section we estimate the overhead cost per preemption. We divide preemption overhead into two parts. The first part is the overhead for the context switch and scheduling policy invocation. This overhead is machine dependent and usually 1% - 2% of a quantum. The second part is the overhead for rebuilding a process' cache entries once the process is rescheduled. The overhead for rebuilding process cache working set can be estimated as:

$$\frac{W \times D}{Q}$$

Where:

W = The number of cache blocks in the process' working set.
D = Delay for servicing a cache miss.
Q = Length of a quantum

This overhead can vary greatly depending on the values of the parameters. Typical or specific values for the number of additional cache block misses resulting from preemption are not known. We view the process cache working set size as the number of cache block misses until the miss rate reaches steady state. In earlier work Agarwal et. al showed that the rate of cold misses drops off after around 1000 - 2000 misses for a variety of uniprocessor programs including system references [AgHH 87] [AgHH 88] [Agar 89]. The decrease in the rate of cold misses signifies the start of steady state. We hypothesize that if a process finds none of its cache entries in the cache, the number of additional misses due to preemption, W, is approximately equal to the number of cache block misses needed to achieve steady state. From Agarwal et. al's results we conclude that the number of additional cache block misses due to preemption will be on the order of 1000 cache blocks.

The parameter Q would be determined by studying what value results in optimal performance. Based on current multiprocessor quantum sizes, we hypothesize that likely values would be around 0.1 to 1.0 seconds. The parameter D is dependent on the machine architecture. For many architectures D is currently about ten times larger than the time to service a cache hit. Thus for a ten MIPS processor D would be approximately 1×10^{-6} seconds.

Figure 7.1 plots percent overhead per quantum versus the process working set size, W, for 3 different values of D with Q set equal to 0.1 seconds. We see that when W is 3000 or less percent preemption overhead is 6% or less. This leads us to conclude that the preemption overhead including operating system context switch overhead will likely be less than 10%. Other input parameters could be used to quickly estimate preemption overhead

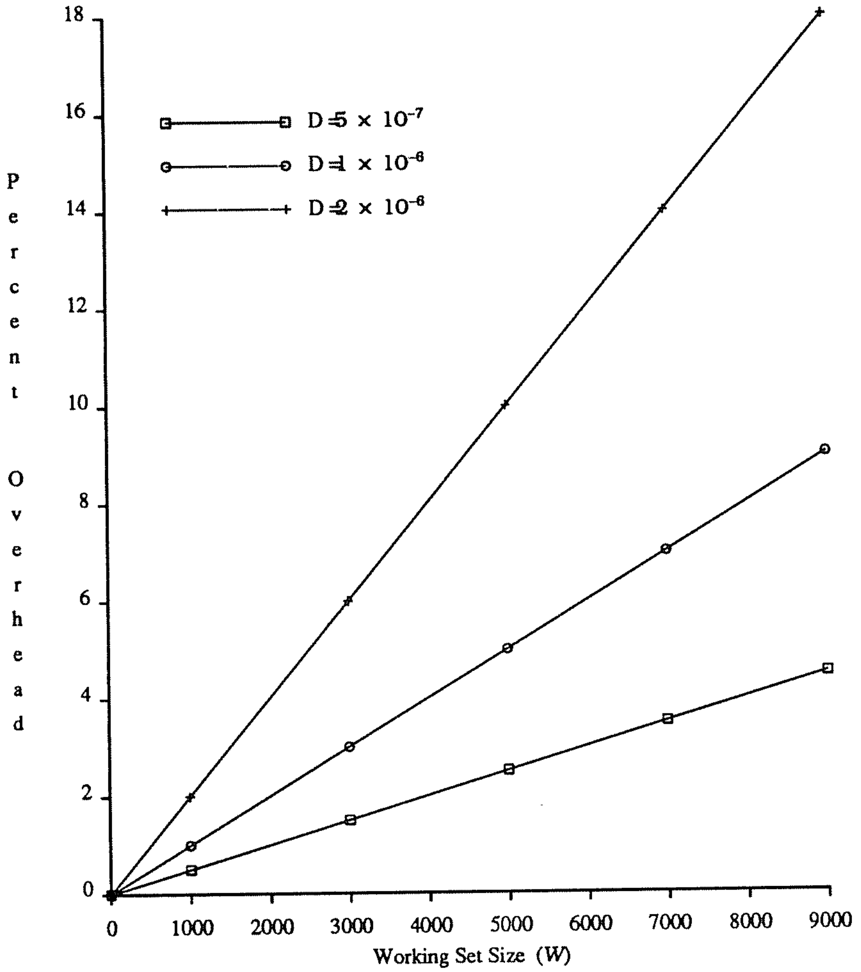


Figure 7.1: Percent Preemption Overhead versus *W*, *Q* = 0.1 seconds

values for real systems.

7.2. Sensitivity of Policies to Preemption Penalty

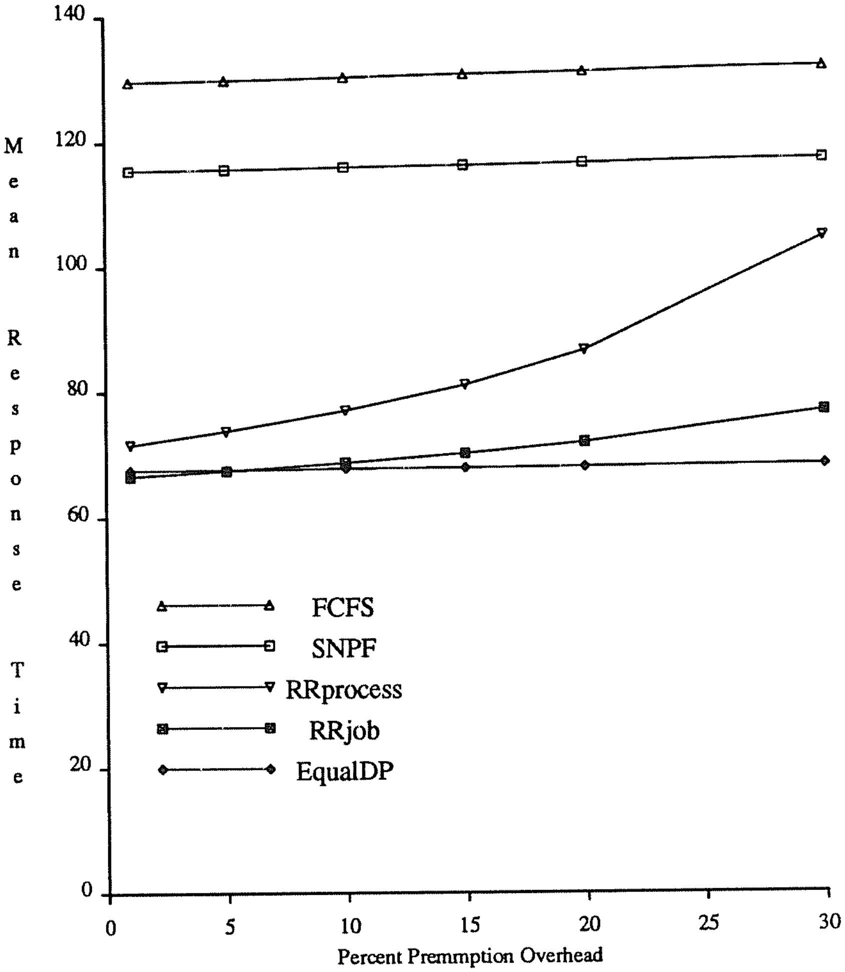
In this section we explore how an increase in preemption penalty affects the relative performance of the scheduling policies. We conduct experiments by varying the percent preemption overhead. We define precent preemption overhead as the percent of a unit lost for preemption overhead, where a unit is the length of the quantum for RRprocess. A 10% preemption overhead means that when a process is scheduled on a processor, $\frac{1}{10}$ of a unit is wasted. The percent preemption overhead is equal to the amount of time for the context switch plus the amount of time to rebuild process cache entries upon rescheduling the preempted process. Section 7.1 suggests that the preemption overhead may be less than 10%.

We consider the five policies FCFS, SNPF, RRprocess, RRjob, and EqualDP. We omit PSCDF because the policy is not a practical policy to implement. We omit Cosched because the policy results in the same number of preemptions as RRprocess, hence the effect of the preemption overhead on Cosched will be the same as the effect of the preemption overhead on RRprocess. We omit the UnequalDP and PSNPF policies because like EqualDP they only preempt processes when new jobs enter the system, hence the effect of the preemption overhead on PSNPF and UnequalDP is the same as the effect of the preemption overhead on EqualDP. We omit the foreground-background policies because they will result in at most one more preemption per job than in their non foreground-background counterparts, therefore the preemption overhead will have a similar effect on them as their non foreground-background counterparts.

Figures 7.2, 7.3, and 7.4 plot mean response time versus percent preemption overhead for the uncorrelated geometric-bounded workload for offered utilizations of 50%, 70%, and 90% respectively. Actual system utilization increases as the percent preemption overhead increases. The policies RRprocess and RRjob are adversely affected by an increase in preemption overhead, yet both RRjob and RRprocess have a lower mean response time than SNPF and FCFS for preemption overheads less than 10%. The RRprocess and RRjob policies become more sensitive to preemption overhead as the system utilization increases. This is because there are more preemptions at higher utilizations since there are fewer times where there are not any processes waiting for service. Note that in figure 7.4 EqualDP performs worse that RRprocess and RRjob at small preemption overhead values due to load queue effects.

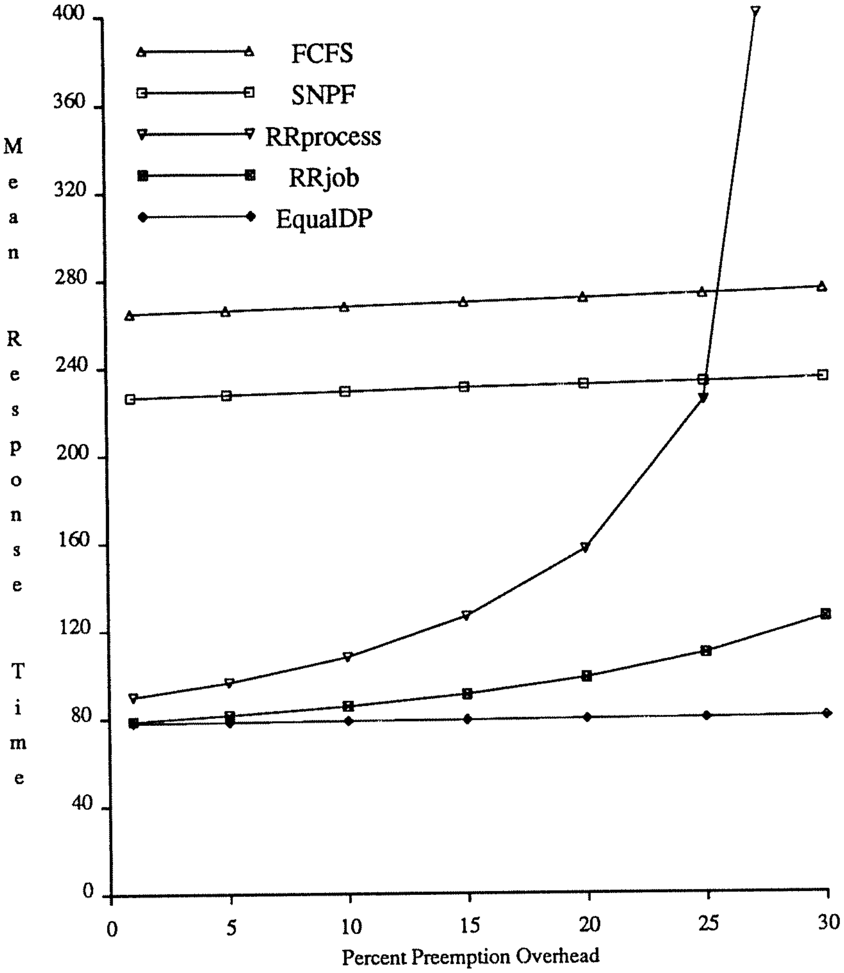
The policy RRjob degrades more slowly than RRprocess because RRjob's average quantum size is larger, resulting in fewer preemptions. Tables 7.1 and 7.2 contain the percent increase in response time for different percent preemption overhead values compared to the response time at a 1% preemption overhead for the RRjob and RRprocess policies respectively. We see that the mean response time of RRjob increases by less than 10% with a preemption overhead of 10% and offered utilizations of 50% and 70%. RRjob appears to be an acceptable policy for preemption overhead values of up to 10%.

Figures 7.5 and 7.6 plot mean response time versus percent preemption overhead for the correlated geometric-bounded workload with offered utilizations of 50% and 70% respectively. The qualitative performance for the correlated workload is the same as the uncorrelated workload.



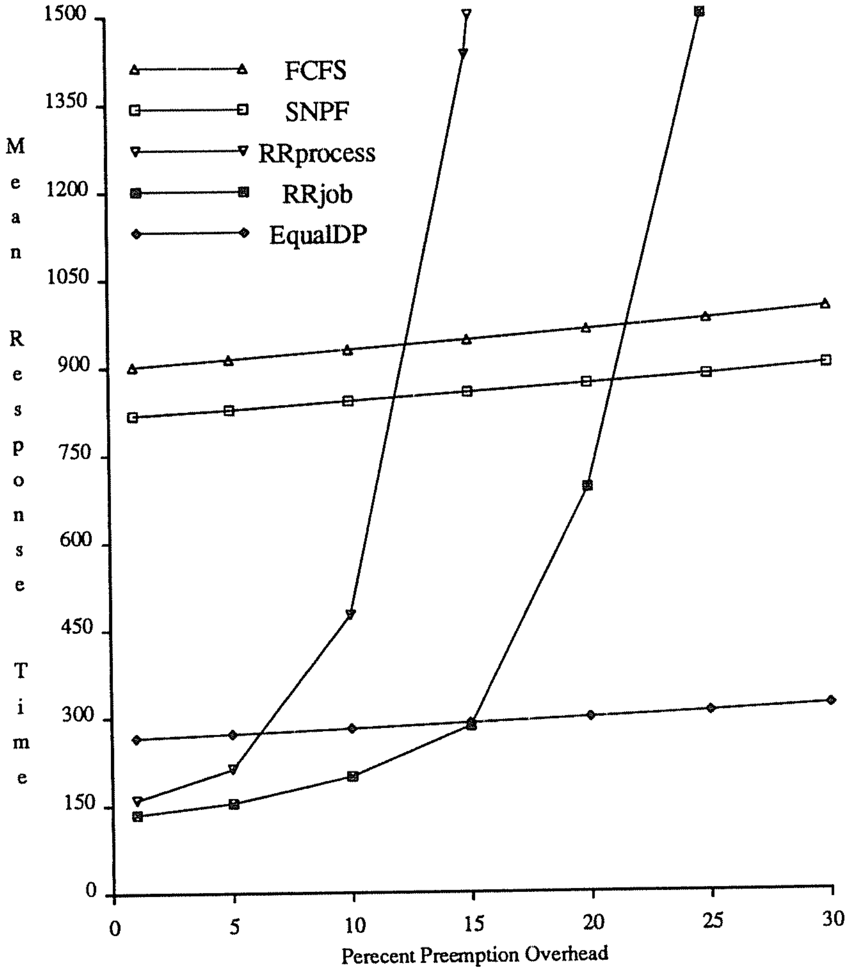
	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		200.0	5.0
Output		10.37	0.79	197.5	4.96

Figure 7.2: Effect of Preemption Overhead, Uncorrelated
Utilization = 50%



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		200.0	5.0
Output		10.37	0.79	197.5	4.96

Figure 7.3: Effect of Preemption Overhead, Uncorrelated
Utilization = 70%



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d
Input	0.4	4.0		200.0	5.0
Output		10.37	0.79	197.5	4.96

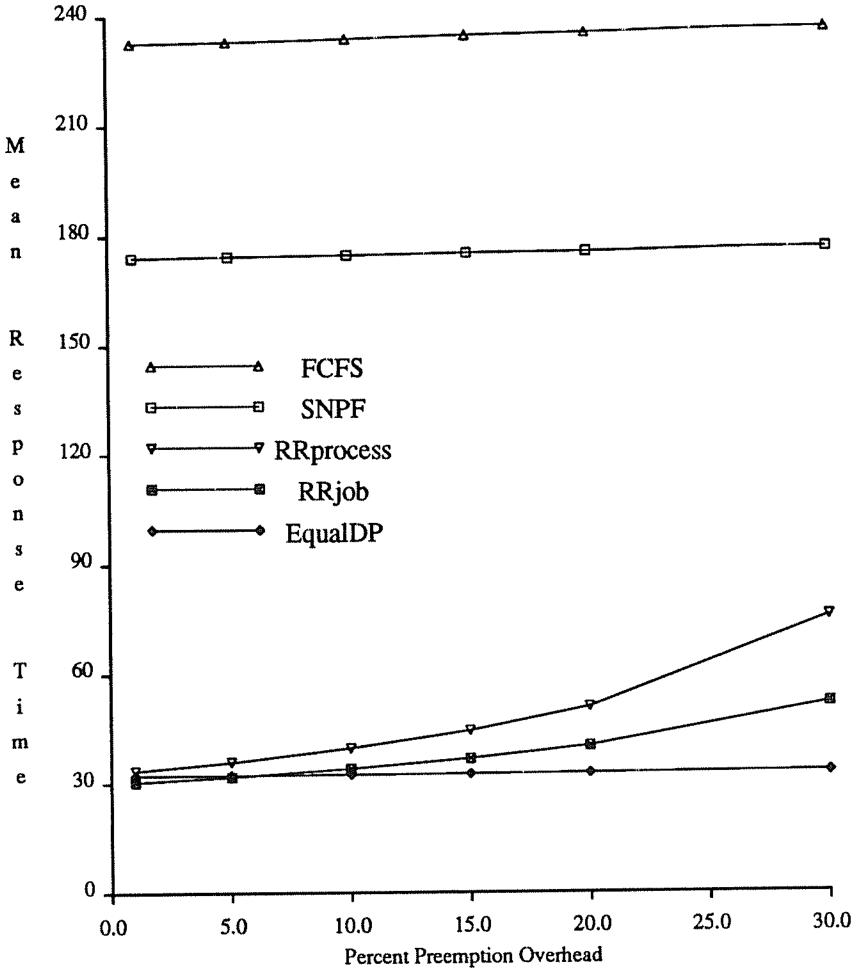
Figure 7.4: Effect of Preemption Overhead, Uncorrelated
Utilization = 90%

Table 7.1: Percent Increase in Response Time for RRprocess
Uncorrelated Workload

	Percent Preemption Overhead				
Offered Utilization	5%	10%	15%	20%	30%
50%	3	8	13	21	46
70%	7	20	41	75	575
90%	33	197	796		

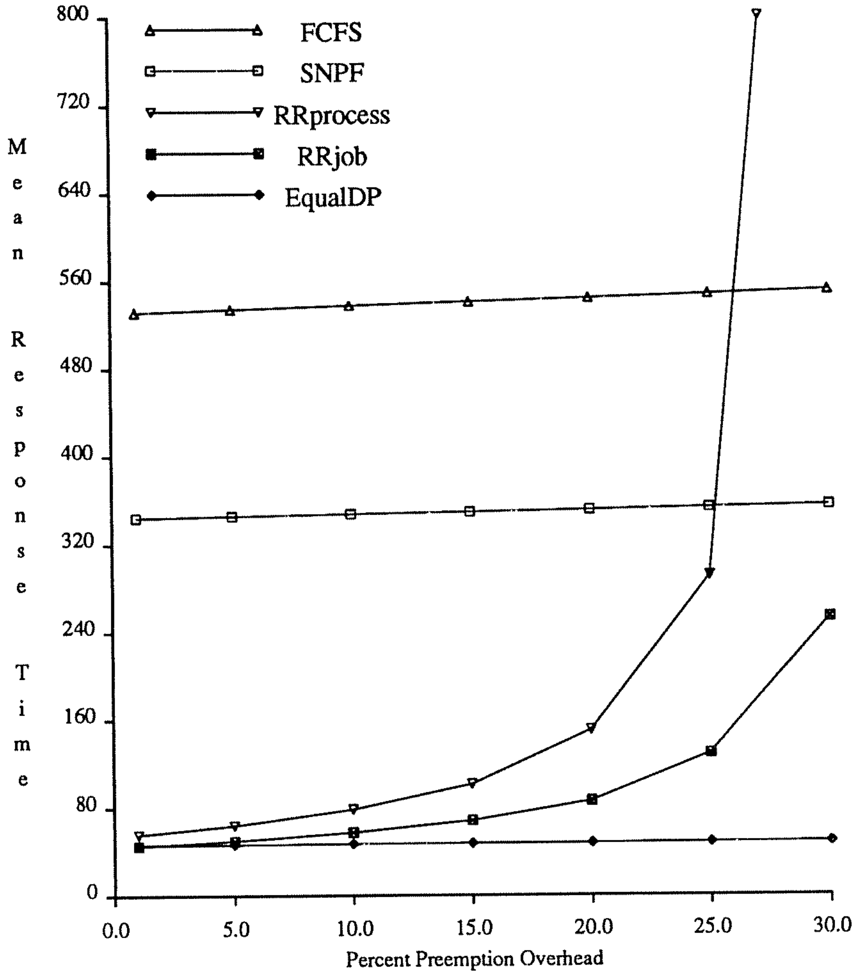
Table 7.2: Percent Increase in Response Time for RRjob
Uncorrelated Workload

	Percent Preemption Overhead				
Offered Utilization	5%	10%	15%	20%	30%
50%	1	3	5	8	16
70%	3	9	15	25	59
90%	14	47	110		



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	C_v	t
Input	0.4	4.0				5.0	20.0
Output		10.36	0.79	210.0	6.48		

Figure 7.5: Effect of Preemption Overhead, Correlated
Utilization = 50%



	P_p	$\bar{n}$	C_n	$\bar{d}$	C_d	C_v	t
Input	0.4	4.0				5.0	20.0
Output		10.36	0.79	210.0	6.48		

Figure 7.6: Effect of Preemption Overhead, Correlated
Utilization = 70%

7.3. Conclusion

We have shown that the policies RRprocess and RRjob perform better than FCFS and SNPF if the penalty per preemption is 10% or less. This implies that the additional overhead for time slicing preemption caused by cache misses is less significant than ramifications of not allocating each job competing for service a fraction of the processing power. We have shown that the performance of RRjob becomes worse relative to the performance of EqualDP as preemption overhead increases. This implies that preemption overhead is an important characteristic of a scheduling policy if the policies being compared allocate processing power equally well.

CHAPTER 8

Conclusions

8.1. Summary

We have identified three characteristics of general purpose multiprogrammed multiprocessor scheduling policies that are important to their resultant performance. These three characteristics are:

1) How the policy allocates processing power among the jobs competing for service.

2) How the policy supports interprocess synchronization.

3) How frequently the policy preempts processes.

To gain insight into the relative importance of these three characteristics we have studied twelve scheduling policies.

We have found that under a variety of workloads, the first characteristic is at least as important as the other two. In particular, we have found that in the absence of accurate job demand knowledge, policies should allocate a fraction of processing power to each job in the system. The policies that allocate a fraction of processing power to each job provide good service to jobs with small demands, which results in smaller mean job response times. In chapter 4 we demonstrate that for many workloads with a system utilization of 70%, policies that do not allocate a fraction of processing power to each job, such as FCFS, SNPF, and PSNPF, result in mean job response times 2 - 8 times larger than policies that do allocate a fraction of processing power to each job. Our results contradict and correct earlier work that concluded that the policies SNPF and PSNPF perform well.

In addition to providing each job with a fraction of the processing power, we propose that each job be allocated an *equal* fraction of the processing power. In chapter 5 we demonstrated that when job demand is correlated with the number of processes per job the policies that do not allocate processing power equally per process result in a mean job response time 10% - 50% larger than policies that do allocate processing power equally per job. The difference in response times is less significant when job demand is not correlated with the number of processes per job.

Scheduling policies may provide a fraction of processing power to each job either temporally (i.e. by time slicing) or spatially (i.e. by dividing the processor among the jobs). The major difference between these two approaches is the resultant preemption frequency. The time slicing policies exhibit a much higher frequency of preemption. Each preemption results in overhead for the operating system context switch plus the time to service extra cache misses for rebuilding a process' cache working set once a descheduled process is rescheduled. In chapter 7 we used a simple model to conclude that the overhead cost per preemption is likely to be 10% of a quantum or less. For preemption overhead values of 10% or less we have found temporal division of processing power (i.e. RRjob) is competitive with spatial division of processing power (i.e. EqualDP)

In chapter 6 we have found that the importance of supporting inter-process synchronization depends on the type and frequency of the synchronization. Explicit support for spin-lock synchronization is not necessary as long as processes holding locks can not be descheduled for long periods of time. As a result spin-lock synchronization does not affect the relative performance of the polices for reasonable workload parameters. Explicit support for barrier synchronization is beneficial when there is frequent synchronization, although based on estimates from today's multiprocessors, the frequency of synchronization where it becomes important to provide explicit support for inter-process synchronization is higher than may be found in a real systems. Application level scheduling results in the best performance, but coscheduling is better than no explicit support for inter-process synchronization. If application level scheduling is included in a policy, we have found that a high degree of coscheduling actually degrades performance when there is barrier synchronization.

Of the policies studied, RRjob and EqualDP have been shown to be the most promising. Both policies provide equal allocation per job, resulting in almost identical performance for most workloads with no synchronization. The choice between these two policies depends the availability of an application level scheduler. Developing an application scheduler is a non-trivial task. The cost of developing and maintaining (or porting) an application level scheduler may lead system designers to not include an application level scheduler. In the absence of an application level scheduler, EqualDP with out time slicing processes within a job performs poorly in the presence of inter-process synchronization. Hence, RRjob would be a better choice for implementation. If all jobs do use application level scheduling, then EqualDP would be preferable since EqualDP has a much lower preemption frequency than RRjob, and better supports interprocess synchronization.

8.2. Future Research Directions

In these section we list several possible directions for future work.

1) Multi-level Queues.

Further work needs to be done in considering multi-level queues. First, more work should be done on the effects of two level queues. We have not considered the effect of two level queues for all the policies. Adding multi-level queues to the other policies will likely decrease the difference in performance of the policies studied. In particular, UnequalDP may rival EqualDP if both policies have multi-level queues. Second, multi-level queues with more levels should be considered. One reason to consider multi-level queues is that they decrease the preemption rate, hence processes remain on processors for longer periods of time. Another reason to consider multi-level queues is that we expect they will eliminate the poor performance of EqualDP and UnequalDP seen at high loads due to only allowing 20 active jobs at a time.

2) Workload Characterization.

Currently there are no measurement studies of general purpose multiprogrammed multiprocessor to draw upon for characterizing workloads. A good understanding of actual workloads would be beneficial in determining what issues are most important. Parameters of especial interest are: the number of processes per job, the demand per job, the demand per process, the I/O behavior of processes, the working set size of processes, and the optimal size of the quantum for time slicing policies.

3) Importance of Preemption Frequency

For today's working set and quantum sizes it appears that time slicing is a competitive approach. It is important to further investigate the amount of time processes need to spend on a processor relative to the process working set size in order for time slicing to remain a competitive approach. In addition to the working set and quantum size, the amount of time to service a cache miss needs to be considered. We anticipate that this parameter will increase as processors become faster. If the cost for moving processes becomes too high, time slicing will not be feasible.

4) Study the effect of I/O.

How the inclusion of I/O will affect these multiprocessor scheduling policies is not known. It is possible that I/O requirements may limit the utility of policies that include application level schedulers. In addition, frequent I/O would result in more frequent context switches, and hence decrease the advantages of dynamic partitioning polices in regards to preemption overhead. Virtual memory may also impact the utility of certain policies. Thrashing may result if the scheduling policy does not consider virtual memory needs.

5) Scheduling for large scale systems.

Systems with thousands of processors may become a reality in the near future. For such large scale multiprocessors the scheduling issues may be significantly different. First, a single centralized ready queue may not be viable. The contention for the lock on the queue could become a bottle neck [LiCh 89]. Second, such large scale multiprocessors have distributed memory for the processors. This may necessitate keeping processes on a specific processors during their entire execution.

6) Analytical Models.

Analytical models may be able to provide some additional insight into the problem of multiprocessor scheduling. To date analytical modeling of multiprocessor scheduling has only meet with limited success. It is difficult to capture the parallelism and synchronization of real multiprocessor system in a tractable analytical model. New models are necessary in order to deal with the difficulty of the problem.

Bibliography

[Agar 87] Agarwal, A., "Analysis of Cache Performance for Operating Systems and Multiprogramming," Ph.D. Thesis, Technical Report No. CSL-TR-87-332, Computer Systems Laboratory, Department of Electrical Engineering, Stanford University, May 1987.

[AgHH 88]Agarwal, A., Horowitz, M., Hennessy, J., "Cache Performance of Operating System and Multiprogramming Workloads," ACM Transactions on Computer Systems, Vol. 6, Number 4, November 1988, pp. 393-431.

[AgHH 89]Agarwal, A., Horowitz, M., Hennessy, J., "An Analytical Cache Model," ACM Transactions on Computer Systems, Vol. 7, Number 2, May 1989, pp. 184-215.

[AnLL 89] Anderson, T.E., Lazowska, E.D., Levy, H.M., "The Performance Implications of Thread Management Alternatives for Shared-Memory Multiprocessors," Proceedings of the ACM SIGMETRICS and Performance 89 joint Conference on Measurement and Modeling of Computer Systems, Berkeley, CA, May 23-26, 1989.

[BeLL 88] Bershad, B.N., Lazowska, E.D., Levy, H.M., "PRESTO: A System for Object-Oriented Parallel Programming," Software: Practice and Experience 18,8, August 1988, pp. 713-732.

[Doep 87] Doeppner, T.W., "Threads: A System for the Support of Concurrent Programming," Technical Report CS-87-11, Department of Computer Science, Brown University, 1987.

[EaZL 89] Eager, D.L, Zahorjan, J., Lazowska, E.D., "Speedup Versus Efficiency in Parallel Systems," I.E.E.E. Transactions on Computers, March, 1989, pp. 408-423.

[GaFF 89] Garcia, A., Foster, D., Freitas, R., "The Advanced Computing Environment Multiprocessor Workstation," Research Report RC-14419, IBM Research Division, March 1989.

[GaJo 79] Garey, M.R., Johnson, D.S., "Computers and Intractability, A Guide to the Theory of NP-Completeness," W.H. Freeman and Company, 1979, pp. 238.

[Gust 88] Gustafson, J.L., "Reevaluating Amdahl's Law," Communications of the ACM, May 1988.

[Koba 78] Kobayashi, "Modeling and Analysis," Addison-Wesley, 1978.

[Klei 76] Kleinrock, L., Queueing Systems, Vol. II: Computer Applications, John Wiley and Sons, 1976.

[KrWe 85] Kruskal, C.P., Weiss, A., "Allocating Independent Subtasks on Parallel Processors," IEEE Transactions on Software Engineering, October 1985, pp. 1001-1016.

[LeVe 90] Leutenegger, S.T, Vernon, M.K., "Multiprogrammed Multiprocessor Scheduling Policies," Proceedings of the ACM SIGMETRICS 1990 Conference on Measurement and Modeling of Computer Systems, Boulder, CO, May 22-25, 1990.

[LiCh 89] Lionel, M.Ni., Ching-Farn, E.WU, "Design Tradeoffs for Process Scheduling in Shared Memory Multiprocessor Systems," IEEE Transactions on Software Engineering, March 1989, pp. 327-334.

[Livn 88] Livny, M., DeNet User's Guide, Version 1.0, Computer Sciences Department, University of Wisconsin, Madison, 1988.

[LoTh 88] Lovett, T., Thakkar, S., "The Symmetry Multiprocessor System," Proceedings of the 1988 International Conference on Parallel Processing, August, 1988.

[Maju 88] Majumdar, S., "Processor Scheduling in Multiprogrammed Parallel Systems," Ph.D. Thesis, Research Report #88-6, Department of Computational Science, Saskatoon, Saskatchewan, Canada, April 1988.

[MaEa 89] Majumdar, S., Eager, D., Private communications, 1989.

[MaEB 88]

Majumdar, S., Eager, D., Bunt, R., "Scheduling in Multiprogrammed Parallel Systems," Proceedings of the ACM SIGMETRICS 1988 Conference on Measurement and Modeling of Computer Systems, Santa Fe, NM, May 24-27, 1988, pp. 104-113.

[NeTa 88] Nelson, R., Tantawi, A.N., "Approximate Analysis of Fork/Join Synchronization in Parallel Queues," IEEE Transactions on Computers, June, 1988, pp. 739-743.

[NeTT 88] Nelson, R., Towsley, D., Tantawi, A.N., "Performance Analysis of Parallel Processing Systems," IEEE Transactions on Software Engineering, April 1988, pp. 532-540.

[Nels 90] Nelson, R., "A Performance Evaluation of a General Parallel Processing Model," Proceedings of the ACM SIGMETRICS 1990 Conference on Measurement and Modeling of Computer Systems, Boulder, CO, May 22-25, 1990.

[NeTo 90] Nelson, R., Towsley, D., "A Performance Evaluation of a Several Priority Policies for Parallel Processing Systems," Preprint, 1990.

[Oust 82] Ousterhout, J., "Scheduling Techniques for Concurrent Systems," Proceedings of the Distributed Computing Systems Conference, 1982, pp. 22-30.

[Shra 68] Shrage, Linus, "A Proof of the Optimality of the Shortest Remaining Processing Time Discipline," Operations Research, 1968, pp. 687-690.

[SaCh 81] Sauer, C.H., Chandy, K.M., "Computer System Performance Modeling", Prentice-Hall, 1981, page 16.

[SeSt 89] Seager, M., Stichnoth, J., "Simulating the Scheduling of Parallel Supercomputer Applications," Preprint UCRL-102059, Lawerence Livermore National Laboratory, September 19, 1989.

[Sevc 89] Sevcik, K., "Characterizations of Parallelism in Applications and Their Use In Scheduling," Proceedings of the ACM SIGMETRICS 1989 Conference on Measurement and Modeling of Computer Systems, Berkeley, CA, May 23-26, 1989, pp. 171-180.

[SqLa 90] Squillante, M., Lazowska, E., "Using Processor-Cache Affinity Information in Shared-Memory Multiprocessor Scheduling," Technical Report, Department of Computer Science, University of Washington, Seattle, 1990.

[TaSS 88] Thacker, C.P., Stewart, L.C., Satterthwaite, E.H., "Firefly: A Multiprocessor Workstation," IEEE Transactions on Computers, August, 1988, pp. 909-920.

[ToRS 90] Towsley, D, Rommel, C.G., Stankovic, J.A., "Analysis of Fork-Join Program Response Times on Multiprocessors," IEEE Transactions on Parallel and Distributed Systems, July, 1990, pp. 286-303.

[TuGu 89]Tucker, A., and Gupta, A., "Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors," Proceedings of the 12th ACM Symposium on Operating System Principles, December 1989.

[ZaLE 88] Zahorjan, J., Lazowska, E.D., Eager, D.L., "Spinning Versus Blocking in Parallel Systems with Uncertainty," Proceedings of the International Seminar on Performance of Distributed and Parallel Systems, Kyoto Japan, December 7-11, 1988, pp. 445-462.

[ZaLE 89] Zahorjan, J., Lazowska, E.D., Eager, D.L., "The Effect of Scheduling Discipline on Spin Overhead in Shared Memory Parallel Systems," Technical Report 89-07-03, Department of Computer Science, University of Washington, Seattle, July 1989.

[ZaMc 90] Zahorjan, J., McCann, C., "Processor Scheduling in Shared Memory Multiprocessors," Proceedings of the ACM SIGMETRICS 1990 Conference on Measurement and Modeling of Computer Systems, Boulder, CO, May 22-25, 1990.