Using Parallel Program Characteristics in Dynamic Multiprocessor Allocation Policies

Kaushik Guha

Technical Report CS-95-03
May, 1995

Graduate Programme in Computer Science
York University
North York, Ontario, Canada
M3J 1P3

Using Parallel Program Characteristics in Dynamic Multiprocessor Allocation Policies

Kaushik Guha

Technical Report CS-95-03
May, 1995

Graduate Programme in Computer Science
York University
North York, Ontario, Canada
M3J 1P3

This report is an adaptation of Kaushik Guha's M.Sc. thesis.

Using Parallel Program Characteristics in Dynamic Multiprocessor Allocation Policies

Kaushik Guha

A thesis submitted in conformity with the requirements
for the Degree of Master of Science
Graduate Program of Computer Science
York University

ABSTRACT

The goal of an effective scheduling policy in a multiprogrammed multiprocessor is to minimize mean response time by sharing the processors among the set of competing parallel applications. In this thesis we perform extensive simulations to demonstrate that application characteristics can be used to make improved allocation decisions in a dynamic scheduling environment. We consider the work to be executed by a job and the efficiency with which the work is executed to determine the advantages that can be obtained by using such job characteristics in making scheduling decisions. The basis of comparison is a widely studied policy which does not use job characteristics and simply equipartitions processors among the active jobs in the system (called Equipartition).

We describe a generalized allocation technique that makes allocation decisions based on application characteristics. This generalization is used to show that policies that make allocation decisions based on only the service demand of applications are unable to improve mean response time when compared with Equipartition for workloads with low average efficiency. However, significant improvements are obtained for workloads with high average efficiency. We also use the generalization to show that using only efficiency in making scheduling decisions result in mean response times that are significantly higher than Equipartition, although small reductions are obtained for some workloads with low average efficiency. We then introduce and evaluate a new family of policies that use both characteristics of the work and the efficiency of an application in making scheduling decisions. The results of our simulation show that these policies reduce mean response times significantly when compared with Equipartition.

In this thesis we identify the workload conditions under which improvements are obtained from using job characteristics in making allocation decisions. We also quantify the reductions in mean response time that can be obtained from using these job characteristics. Although the allocation policies introduced in this thesis cannot be directly implemented in a multiprocessor, we provide insights into the scheduling problem that will be fundamental in the development of future multiprocessor schedulers.

Acknowledgments

I am thankful to my thesis supervisor, Tim Brecht, for his constant support and encouragement throughout this research. It has been my good fortune to be able to work with Tim. Thanks Tim for everything you have taught me in these two years.

I am also thankful to the committee members for spending their valuable time and consenting to be in my examination committee. I would also like to extend my special thanks to Professor Minas Spetsakis who was always there to answer my questions. I also thank Professor Mary Vernon for her insightful comments that helped us to resolve some outstanding issues and improve the quality of this thesis.

I would like to thank the Department of Computer Science, York University for providing financial support. Of course, no words are enough to thank Pat for the help she has provided. I am really grateful to you Pat, for all that you have done for me.

Amongst my friends, I would like to thank Nian, Khuzaima, Roy, Benny, 'buddy' Choi, Ken and Jason for their support. Special thanks go to Nian and Khuzaima for helping me out during hard times.

I bow in gratitude before my parents: it is only because of them that I have achieved this goal. I also thank Biba, Khokada, Tabai and Tia for the love and the understanding that they have provided.

Table of Contents

Glossary

P	- total number of processors in the system
n	- total number of jobs in the system
J_i	- job i
p_i	- number of processors allocated to J_i
W_i	- work executed by J_i
β_i	- execution rate parameter of J_i (indicates efficiency)
N_i	- used to model limits to J_i's parallelism and speedup
k_i	- knee of J_i in the execution time - efficiency profile
ε_i	- effective efficiency of $J_i = E_i(P) \cdot 100$
Acc_i	- CPU service accumulated by J_i
$Time_i$	- time spent by J_i in the system since its arrival
γ	- execution rate function $= \dfrac{(1+\beta_i)p_i}{\beta_i + p_i}$
$T_i(1)$	- execution time on 1 processor (sequential version)
$T_i(p_i)$	- execution time on p_i processors $= \dfrac{W_i(\beta_i + p_i)}{(1+\beta_i)p_i}$
$S_i(p_i)$	- speedup on p_i processors $= \dfrac{T_i(1)}{T_i(p_i)}$
$E_i(p_i)$	- efficiency on p_i processors $= \dfrac{S_i(p_i)}{p_i}$
R	- mean response time
W	- mean service demand
N	- mean N_i

ε - mean effective efficiency

C_W - coefficient of variation of W_i

D_W - distribution used to obtain W_i

C_ε - coefficient of variation of ε_i

D_ε - distribution used to obtain ε_i

C_N - coefficient of variation of N_i

D_N - distribution used to obtain N_i

λ - job arrival rate

M - number of jobs executed in a simulation run

S - number of trials to execute for generating confidence intervals

Chapter 1

Introduction

1.1. Motivation

This thesis is concerned with the scheduling of multiple parallel applications in multiprocessors. In particular, the focus is on the advantages of using application characteristics in determining the number of processors to allocate to each application.

The use of multiprocessors has become widespread in the user community over the past few years. The primary motivation behind multiprocessors is the demand for greater performance. *Scheduling* in multiprocessors involves determining the number of jobs to execute simultaneously (i.e., when to activate jobs) as well as the number of processors to allocate to each active application. An appropriate scheduling policy is necessary for the effective utilization of processors and also satisfactory response time for parallel applications. Ineffective scheduling methods may lead to the underutilization of a parallel system and may seriously undermine the success of this class of machines.

One method of multiprogramming parallel applications is to execute different applications on the same processors during different time intervals. Hence, processors are time-shared among applications. Using this technique, all processors incur context switching overheads during every reallocation. A space-sharing scheme avoids these overheads by allocating different portions of the processors to different applications [27]. In a time-sharing approach the scheduler only decides when to activate an application and when to suspend it. In a space-sharing approach the scheduler also has to determine the number of processors to allocate to each application. This is called the allocation problem [1].

This thesis addresses the allocation problem. The number of processors allocated to each job will determine the execution time of an application and consequently overall system performance. From the user point of view, satisfactory response time of applications implies good system performance. Hence, mean response time is the objective function we seek to minimize in this thesis. Note that the *response time* of a job is the time that elapses from its arrival until its completion.

When trying to minimize the mean response time of parallel programs in a dynamic multiprogrammed environment a fundamental tension exists between allocating a sufficiently large number of processors to a job (in order to minimize its execution time) and allocating a sufficiently small number of processors (in order to maximize the efficiency with which the processors are used). This tension exists because the processors which are not efficiently utilized by one job can be allocated to another job (which might be able to make better use of them). In uniprocessor systems mean response time can be reduced by allocating less processing power (fewer time-slices) to larger jobs (e.g., to ensure that shorter jobs finish execution as soon as possible). In a multiprocessor it may also be desirable to reduce the processing power allocated to long running jobs. Hence, the difficulty in making processor allocation decisions lies in trying to allocate fewer processors to larger jobs in order to expedite the completion of smaller jobs while also considering the efficiency with which these jobs can execute.

Scheduling policies which use knowledge of the job size (work) and efficiency in making allocation decisions might be able to improve mean response time over policies which use no such information. Note that although such information (work and efficiency) may not be available to the system at run time, we obtain insights into the allocation problem that we expect can be used to improve future implementations of multiprocessor schedulers. Moreover, estimates of work and efficiency may be used to reduce mean response times when compared with policies that use no information about job characteristics. (Previous studies suggest that the service demand of a job can be estimated by user supplied execution time estimates, system maintained logs of previous executions, using the run time system or using the past execution history of the application [1].)

It has been demonstrated in previous studies that the use of job characteristics in making allocation decisions improves mean response time over policies which do not use such information [1][13][6][14][24]. However, the study by Chiang, et al.[2] report the opposite conclusion and assert that policies using execution rate characteristics of a job do not improve performance (compared with policies that use none). Similar conclusions can be derived from the results reported in Setia and Tripathi's studies [21][22].

Note that these studies have been conducted in a static scheduling environment where processors are allocated for the lifetime of a job. (In a dynamic scheduling environment, processors can be reallocated at any point during the execution of a job.) In this thesis we investigate the advantages of using job characteristics (work and efficiency) for making scheduling decisions in a dynamic environment. Our research is conducted in a dynamic scheduling environment since dynamic scheduling techniques are generally preferred over static scheduling techniques [31][16]. We compare policies which use work and efficiency in making scheduling decisions with a policy that requires no such characteristics and shares processing power equally among jobs (Equipartition). (Note that, it has been concluded by some researchers that the equal sharing of processing power is a desirable property of a good scheduler [12][11].)

Much of the work in this thesis is driven by the investigation of the following questions:

1) How useful is it to know (or have an estimate of) the amount of work a job has left to execute?

2) How useful is it to know (or have an estimate of) the efficiency with which a job executes?

3) Is one of these characteristics more important (or useful) than the other?

4) If we have information about the work a job executes and the efficiency with which it is executed, how can that information be used in making dynamic processor allocation decisions in a multiprogrammed environment?

5) What are the size of the improvements in mean response times that can be obtained by policies using job characteristics in making partitioning decisions over the Equipartition policy?

1.2. Goals

The main goal of this thesis is to gain a better understanding of the factors involved in designing scheduling methods for multiprogrammed multiprocessors.

Previous studies have investigated the potential benefits that may be obtained by using application characteristics in a static scheduling environment [2][6][14][24][25]. In this thesis we consider a dynamic scheduling environment and conduct simulation studies to investigate the utility of using job characteristics in making scheduling decisions. Specifically, we want to understand the relationship between the service demand of a parallel application, the efficiency with which the service requirement can be satisfied and the processing power that should be allocated to each job in order to minimize the mean response time. We use a simple job model in which the service demand and the efficiency of jobs are characterized with a single parameter to quantify the reductions in mean response time that might be obtained by using these job characteristics when compared with an allocation policy that does not. The models used in this thesis are approximate representations of parallel jobs and multiprocessors chosen for their simplicity and to make our study and analysis of the problem tractable. This will provide a first-order understanding and insights into how allocation decisions should be made in order to improve future practical implementations of a multiprocessor scheduler.

Previous research has shown that, the relative performance of scheduling policies can be sensitive to workload conditions [2][13][1]. An additional goal of this thesis is to identify the conditions under which improvements can be obtained by using job characteristics in making allocation decisions.

1.3. Contributions

In this thesis we use relatively simple models of a parallel program's execution, the workload, and the system, and perform extensive simulation studies in order to determine the advantages of using application characteristics in a dynamic space-sharing scheduling environment. The main contributions of this thesis are outlined below.

In this thesis we demonstrate that policies which use only information about the work jobs execute to make processor allocation decisions do not reduce mean response time unless the average efficiency of jobs are relatively high. The basis for comparison used in this thesis is the Equipartition policy [27][7], which uses no job characteristics and equally partitions the processors among the applications. The size of the improvements observed by making partitioning decisions based on the amount of work each job executes increases with the average efficiency of the jobs, the coefficient of variation of the work, and the system load. Reductions in mean response time are maximized and are substantial (up to 70%) when all jobs execute with perfect efficiency.

We also demonstrate that policies that use only information about the efficiency with which a job executes are unable to substantially reduce mean response times when compared with the Equipartition policy, although relatively small reductions are obtained for some workloads (≈14%). These policies are unable to attain significant reductions in mean response time because, they fail to consider the amount of remaining work each job has to execute.

We demonstrate the advantages of using job characteristics in scheduling by introducing a new class of policies called $W\&E$ (Work and Efficiency) which use work as well as efficiency in making processor allocations. Although information about a job's remaining work and its efficiency may not be readily available we believe that it is important to understand the properties of effective processor allocation policies in order to develop more effective and practical scheduling algorithms. We demonstrate that these policies can yield substantial improvements when compared with the Equipartition policy. However, the improvements are limited by the coefficient of variation of the work jobs execute, the efficiency with which jobs execute their work, and the system load.

Eager, et al. [5], report that the knee (where the maximum benefit per unit cost of processor allocation is obtained) might be useful in determining processor allocations in a static scheduling environment. However, our results indicate that the knee might not be a suitable point for processor allocation in a dynamic scheduling environment at high loads, especially for workloads with high average efficiency.

We demonstrate in this thesis that it is not desirable to blindly equipartition processors and that considerable improvements in mean response time can be obtained by using information about a job's work and efficiency. However, equipartitioning processors is a compromise that is safe in the absence of such characteristics. Equipartition is also likely to remain the practical algorithm of choice until more studies of multiprocessor workloads are performed and better techniques for obtaining job characteristics are developed.

1.4. Overview of the Thesis

The widespread use of parallel systems has led to extensive research in high performance scheduling policies. This thesis investigates the advantages of using job characteristics in making scheduling decisions.

The structure of the thesis is as follows: in Chapter 2 we discuss the relevant background in multiprocessor scheduling and survey some of the scheduling policies studied in the literature. In Chapter 3 we describe the job, workload and the multiprocessor models used to capture the essential features of a parallel system. Chapter 4 describes the scheduling disciplines examined in this thesis. The results of comparisons between the different policies are presented in Chapters 5, 6, and 7. In Chapter 5 we discuss policies which use only knowledge of the work in making scheduling decisions and show that they are unable to reduce mean response time over Equipartition for all workloads considered in this thesis. Policies which use only knowledge of the efficiency in scheduling are discussed in Chapter 6. The results in Chapter 6 show that using only efficiency in making scheduling decisions results in relatively small reductions in mean response times over Equipartition for some workloads. In Chapter 7 we introduce and evaluate new policies which uses both the knowledge of work and efficiency in making scheduling decisions. These policies are shown to achieve significant reductions in mean response time over Equipartition. Chapter 8 summarizes the contributions and concludes the thesis.

Chapter 2

Background

2.1. Introduction

The widespread use of multiprocessors has created the need for job scheduling policies in order to reduce job response time and make efficient use of the system processors. As a result, a number of studies have investigated different multiprocessor scheduling policies.

In this chapter we present an overview of existing research in the field of multiprocessor scheduling. We first describe some uniprocessor scheduling algorithms that have influenced multiprocessor scheduling. Different types of multiprocessor schedulers such as static and dynamic schedulers are then explained. We also describe parallel program characteristics which model the execution of a job after which we discuss scheduling policies that use job characteristics in making allocation decisions. The common conclusions derived in previous research regarding the use of job characteristics in multiprocessor scheduling are then summarized to conclude the chapter.

2.2. Uniprocessor Scheduling

The insights gained by understanding uniprocessor scheduling techniques could provide a better understanding of multiprocessor scheduling principles. This may lead to the design of effective multiprocessor schedulers.

The First-Come-First-Served (FCFS) policy has been used commonly in uniprocessor systems. This policy does not perform well, especially when the coefficient of variation in the service time of applications is high. (The coefficient of variation is defined as the ratio of the standard deviation to the mean.) This is due to monopolization of the system by large applications. Small jobs are forced to wait behind large applications thus resulting in increased mean response times. The FCFS discipline performs quite well under workloads with low coefficient of variation (typically less than or equal to 1), since the job with the most acquired service is expected to be the closest to completion. The Preemptive Shortest Job First (PSJF) policy which allocates the processor to the smallest job (in a preemptive fashion) is optimal in uniprocessor scheduling. However, this policy requires precise information about the service requirement of jobs.

Round Robin (RR) is an effective approximate to the PSJF policy especially when no information is available about the service demand of jobs and when the coefficient of variation in service demand is high. In the *Round Robin* policy, the processor is time-multiplexed among applications in the system, giving quantums of service to each application.

Although precise information about the service demand of an application may not be known at run time, estimates of the service demand can be used in making scheduling decisions. The *foreground-background* algorithm (FB), also commonly referred to as the *multi-level feedback queue* policy [3], uses the acquired processor service time of an application in scheduling since the job with the most acquired service is expected to be the one with the most remaining service time. The FB policy orders the applications in multiple queues according to the acquired processor service (CPU time). Jobs with smaller amounts of acquired service are given higher priority than those with larger amounts of acquired service time. This policy has been shown to improve mean response times over Round-Robin, especially for workloads with a high coefficient of variation (typically greater than one).

The principles of uniprocessor scheduling policies have been used to develop several multiprocessor scheduling policies. These are described in Section 2.6. In general, uniprocessor scheduling policies which use precise knowledge or estimates of the service demand of an application can improve mean response time over policies that divide processing power equally among the applications, especially for workloads with high coefficient of variation in service demand. This may also be true in the case of multiprocessors where policies using job characteristics may improve mean response times significantly over Equipartition.

2.3. Time-Sharing versus Space-Sharing

Previous studies have identified two general categories for multiprogramming parallel applications in shared-memory multiprocessors, *time-sharing* and *space-sharing*. In a *time-sharing* approach different applications are run on the same processors during different intervals of time. Thus, the processors are time-shared by the running applications. A *space-sharing* scheme divides the processors among jobs so that each job executes on a portion of the processors. Several studies have compared time-sharing and space-sharing policies. Round-Robin (RR) is an example of a time-sharing policy which has been studied extensively in previous research [13][12][11]. Among the space-sharing schemes, one of the most studied is the Equipartition policy [27][7]. Tucker and Gupta [27] try to dynamically control the number of active processes of each job so that the total number of running processes in the system closely matches the number of processors in the system (this technique is called *Process Control*). At the same time the scheduler dynamically allocates an equal fraction of the processing power to each job (this is the *Equipartition* policy). Space-sharing schemes avoid the problem of time-sharing processors among multiple applications. Time-sharing processors among applications might degrade system performance due to the overhead of frequent context switches and processor cache corruption. Experimental and simulation studies by Tucker and Gupta [27], Gupta, et al. [7], Zahorjan and McCann [31] and McCann, et al. [16], have lead to the general conclusion that space-sharing is preferable to time-sharing in Uniform Memory Access (UMA) multiprocessors, since space-sharing avoids the costs associated with preemptions required by time-sharing techniques. Hence our research focuses only on policies which space-share processors among jobs.

2.4. Dynamic and Static Scheduling

Another classification of multiprocessor schedulers is based on the frequency of processor reallocations performed by the scheduler. At two extremes are the static and dynamic scheduling approaches. In *static* scheduling processors are allocated for the lifetime of the application. The allocated processors are not relinquished until the job is completed. These are also referred to as *Run-To-Completion (RTC)* algorithms. RTC algorithms are simple to implement and have low scheduling overhead. This approach is popular in systems where the cost of reallocating processors is quite high, for example in message passing systems. However, these algorithms may allocate too many processors to applications which cannot use them effectively. Moreover, processors are not preempted from an application until they terminate and hence new applications cannot be started immediately. Both of these factors might lead to performance degradation, especially at high system loads. Moreover, static scheduling policies typically hold back processors in order to effectively handle newly arriving jobs. Hence processors may not be effectively utilized especially when the variability in system load is quite high [23][6][14][25].

In *dynamic* scheduling processors can be reallocated at any point during a job's execution. Changes in processor allocation are usually triggered by events such as the completion of an application (released processors are redistributed among other applications), arrival of an application (processors may be taken from other applications in order to facilitate the immediate start of a new job), or due to changes in the parallelism of the job (changes in parallelism may result in an increase or decrease in the number of allocated processors). The frequent reallocation of processors introduces extra overhead due to context switches and the resultant loss of processor cache context. However, the advantage of dynamic scheduling lies in its ability to adapt itself to changing system conditions. Simulation and experimental studies by Zahorjan and McCann [31] and McCann, et al. [16], show that although dynamic scheduling incurs extra overhead, it improves performance over static scheduling methods in UMA multiprocessors. They report that the increased processor utilization more than offsets the overhead resulting from frequent processor reallocations. Hence, dynamic scheduling is generally preferred to static scheduling techniques. Moreover, static policies are greatly affected by new arrivals (as mentioned before, static policies hold back processors to effectively handle new arrivals) while dynamic policies are able to handle newly arriving jobs in a more effective manner (since processor preemption can occur at any point during a job's execution). For these reason we investigate the utility of using application characteristics in dynamic processor allocation policies.

2.5. Workload Characterization

Traditional models of uniprocessor systems have used only a single parameter to characterize the processor service demand of an application. When applications are parallelized the service demand is an inadequate characterization since it is the application which determines how many processors can be used at various points of its execution. As well, the patterns and frequency of synchronization and communication govern the execution of the application. Different characterizations of parallel applications have been proposed in the literature. At one extreme, complete application characterizations such as data dependency graphs [28] and Petri nets [15] have been proposed. However, these are not easy to use in practical situations since they require handling large amounts of data. At the opposite extreme, high-level characterizations of applications have been proposed and studied in the literature. These use a small number of parameters to model a parallel application. The following sections describe some of the high-level workload characterizations proposed and studied in the literature.

2.5.1. Parallelism Profile

A *parallelism profile* is defined as the number of processors an application is capable of using at any point in time during its execution [9](see Figure 2.1). Kumar describes a method for determining the parallelism profile of existing scientific applications [9]. Although a parallelism profile can provide valuable information about an application (e.g., maximum parallelism, fraction of sequential computation), it may be difficult to use in making scheduling decisions because of the amount of data required for its representation.

The average parallelism of a job, in some sense, provides information about the average number of processors that are busy during the lifetime of the job. It is defined as the average number of busy processors during the execution of an application when an unlimited number of processors are allocated to the job [5].

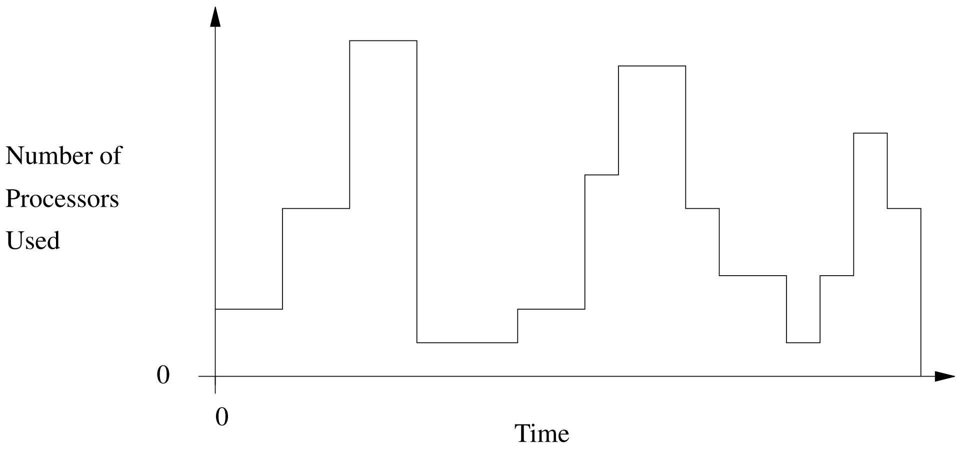

Figure 2.1: Parallelism profile

2.5.2. Speedup

The *speedup* of job J_i is defined as the ratio of the execution time attained using one processor, $T_i(1)$, to the execution time using p_i processors, $T_i(p_i)$.

$$S_i(p_i) = \frac{T_i(1)}{T_i(p_i)}$$

Here, $T_i(p_i)$ denotes the execution time of application, J_i, when executed in isolation using p_i processors. If the speedup of J_i is $S_i(p_i) = p_i$, it is said to execute with perfect speedup. It is assumed that a speedup of greater than p_i is not possible (for an allocation of p_i processors). However, perfect speedup is not achievable for most applications due to factors such as contention for shared resources, communication overhead, synchronization costs and inadequate parallelism in the application. As more processors are allocated to a parallel application the benefits of reduced execution time begin to decrease as these overheads increase. In some cases, the allocation of too many processors may result in a decrease in speedup (see Figure 2.2). This is because the increases in communication and synchronization costs associated with the use of additional processors may outweigh the benefits of decreased execution time. For a certain allocation of processors, commonly referred to as *pmax*, the speedup attains a maximum value after which it decreases for any extra allocation of processors. Figure 2.2 shows an example speedup curve including the maximum speedup, $S_i(pmax)$, which is obtained for an allocation of *pmax* processors. In this thesis we use the speedup characteristic to model the execution of a parallel job. Note that the insights obtained from using such high-level characterizations may lead to a better understanding of how to schedule with more detailed characteristics of parallel jobs such as the parallelism profile.

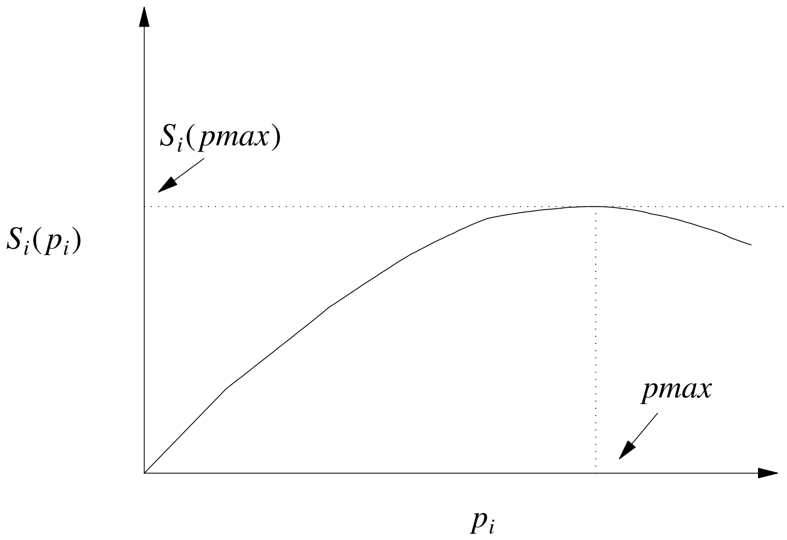

Figure 2.2: Speedup versus number of processors used

2.5.3. Efficiency

Efficiency is the mean effective utilization of the p_i processors allocated to job J_i [5]. The relationship between efficiency and speedup is given as:

$$E_i(p_i) = \frac{S_i(p_i)}{p_i} \, ,$$

where p_i denotes the number of processors allocated to job J_i and $S_i(p_i)$ is the speedup.

In a multiprogrammed environment, processors can be allocated to benefit one job at the expense of other jobs in the system. Speedup measures the benefit of using a number of processors for a parallel application, while efficiency measures the cost of using those processors [5]. The graph of execution time versus efficiency, called the execution time - efficiency profile, proposed by Eager, et al. [5], is shown in Figure 2.3. It shows the cost-benefit tradeoff in allocating additional processors to applications. The *knee* of the profile is that allocation of processors which results in the maximum benefit per unit cost of processor allocation (i.e., maximizes the ratio of efficiency to execution time, $E_i(p_i)/T_i(p_i)$). As pointed out by Eager, et al. [5], this may be useful in determining effective processor allocations in multiprocessor systems.

Ghosal, et al. [6] introduce another characterization of a parallel program called the processor working set (pws). They define the *pws* as the minimum number of processors that maximizes the speedup per unit cost function associated with the allocation of a processor. Ghosal, et al. assert that the pws coincides with the number of processors corresponding to the knee of the execution time - efficiency profile for a linear cost function and conclude that the pws

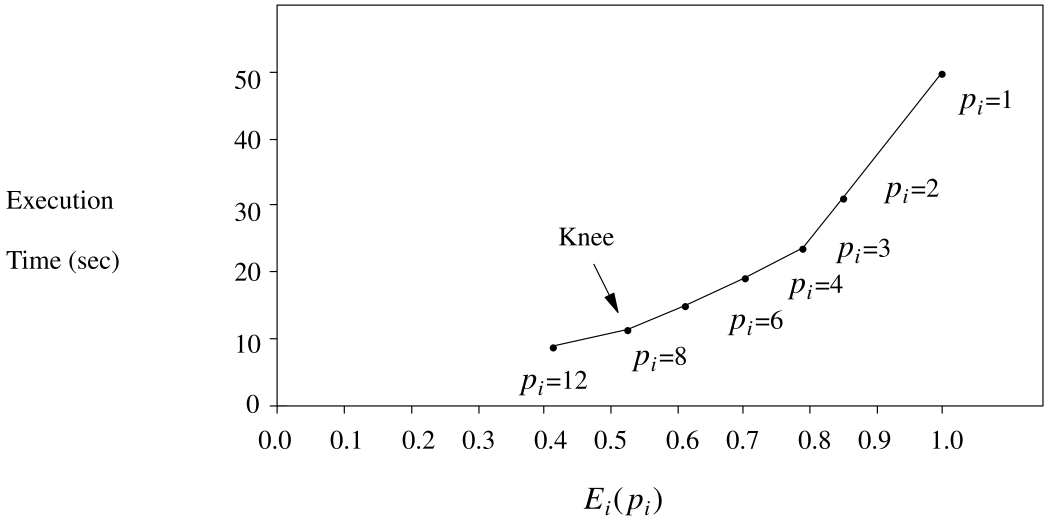

Figure 2.3: Execution time - efficiency profile

identifies an optimal system operating point. In this thesis we investigate the advantages of using information about a job's efficiency in making processor allocation decisions. We also examine the performance of a number of dynamic scheduling policies that allocate processors according to the knee.

2.5.4. Execution Rate Functions

An Execution Rate Function (ERF), γ, models the rate at which a program executes when allocated a specified number of processors. If p_i processors are allocated to job J_i, then its demand is satisfied at the rate γ. Note that the processor allocation, p_i, may change with time in a dynamic scheduling environment.

The execution time of a parallel program depends on a number of factors. These factors include the amount of basic work to be executed by the job, the imbalance of the load (service demand), the amount and type of synchronization and communication among the cooperating threads, as well as the costs incurred in creating multiple threads on different processors. These overheads can be modeled as inefficiencies of a job. Inefficiencies refer to the inability of a job to fully utilize all of the allocated processors. Due to inefficiencies, most jobs are unable to achieve perfect speedup. Execution rate functions are used to model the speedup curve of an application.

A number of different execution rate functions and job characteristics have been proposed and studied in the literature [2][25][4]. Dowdy [4] proposes an ERF, also called an *execution signature*, that has the following form:

$$\mu_i(p_i) = \frac{p_i}{D_{i1} + D_{i2} p_i} ,$$

where $\mu_i(p_i)$ is the execution rate of job J_i when it is assigned p_i processors and D_{i1} and D_{i2} are two empirically determined parameters reflecting the characteristics of job J_i and the underlying architecture. Chiang, et al. [2] use a different form of the Dowdy ERF which has been derived in the following manner:

The execution time of job J_i using one processor is, $T_i(1) = \dfrac{1}{\mu_i(1)} = D_{i1} + D_{i2}$ and the execution time using p_i processors is, $T_i(p_i) = \dfrac{1}{\mu_i(p_i)} = \dfrac{D_{i1} + D_{i2}p_i}{p_i}$. Therefore, the speedup of job J_i is $S_i(p_i) = \dfrac{T_i(1)}{T_i(p_i)}$, which is equivalent to $S_i(p_i) = \dfrac{D_{i1} + D_{i2}}{\dfrac{D_{i1} + D_{i2}p_i}{p_i}}$. Dividing the numerator and denominator by D_{i2}, $S_i(p_i) = \dfrac{(D_{i1}/D_{i2} + 1)p_i}{(D_{i1}/D_{i2} + p_i)}$. Now using $\beta_i = \dfrac{D_{i1}}{D_{i2}}$ we have $S_i(p_i) = \dfrac{(1 + \beta_i)p_i}{\beta_i + p_i}$ which is the form used by Chiang, et al. [2].

The advantage of this execution rate function is that it uses a single parameter, β_i $(0 \le \beta_i \le \infty)$ to characterize the parallel programming overheads of an application. Hence a parallel job, J_i, having the efficiency parameter β_i has an execution rate defined by:

$$\gamma = \frac{(1 + \beta_i)p_i}{\beta_i + p_i} .$$

Figure 2.4 plots the execution rate function for several values of β_i. The figure shows that all speedup curves are non-decreasing. It also shows that curves having lower β_i values (e.g., β_i = 10 and 41 in Figure 2.4) approach an asymptote as the allocation of processors approaches the maximum number of processors in the system (one hundred in this example).

One potential problem with the above execution rate function is that it is a non-decreasing function. In a multiprocessing environment speedup of some applications will decrease when more than *pmax* (as defined in Section 2.5.2) processors are allocated. This drop in speedup is due to an increase in synchronization and communication costs. Hence, this model is not a true representation of all types of jobs in multiprocessor workloads.

Sevcik [25] proposes a model of an application's execution time which takes into account the increased synchronization and communication costs.

$$T_i(p_i) = \phi_i(p_i)\,\frac{W_i}{p_i} + \alpha_i + \beta_i p_i .$$

Here T_i is the execution time of job J_i given an allocation of p_i processors. The parameters $\phi_i(p_i)$, α_i and $\beta_i(p_i)$ model the different parallel programming overheads. Figure 2.5 plots the

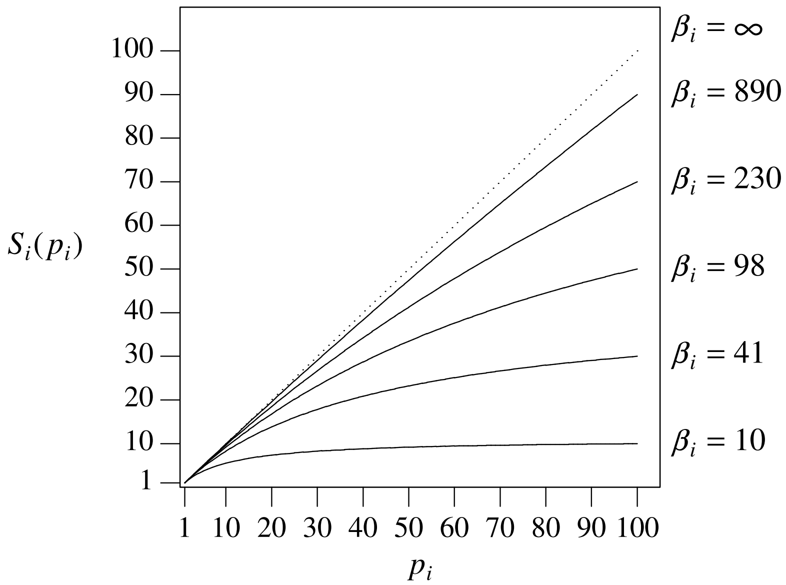

Figure 2.4: Curves for ERF $\gamma = \dfrac{(1+\beta_i)p_i}{\beta_i + p_i}$

execution time function for different parameters of the Sevcik model.

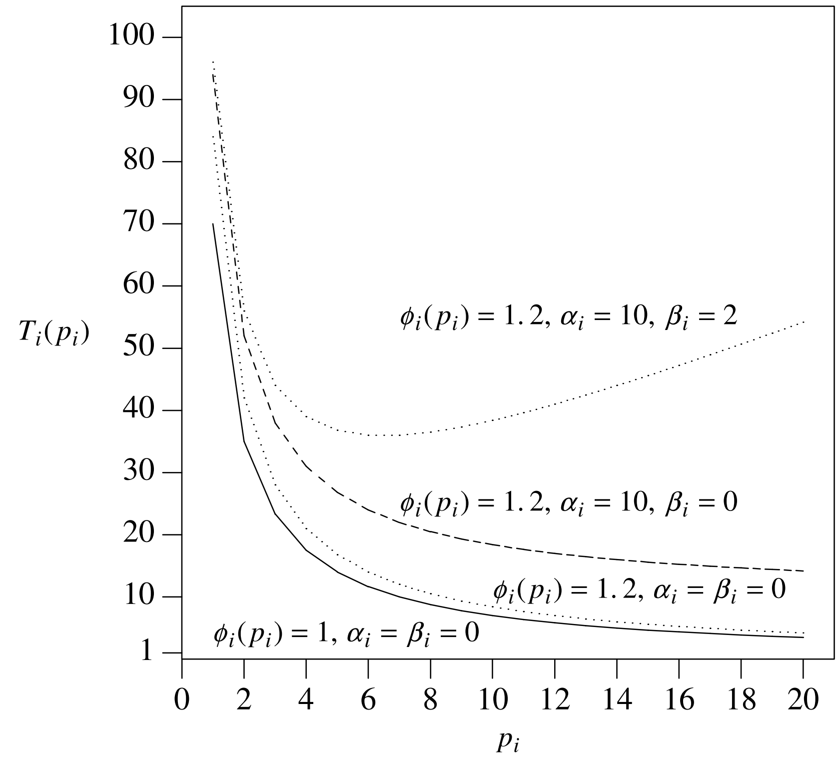

Figure 2.5: Curves for the Sevcik execution time function

Wu [30] examines the accuracy of the models proposed by Dowdy and Sevcik. He reports that although the curves obtained using the Dowdy model closely match the speedup of some of the applications considered, the model is inaccurate in other cases. This is true especially for those applications where the synchronization and communication costs are significantly high. However, the curves obtained using the Sevcik model and appropriate parameters are reported to match the actual speedup curves of the examined applications with surprising accuracy.

However, the Sevcik model is not used in this thesis for the following reasons:

1) There is a correlation between W_i and the speedup (or efficiency) of job J_i. In other words, for two jobs J_i and J_k where $\phi(p_i) = \phi(p_k)$, $\alpha_i = \alpha_k$, $\beta_i = \beta_k$ but $W_i > W_k$, job J_i will execute with higher efficiency (and consequently have better speedup) than job J_k. Although there might be a strong correlation between work and efficiency for some parallel applications, we feel that the correlation for other applications may be weak or non-existent. This may be especially true considering that different jobs will very likely be executing different applications. Ghosal, et al. [6] experimentally determine the speedup curves of some parallel applications and point out examples where the efficiency of an application is correlated with its work. They also cite an example where the application is not scalable and report that there is no significant increase in speedup with an increase in problem size.

2) The increased number of parameters for the Sevcik model including appropriate choices of means and distributions for each of the parameters would considerably increase the complexity of the experimentation and likely the interpretation of the results.

It should be noted at this point that the Dowdy model is simpler than the Sevcik model as fewer parameters are needed to represent a parallel application. In essence, choosing between the two models represents a tradeoff between the simplicity and the accuracy of the function, since the use of more parameters more accurately model a parallel application's execution thus increasing the complexity of the model. We emphasize the simplicity of the execution rate function in order to obtain insights into the scheduling problem. Therefore, the function proposed by Dowdy [4] is used for modeling the execution of parallel jobs.

2.6. Scheduling Policies

This section describes some of the multiprocessor scheduling policies investigated in previous research. Some of these policies use job characteristics in making scheduling decisions. As well, the principles behind some of these policies have been derived from uniprocessor scheduling policies. In this section, we examine the importance of using job characteristics in scheduling.

2.6.1. Policies Using No Job Characteristics

The Equipartition policy [27][7], has been widely studied and extensively compared with other processor allocation policies [31][12][11][16][2]. Equipartition is a dynamic space-sharing algorithm and is very practical in the sense that it can be implemented easily, since no job characteristics are required when making allocation decisions. Moreover, it has been observed to be robust in that it provides quite good mean response times over a wide variety of workloads [31][12][16]. *Equipartition* [27][7] allocates an equal fraction of the processing power to all jobs in the system. It has been claimed by some researchers that this equal sharing of processing power is a desirable property of a good scheduler [12][11]. In the Equipartition policy reallocations are done only at job arrivals and departures. However, in Zahorjan and McCann's Dynamic Equipartition policy [31], reallocations are also possible whenever the parallelism of a job changes during execution. Because the job model used in this thesis does not explicitly model variations in job parallelism, we examine a policy which reallocates processors only at job arrivals and departures and call it Equipartition.

The Equipartition policy can be expressed as:

$$p_i = \frac{P}{n},$$

where p_i is the number of processors allocated to job J_i, P is the number of processors in the system and n denotes the number of jobs.

2.6.2. Using Characteristics of Work

The amount of work to be executed and the efficiency of an application, along with the number of processors it is allocated, will determine the execution time of a parallel application. Hence, precise knowledge (if available) or estimates of the application's service demand may be helpful when making scheduling decisions.

Majumdar, et al. [13] have investigated the importance of using work characteristics such as cumulative job demand or predictors such as the number of processes (in a job) in making scheduling decisions. In their simulation study, they compare a Round-Robin time-sharing policy with policies that either know or estimate a job's cumulative service demand. They find that their policies SCDF (Shortest Cumulative Demand First) and SCDF* (pre-emptive SCDF) improve mean response time significantly when compared with the Round-Robin policy and that the improvements increase with the coefficient of variation of the cumulative job service demand. They also propose a policy called Smallest Number of Processes First (SNPF) and a preemptive version (SNPF*). These policies are designed to emulate the behavior of the SCDF policies by estimating a job's cumulative service demand (using the number of processes) and making scheduling decisions based on that estimate. They find that if the total amount of work a job executes is correlated with the number of parallel processes it executes, SNPF* performs quite well and improves performance significantly when compared with FCFS and RR (when the coefficient of variation in service demand is high).

Leutenegger and Vernon also perform a simulation study of a number of scheduling policies. Their simulation results also show that SCDF* policy performs well. However, they express reservations about the feasibility of implementing the SCDF* policy and hence conclude that policies that allocate processing power equally among all jobs (i.e., RR-based) perform best [12].

Leutenegger and Nelson [11] use a simple job model and a small number of jobs (two or three) in order to compare a number of semi-static and dynamic policies with an optimal scheduling policy. The small number of jobs enables them to exhaustively determine the optimal schedule. They also conclude that policies that attempt to allocate processing power equally among jobs perform well in both environments and conclude that this is a desirable property of a good scheduler.

Sevcik [25] has proven that if jobs execute with perfect efficiency and there are no new arrivals, the Least Work First (LWF) policy is optimal. With new arrivals, however, a Least Remaining Work First (LRWF) policy is optimal (preemption overhead is neglected) [1]. However, no optimal policy is known which takes into account the overheads associated with parallel programs (imperfect job efficiency). This is due to the fact that, a combinatorial number of possibilities exist when considering different overhead factors.

Brecht [1] makes simplifying assumptions about the workload in order to obtain insights into the problem. He performs analytic studies to show that under conditions in which all jobs arrive simultaneously and execute with perfect efficiency, Equipartition is two-competitive. That is, it is guaranteed to be within a factor of two of the optimal algorithm (LWF). He also shows that when the assumption of simultaneous arrivals is relaxed, Equipartition is not guaranteed to be within a constant factor of optimal. Brecht also conducts experimental studies to confirm some of his analytical results. From Brecht's study, the following conclusions are quite apparent:

1) At heavy loads LRWF performs significantly better than Equipartition (assuming perfect efficiency). The difference in response times of the two policies increases with the system load.
2) At lighter loads, the performance of Equipartition approaches that of LRWF (assuming perfect efficiency). Moreover, at extremely light loads performance of Equipartition equals that of LRWF.

3) Performance of LRWF degrades with the decrease in efficiency of jobs until, at some point, Equipartition performs better than LRWF. The intuition behind this is that, with low efficiency, jobs cannot effectively use all of the processors in the system. In such cases it is preferable to space-share the processors (Equipartition) leading to more effective processor utilization.

In this thesis we approximate the LRWF policy by using a generalized processor allocation policy proposed by Brecht [1] (described below). Policies which share processors among jobs while at the same time biasing against larger jobs or less efficient jobs (the bias depending on a control parameter) are also examined by using the generalized allocation scheme. All of these policies (which use job characteristics) are compared against Equipartition to determine the advantages of using job characteristics in dynamic scheduling.

The generalized processor allocation policy proposed by Brecht is described below:

$$p_i = \frac{P W_i^{\alpha}}{\sum_{j=1}^{n} W_j^{\alpha}}$$

In this generalization, W_i is the amount of remaining work to be executed by job J_i, P is the number of processors in the system, n is the number of jobs currently executing, α is the control that determines the actual allocation policy, and p_i is the number of processors allocated to job J_i as a result of the policy. Different values of α represent various points on a spectrum of processor allocation strategies. For example, larger positive values of α allocate a greater portion of processors to jobs with more work (larger jobs) while larger negative values of α allocate a greater portion of processors to jobs with less work (smaller jobs). Values of $\alpha = 0$ and $\alpha = -\infty$ represent the Equipartition and the LRWF policies respectively. (A value of $\alpha = \infty$ represents a Most Remaining Work First policy.) Other specific values of α worth noting are $\alpha = 1$ and $\alpha = 0.5$. The policy obtained when $\alpha = 1$ allocates processors in direct proportion to the work being executed by each job [1]. The policy obtained when $\alpha = 0.5$ allocates processors in proportion to the square root of the work being executed by each application and has been examined by Sevcik [25] and Brecht [1] in the context of static scheduling algorithms. In Chapters 5 and 6 of this thesis we use a modified version of the generalized allocation policy in order to determine the importance of using characteristics of work or efficiency in making scheduling decisions.

The policies described to this point use precise information about the service demand of a parallel job. If precise information is not available to the scheduler at run time, estimates of the service demand can be obtained by the system. Leland and Ott [10] study the behavior of UNIX processes and conclude that there is an almost perfect linear correlation between the CPU time used by a process and the amount of processing time they will require in the future. The multi-level feedback policy used in uniprocessors systems makes use of this observation. Brecht [1] combines the idea of estimating the remaining work using the accumulated CPU time of the application with the LRWF policy. This policy is more realistic than policies which assume they have precise information of the service requirement of a parallel job, in the sense that it can be implemented in a real multiprocessor. However, extensive experimentation using these policies was not carried out with realistic workloads.

In this thesis we use similar estimates to determine a job's remaining execution time. We show, in Chapter 5, that these estimates of work can be used to reduce mean response times when compared with policies that share processing power equally among jobs, provided jobs execute with perfect efficiency. We also expect that these policies (that use estimates of work) can improve performance over Equipartition for workloads that have relatively high average efficiency. This is because for workloads with high average efficiency, policies that use precise knowledge of work can significantly reduce mean response times over Equipartition (this is shown in Chapter 5).

2.6.3. Using Other Job Characteristics

Other job characteristics (e.g., pws, average parallelism and efficiency) have been used by schedulers to determine processor allocations in multiprocessors [6][25][2][24][14]. A number of allocation policies that use job characteristics have been investigated in a static environment.

Ghosal, et al. [6] study a number of static scheduling algorithms that use information about a job's processor working set (pws) and show that these policies perform well under a varying system load. Majumdar, et al. [14] also report that in a static scheduling environment, allocating processors near the knee (which is equivalent to the pws for a linear cost function) is effective in producing low mean response times over a broad range of system loads. A study by Sevcik [24] reports that policies considering additional information about an application's parallelism, such as the minimum, maximum, and variation in parallelism, improve mean response time over methods that only consider average parallelism.

The studies by Ghosal, et al. [6], Majumdar, et al. [14] and Sevcik [24] conclude that policies that use application characteristics perform better than policies that use none. However, the opposite conclusion is drawn by some researchers[2]. Chiang, et al. [2] assert that policies which do not use application characteristics can still improve mean response time over policies that use knowledge of job characteristics in a static scheduling environment.

The results reported in the study by Setia, et al. [22] also indicate that the policies considered using job characteristics do not improve mean response times when compared to policies that use none. In this thesis we demonstrate the substantial benefits that can be obtained in a dynamic scheduling environment by using job characteristics (work and efficiency).

The Adaptive Static Partitioning policy (ASP) [21] uses no job characteristics (only the available parallelism is used) in making scheduling decisions and has been examined in several studies [21][22][2][17][18][19]. Setia, et al. [22] show that ASP outperforms policies that use knowledge of the pws (at moderate to high system loads). Chiang, et al. [2] also report similar results. Chiang, et al. also study a number of run-to-completion (static) algorithms that use information about a job's execution rate characteristics (average parallelism and pws). They assert that policies that use execution rate characteristics do not improve performance (compared with policies that use none), especially when the coefficient of variation of job service demand is greater than one.

Zahorjan and McCann [31] and McCann, et al. [16] use simple information about an job's current degree of parallelism to reallocate processors in response to changes in the parallelism of the job; in a sense attempting to utilize processors more effectively. This dynamic allocation policy reduces mean response time when compared with strictly repartitioning processors upon job arrivals and departures [27][7]. The advantage of this dynamic scheduling policy is that it may adjust to a job's demands for processors over time. This indicates that using simple information about a job's parallelism can improve mean response time over policies that use no information.

Most of the studies discussed in this section investigate the utility of using application characteristics in a static environment. Although some studies report that using job characteristics in a static scheduling environment is useful [6][14][24], the opposite conclusion is also reported in the literature [21][2]. We believe that this is mainly due to the difference in the workloads considered in these studies. In this thesis we consider simplistic models of a job and the workload and try to gain a first-order understanding of the problem in a dynamic scheduling environment. (We investigate the advantages of using job characteristics in a dynamic environment because of the reasons mentioned in Section 2.4.)

2.6.4. Using Characteristics of Work and Efficiency

In this section, we present an overview of a static policy that uses characteristics of both work and efficiency in making scheduling decisions. Sevcik [25] uses the model described in Section 2.5.4 to characterize parallel applications and extensively examines the problem of statically allocating processors to a batch of parallel applications given *a priori* knowledge of these parameters. A number of simplified cases are explored. The insights gained by studying these problems are used to develop a general allocation scheme which uses the characteristics of work and efficiency of a job in making allocation decisions. However, the effectiveness of this scheduling policy is not evaluated in Sevcik's study.

2.7. Summary

In this chapter we describe different techniques for characterizing parallel applications and a number of policies for scheduling processors in multiprogrammed multiprocessors. From previous research, the following conclusions are apparent:

1) Space-sharing policies are preferred to time-sharing policies, since they allow equal sharing of processing power without incurring the overheads due to preemption.

2) Dynamic scheduling policies are preferred over static policies.

3) If the work to be executed by the application is known *a priori*, optimal scheduling using the Least Remaining Work First (LRWF) policy is only possible if the jobs execute with perfect efficiency. The optimal schedule is not known for jobs executing with imperfect efficiency.

A number of recent studies have demonstrated that, in a static environment, policies that use job characteristics perform better than policies that do not. However, the opposite conclusion is also reported in the literature. In this thesis we investigate the advantages of using job characteristics in a dynamic environment. Specifically, we want to demonstrate and quantify the reductions in mean response time that can be obtained by using knowledge of the work and efficiency in making allocation decisions (when compared with the Equipartition policy, which does not use such knowledge).

Chapter 3

The Job, Workload, and System Models

3.1. Introduction

This thesis tries to obtain a first order understanding of the multiprocessor scheduling problem. In order to achieve this objective different scheduling policies are analyzed using a simulated model of a multiprocessor. This chapter discusses the model used to capture the essential features of parallel jobs and the system. The goal is to use a simple model that broadly represents multiprocessors, characterizes the essential features of parallel workloads, contains a small number of parameters, and is easy to analyze. Thus a simplistic model is adopted which is described in the subsequent sections of this chapter.

3.2. The Job Model

We characterize jobs in the system with the following two parameters:

1) Work: W_i is the amount of basic work executed by job J_i. This corresponds to the execution time required to execute the program sequentially (i.e., the service demand). In other words, $W_i = T_i(1)$.

2) Execution rate parameter: β_i is used to characterize the rate at which the work, W_i, is executed by job J_i when allocated a specified number of processors, p_i. That is, the execution rate function along with p_i and β_i determine the speedup of the parallel job.

The parallel programming overheads of job J_i are modeled as inefficiencies which are parameterized by the job characteristic β_i. The characteristic β_i is a parameter to the function γ, which determines the execution rate of job J_i. If p_i processors (possibly fractional) are allocated to job J_i, then its demand is satisfied at the rate $\gamma(p_i, \beta_i)$. Note that the execution rate function, $\gamma(p_i, \beta_i)$, is referred to as γ in this thesis. The execution rate function, γ, models an application's speedup. The function, γ, is the same across all jobs while different execution rate parameters, β_i, characterize the speedup of the different parallel jobs.

The following execution rate function, proposed by Dowdy [4] and also used by Chiang, et al. [2], is also used in this thesis to model the speedup of parallel jobs:

$$\gamma = \frac{(1 + \beta_i) p_i}{p_i + \beta_i}, \qquad 0 \le p_i \le P \text{ and } 0 \le \beta_i \le \infty$$

Figure 3.1 shows speedup curves for different values of β_i.

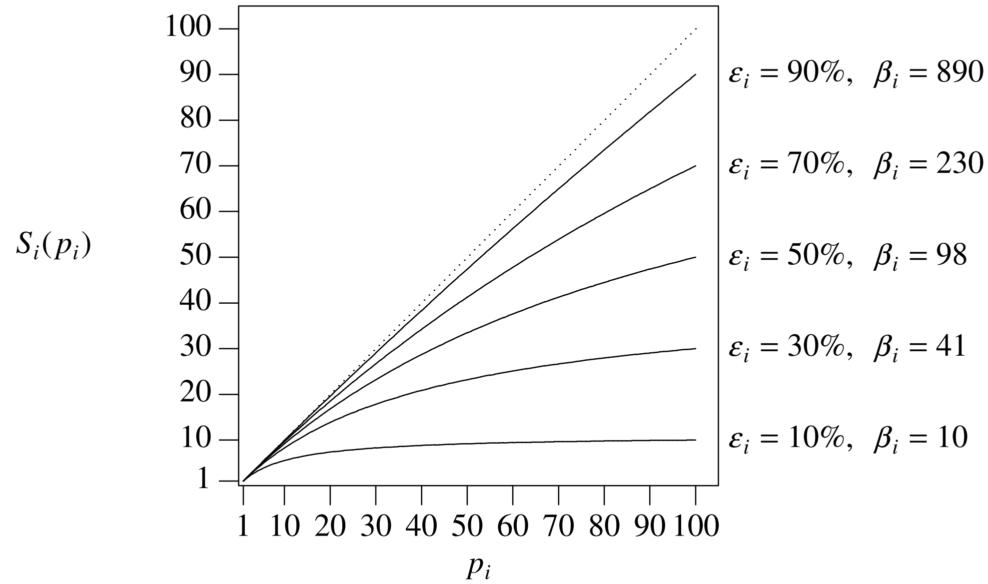

Figure 3.1: Speedup curves for execution rate function $\gamma = \dfrac{(1+\beta_i)p_i}{\beta_i + p_i}$

The value ε_i (called the *effective efficiency*) shown in Figure 3.1 is defined to be the number of processors effectively used if all P processors are allocated to the job ($\varepsilon_i = E_i(P) \cdot 100$). When working with the execution rate function, γ, we found it difficult to think in terms of β_i values for the execution rate. (For example, does $\beta_i = 300$ mean the job attains good speedup? The answer depends on the number of processors used.) Therefore, we use effective efficiency to express the speedup attained as a percentage of the number of processors used if all P processors are allocated to the job. Therefore, $\varepsilon_i = 90\%$ means that if job J_i is executed in isolation using all of the processors, its speedup would be 90 percent of P.

$$\varepsilon_i \;=\; E_i(P) \cdot 100 \;=\; \frac{100\ S_i(P)}{P} \;=\; \frac{100\ (1+\beta_i)}{(P+\beta_i)}$$

For a given a value ε_i we can compute the corresponding β_i as follows:

$$\beta_i \;=\; \frac{P\ \varepsilon_i - 100}{100 - \varepsilon_i},\quad \frac{100}{P} \le \varepsilon_i < 100 \text{ and } P > 1 \text{ (if } \varepsilon_i = 100,\ \beta_i = \infty \text{)}.$$

(Note that a lower bound on ε_i is $\dfrac{100}{P}$, since the efficiency parameter, β_i, is bounded by zero.) Using this approach (i.e., using effective efficiency) we gain an intuitive feeling for possible values and perhaps even representative distributions of ε_i. Therefore we can think and perform experiments using these ε_i values while the simulator internally uses the corresponding β_i values.

The main reason that this function is not a completely accurate representative of the execution rate function of all jobs is that it is a non-decreasing function. In a large multiprocessing environment the speedup of most applications will almost certainly decrease (if enough processors are allocated) due to increases in synchronization and communications costs (described in Chapter2). However, the advantage of this model lies in its simplicity since it essentially captures all parallel programming overheads of a job by means of a single parameter β_i.

In our job model we do not explicitly model the individual threads within a parallel program. In fact there is an implicit assumption, informally based on a user-level thread programming model, that there are always at least P threads available to execute the remaining work. This results in considerable simplification of the job model. The purpose of this thesis is to gain an insight into scheduling policy behavior. Hence we feel that, these assumptions which approximately represent the essential characteristics of a job will provide us with a high-level first-order understanding while keeping the problem tractable. However, we revisit the job model in Chapter 7 and consider factors that limit processor allocation to a job; namely the amount of available parallelism and pmax.

3.3. The Workload model

The workload consists of a set of jobs, each job, J_i, being characterized by its work, W_i, and execution rate parameter, β_i. The system executes the workload defined by the parameters described subsequently in this section.

Each trial is conducted by executing M jobs. A number of different trials, S, are performed by incrementing the seed value of the random number generators used to generate the job and workload parameters. The number of jobs in each run, M, and the number of runs, S, are adjusted in order to obtain 90% confidence intervals that are within 5% of the mean (for all but few cases where the coefficient of variation in service demand, C_W, is 30). We mostly use the classical method for computing confidence intervals although it requires a large number of trials (usually greater than 30). A large number of trials ensures that the observed means follow a normal distribution [8]. (The classical method for computing confidence intervals assumes that the samples are normally distributed.) When the experimentation time required for a large number of trials is prohibitive and when the sample means are not distributed according to a normal distribution, we use the bootstrap method [29] to compute confidence intervals. Note that the bootstrap method is robust for a small number of trials and also for non-normal distribution of observed means.

Job arrivals are assumed to follow a Poisson distribution with mean λ. If the job arrival process is Poisson, the inter-arrival times are distributed exponentially with a mean of $\frac{1}{\lambda}$. We choose the job-arrival times from an exponential distribution with a mean of $\frac{1}{\lambda}$ where λ is computed based on the desired system load. We use the same job arrival rate for all scheduling policies evaluated within an individual experiment. However, the processor utilization may vary across the different allocation policies. This is because the execution rate (the efficiency) of a job is a function of the processor allocation, p_i, and the processor allocation depends on the scheduling policy. Hence, jobs execute with different execution rates for different policies and therefore, the system load (or processor utilization) varies over different scheduling policies for a fixed job arrival rate.

We experimentally determine the arrival rate so that the desired processor utilization is obtained. This is done by determining the scheduling policy which yields the maximum processor utilization (among all policies investigated within the individual experiment) and then using that scheduling policy to determine the arrival rate required to achieve the desired processor utilization. This arrival rate is then uniformly used for all of the scheduling policies in that experiment. A list of arrival rates used for each experiment can be found in the appendix.

The work to be executed, W_i, by job J_i, is drawn from a Hypergeometric distribution with mean $\bar{W} = 1000$. The mean was chosen so that jobs finish execution within a reasonable amount of time in a system of one hundred processors and is also consistent with values used by Parsons and Sevcik [26]. We consider coefficients of variation in service demand, C_W, of 1, 5 and 30. Higher values of C_W signify higher variations in service demand of the jobs. Table 3.1 shows a breakdown of the service demands for a workload of $M = 500{,}000$ jobs and C_W values of 1, 5 and 30. For higher values of C_W, the number of smaller jobs is greater while the number of medium sized jobs is greater for a lower value of C_W. For very high values of C_W there are some jobs with very large service requirements which are absent for lower values of C_W. Hence, a higher value of C_W implies a higher degree of polarization in job service demand.

Chiang et al. [2] provide justification for using such values of coefficient of variation. They report that the coefficient of variation in service demand on their CM-5 machine ranges from 2.5 to about 6, with 40% of the measures being above 4.0. They also report that measurements taken at a Cray YMP site have observed C_W to be in the range of 30-70.

We are not aware of studies that characterize parallel programs in terms of the average speedup or efficiency of a job or in terms of the distribution of efficiency across different jobs (i.e., $\bar{\varepsilon}$ or its distribution, D_ε). In most parts of the study by Chiang, et. al [2] the efficiency parameter of parallel jobs are constant. That is, all jobs in the workload have the same efficiency

Table of Service Demand, C_w = 1, 5, 30			
Range	C_W=1 (Jobs %)	C_W=5 (Jobs %)	C_W=30 (Jobs %)
0-1000	63.199	84.251	86.341
1000-5000	36.125	14.037	13.595
5000-50,000	0.675	1.346	0.008
50,000-1,000,000	0.000	0.278	0.035
>1,000,000	0.000	0.000	0.033

Table 3.1: Table of service demand for 500,00 jobs and C_W = 1, 5, 30

parameter. We feel that this assumption is unrealistic since jobs will not all execute with the same efficiency. We also hypothesize that differences in efficiency will possibly affect the outcome of experiments. In this thesis the effective efficiency parameter, ε_i, for job J_i is randomly chosen from a specified distribution.

The distributions considered for ε_i are as follows:

1) Uniform Distribution
   In this case, the effective efficiency of applications is uniformly distributed between $\varepsilon_{\min}$ and $\varepsilon_{\max}$ where $\varepsilon_{\min}$ and $\varepsilon_{\max}$ are input parameters to the workload. The uniform distribution is conceptually simple to understand and permits us to easily and directly control the mean and, to the extent possible, the variance. This permits us to examine a wide variety of workloads. Note that the efficiency parameter, β_i, of applications are not uniformly distributed. However, the efficiency parameters are assumed to be uniformly distributed in the studies by Setia [20] and McCann and Zahorjan [32]. Note that a uniform distribution of β_i values will not result in a uniform distribution of effective efficiency values.

2) Beta Distribution
   We hypothesize that if a number of well tuned parallel programs were examined, the distribution, D_ε, of their effective efficiency values, ε_i, would approximately follow the probability distribution function of a Beta distribution. The Beta distribution uses two input parameters p and q, which determine the shape of the probability distribution function (examples of a number of parameters are shown in Figure 3.2).

Table 3.2 lists the parameters, their respective means and the coefficients of variation for the Uniform and the Beta distributions. Note that these parameters are used in this thesis for generating the effective efficiency of parallel applications. The probability density functions shown in Figure 3.2 have been obtained using parameters that are a close approximation of the Beta distribution parameters shown in Table 3.2. The parameters for the Beta distribution were chosen so that the $\bar{\varepsilon}$ and C_ε values are similar for both the Uniform and Beta distributions. Note

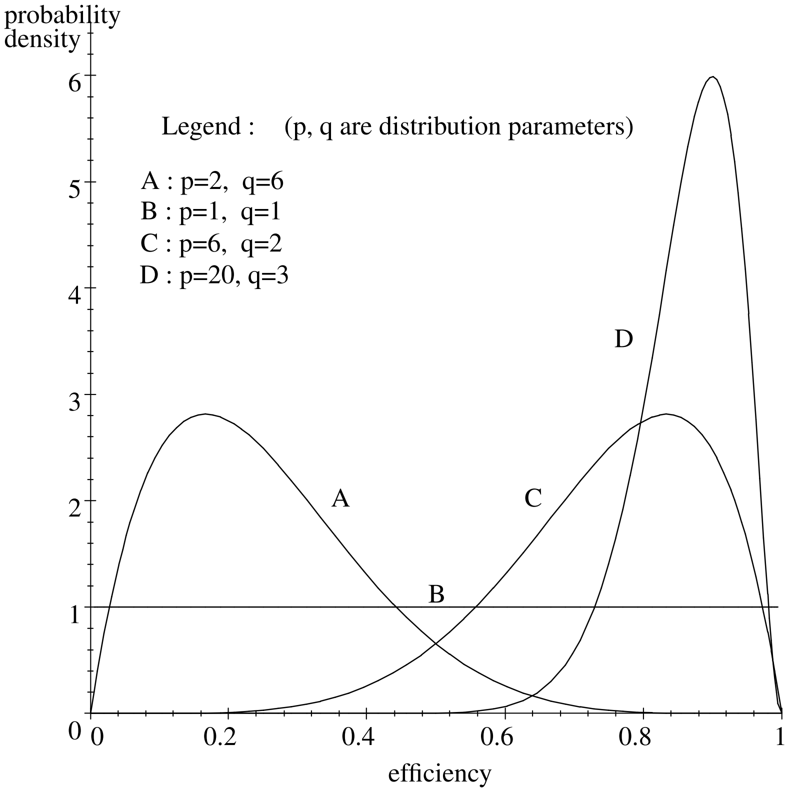

Figure 3.2: Density Function for the Beta distribution

$\bar{\varepsilon}$	C_ε	Uniform		Beta	
		$\varepsilon_{\min}$	$\varepsilon_{\max}$	p	q
25.5	0.566	1	50	2.166	6.328
50.0	0.571	1	99	1.062	1.062
74.5	0.194	50	99	6.328	2.166
87.0	0.082	75	99	19.629	2.933

Table 3.2: Distributions and parameters used in generating effective efficiency

that Beta distribution parameters of $p = 1$ and $q = 1$ and Uniform distribution parameters $\varepsilon_{\min} = 0$ and $\varepsilon_{\max} = 100$ have similar values of $\bar{\varepsilon}$ and C_ε (not shown in Table 3.2). We hypothesize that a Beta distribution with parameters of approximately $p = 6$ and $q = 2$ (the curve labeled C in Figure 3.2) would provide a close representation of real workloads in parallel machines. The reasons for this are:

a) As multiprocessors are becoming more prevalent, improved parallel programming techniques have resulted in the development of more efficient parallel programs. We feel that further improvements in parallel programming techniques and multiprocessor design will likely increase the efficiency of future parallel applications.

b) We speculate that inefficient jobs would be relegated to sequential machines and hence the number of jobs with low efficiency would be relatively small in multiprocessor workloads.

c) On the other hand, perfect (100%) or very high efficiency (95%) are not realizable for most parallel applications. This may be due to architectural constraints, sequential phases in the application and other parallel programming overheads.

Although we hypothesize that the effective efficiencies of parallel programs may be distributed according to a Beta distribution, we use a Uniform distribution in most of our experiments. This is because it is conceptually simple and also because it has been used by other researchers [20][32]. The Beta distribution is only used in Chapter 5 to examine the sensitivity of some of the results to the distribution of effective efficiencies. (The results were found to be qualitatively insensitive to this change in the distribution.)

The following list summarizes the workload parameters:

- $\bar{W}$ mean service demand (1000 time units)
- C_W coefficient of variation of W_i (we examine 1, 5 and 30)
- D_W distribution used to obtain W_i (Hypergeometric)
- $\bar{\varepsilon}$ mean effective efficiency (varies with experiment)
- C_ε coefficient of variation of ε_i (if applicable)
- D_ε distribution used to obtain ε_i (Uniform and Beta)
- λ job arrival rate
- M number of jobs to be executed (usually $\geq$ 500,000) for one run
- S number of trials to execute (used to generate confidence intervals)

3.4. The System Model

The primary objective of this research is to obtain a first-order understanding of scheduling policy behavior in multiprocessor systems. Hence, we only model the processors of a parallel system. Consideration of I/O and memory requirements are not modeled or simulated since they would complicate the simulation model and significantly increase the running time of the simulation. Moreover, the effects of the scheduling policy on mean response time would have been extremely difficult to isolate. Therefore, we consider a simplistic model of a medium-scale multiprocessor with one hundred processors.

In order to determine how different allocations of processing power affect the mean response time we assume that scheduling decisions can be performed instantaneously with negligible overhead. McCann, et al. [16] report that in their implementation of the Dynamic Equipartition policy on the Sequent Symmetry, scheduling overhead adds less than 1% to the response time. We also assume that the number of processors allocated to a job can be fractional. Applications that are allocated a fraction of a processor are assumed to be time-shared in appropriate proportions with negligible overheads.

3.5. Summary

The analysis of scheduling policy behavior using simulation techniques involves the design of a job, workload and system model, this chapter discusses the models used in this thesis.

Our characterization of a job J_i consists of two components: the Work, W_i, and the execution rate parameter, β_i. The parameter β_i determines the rate at which the service requirement, W_i, of job, J_i, is satisfied.

The workload consists of a set of jobs with each job being parameterized by the characteristics of work and efficiency. The parameter for work is a stochastic variable in our model, whose distributions is chosen to resemble those found in real parallel systems. Unfortunately, we are unaware of any study that determines the distribution of the effective efficiency parameter in the workload of a production multiprocessor. Due to its conceptual simplicity, an Uniform distribution is used for generating efficiency parameters for jobs in the workload. We also use the Beta distribution to analyze the sensitivity of the results to the distribution of the efficiency parameter.

The system model assumes that the scheduler allocates processors to the jobs in the system with negligible overhead. We emphasize the simplicity of the model to isolate the effects of the scheduling policy on the behavior and performance of the system.

Chapter 4

The Scheduling Algorithms

4.1. Introduction

This chapter describes the scheduling algorithms investigated in this thesis. All of the policies studied in this thesis are dynamic scheduling algorithms and follow the space-sharing approach to allocate processors among the applications in the system. These scheduling policies can be broadly classified into three families of algorithms; namely the Equi-Allocation, Generalized Allocation and the Work and Efficiency families. The principle behind each of the families along with detailed descriptions of the policies are presented in this chapter.

4.2. The Equi-Allocation Family

The guiding principle of this family of algorithms is to allocate an equal share of the processing power to all applications. This family of algorithms consists of the following three algorithms. The Equipartition policy [27][7] (described in Chapter 2) has no *a priori* knowledge of job characteristics and allocates an equal number of processors to each job. Chiang, et al. [2] use a modified version of Equipartition where knowledge of the amount of parallelism available in a job is used to bound the number of allocated processors. Note that processor reallocations in both of these policies are only made at job arrivals and departures. Zahorjan and McCann's [31] Dynamic Equipartition Policy also uses knowledge of a job's current degree of parallelism and allocates equal processing power to the jobs while also ensuring that jobs are not allocated more processors than they have parallel threads. Note that in this policy, processor reallocations can be made during job arrivals, departures and when the degree of parallelism changes during the execution of the job.

The Equipartition policy is used in this thesis as a basis of comparison against algorithms which use job characteristics in making scheduling decisions. The Equipartition policy can also be represented using the Generalized Allocation family of algorithms which is described in the next section.

4.3. The Generalized Allocation Family

In order to explore different methods of allocating processing power to jobs we employ the generalization of processor allocation policies proposed by Brecht [1] (described in Chapter 2). The generalization is defined as:

$$p_i = \frac{P\,W_i^{\alpha}}{\sum_{j=1}^{n} W_j^{\alpha}} .$$

We make a further generalization by substituting X_i for W_i, where X_i is a function of some characteristics of the parallel execution of job J_i. In this thesis we consider characteristics of work and efficiency and perform simulation studies in order to evaluate their use in determining processor allocations and the effectiveness of these allocations in minimizing the mean response time. Note that we only consider those cases where X_i is either the work or the efficiency.

The generalized form used throughout this thesis is:

$$p_i = \frac{P\,X_i^{\alpha}}{\sum_{j=1}^{n} X_j^{\alpha}}$$

The system scheduler uses the above generalized form in distributing the processors among the active jobs in the system. The general algorithm for these policies is now described.

Jobs in the system are kept in an active list. The scheduler is invoked and it partitions the processors among the jobs in the active list. The scheduling algorithm limits the number of jobs in the active list to be less than or equal to the number of processors, P. If the number of jobs in the active list is equal to the number of processors, subsequent jobs are placed in a wait queue. When a job finishes execution a new job is transferred from the wait queue (if there is one) to the active list and the scheduler is invoked. In other words, the system always ensures that the active list size is always less than or equal to the number of processors in the system. Note that processor reallocations are done at job arrivals and departures only.

The Generalized Algorithm:

- Upon the arrival of job J_i:

  if the number of jobs in the active list < P {

  add J_i to the active list and

  repartition processors among jobs in the active list according to $p_i = \frac{P\,X_i^{\alpha}}{\sum_{j=1}^{n} X_j^{\alpha}}$

  }

  else add J_i to the wait queue

- Upon the departure of job J_i:

  if the wait queue is not empty {

  move the first job from the wait queue to the active list

  }

  now repartition processors among jobs in the active list according to $p_i = \frac{P\ X_i^{\alpha}}{\sum_{j=1}^{n} X_j^{\alpha}}$

Note, that a job in the active list may be allocated zero processors by the partitioning scheme. In this algorithm once a job is added to the active list it is never moved back into the wait queue. Obvious variations of this algorithm exist. However, during our simulated experiments we found that the wait queue was almost always empty. Therefore, different techniques for maintaining the active list and the wait queue were not investigated.

4.4. The Work and Efficiency Family

In this thesis we introduce a new family of algorithms called the Work and Efficiency family (W&E). This family is similar to the generalized allocation algorithms in that it also uses application characteristics, namely the work and the efficiency in making processor allocations. However, the major difference is that the W&E family of algorithms use knowledge of both work and efficiency in making processor allocations while the generalized allocation algorithms we consider use only the work or the efficiency. Our results show that algorithms which use only work or only efficiency are not likely to substantially improve mean response time over Equipartition across the range of workloads considered in this thesis. This motivates the use of both work and efficiency in making effective processor allocations. The allocation of processors to jobs in the W&E class is done in two independent steps or phases with the allocation in the first phase being dependent on work and efficiency. The second phase, however, employs the Equipartition policy to distribute any processors that were not allocated during the first phase among the jobs in the active list.

The W&E algorithms maintain a sorted list of jobs in the system (sorted by increasing W_i). This ordered list is used to try to prevent short jobs from becoming stuck behind large jobs, as can be the case when considering only efficiency. During the first phase of allocation, jobs are considered for activation in this order and are assigned f_i processors, where f_i is determined by considering the efficiency with which the jobs execute. (We consider three different approaches for determining f_i.) Processors are assigned until either all P processors have been assigned or until all jobs have been activated. If all jobs have been activated and all P processors have not been assigned, a second phase is performed to assign the remaining processors to the activated jobs. A number of different assignment policies are possible for the first and second phases. In this thesis we focus on policies for the first phase, since we believe that the number of unassigned processors left for the second phase will be relatively small (especially under workloads with relatively high loads and assuming the average efficiency is not unreasonably low). Therefore, we use a simple policy for the second phase that divides the remaining processors equally among the active jobs (since Equipartitioning is a relatively safe and robust approach).

We consider three different approaches for computing f_i, thus introducing three new algorithms. In the first approach, we let f_i equal the knee, k_i, of the execution time - efficiency profile ($f_i = k_i$). This policy is therefore referred as the $W\&k_i$ algorithm. In the second approach we consider the effective efficiency of applications. Hence, f_i is equal to $\frac{\varepsilon_i P}{100}$. (Note that $\varepsilon_i \neq \beta_i$.) This policy is referred as the $W\&\varepsilon_i$ algorithm. The third approach uses a function of the effective efficiency, $F(\varepsilon_i)$. That is $f_i = F(\varepsilon_i)$. This policy is therefore called the $W\&F(\varepsilon_i)$ algorithm. The specific function used in this thesis is a piecewise linear function, the derivation of which is justified in Chapter 7.

In the following description of the W&E family of algorithms let r be the remaining number of processors (initially $r = p$).

- Upon the arrival of job J_i:

```
if the number of jobs in the active list < P {
    add J_i to the active list in sorted order (sorted by increasing W_i)
    for all jobs J_i in the active list {
        allocate p_i = min(r, f_i)
        r = r − p_i
    }
    if r > 0 {
        equally divide the r processors among the jobs in the active list
    }
}
else add J_i to the wait queue
```

- Upon the departure of job J_i:

```
If the wait queue is not empty {
    move the first job from the wait queue to the active list (in sorted order)
}
for all jobs J_i in the active list {
    allocate p_i = min(r, f_i)
    r = r − p_i
```

```
}
if r > 0 {
    equally divide the r processors among the jobs in the active list
}
```

4.5. Summary

The scheduling policies used in this thesis are divided into three broad families, Equi-Allocation, Generalized Allocation and the Work and Efficiency family. The Equi-Allocation family divides processing power equally among applications and uses no job characteristics (or minimal information such as the parallelism of the job) in making scheduling decisions. The Generalized Allocation family uses a control, α, in determining the actual processor allocation policy. These algorithms are used to explore a spectrum of policies that use only the work or only the efficiency to provide evidence that the use of these characteristics alone are unlikely to yield mean response times which are significantly lower than those obtained with Equipartition. However, the use of both work and efficiency in making allocation decisions can improve performance significantly over the Equipartition family. This is demonstrated by using the W&E family of algorithms which use both the work and the efficiency in making scheduling decisions. In this family of algorithms, jobs are sorted according to their remaining work to ensure that the smaller jobs are not delayed by the execution of larger jobs. The processors are then allocated according to the efficiency of the applications in the sorted list. In the following chapters we present results of the simulation studies conducted to compare policies that use application characteristics (Generalized Allocation and W&E) with those that use no application characteristics (Equipartition).

Chapter 5

Using Characteristics of Work Only

5.1. Introduction

In this chapter we present the results of simulation studies conducted in order to investigate if algorithms that make processor allocation decisions based only on the characteristics of a job's work can improve mean response time over the Equipartition policy (which uses no job characteristics). We also obtain bounds on the maximum improvement that can be obtained by considering jobs that execute with perfect efficiency. We then relax the assumption of perfect efficiency and conduct experiments to show that using only work in making allocation decisions can not improve performance across all of the workloads considered in this thesis. Policies that consider only work are shown to produce mean response times which are significantly higher than Equipartition unless the workloads have high average efficiency. However, the results of this chapter demonstrate the importance of service demand and motivate the work, in Chapter 7, on the development of algorithms that consider both work and efficiency.

5.2. Perfectly Efficient Workloads

We begin by considering jobs that execute with perfect efficiency. This is an admittedly unrealistic assumption but one that is useful in understanding and quantifying the importance of the service demand characteristic of a job (i.e., its work). Moreover, it is a means of obtaining a bound on the improvement that can be achieved by using job characteristics in making scheduling decisions over policies that use none.

The first experiment is designed to compare the performance of policies that use only work in making scheduling decisions under different system loads and different coefficients of variation in service demand. The generalized allocation policy (as described in Chapter 4) is used in comparing a spectrum of allocation algorithms that adaptively reallocate processing power to jobs according to their remaining work, W_i. The basis for comparison is the Equipartition policy (i.e., $\alpha = 0$). We use $X_i = W_i$ in order to determine processor allocations and examine a range of α values. The maximum improvement in mean response time that can be obtained by using work in making allocation decisions is also determined from this experiment.

The results of this experiment (shown in Figure 5.1) have been obtained using a mean work requirement of $\bar{W} = 1000$, and a coefficient of variation in service demand being equal to one ($C_W = 1$). As mentioned previously, all of the results presented in this thesis have been obtained by simulating a one-hundred node multiprocessor ($P = 100$). The graphs plot the mean response time as a function of the scheduling policy (α value) and shows results for loads corresponding to approximate observed loads (or processor utilizations) of 30, 50, 70, and 90 percent.

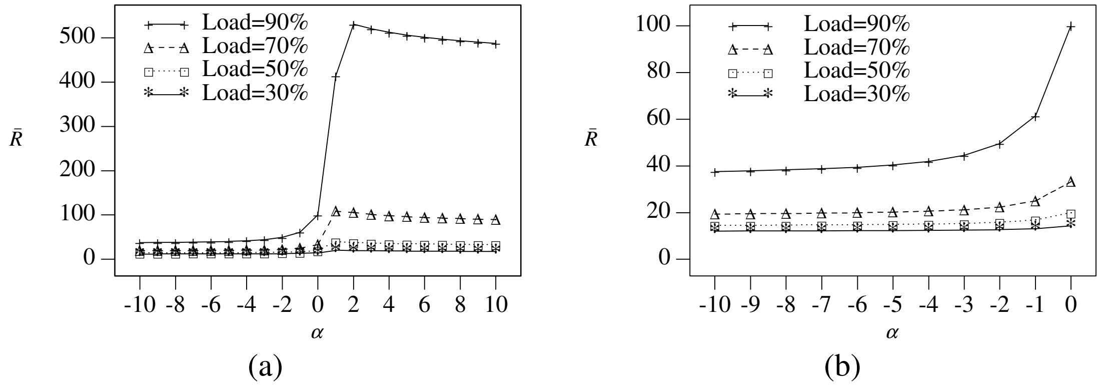

Figure 5.1: Mean response time versus α, $C_W = 1$, perfect efficiency

We first note that at high system loads the mean response times, $\bar{R}$, obtained with positive values of α, as seen in Figure 5.1(a), are significantly higher than those obtained for $\alpha \leq 0$. This is because large jobs (high W_i) are assigned a larger fraction of the processing power for positive values of α. Therefore, smaller jobs get a smaller share of the processors and have to wait behind the large jobs. Hence, the smaller jobs stay in the system for a longer period of time resulting in increased mean response times. The larger the α value the larger the portion of processors assigned to the largest job. With large enough α, jobs are essentially executed in a Most Remaining Work First (MRWF) fashion.

For negative values of α a larger share of the processors are allocated to the smaller jobs. Thus the scheduler ensures that smaller jobs finish execution as quickly as possible, resulting in reduced mean response times. Scheduling algorithms with large negative values of α closely approximate a Least Remaining Work First (LRWF) policy. Previous studies have shown that the LRWF policy is optimal when jobs execute with perfect efficiency and job preemption overhead is considered to be negligible [25] [1]. The graph in Figure 5.1(b) demonstrates clearly the difference between Equipartition and those policies corresponding to negative values of α. This demonstrates the importance of using W_i in making scheduling decisions.

We have also conducted experiments to investigate the difference between positive and negative values of α for C_W values of 5 and 30 (these results are not shown here). Workloads with higher C_W contain a higher degree of polarization in the service demand of jobs. The performance of policies corresponding to positive values of α degrades significantly with an increase in the coefficient of variation of service demand, C_W, resulting in an even larger difference in performance between positive and negative values of α. This is not surprising as more and more small jobs wait behind jobs with even larger service demands resulting in an increase in the response time of small jobs.

The graphs in Figure 5.1 also show that the difference in behavior across different scheduling policies increases with increases in the system load. This is because as the load increases the degree of multiprogramming increases. Under extremely low loads there will only be one job in the system at any point in time, in which case all of the allocation policies behave in the same fashion (i.e., the job is allocated all P processors). Henceforth, most of our experiments are conducted at high loads of 90%. We occasionally include experiments conducted with loads of 30% in order to demonstrate the differences in behavior under different loads.

Figure 5.1(a) shows that, with a load of 90 percent, $\bar{R}$ is maximized when $\alpha = 2$. This is because the mean response time of a MRWF policy (i.e., $\alpha = \infty$) can be increased by introducing processor sharing among the largest jobs (by moving α towards zero), thus increasing the mean response time.

The next experiment is designed to compare the performance of three different policies as a function of the coefficient of variation in service demand. The $\alpha = 0$, $\alpha = -10$ and LRWF ($\alpha = \infty$) policies are considered. The optimal LRWF is considered to demonstrate that although $\alpha = -10$ approximates LRWF, the approximation is quite close.

The results of this experiment are shown in Figure 5.2 and have been obtained using a mean work requirement of $\bar{W} = 1000$, with an observed load of approximately 90%. The vertical bars in the graphs represent 90 percent confidence intervals.

The graph shows that $\alpha = -10$ closely approximates the LRWF policy even at low values of C_W. The maximum difference between the two policies is 16 percent at $C_W = 0$ (all jobs have same service demand), an extremely unlikely C_W for realistic workloads. Moreover, for $C_W \geq 1$, the difference between the two policies becomes statistically insignificant. As mentioned in Chapter 2, experimental results by Chiang et al. [2] on a CM-5 machine, show that C_W ranges from 2.5 to about 6 with 40% of the measure being above 4.0. Hence, we can safely conclude that the LRWF policy ($\alpha = \infty$) can be closely approximated with $\alpha = -10$ in realistic workloads. Therefore, the policy corresponding to $\alpha = -10$ will yield the maximum improvement in mean response time over Equipartition for perfectly efficient workloads.

The difference between Equipartition ($\alpha = 0$) and $\alpha = -10$ increases with the coefficient of variation in service demand. However, the difference between these two policies does not

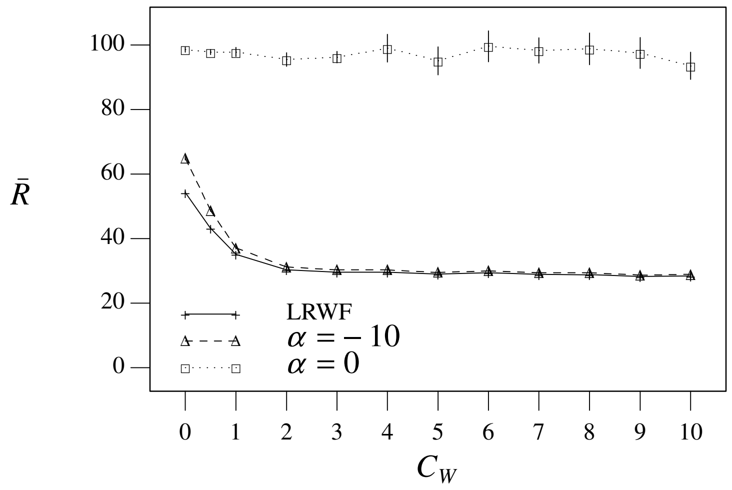

Figure 5.2: Mean response time versus C_W for policies LRWF, $\alpha = -10$ and $\alpha = 0$

increase significantly beyond $C_W = 5$. Experiments were also conducted for higher coefficients of variation in service demand ($C_W = 20$ and $C_W = 30$) but the results (not shown in Figure 5.2) are similar to those obtained for $C_W = 10$. We also note that Equipartition is insensitive to the coefficient of variation in the mean service demand of the workload since no statistically significant variations in the mean response times are observed with changes in C_W.

Table 5.1 provides a rough quantification of the performance gains that can be achieved by knowing and utilizing W_i for loads of 30, 50, 70 and 90 percent when jobs are perfectly efficient. This table contains one column for the mean response times and 90% confidence intervals for each of $\alpha = 0$ and $\alpha = -10$ as well as one column showing the percentage improvement (%Impr) that is obtained by using $\alpha = -10$ instead of $\alpha = 0$.

This rough quantification is of interest because:

1) It provides an approximate bound on the reduction in mean response time that can be obtained by using W_i. This is because workloads with less efficient jobs will receive less benefit from allocating all processors to one job at a time (assuming a non-decreasing execution rate function).

2) Previous comparisons between LRWF and Equipartition policies by Brecht [1] show that the reduction in mean response time obtained when all jobs arrive simultaneously can be as large as 50% (i.e., Equipartition is 2-competitive under the model and assumptions used). However, when jobs do not arrive simultaneously the difference can grow with the number of jobs (i.e., Equipartition is not a competitive approach). Since Brecht's result showing that Equipartition is not competitive assumes that arrival times and job sizes are designed to demonstrate maximum differences between the two policies, our comparison provides more reasonable estimates on the performance improvements that could be expected from using

Load	C_W	$\bar{R}$ ($\alpha = 0$)	$\bar{R}$ ($\alpha = -10$)	%Impr
90%	1	100.1 +/- 1.7	36.5 +/- 0.4	64
	5	100.4 +/- 5.2	29.8 +/- 0.7	70
	30	98.1 +/- 5.1	28.2 +/- 2.5	71
70%	1	33.3 +/- 0.1	19.4 +/- 0.0	41
	5	33.3 +/- 0.5	17.9 +/- 0.1	46
	30	32.1 +/- 2.0	17.5 +/- 0.7	45
50%	1	20.0 +/- 0.0	14.6 +/- 0.0	27
	5	19.9 +/- 0.1	14.1 +/- 0.1	29
	30	19.7 +/- 0.7	13.9 +/- 0.4	29
30%	1	14.3 +/- 0.0	12.1 +/- 0.0	15
	5	14.3 +/- 0.1	12.0 +/- 0.0	16
	30	14.1 +/- 0.3	11.9 +/- 0.2	15

Table 5.1: Comparing $\alpha = -10$ and $\alpha = 0$ for different C_W, load = 90%

W_i.

3) It shows that greater reductions in mean response times can be observed at higher system loads when using knowledge of the service demand in making scheduling decisions. Moreover, the improvements are likely to increase with increases in C_W. However, we found that the improvements for $C_W = 30$ are nearly the same as for $C_W = 5$, indicating that the improvements do not increase linearly with increases in coefficient of variation in service demand.

5.3. Estimates of Service Demand

In Section 5.2 a job's remaining work is used in making scheduling decisions. However, precise information about a job's remaining work may not be available to the scheduler. Instead, estimates of the remaining work may be obtained at run-time by monitoring the current execution of the job. We perform experiments using two methods for estimating the service demand of a parallel job proposed by Brecht [1]. The first method uses the total processing power received by the job to estimate its service demand. The second method estimates work by using the total time spent in the system since the arrival of the job. These methods are based on the principle that a job that has been running in the system for a long period of time is expected to run for an even longer period of time. Both of these methods are investigated to determine which provides the better estimates (of service demand). Our objective is to demonstrate that the use of these estimates can result in reductions in mean response time over the Equipartition policy.

In the first method, the system keeps track of the CPU service accumulated by a job. The CPU service accumulated by J_i to the current point in time is denoted by Acc_i. The generalized allocation policy is used to determine processor allocations with $X_i = Acc_i$. In the second method the system keeps track of the time spent by the job in the system since its arrival. As with the previous estimate, the generalized allocation policy is used to determine processor allocations with $X_i = Time_i$, where $Time_i$ denotes the elapsed time in the system for job J_i. The results for both experiments are shown in Figure 5.3. The graphs for both of the experiments have been obtained using a mean work requirement of $\bar{W} = 1000$, a load of 90 percent and C_W values of 1, 5 and 30. Note that all jobs in these experiments execute with perfect efficiency.

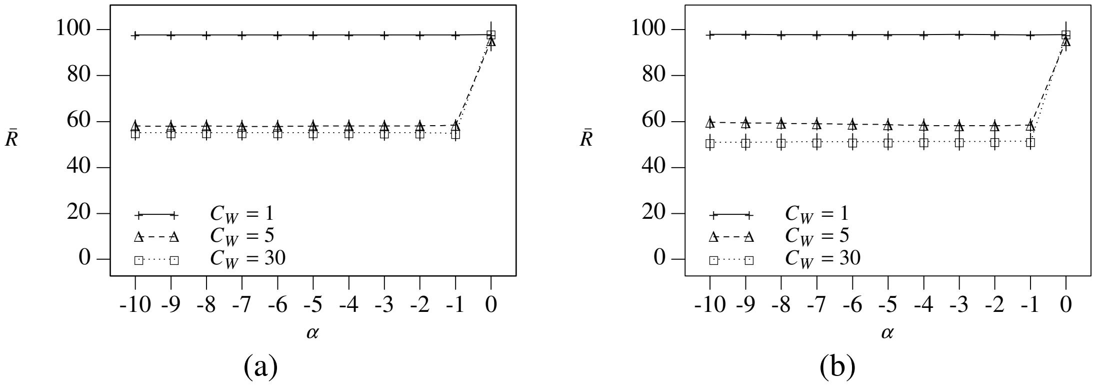

Figure 5.3: Mean response time versus α, $X_i = Acc_i$ (a) and $X_i = Time_i$ (b)

The graphs show that using estimates of the service demand of an application results in a significant reduction in mean response time over Equipartition for C_W values of 5 and 30. However, no improvements are obtained at $C_W = 1$. This is because the scheduler uses estimates rather than exact knowledge of the service demand of an application and therefore, cannot distinguish between jobs with a workload having a low value of C_W. We also note that for $C_W =$ 5 and 30 the response times obtained for $\alpha = -1$ are almost equal to those obtained with $\alpha = -10$. Although it may not be clearly visible in the graphs in Figures 5.3(a) and 5.3(b), the results obtained using time in system are statistically similar to those obtained using the accumulated CPU service. Therefore, it appears as though each of these techniques provides similar estimates of the work to be executed by an application.

5.4. Inefficient Workloads

In this section we relax the requirement that all jobs execute with perfect efficiency and again compare Equipartition with the policies that use work in making processor allocations. The distribution of the effective efficiency of applications in a workload executing on a real multiprocessor is not known. Hence, we first assume that the effective efficiency of applications is distributed uniformly between $\varepsilon_{\min}$ and $\varepsilon_{\max}$. Then we investigate the sensitivity of the results to the distribution of efficiency values by using a Beta distribution for generating the effective efficiency. Among a number of combinations of ε_{low} and ε_{high} parameters for the Uniform distribution, we investigate the following ranges in our experiments (C_ε denotes the coefficient in variation in effective efficiency):

1 - 50% - workload with low efficiency and medium C_ε ($\bar{\varepsilon} = 25.5$, $C_\varepsilon = 0.566$)
1 - 99% - workload with medium efficiency and high C_ε ($\bar{\varepsilon} = 50.0$, $C_\varepsilon = 0.571$)
50 - 90% - workload with high efficiency and medium C_ε ($\bar{\varepsilon} = 74.5$, $C_\varepsilon = 0.194$)
75 - 99% - workload with very high efficiency and low C_ε ($\bar{\varepsilon} = 87.0$, $C_\varepsilon = 0.082$)

By using the above ranges we hope to examine the behavior of scheduling policies as the nature of the workload changes with respect to efficiency.

The graphs shown in Figure 5.4 have been obtained using a mean work requirement of $\bar{W} = 1000$, a load of 90 percent and $C_W = 1$, 5 and 30. Note that although the observed load is approximately 90 percent in Figures 5.4 (a), (b), (c) and (d), the arrival rates used in each of the experiments represented by the graphs are different. Since the workloads are different for the experiments conducted to generate each graph, direct comparisons among these graphs are not possible. The parameters for the uniform distribution used in generating the effective efficiency are shown in the bottom right corner of each graph. The vertical bars in each graph represent 90 percent confidence intervals.

The graphs in Figure 5.4(a) and Figure 5.4(b) show that for workloads with relatively low average efficiency (i.e., $\bar{\varepsilon} = 50\%$ for an ε_i range of 1-99% and $\bar{\varepsilon} = 25.5\%$ for an ε_i range of 1-50%) using only W_i to make scheduling decisions leads to very large increases in mean response times (versus equipartitioning processors, $\alpha = 0$). This is because these polices ($-10 \le \alpha \le -1$) are allocating larger portions of the processors to small jobs even though they may not be capable of utilizing them effectively. However, the graphs in Figure 5.4(d) show that for workloads with high average efficiency (75-99% and $\bar{\varepsilon} = 87\%$), using only W_i to make scheduling decisions can reduce mean response times (versus using no job characteristics and equipartitioning processors, $\alpha = 0$). This is because, under this workload, the average efficiency of jobs is high enough that the benefits from allocating large portions of the processors to these jobs (reduced response time) outweigh the costs (under-utilized processors). For the workload of $\varepsilon_{low} = 50\%$ and $\varepsilon_{high} = 99\%$, shown in Figure 5.4(c), the cost of allocating a larger share of the

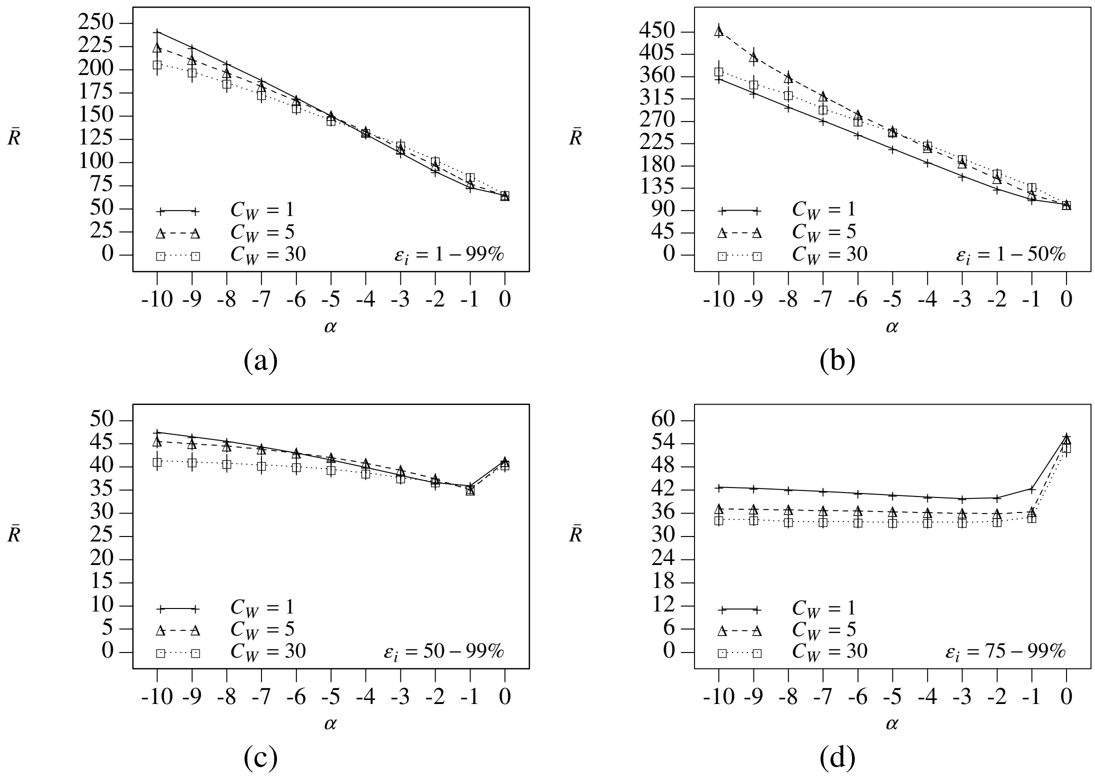

Figure 5.4: Mean response time versus α, imperfect efficiency (D_ε=Uniform), load≈90%

processors is increased due to a decrease in the average efficiency of the workload. Under this workload the benefits of allocating larger portions of the processors is close too matching the costs of such an allocation.

The graphs in Figure 5.4(c) also show that $\alpha = -1$ reduces mean response time over Equipartition (and all other policies) although the improvement is relatively small. This shows that under this workload it is desirable to allocate more processors to smaller jobs but only slightly less would be allocated using an Equipartition policy. The value used to select the policy being used, α, in some sense controls the degree of processor sharing among the jobs in the system. As can be seen from the graphs for the range $\varepsilon_i = 50 - 99\%$, in Figure 5.4(c), reduced processor sharing for this workload is desirable since $\alpha = -1$ results in reduced mean response times as compared to those for $\alpha < -1$.

We now investigate the sensitivity of the results shown in Figure 5.4 to the distribution used for generating effective efficiency values. The parameters of the Beta distribution are chosen so that $\bar{\varepsilon}$ and C_ε correspond to those values used for the experiments depicted in the graphs in Figure 5.4 (i.e., the $\bar{\varepsilon}$ and C_ε for the Uniform distribution). The parameters used for the Uniform and Beta distributions and their means and coefficients of variation are shown in Table 3.2.

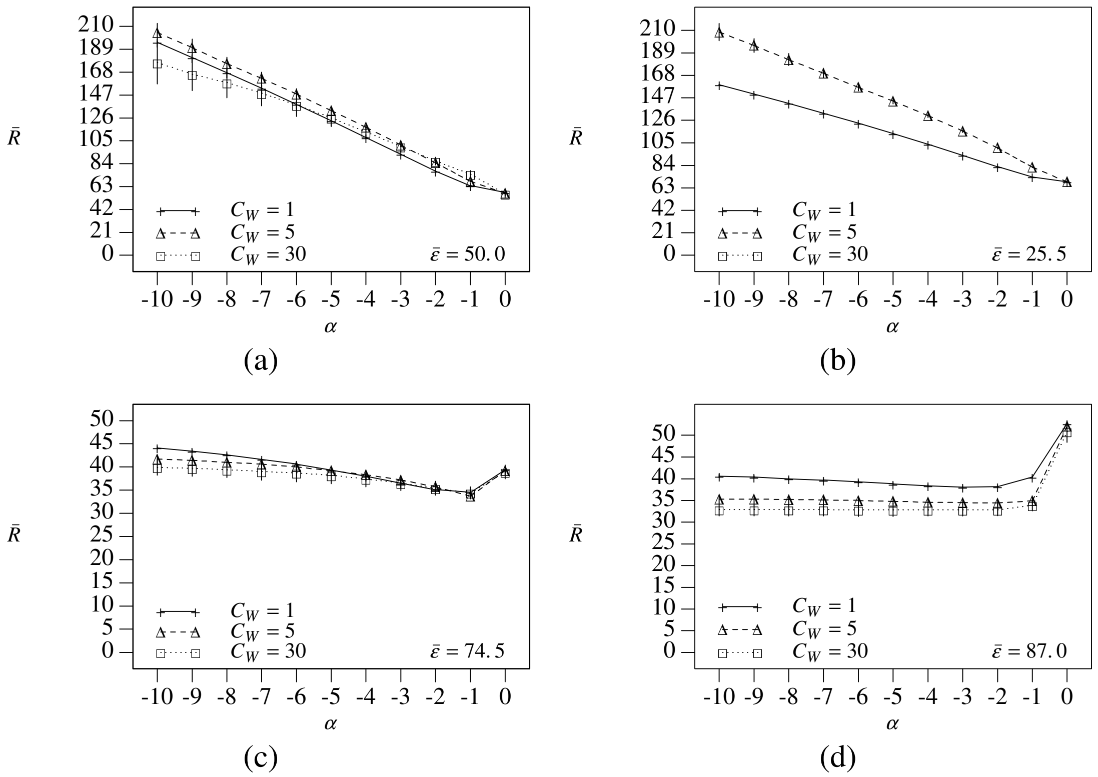

Figure 5.5: Mean response time versus α, imperfect efficiency (D_ε = Beta), load ≈ 90%

The graphs in Figure 5.5 indicate that the results are qualitatively similar to the ones shown in Figure 5.4. This is true of all of the graphs shown in Figure 5.5. Hence the results obtained for using only work in making scheduling decisions are relatively insensitive to the two distribution used to generate the effective efficiency values.

5.5. Summary

In this chapter we present results obtained by simulating a family of scheduling policies that makes allocation decisions based on knowledge of the service requirement of each job. If jobs execute with perfect efficiency, reductions in mean response time of as much as 70% are obtained when compared with the Equipartition policy. If the scheduler has precise knowledge about the service requirement of each job, improvements are maximized under high loads and high coefficients of variation in the service demand. If precise information is not available, estimates such as the accumulated CPU time or the time spent in the system can be used in making scheduling decisions. The results show that both estimates perform equally well with the maximum reduction in mean response time (when compared with Equipartition) being approximately 45%.

For inefficient workloads processor sharing becomes more desirable as the average efficiency of the jobs in the workload decreases. Specifically, at low average efficiency, Equipartition performs substantially better than policies corresponding to $\alpha < 0$ (which use knowledge of the service demand of an application to allocate a larger fraction of the processing power to a single application). For workloads with high average efficiency the benefits of allocating large portions of the processors to the smaller jobs (reduced response time) outweigh the costs (under-utilized processors). Hence, under these circumstances allocation policies which use work in making allocation decisions perform better than the Equipartition policy. These results are also shown to be insensitive to the two distributions considered for generating effective efficiency values.

The results from this chapter shows that allocation policies which only use work in making scheduling decisions can not improve mean response time over Equipartition for all potential workloads. However, reductions in mean response time are obtained for workloads with high average efficiency when compared with Equipartition. The size of the reductions obtained depend on the average efficiency of the workload, the coefficient of variation in service demand and the system load. This demonstrates the importance of knowing or having estimates of work and utilizing them in making allocation decisions.

Chapter 6

Using Characteristics of Efficiency

6.1. Introduction

In the previous chapter, knowledge of each job's remaining work is used by the scheduler when determining processor allocations. Our simulation results show that using the remaining service demand in scheduling improves mean response time over Equipartition only when workloads have high average efficiency. Under workloads with low average efficiency the performance of these policies degrades since jobs are allocated processors which they are not able to use effectively. Results from the previous chapter indicate that the efficiency is another important workload characteristic which determines the execution time of parallel applications and also the performance of scheduling policies. However, recent studies in static scheduling report that characteristics of execution rate were not successful in improving mean response times [2]. This motivates the investigation of policies which only use information about a job's efficiency.

6.2. Using Efficiency

In this chapter we use the generalized allocation policies in determining the advantages of using only the efficiency characteristic of a job in making scheduling decisions. The efficiency parameter, β_i, is used in the generalized allocation policy to determine processor allocations. We set $X_i = \beta_i$ and use:

$$p_i \;=\; \frac{P\ {\beta_i}^{\alpha}}{\sum\limits_{j=1}^{n} {\beta_j}^{\alpha}}\ .$$

As was done in the previous chapter, various policies are considered by using different values of α to allocate processing power according to the efficiency with which a job executes. Positive values of α will allocate more processors to jobs with higher efficiency while negative α values allocate more processors to jobs with low efficiency (an obviously bad approach).

The parameter β_i seems to be a good choice for making allocation decisions since the knee, k_i, of application J_i corresponds to β_i for the job model considered in this thesis. As mentioned in Chapter 2, the knee, k_i, identifies an optimal allocation point in the execution time - efficiency profile where the ratio of the application's efficiency to its execution time, $E_i(p_i)/T_i(p_i)$, is maximized. Therefore, our study also examines the effects of using the knee in making allocation decisions. The correspondence between β_i and the knee, k_i, of application J_i can be shown in the following manner. Given the function that describes the execution rate of a parallel job, J_i, we can compute the knee as follows:

$$E_i(p_i) = \frac{S_i(p_i)}{p_i} = \frac{(1+\beta_i)}{\beta_i + p_i} \quad \text{and} \quad T_i(p_i) = \frac{W_i}{S_i(p_i)} = \frac{W_i\ (\beta_i + p_i)}{(1+\beta_i)\ p_i} .$$

$$\text{Therefore,} \quad \frac{E_i(p_i)}{T_i(p_i)} = \frac{(1+\beta_i)^2\ p_i}{(\beta_i + p_i)^2\ W_i} .$$

The value of p_i which maximizes $\frac{E_i(p_i)}{T_i(p_i)}$ is the knee, k_i. Hence, solving the equation $\frac{\partial \frac{E_i(p_i)}{T_i(p_i)}}{\partial\ p_i} = 0$ in terms of p_i gives us k_i, which is equal to β_i in this case. Therefore, for the execution rate function used in this thesis, the knee of the execution time - efficiency profile is $k_i = \beta_i$. (Chiang, et al. [2] also point out that the knee, k_i, for job J_i can be shown to be equal to β_i.)

6.3. Experimental Results

The first experiment in this chapter is designed to compare the performance of policies corresponding to $\alpha < 0$ and $\alpha > 0$ with Equipartition ($\alpha = 0$) under low loads and different coefficients of variation in service demand. This experiment has been conducted with a load of approximately 30% to demonstrate that even under light loads mean response times increase significantly for policies corresponding to $\alpha < 0$ (which is not surprising since these policies allocate more processors to jobs with low efficiency). The light load is also used to demonstrate the general v-shape of the response time curves since at higher loads the differences in mean response times for the various algorithms changes drastically. The graph in Figure 6.1 plots the mean response time against different α values (representing different scheduling policies) and shows results for $C_W = 1$ and $C_W = 5$ (results for $C_W = 30$ yield very high mean response times which if included in the same graph hides the distinction between $C_W = 1$ and $C_W = 5$). The efficiency range used for each experiment (1-50%, 1-99%, 50-99% or 75-99%) is shown on the bottom left corner of each graph.

The results show that for $\alpha < 0$ mean response times increase significantly compared with Equipartition, especially for workloads with low average efficiency (Figures 6.1(a), 1-50% and 6.1(b), 1-99%). Performance of these policies ($\alpha < 0$) degrades significantly at higher coefficients of variation in service demand ($C_W = 5$). The figures also show that policies

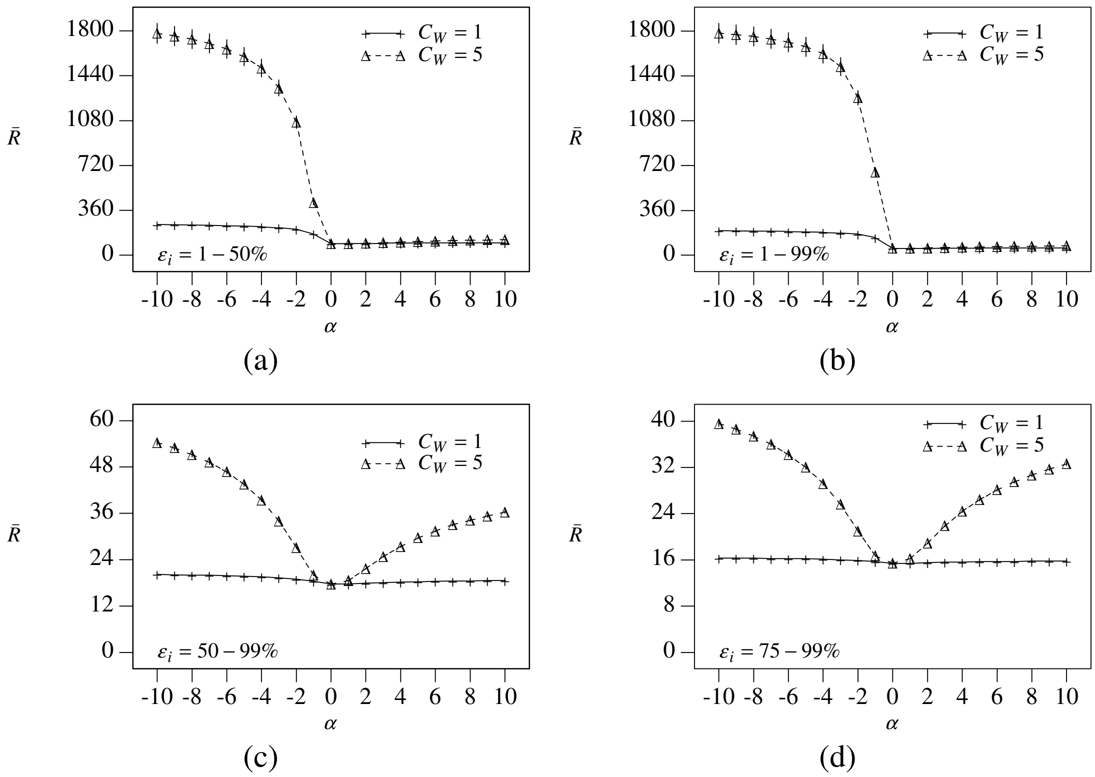

Figure 6.1: Mean response time versus α, imperfect efficiency, load ≈ 30%

corresponding to $\alpha > 0$ result in mean response times which are significantly lower than $\alpha < 0$. This is not surprising since these policies ($\alpha > 0$) allocate a larger fraction of the processing power to jobs with higher efficiency. Allocating more processors to jobs with low efficiency ($\alpha < 0$) is obviously a bad approach and is therefore not considered further.

The policies corresponding to $\alpha > 0$ are unable to reduce mean response times when compared with Equipartition, although the differences between these policies are smaller than the differences between Equipartition and $\alpha < 0$ policies. This is because the scheduler may allocate a large fraction of the processing power to large jobs (since allocations are based only on the efficiency of jobs) and therefore small jobs may have to wait behind the larger jobs resulting in increased mean response times. This can also be seen by the large degradation in performance that occurs when the coefficient of variation is increased from $C_W = 1$ to $C_W = 5$.

A second experiment is designed to more closely examine the differences between Equipartition and policies which allocate more processing power to jobs with higher efficiency (policies corresponding to $\alpha > 0$). This experiment is performed using a higher observed load (≈90%). Note that the arrival rates corresponding to processor utilizations of 90% were determined using the policy $\alpha = 10$ and then used for all other policies (including Equipartition).

The graphs in Figure 6.2 also demonstrate that significantly increasing the number of processors allocated to efficient jobs does not improve mean response time. This is because jobs that might potentially take a short amount of time to execute can become stuck behind jobs that take a long time to execute since processor allocations are not based on the remaining amount of work. The mean response times of policies corresponding to $\alpha > 1$ increase with the coefficient of variation in service demand. This is because, W_i is polarized for higher coefficients of variation and therefore more small jobs get stuck behind large jobs resulting in increased mean response times.

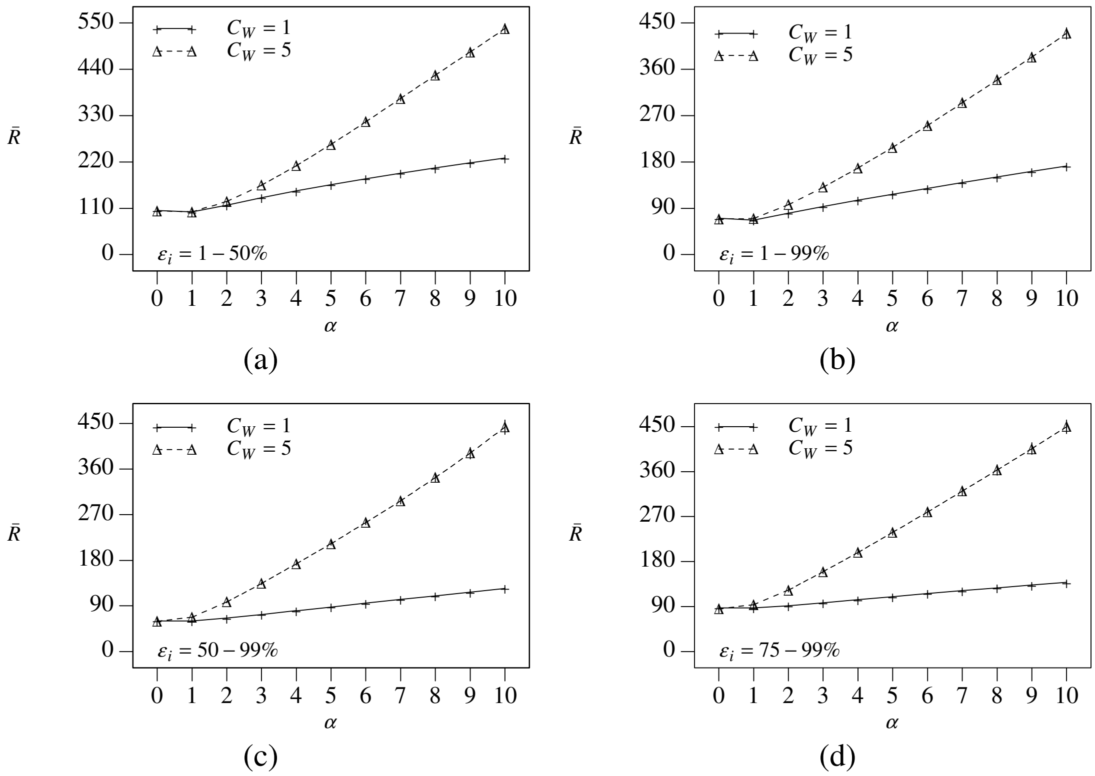

Figure 6.2: Mean response time versus α, imperfect efficiency, load $\approx$ 90%

The graphs in Figure 6.2 show that the mean response time obtained with $\alpha = 1$ is relatively close to that obtained with $\alpha = 0$ for $C_W = 1$ and $C_W = 5$. The proximity of the results obtained with $\alpha = 0$ and $\alpha = 1$ motivates a closer investigation of these algorithms along with another point on the spectrum of allocation policies (we use $\alpha = 0.5$ since it lies between $\alpha = 0$ and $\alpha = 1$).

The results of the experiments conducted to compare the performances of the $\alpha = 1.0$, $\alpha = 0.5$ and Equipartition ($\alpha = 0$) policies are shown in Figure 6.3. These experiments were conducted using arrival rates that produced loads of approximately 30% and 90% respectively. (The arrival rate was determined using $\alpha = 1.0$ and was then used for the other policies.) Each bar shown in these graphs has been normalized with respect to the mean response time obtained using Equipartition under the specified workload. Displayed below the x-axis are the workload parameters C_W, the effective efficiency range, $\varepsilon_{\min} - \varepsilon_{\max}$, and the α value denoting the scheduling policy.

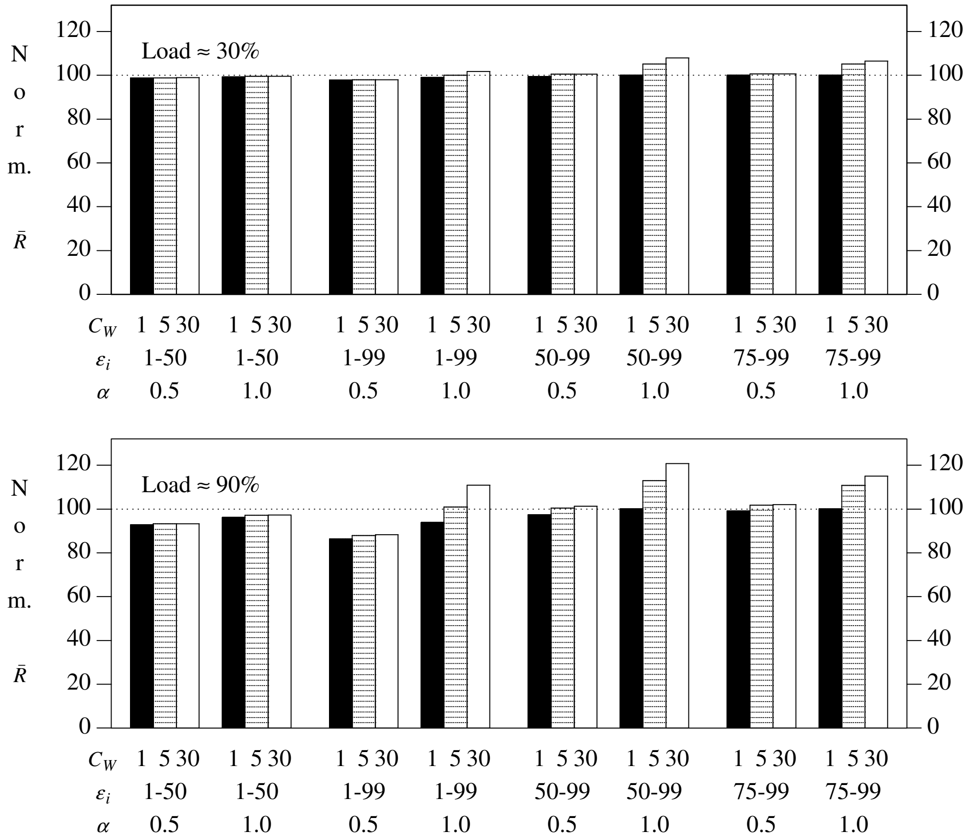

Figure 6.3: Comparing α = 0.5 and α = 1.0 against Equipartition, load ≈ 30 and 90%

As expected under low loads (≈30%), the performance of Equipartition is comparable to $\alpha = 0.5$ and $\alpha = 1.0$ for nearly all ε_i ranges. However, when the degree of multiprogramming increases (load ≈ 90%), $\alpha = 0.5$ reduces mean response times over Equipartition for workloads with low average efficiency (1-50% and 1-99%) while for workloads with high average efficiency (50-99% and 75-99%), Equipartition and $\alpha = 0.5$ produce mean response times that are statistically similar. (Confidence intervals were computed but are not shown). We note that the improvement in mean response time over Equipartition is larger for the range of ε_i = 1-99% than for ε_i = 1-50%. We expect that this is because the range 1-99% has a higher coefficient of variation in effective efficiency, ε_i, than the 1-50% range. The policy corresponding to $\alpha = 1$ reduces mean response times when compared with Equipartition only for workloads with low average efficiency and a low coefficient of variation in service demand (C_W = 1). We believe that with $\alpha = 1$ too many processors are allocated to jobs that are not capable of utilizing them effectively. Because of the nature of how β_i characterizes the efficiency of each job we expect that the low mean response times obtained for $\alpha = 0.5$ are more indicative of how to allocate processors using β_i than how to allocate processors using other characteristics of a job's efficiency. (For example, we expect that if ε_i was used $\alpha = 1$ might perform better than $\alpha = 0.5$).

The performance of the policy corresponding to $\alpha = 0.5$, which allocates processors proportional to the square root of the efficiency, is at least equal to that of Equipartition and even better for workloads with low average efficiency. (Note that the policy $\alpha = 1$ allocates processors proportional to the efficiency of the jobs.) In general, it is possible to improve mean response time over Equipartition by using only the knowledge of efficiency in making allocation decisions although the improvements are relatively small (≈14%) and are only obtained for workloads with low average efficiency. This also indicates that using the knee in making scheduling decisions might not result in good performance especially for workloads with high average efficiency (since the efficiency characteristic, β_i, is equal to k_i).

6.4. Summary

In this chapter we present results for scheduling policies which use only a characteristic of a job's efficiency in making allocation decisions. The efficiency characteristic, β_i, is used in the generalized allocation policy. Using β_i, we also examine the effects of using the knee in making processor allocations since β_i corresponds to the knee, k_i, for the job model considered in this thesis. The results show that the mean response times for policies corresponding to $\alpha > 1$ and $\alpha \leq -1$ are significantly higher than for Equipartition. This is because the scheduler does not consider the remaining work while making scheduling decisions and may allocate too large a fraction of the processing power to jobs that may execute for a long period of time. Thus, jobs which may have potentially finished execution quickly are forced to wait, resulting in increased mean response times. This shows that work is an important characteristic which should be taken into consideration while making allocation decisions.

The policy $\alpha = 0.5$ reduces or equals the mean response times of Equipartition, with improvements occurring for workloads with low average efficiency. The improvements also seem to be reduced for lower coefficient of variations in effective efficiency. The results show that using only efficiency in making processor allocations will reduce or equal the mean response times of Equipartition across the workloads considered. However, the improvements are obtained for only a few workloads and are quite small (≈ 14%). These results and the results from Chapter 5 motivate the need for new policies that consider both work and efficiency in making processor allocation decisions.

Chapter 7

Using Characteristics of Work and Efficiency

7.1. Introduction

As shown in Chapter 5, for workloads with high average efficiency the use of the application service demand in making scheduling decisions leads to significant improvements in mean response time over the Equipartition policy. However, the performance of these policies (based on using information about a job's service demand) degrades as the average efficiency of the workload decreases. The results from Chapter 6 indicate that using only efficiency in making processor allocations results in mean response times which are equal to or yield only small reductions (up to 14 percent) over Equipartition. This is because these policies fail to consider the amount of remaining work each job has to execute. The results obtained in Chapter 5 and Chapter 6 strongly motivate the need for scheduling policies which use characteristics of both the work and the efficiency in making processor allocation decisions.

In this chapter we present the results of experiments conducted with new policies we have developed that are designed to take into account both the efficiency of a job and its work. The policies comprise what we call the $W\&E$ family of algorithms. The $W\&E$ family has evolved from our observations that there are two main characteristics that contribute to the response time of a parallel application, the remaining work, W_i, and the efficiency with which that work can be executed, β_i, and that policies based on using either one of these characteristics in isolation is unable to significantly reduce mean response times over the range of workloads considered in Chapters 5 and 6.

After performing a number of experiments to evaluate the performance of these new policies we also consider a new job model to examine the sensitivity of the results to the job model. The performance of the $W\&E$ policies are evaluated using this new job model and some general conclusions are drawn about the advantages of using job characteristics in scheduling by unifying the results obtained from using both job models.

7.2. Using $\mathbf{W\&k_i}$

As described in Chapter 4, the $W\&E$ family of algorithms operates by maintaining a sorted list of jobs in the system (sorted by increasing W_i). Jobs are activated in this order and are assigned f_i processors. In this section, we consider an allocation policy which assigns processors according to the knee of the execution time - efficiency profile (i.e., $f_i = k_i = \beta_i$). Note that β_i can be greater than P, in which case $f_i = P$.

The results of the experiments performed with the $W\&k_i$ policy are shown in Figure 7.1. These experiments are performed with arrival rates corresponding to loads of approximately 30% and 90%. The arrival rates corresponding to processor utilizations of 30% and 90% were determined using the $W\&k_i$ policy and then used for the Equipartition policy. The bars in Figure 7.1 have been normalized with respect to the mean response time obtained using the Equipartition policy under the specified workload (the dotted line in the figure represents the performance of Equipartition). The workload parameters, C_W and the effective efficiency range, $\varepsilon_{\min} - \varepsilon_{\max}$, are shown below the x-axis for each of the experiments. Note that the Uniform distribution is used to generate effective efficiency parameters for the experiments in this Chapter.

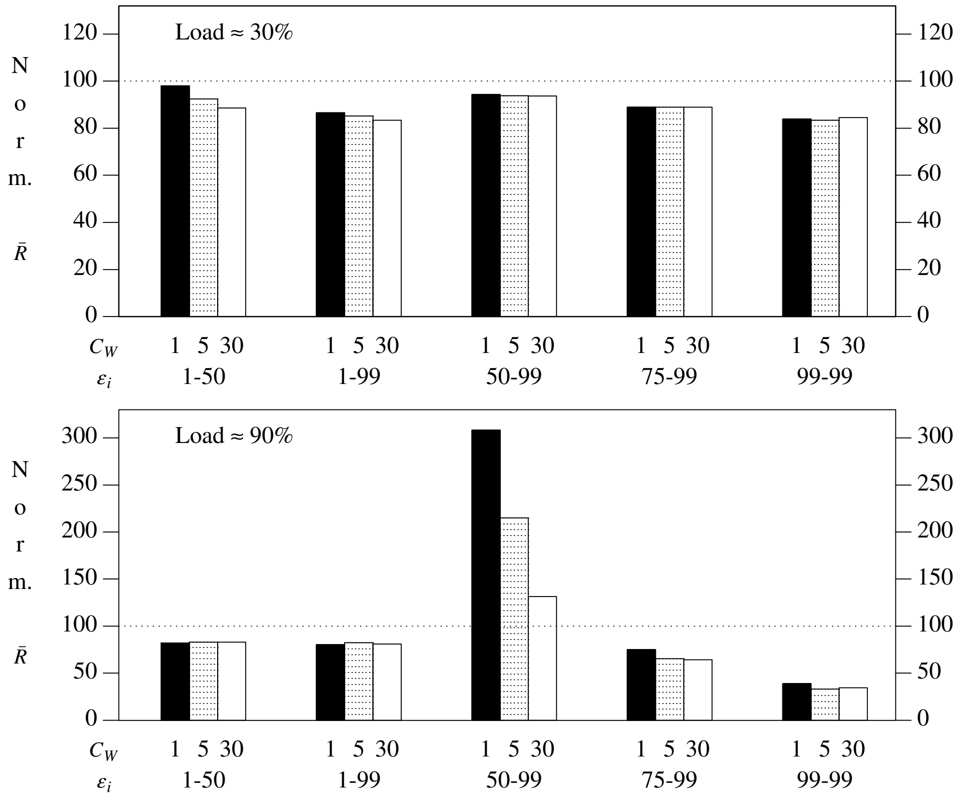

Figure 7.1: Comparing Equipartition and $W\&k_i$

As seen in Figure 7.1, for a load of 30%, the $W\&k_i$ policy improves mean response time over Equipartition across all ranges of ε_i. However, the reductions obtained in mean response time are relatively small (less than 20%). For a load of 90% the improvements in mean response time over Equipartition increase for ε_i ranges of 1-50%, 1-99%, 75-99% and 99-99%. The range of 99-99% has been considered to indicate the upper bound on improvements that can be obtained using the $W\&k_i$ policy since jobs execute with nearly perfect efficiency and the policy behaves nearly optimally in this case. Note that we do not consider the range of ε_i = 100-100% to avoid the problem of dealing with infinity (since at ε_i = 100%, $\beta_i = \infty$).

For the range of ε_i = 50-99%, as the degree of multiprogramming increases (i.e., for the high load of 90%), the $W\&k_i$ policy produces mean response times which are considerably higher than Equipartition. This is because the $W\&k_i$ policy allocates all P processors to any job J_i for which $\varepsilon_i \geq 50.5\%$ (because $k_i \geq P$). This is illustrated in Figure 7.2, which shows processor allocation, f_i, as a function of the effective efficiency (P = 100).

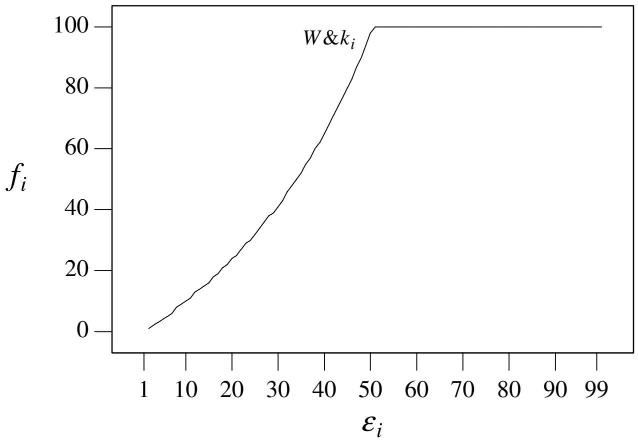

Figure 7.2: Processor allocation as a function of ε_i in the $W\&k_i$ policy

This results in each job in turn being allocated P processors (or very nearly P, since some jobs may have $50 \leq \varepsilon_i \leq 50.5$) which it may not be able to utilize effectively while the other jobs in the system wait. The result is increased response times. For the range of $\varepsilon_i = 50 - 99\%$, the $W\&k_i$ and the LRWF policies are very nearly equivalent since the number of processors allocated to the smallest job is P (or very close to P). Workloads with higher C_W contain more small jobs which are executed essentially in a Least Remaining Work First fashion. For the range ε_i = 50-99%, better performance is observed for workloads with high C_W than that with low values of C_W. This is due to the large number of small jobs in the workloads with high C_W. Note that although the processor allocation for all jobs is P for the 75-99% range, performance of the $W\&k_i$ policy shows a considerable improvement in mean response time over the Equipartition policy. This is because, under this workload, the average efficiency of jobs is high enough that the benefits of allocating all P processors to one job (reduced response time) outweigh the costs of the processors being under-utilized. This was also observed in the experiments conducted in Section 5.4.

As seen from the results, the $W\&k_i$ policy improves mean response times over Equipartition for workloads with either low or very high average efficiency. However, for the workload with ε_i = 50-99%, Equipartition has a mean response time which is much lower than the $W\&k_i$ policy (as much as three times lower). Obviously, for such workloads, the knee is not a suitable point for processor allocation. This may be due to the execution rate function used in this thesis. However, no matter what execution rate function is used there are likely to be some jobs whose communication and synchronization overhead is relatively small and therefore execute with relatively high efficiency. In these cases their knee will be greater than the number of processors in the system. For such jobs the knee must be mapped into a number of processors $\leq P$. Therefore, we require a function that more evenly maps the range, β_i, to the domain, f_i.

7.3. Using W&ε_i

In this section, we consider a very simple and obvious mapping, $f_i = \dfrac{\varepsilon_i P}{100}$. The $W\&\varepsilon_i$ policy uses this mapping in processor allocation. The use of this mapping is now explained.

Let $x_i = min(\beta_i, P)$ and then consider the value $\chi = E(x_i)/T(x_i)$. Note that x_i is equal to the knee, k_i, when $E(x_i)/T(x_i)$ is maximized. Since we do not want to allocate all P processors to the jobs with $\varepsilon_i \geq 50.5\%$ we choose a new point x'_i with a value $\chi' = E(x'_i)/T(x'_i)$. This new point x'_i is chosen such that χ' is sufficiently close to but less than χ. That is, the new point x'_i will be guaranteed to be relatively close to the knee. For the $W\&\varepsilon_i$ policy we use $x'_i = \dfrac{\varepsilon_i P}{100}$. This gives values of χ' that are less than χ by at most 15.4% and on average 7.4%. We also found, by conducting a series of experiments, that choosing χ' to be closer to χ did not result in substantial reductions in mean response time. (In fact we will see later that it is better to choose χ' farther from χ.)

Figure 7.3 shows the processor allocations, f_i, for the $W\&\varepsilon_i$ policy as a function of the effective efficiency, ε_i. The processor allocations for the $W\&k_i$ policy are also included in Figure 7.3 for ease of comparison between the two policies.

The advantage of the $W\&\varepsilon_i$ policy is that only jobs with high efficiency will be allocated a large number of processors. Also, as the average efficiency of the workload increases this algorithm asymptotically behaves optimally (i.e., if all jobs execute with perfect efficiency they would be executed in a LRWF fashion). Note that the maximum difference in allocation between the two policies is for $\varepsilon_i \approx 50\%$ and is a difference of nearly 50 processors.

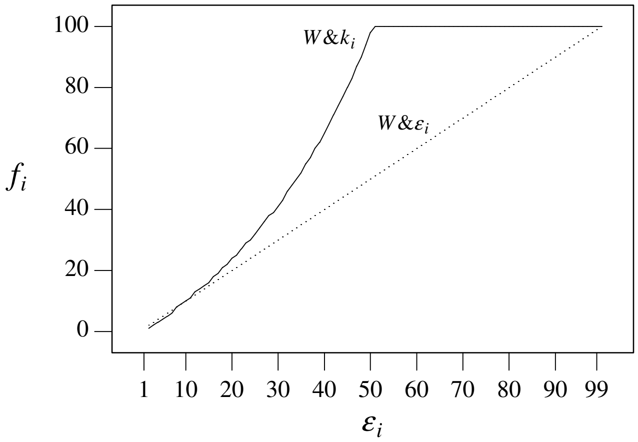

Figure 7.3: Processor allocation as a function of ε_i ($W\&k_i$ and $W\&\varepsilon_i$ policies)

We have conducted a series of experiments to compare $W\&\varepsilon_i$ with Equipartition. These experiments were conducted using arrival rates that produced observed loads of approximately 30 and 90% respectively. The arrival rate for each range of $\varepsilon_{\min}$ - $\varepsilon_{\max}$ was first determined using $W\&\varepsilon_i$ and was then used for Equipartition (resulting utilizations differ for different scheduling policies). The results of these experiments are shown in Figure 7.4. Again, each bar shown in the graphs of Figure 7.4 has been normalized with respect to the mean response time obtained using Equipartition under the specified workload and therefore the dotted line represents the performance of Equipartition.

As expected under low loads (≈30%), the reductions in mean response time obtained by using $W\&\varepsilon_i$ instead of Equipartition are relatively small, but statistically significant. Again, this is because of the relatively low degree of multiprogramming, which reduces the allocation alternatives. We also note that the mean response times obtained for the $W\&\varepsilon_i$ and $W\&k_i$ policies are similar at this load (30%). Under this load, Equipartition is relatively robust since the maximum observed improvement is approximately 20% (for the range of 99-99%). The size of the reductions in mean response time increase when the degree of multiprogramming is increased (load ≈ 90%). The results for the 50-99% range show an improvement of about 20% over Equipartition at $C_W = 30$. Hence, for the range of ε_i = 50-99%, $W\&\varepsilon_i$ performs significantly better than the $W\&k_i$ policy. Moreover, the improvements in mean response time over Equipartition for $W\&\varepsilon_i$ are greater than the improvements obtained for the $W\&k_i$ policy for the ranges of ε_i = 1-99% and 75-99%. The results in Figure 7.4 demonstrate the substantial reductions in mean response time that are possible and illustrate how the size of these reductions grows with increases in average efficiency.

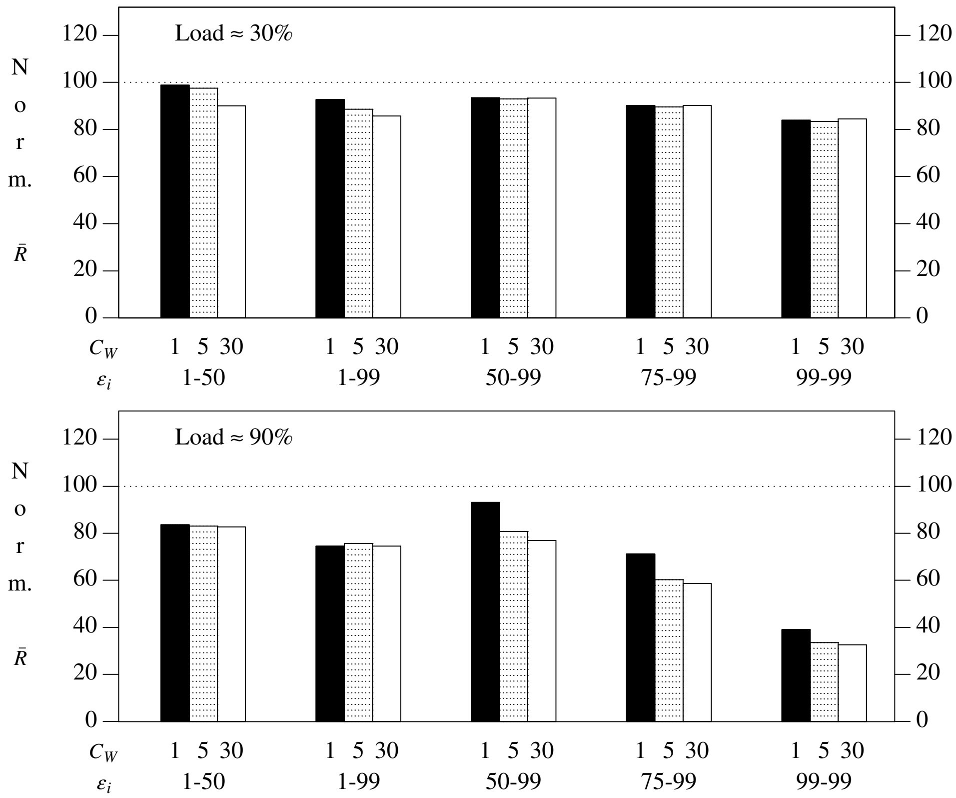

Figure 7.4: Comparing Equipartition and $W\&\varepsilon_i$

Additionally, we study the effect that the variation in job efficiencies, C_ε, might have on our results. This is done by fixing a mean efficiency and examining different efficiency ranges. We choose a fixed mean of 50%, since it offers the greatest potential variation for the Uniform distribution and compare the ranges 1-99%, 30-70%, and 50-50% (which produce a C_ε of 0.57, 0.23, and 0.0 respectively).

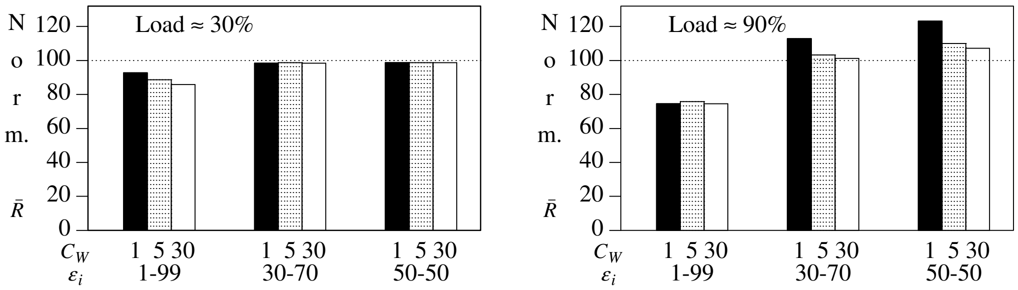

Figure 7.5: Comparing Equipartition and $W\&\varepsilon_i$

The results in Figure 7.5 show that as the variation in efficiency decreases, mean response times of the $W\&\varepsilon_i$ policy increases substantially relative to Equipartition and can eventually become significantly greater (for high loads). At low loads, the mean response time of the $W\&\varepsilon_i$ policy is nearly equal to that of the Equipartition policy for ε_i ranges of 30-70% and 50-50%. However, at high loads (90%), Equipartition performs better than $W\&\varepsilon_i$ for the ranges of 30-70% and 50-50%. Note that some previous studies which examine the advantages of using job characteristics in static scheduling may not adequately consider the effect of C_ε on their results (the efficiency parameter of jobs are not stochastic variables in Chiang, et al's study [2]).

The performance degradation of the $W\&\varepsilon_i$ policy with the decrease in the variation in efficiency might be because the $W\&\varepsilon_i$ policy is unable to handle workloads with low variation in efficiency. On the other hand, it is also possible that the allocation of processors during the first phase of the $W\&\varepsilon_i$ policy could be improved, resulting in improved performance over Equipartition. In order to determine the cause for the degradation in performance we conducted a series of experiments (whose results are not shown here) to see if a value of f_i could be found that improves the mean response time over Equipartition for the range 50-50% (under a load of 90%). We found that by using the value $f_i = 30$, we were able to produce mean response times that were lower than those obtained using Equipartition. Further investigation revealed that $f_i = \varepsilon_i$ performed reasonably well for $1 \le \varepsilon_i \le 20$ and for $80 \le \varepsilon_i \le 99$. Moreover, the mean response time of the $W\&\varepsilon_i$ policy equals (statistically) that obtained with the Equipartition policy for the range of ε_i = 20-80%. Using these results we develop a simple assignment policy that we hope uniformly produces mean response times equal to or lower than Equipartition. This policy is described in the next section.

7.4. Using W&F(ε_i)

A piecewise linear function is chosen (rather naively) as the basis for the new algorithm. The $W\&F(\varepsilon_i)$ algorithm is not designed to be the best possible algorithm under the given job and workload conditions. Rather, it is hoped that it would reduce mean response times over Equipartition for the ranges of ε_i considered in this thesis in order to demonstrate that the use of job characteristics can lead to reductions in mean response times over the Equipartition policy. This function is described, for $P = 100$, as follows:

$$f_i = \begin{cases} \dfrac{\varepsilon_i P}{100} & \text{if } 1 \le \varepsilon_i \le 20 \\ \dfrac{((\varepsilon_i - 20)\,(10/30) + 20)P}{100} = \dfrac{(\varepsilon_i + 40)P}{300} & \text{if } 20 < \varepsilon_i \le 50 \\ \dfrac{((\varepsilon_i - 50)\,(50/30) + 30)P}{100} = \dfrac{(\varepsilon_i - 32)P}{60} & \text{if } 50 < \varepsilon_i < 80 \\ \dfrac{\varepsilon_i P}{100} & \text{if } 80 \le \varepsilon_i \le 99 \end{cases}$$

We expect that a similar function could be used when $P \neq 100$. Figure 7.6 shows the processor allocation, f_i, as a function of the effective efficiency, ε_i for all of the $W\&E$ algorithms.

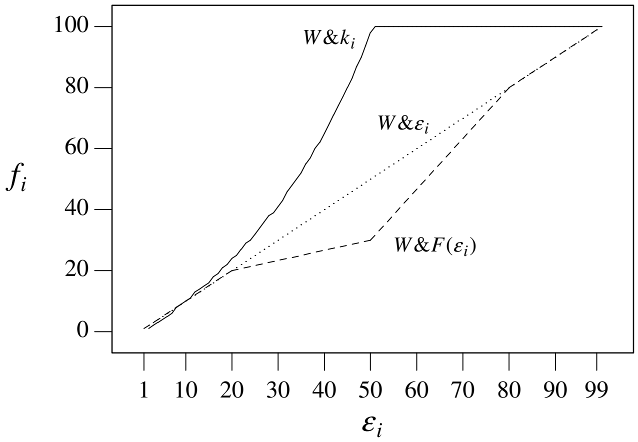

Figure 7.6: Processor allocation as a function of ε_i ($W\&k_i$, $W\&\varepsilon_i$ and $W\&F(\varepsilon_i)$)

We conduct experiments similar to those conducted with the $W\&\varepsilon_i$ policy in order to compare $W\&F(\varepsilon_i)$ with the Equipartition policy. The results of using this policy, $W\&F(\varepsilon_i)$, for a wide variety of workloads are shown in Figure 7.7. The results show that the $W\&F(\varepsilon_i)$ policy does produce mean response times that are always as good and are in many cases substantially better than those obtained with the Equipartition policy. This policy therefore performs better across the variety of workloads considered than the $W\&\beta_i$, $W\&\varepsilon_i$ and the Equipartition policies. This suggests that, in a dynamic scheduling environment, it may not be desirable to allocate processors according to the knee of the execution time - efficiency profile. Although this might be due to the execution rate function used in this thesis, no matter what model is used there will exist some jobs whose knee is greater than the number of processors in the system. In such cases, processor allocation should likely be made far below the knee. Moreover, the graphs in Figure 7.6 indicate that when ε_i = 50-50%, there is a large difference between the number of processors allocated using $W\&\beta_i$ (i.e., the knee) and the number of processors allocated using

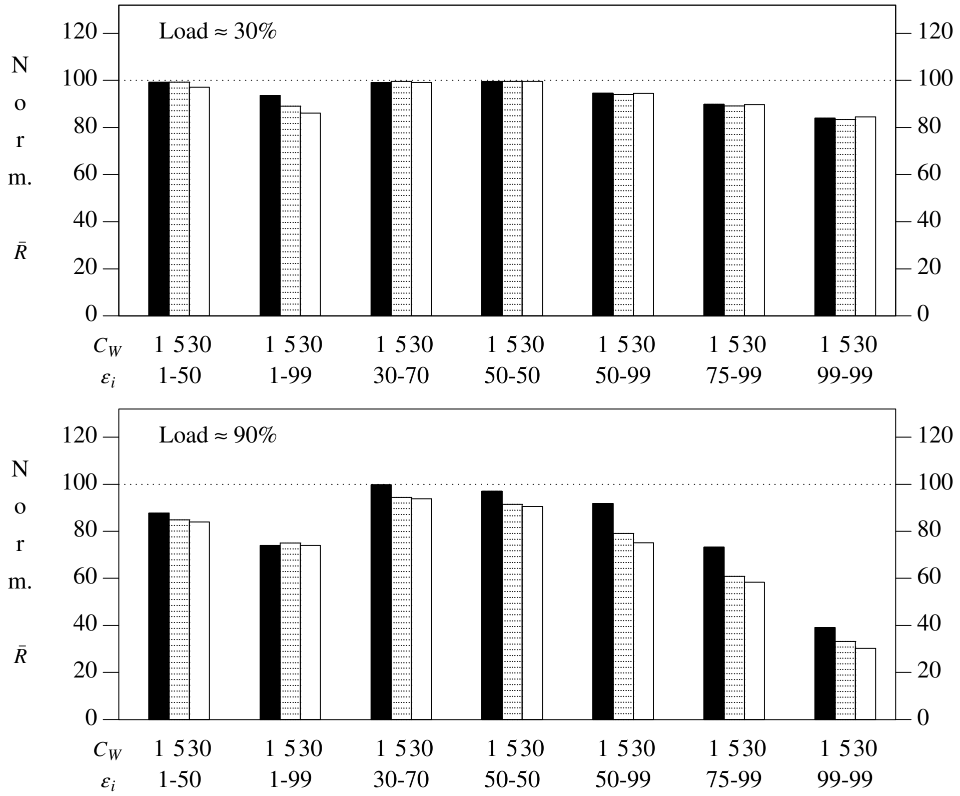

Figure 7.7: Comparing Equipartition and $W\&F(\varepsilon_i)$

either $W\&\varepsilon_i$ or $W\&F(\varepsilon_i)$. This substantiates our belief that under the job and workload models used in this thesis and under high loads, processor allocations in a dynamic scheduling environment might be better made at a point quite far below the knee.

The largest improvement observed in Figure 7.7 is approximately 70% for range of ε_i = 99-99% (for load ≈ 90%). Significant improvements in mean response times are also observed for the ranges of 50-99% and 1-99%. The improvements obtained for efficiency ranges 30-70% and 50-50% are relatively small. However, the $W\&F(\varepsilon_i)$ policy does reduce mean response times over Equipartition for all workload conditions considered.

A general trend that can be seen by examining the ranges 1-50%, 50-99%, 75-99% and 99-99%, is that the size of the reductions in mean response time obtained by using $W\&F(\varepsilon_i)$ increases as the average efficiency increases. It also seems that the size of the reduction is correlated with the coefficient of variation of work, C_W, especially when the load and average efficiency of the jobs is relatively high. Workloads with higher C_W consist of jobs which are more polarized and therefore, the execution of smaller jobs are expedited (if average efficiency is high) resulting in decreased mean response times. With low average efficiency workloads (1-50% and 1-99%), this is not possible since jobs execute with low efficiency (on the average) and hence the improvements of the $W\&F(\varepsilon_i)$ policy over Equipartition are statistically similar for C_W = 1, 5, and 30. We also reiterate that the reductions in mean response time are larger for the workload whose range of efficiency is 1-99% than for the range 30-70% even though the mean efficiency has not changed. This indicates that the results are affected by the coefficient of variation in effective efficiency. Note that the ranges of ε_i = 50-50% and 30-70% result in mean response times that are relatively close even though C_ε for the 30-70% range (0.234) is greater than that for the 50-50% range (0.0). We expect that this is because the allocation for the 50-50% range was specifically tuned to produce the maximum improvement over Equipartition while this was not done for the 30-70% range.

Although the $W\&F(\varepsilon_i)$ algorithm can likely be improved (since it was chosen quite naively) these experiments demonstrate that the $W\&E$ strategy is an attractive approach to using job characteristics in making processor allocation decisions in a dynamic scheduling environment. If the number of processors assigned during the first phase of the algorithm is chosen properly, this method can produce mean response times that are significantly lower than those obtained by using Equipartition across a wide variety of workloads. As well, the improvements are greater than those that can be obtained using only characteristics of work, W_i, or efficiency, β_i, in isolation. We believe that these results demonstrate not only the importance of using job characteristics correctly but also the need for allocation policies that effectively use characteristics of both work and efficiency in scheduling parallel programs in multiprogrammed multiprocessors.

7.5. Revisiting the Job Model

Throughout this thesis we use the function proposed by Dowdy [4] in modeling the execution rate of a parallel job. One of the problems with the Dowdy model is that it does not model factors for which one might wish to limit the number of processors allocated to a parallel job. More realistically, it might be desirable to limit processor allocation to job J_i because of two factors:

1) The first factor is the available parallelism of a job, which is the maximum number of processors that can be used simultaneously to execute a parallel program. (e.g., a job may have a number of parallel threads that is less than P.)

2) The second factor we call the *pmax* of an application. The allocation of *pmax* processors results in maximum speedup that can be achieved by a job. Processor allocations of more than *pmax* increases program execution time (and results in a decrease in speedup). Therefore, it is not desirable to allocate more than *pmax* processors to a job.

We have used the Dowdy model to this point since it is relatively simple and models all parallel programming overheads by means of a single parameter, β_i. This has provided us with fundamental insights into the scheduling problem. Now that we have a better understanding of the scheduling problem we revisit the job model by considering another parameter, N_i, which is used to model the factors for which the number of processors allocated to job J_i should be limited. The parameter N_i is used to model both pmax and the parallelism in job J_i. We assume N_i to be constant throughout the lifetime of the job. A similar job model is used in the study by Chiang, et al. [2].

The parameter N_i follows a bounded geometric distribution with observed mean $\bar{N}$ and an upper bound being equal to the number of processors in the system, P. The input mean to the distribution is somewhat greater than the observed mean, $\bar{N}$, since the generated N_i values are bounded by P. In other words, if a value greater than P is generated, another random value is generated until the value obtained is less than or equal to P. The resulting distribution has a mean of $\bar{N}$. Similar methods for generating N_i are used in the study by Zhou and Brecht [33]. Leutenegger and Vernon [12] and Chiang, et al. [2] also assume that the parameter N_i follows a bounded geometric distribution. Note that Zhou and Brecht [33], Leutenegger and Vernon [12] and Chiang, et al. [2] call the parameter N_i the available parallelism, since only the parallelism of a parallel job is modeled in their studies. We conduct experiments using $\bar{N}$ values of approximately 25, 45 and 73. (They are not at regular intervals because a regenerative method is used to generate bounded N_i values.) Note that in our new model the service demand of an application, W_i, is not correlated with N_i.

We devise and evaluate new versions of the $W\&k_i$, $W\&\varepsilon_i$ and the $W\&F(\varepsilon_i)$ policies that use the parameter N_i (although naively) while making scheduling decisions. These policies for the most part are the same as the ones described in Chapter 4, except that processor allocations to job, J_i, is bounded by the N_i value. This is because, either the job has insufficient parallelism or the allocation of more than N_i processors would increase the job's execution time. The $W\&E$ policies are compared to a modified version of Equipartition which computes the processor allocation to each job in the following manner. The initial allocation to all jobs is zero. Each job is then allocated an equal number of processors unless a job has a smaller N_i than the equipartition value $(\frac{P}{n})$. In that case, N_i processors are allocated to those jobs for which N_i is less than the equipartition value. Note that the equipartition value is recursively re-computed for the remaining jobs. In this section we refer to this policy as Equipartition. Also note that although this variant of Equipartition uses a job characteristic, namely N_i, we use this algorithm to more fairly determine the benefits of using application characteristics. The use of a naive Equipartition policy which has no knowledge of N_i would produce results that may be considered biased in favor of policies that use application characteristics. Although it could be argued that a naive Equipartition should be the basis for comparison, we use an algorithm that assumes the Equipartition policy knows N_i.

We now present the results of the experiments conducted with the new job model in order to evaluate the performance of the $W\&E$ policies. A series of experiments were conducted with $\bar{N}$ = 25, 45 and 73 in order to examine the impact of $\bar{N}$ on our results. Note that $P = 100$ for all of our experiments and that all jobs have $N_i \leq P$. The arrival rates corresponding to processor utilizations of 90% were determined using the $W\&E$ policies and then used for the Equipartition policy. Again, all results are normalized with respect to the mean response time of the Equipartition policy. The coefficient of variation in service demand, C_W, and the effective efficiency range, $\varepsilon_{\min} - \varepsilon_{\max}$, are shown below the x-axis for each of these experiments.

The results of these experiments, shown in Figures 7.8, 7.9 and 7.10, exhibit trends which are similar to those observed in Section 7.2, 7.3 and 7.4. However, we do observe that the parameter $\bar{N}$ affects the performance of the $W\&E$ policies. For low $\bar{N}$ (e.g., $\bar{N} = 25$), the processor allocation to job J_i is constrained by N_i. In other words, if f_i is greater than N_i, only N_i processors are allocated to J_i. This implies that the scheduler does not actually make use of the knowledge of job efficiency (β_i) while making scheduling decisions for such jobs. In Figure 7.8, when $\bar{N} = 25$, 72% of the processor allocations (in a simulation of 500,000 jobs) were made according to N_i (i.e., $p_i = N_i$ for J_i in 72% of the cases). For $\bar{N} = 45$, $p_i = N_i$ for 55% of the cases and $p_i = N_i$ for 40% of the cases when $\bar{N} = 73$. For low values of $\bar{N}$, this results in increased processor sharing among the jobs in the system and consequently reduces the gap between the $W\&E$ policies and Equipartition. However, the performance of the $W\&E$ policies improves as $\bar{N}$ increases. Note that only the $W\&F(\varepsilon_i)$ policy is able to improve mean response times when compared with Equipartition for $\bar{N} = 73$ across all the workloads considered.

In general, the $W\&E$ policies perform surprisingly well (especially at higher values of $\bar{N}$) even though they do not intelligently use knowledge of N_i in making partitioning decisions (processor allocations are only bounded by N_i). Moreover, the $W\&E$ policies no longer use knowledge of the efficiency effectively, especially at low values of $\bar{N}$. These results indicate that the $W\&E$ family of policies can be further enhanced by proper use of knowledge of N_i. However, this reinforces the conclusion drawn in Section 7.4 that proper use of application characteristics is important in dynamic scheduling and can lead to significant improvements in mean response times when compared with the Equipartition policy.

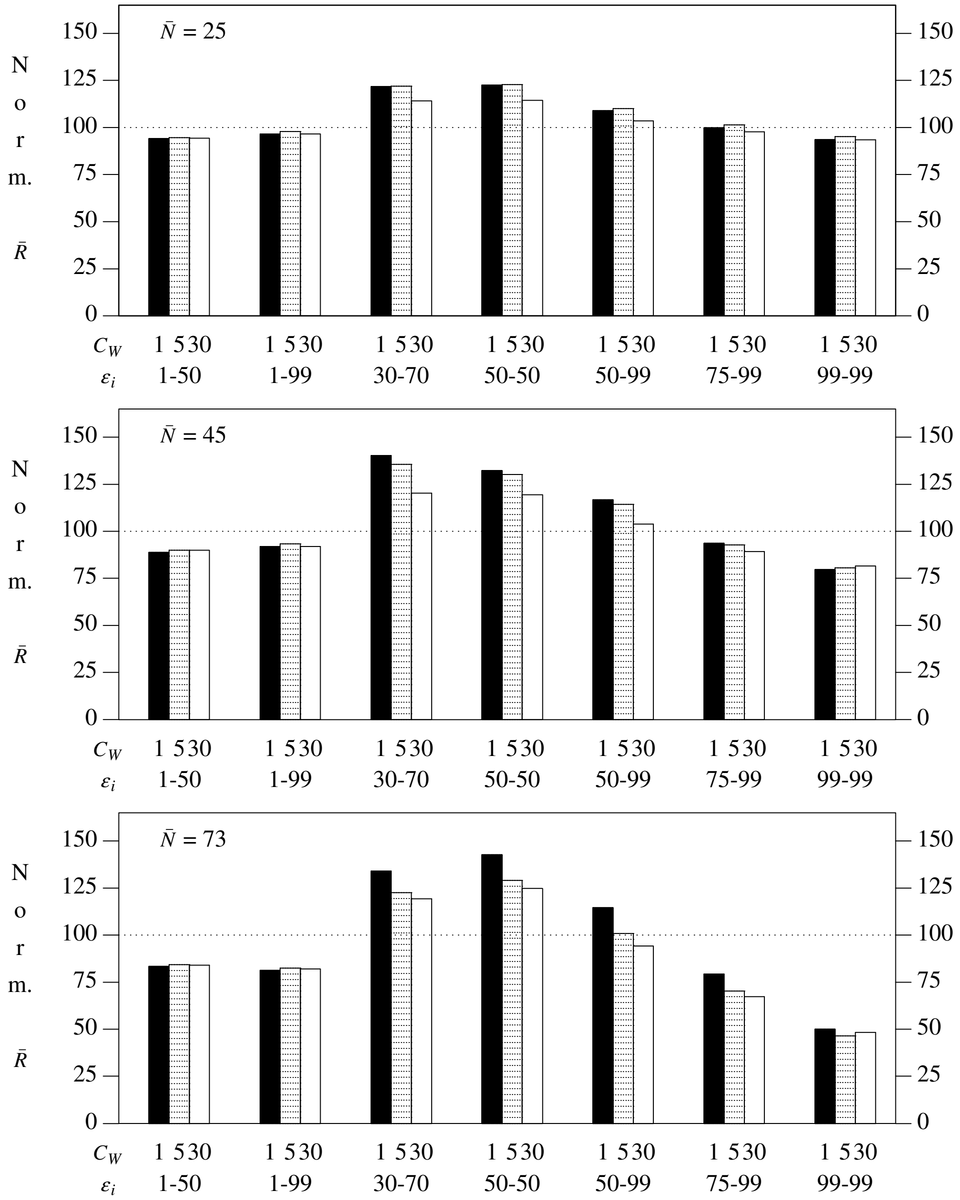

Figure 7.8: Comparing Equipartition and $W\&k_i$, Load ≈ 90

7.6. Summary

In this chapter we introduce a new family of dynamic processor allocation policies that use characteristics of a job's work and its efficiency in making scheduling decisions. The $W\&k_i$ policy, which allocates processors according to the knee, improves mean response times over Equipartition for workloads with either low average efficiency or very high average efficiency. However, for workloads with medium to high average efficiency (the 50-99% range) the Equipartition policy produces mean response times which are significantly lower than the $W\&k_i$ policy. This indicates that the knee might not be a suitable point for processor allocations,

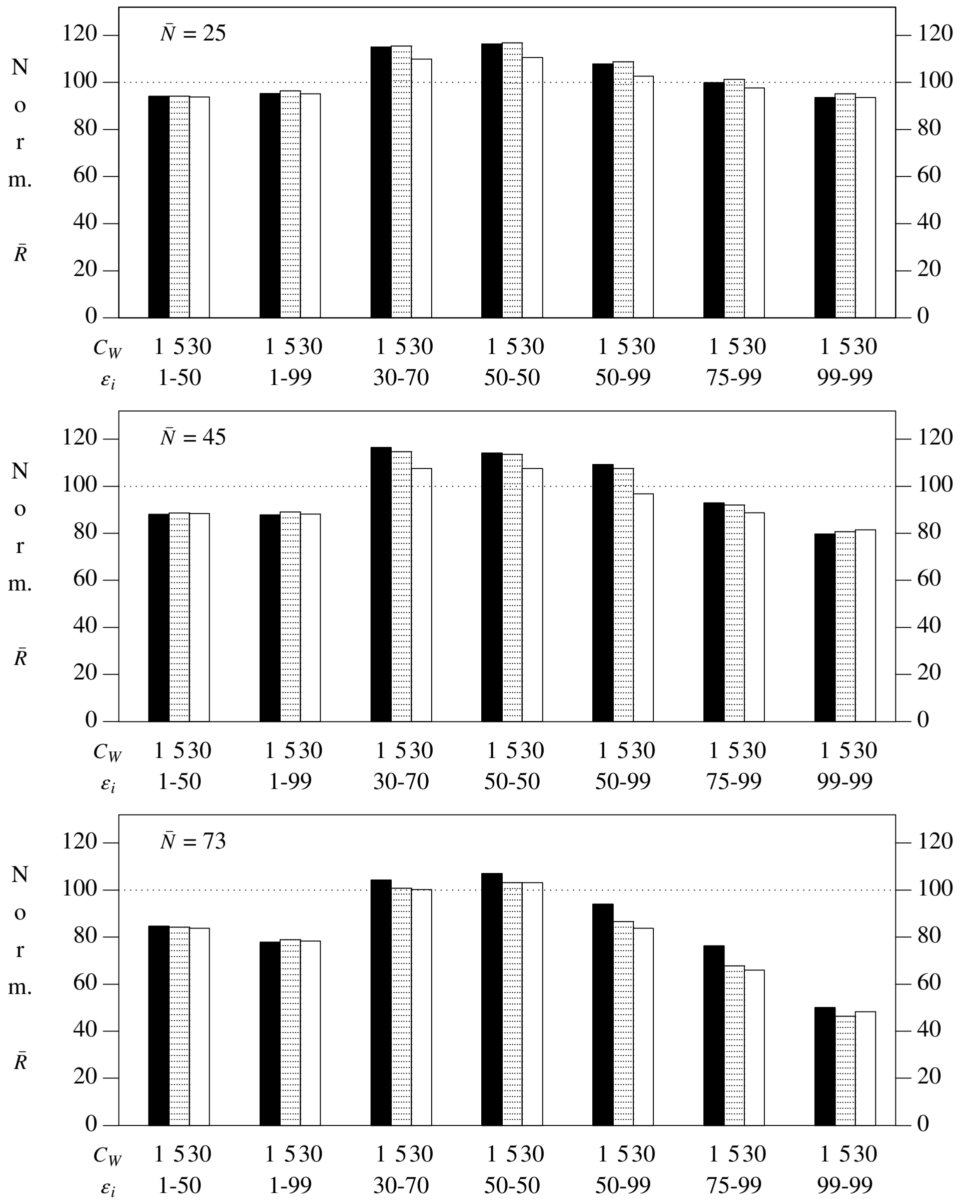

Figure 7.9: Comparing Equipartition and $W\&\varepsilon_i$, Load ≈ 90

especially in those cases where the knee is greater than the number of processors in the system.

The $W\&\varepsilon_i$ policy allocates processors below the knee (i.e., according to ε_i) and improves mean response times over Equipartition for most of the workloads examined. However, it is unable to improve mean response times for workloads with low coefficient of variation in efficiency. To improve performance across all workloads considered in this thesis we introduce the $W\&F(\varepsilon_i)$ policy which uses a piecewise linear function to allocate processors to jobs. The $W\&F(\varepsilon_i)$ policy demonstrates that proper use of application characteristics such as work and efficiency can lead to improved mean response time over the Equipartition policy and that processor allocations in a dynamic scheduling environment should likely be made at points

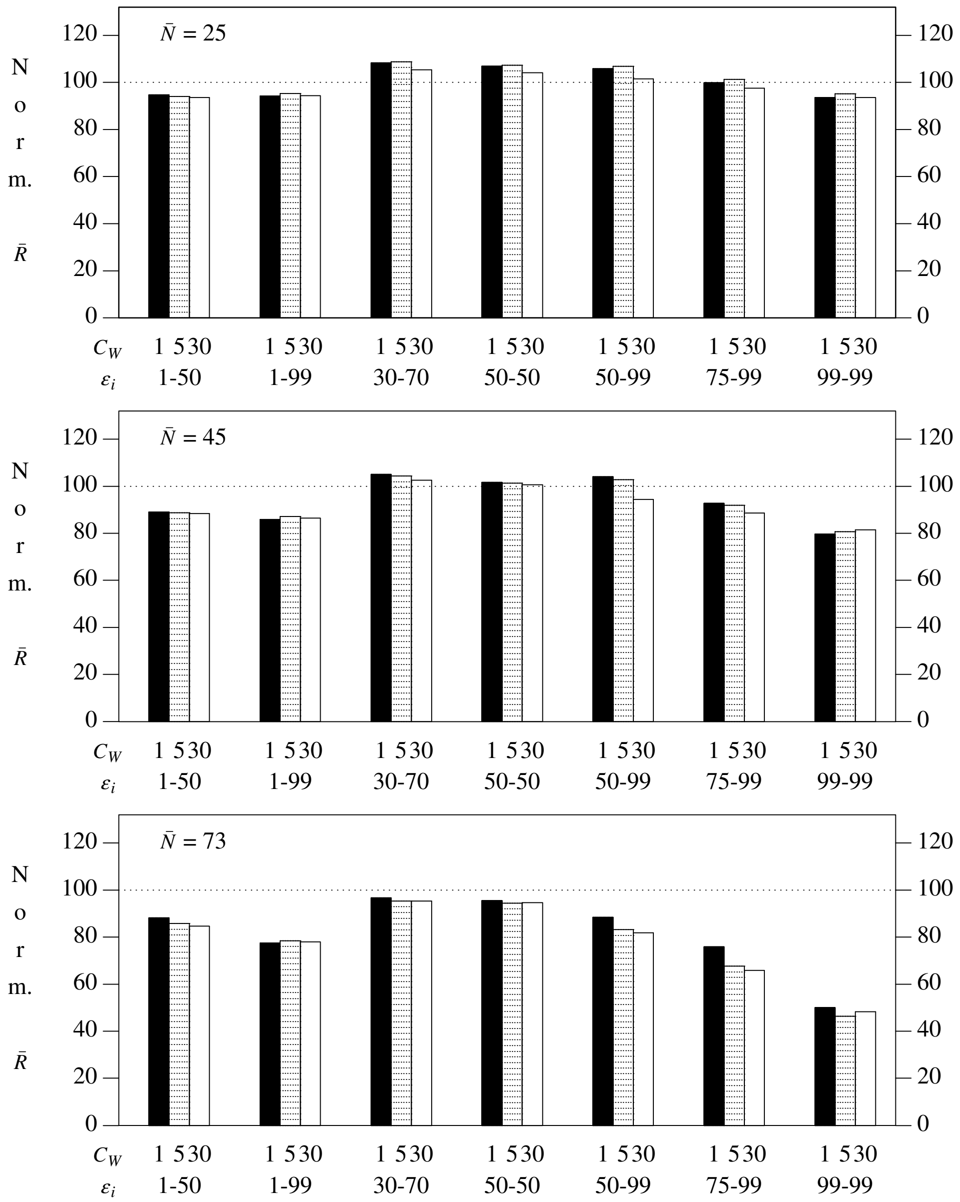

Figure 7.10: Comparing Equipartition and $W\&F(\varepsilon_i)$, Load ≈ 90

which are below the knee. The results obtained with the $W\&E$ family of policies also demonstrate that the reductions in mean response time obtained by using job characteristics (over Equipartition) increase with the coefficient of variation in service demand and coefficient of variation in effective efficiency.

We also briefly consider another job model to demonstrate that qualitatively our results are relatively insensitive to the job model. This additional job model uses a parameter, N_i, to model some of the factors for which it might be desirable to limit the number of processors allocated to a job; namely pmax and available parallelism. However, this new model does not seem to qualitatively affect our results, indicating that the $W\&E$ policies can likely be further enhanced by considering N_i while making scheduling decisions. This strengthens our conclusion that proper use of job characteristics in a dynamic scheduling environment can lead to significant improvements in mean response time when compared with policies that use no characteristics in making allocation decisions.

Chapter 8

Conclusions

8.1. Introduction

The goals of this thesis are to gain insights into the factors involved in designing scheduling policies for multiprogrammed multiprocessors and to demonstrate that application characteristics can be used to make improved allocation decisions in a dynamic scheduling environment. We perform simulation studies to demonstrate and quantify the advantages of using job characteristics in making scheduling decisions over policies that use none. The following job characteristics are used:

- **Work:** The total amount of work to be executed by an application.
- **Efficiency:** The efficiency with which processors can be effectively utilized by an application.

We consider work and efficiency since these are the key characteristics which directly determine the execution time of a parallel program and ultimately overall system performance. The use of these job characteristics in making processor allocations has led to the contributions outlined in the following section.

8.2. Contributions

In this thesis we demonstrate how and when application characteristics can be used to make scheduling decisions in order to improve mean response time over existing scheduling techniques.

We first show, in Chapter 5, that if jobs execute with perfect efficiency, policies that use precise knowledge of service demands reduce mean response time significantly over Equipartition. The size of the improvements increase with the coefficient of variation in service demand as well as the system load. The maximum improvement is 70 percent which provides an approximate bound on the reductions that can be obtained by using job characteristics in making scheduling decisions. If precise information is not available, the accumulated CPU time or the time spent in the system can be used as estimates of the expected remaining execution time resulting in a 45% reduction when compared with Equipartition.

For workloads with low average efficiency, policies that make partitioning decisions based on the work jobs execute do not reduce mean response times when compared to Equipartition. This is because the costs of allocating large portions of the processors to smaller jobs (under-utilized processors) outweigh the benefits (reduced response time). However, for workloads with high average efficiency the benefits increase and therefore, allocation policies which use work in making allocation decisions perform better than the Equipartition policy. These results are also shown to be insensitive to two distributions of effective efficiency. Hence, when jobs execute with imperfect efficiency, using only work in making scheduling decisions does not improve performance over the Equipartition policy unless the average efficiency of the workload is very high.

In Chapter 6 we see that for the job models considered in this thesis, the parameter which characterizes the efficiency of an application, β_i, corresponds to the knee of job J_i. Thus, we also examine the effects of making processor allocations according to the knee while investigating allocation policies that make partitioning decisions based on the efficiency parameter, β_i. We show that policies that use only efficiency (β_i) in making scheduling decisions are unable to substantially reduce mean response times when compared with Equipartition, although, in some cases relatively small reductions are obtained. The improvements are obtained only for workloads with low average efficiency and are also limited by the coefficient of variation in effective efficiency. This is because the scheduler does not consider the remaining work while making scheduling decisions.

In Chapter 7 we introduce a new family of policies called the $W\&E$ policies which use both work and efficiency in making processor allocations. These policies are shown to reduce mean response times significantly when compared with Equipartition. The size of these improvements increases with the average efficiency of the workload. The reductions obtained by using work and efficiency in making processor allocations are relatively small for workloads having a low coefficient of variation in efficiency. However, the size of the reductions increase with the coefficient of variation in efficiency as well as the coefficient of variation in service demand. Our results suggest that the knee might not be a suitable point for processor allocation especially for workloads with high average efficiency. Our observation could be influenced by the execution rate function used, although, many jobs with relatively high efficiency will have $k_i \geq P$. In such cases processor allocation at the knee (which is bounded by P) is certainly not suitable.

We conclude that using job characteristics in a dynamic scheduling environment is important and results in significant improvements in mean response times when compared with policies that do not use such characteristics. As well, these characteristics need to be used properly in order to improve mean response time over existing space-sharing schemes (i.e., Equipartition). We also employ another job model to show that the general conclusions derived from this thesis are independent of the two job models considered.

Although we demonstrate the importance of using characteristics of work and efficiency and the importance of using them properly, we do realize that precise information about a job's remaining work and efficiency may not be readily available. However, it is important to understand the properties of effective scheduling algorithms in order to design and implement improved techniques. As well, we believe that future studies may lead to improved estimation techniques of job characteristics. The insights obtained from this thesis can and will be used to improve practical implementations of future multiprocessor schedulers.

We observe in this thesis that equipartitioning processors is a relatively robust approach since it produces acceptable mean response times over a wide range of workloads. It is shown to produce mean response times that are quite close to those obtained with policies that use job characteristics, specifically for workloads having low loads, low coefficients of variation in efficiency and low coefficients of variation in service demand. Therefore, in the absence of job characteristics Equipartition is likely to be used in current dynamic scheduling implementations.

8.3. Future Work

Studies of real multiprocessor workloads are required to obtain precise knowledge of means, distributions and coefficients of variation of workload characteristics (e.g., work, efficiency and available parallelism). This would alleviate the complexity of performing much of the sensitivity analysis performed in this thesis, considerably simplify the problem and increase the strength and reliability of the conclusions.

The performance of policies that use application characteristics in making allocation decisions was investigated using the model proposed by Dowdy [4] (actually the modified version of that model used by Chiang, et al. [2]). The importance of job characteristics and the relative performance of the policies evaluated in this thesis could be further investigated using other models (e.g., the model proposed by Sevcik [25]).

The analysis of techniques that utilize application characteristics are of little benefit if they can not be applied in practical situations. In this thesis we provide a simple example and use estimates of service demand proposed by Brecht [1] to demonstrate that such estimates can improve mean response time over Equipartition when jobs execute with perfect efficiency. More effective methods need to be investigated for distinguishing different jobs according to their remaining work. Moreover, methods for estimating the efficiency of jobs at run time, which would enable policies analogous to the $W\&E$ family to be used in real multiprocessors, should be investigated.

Policies which use only efficiency in making partitioning decisions obtain relatively small reduction in mean response times over the Equipartition policy in some instances. Further investigation is needed to improve these policies so that larger reductions are obtained over Equipartition.

We use the $W\&E$ policies to demonstrate that proper use of work and efficiency may lead to improved mean response time over policies that use no job characteristics. It is likely that the $W\&E$ policies could be further improved. Further research is needed to enhance the existing allocation schemes of the first phase of the $W\&E$ family of scheduling policies. As well, further efforts could be directed into investigating the importance of the second phase of allocation in the $W\&E$ policies. Moreover, the $W\&E$ policies could also be further enhanced by using the knowledge of the parameter N_i (which models pmax and the available parallelism of applications).

In this thesis we identify the conditions under which characteristics of parallel programs can be used to reduce mean response time in a dynamic scheduling environment. We believe that these insights can be used to enhance static allocation policies as well as policies which time-share processors among jobs.

Appendix

In this thesis job arrivals are assumed to follow a Poisson distribution with mean λ and therefore the inter-arrival times are distributed exponentially with a mean of $\frac{1}{\lambda}$. The mean values used for generating the inter-arrival times of the jobs in our simulation are noted in the following tables. Each table contains the arrival rate for the experiments conducted to generate the graphs in the figures shown in Chapters 5, 6 and 7. The table header shows the figure number.

Figure 5.1	
Load	Inter Arrival Time
30	33.33
50	20.00
70	14.29
90	11.11

Figure 5.2	
Load	Inter Arrival Time
90	11.11

Figure 5.3(a)&(b)	
Load	Inter Arrival Time
90	11.11

Figure 5.4		
Figure	Range	Inter Arrival Time
5.4(a)	1-99%	49.37
5.4(b)	1-50%	76.52
5.4(c)	50-99%	15.00
5.4(d)	75-99%	12.56

Figure 5.5		
Figure	$\bar{\varepsilon}$	Inter Arrival Time
5.5(a)	50	44.00
5.5(b)	25.5	57.5
5.5(c)	74.5	15.5
5.5(d)	87.0	12.8

Figure 6.1		
Figure	Range	Inter Arrival Time
6.1(a)	1-50%	306.07
6.1(b)	1-99%	176.10
6.1(c)	50-99%	47.49
6.1(d)	75-99%	40.19

Figure 6.2		
Figure	Range	Inter Arrival Time
6.2(a)	1-50%	64.66
6.2(b)	1-99%	37.20
6.2(c)	50-99%	12.95
6.2(d)	75-99%	11.48

Figure 6.3		
Figure	Range	Inter Arrival Time
6.3(a)	1-50%	51.01
6.3(b)	1-99%	29.35
6.3(c)	50-99%	11.87
6.3(d)	75-99%	12.06

Figure 7.1		
Load	Range	Inter Arrival Time
30	1-50%	306.07
	1-99%	176.10
	50-99%	47.49
	75-99%	40.19
	99-99%	33.67
90	1-50%	36.73
	1-99%	26.42
	50-99%	14.25
	75-99%	13.40
	99-99%	11.22

Figure 7.4		
Load	Range	Inter Arrival Time
30	1-50%	229.55
	1-99%	132.08
	50-99%	40.71
	75-99%	40.19
	99-99%	33.67
90	1-50%	36.73
	1-99%	23.48
	50-99%	13.57
	75-99%	12.30
	99-99%	11.22

Figure 7.5		
Load	Range	Inter Arrival Time
30	1-99%	132.08
	30-70%	62.23
	50-50%	57.14
90	1-99%	23.48
	30-70%	16.75
	50-50%	16.95

Figure 7.7		
Load	Range	Inter Arrival Time
30	1-50%	229.55
	1-99%	132.08
	30-70%	62.23
	50-50%	57.14
	50-99%	40.71
	75-99%	37.68
	99-99%	33.67
90	1-50%	36.73
	1-99%	22.97
	30-70%	15.56
	50-50%	14.81
	50-99%	12.95
	75-99%	12.06
	99-99%	11.22

Figure 7.8		
$\bar{N}$	Range	Inter Arrival Time
25	1-50%	25.51
	1-99%	17.04
	30-70%	13.61
	50-50%	13.33
	50-99%	11.87
	75-99%	11.48
	99-99%	11.22
45	1-50%	26.23
	1-99%	17.32
	30-70%	15.02
	50-50%	15.38
	50-99%	12.39
	75-99%	11.37
	99-99%	11.22
73	1-50%	35.32
	1-99%	23.48
	30-70%	17.85
	50-50%	18.18
	50-99%	13.57
	75-99%	11.82
	99-99%	11.22

Figure 7.9		
$\bar{N}$	Range	Inter Arrival Time
25	1-50%	25.51
	1-99%	17.04
	30-70%	13.61
	50-50%	13.33
	50-99%	11.87
	75-99%	11.48
	99-99%	11.22
45	1-50%	26.23
	1-99%	17.32
	30-70%	15.02
	50-50%	15.38
	50-99%	12.39
	75-99%	11.37
	99-99%	11.22
73	1-50%	35.32
	1-99%	23.48
	30-70%	17.85
	50-50%	18.18
	50-99%	13.57
	75-99%	11.82
	99-99%	11.22

Figure 7.10		
$\bar{N}$	Range	Inter Arrival Time
25	1-50%	25.51
	1-99%	17.04
	30-70%	13.61
	50-50%	13.33
	50-99%	11.87
	75-99%	11.48
	99-99%	11.22
45	1-50%	26.23
	1-99%	17.32
	30-70%	15.02
	50-50%	15.38
	50-99%	12.39
	75-99%	11.37
	99-99%	11.22
73	1-50%	35.32
	1-99%	23.48
	30-70%	17.85
	50-50%	18.18
	50-99%	13.57
	75-99%	11.82
	99-99%	11.22

Bibliography

[1] T. B. Brecht, ‘‘Multiprogrammed Parallel Application Scheduling in NUMA Multiprocessors’’, CSRI-303, Computer Systems Research Institute, University of Toronto, Toronto, April, 1994.

[2] S. Chiang, R. K. Mansharamani, and M. K. Vernon, ‘‘Use of Application Characteristics and Limited Preemption for Run-To-Completion Parallel Processor Scheduling Policies’’, *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 33-55, May, 1994.

[3] E. Coffman and L. Kleinrock, ‘‘Feedback Queueing Models for Time-Shared Systems’’, *Journal of the ACM*, *Vol*. 15, *No*. 4, pp. 549-576, October, 1968.

[4] L. W. Dowdy and M. R. Leuze, ‘‘On Modeling Partitioned Multiprocessor Systems’’, *International Journal of High Speed Computing*, *Vol*. 6, pp. 31-53, 1994.

[5] D. L. Eager, J. Zahorjan, and E. D. Lazowska, ‘‘Speedup Versus Efficiency in Parallel Systems’’, *IEEE Transactions on Computers*, *Vol*. 38, *No*. 3, pp. 408-423, March, 1989.

[6] D. Ghosal, G. Serazzi, and S. K. Tripathi, ‘‘The Processor Working Set and Its Use in Scheduling Multiprocessor Systems’’, *IEEE Transactions on Software Engineering*, *Vol*. 17, *No*. 5, pp. 443-453, May, 1991.

[7] A. Gupta, A. Tucker, and S. Urushibara, ‘‘The Impact of Operating System Scheduling Policies and Synchronization Methods on the Performance of Parallel Applications’’, *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 120-132, San Diego, CA, May, 1991.

[8] R. Jain, **The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling,** New York: Wiley, c1991.

[9] M. Kumar, ‘‘Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications’’, *IEEE Transactions on Computers*, *Vol*. 37, *No*. 9, pp. 1088-1098, September, 1988.

[10] W. E. Leland and T. J. Ott, ‘‘Load-balancing Heuristics and Process Behaviour’’, *Proceedings of the 1986 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 54-69, Raleigh, NC, 1986.

[11] S. T. Leutenegger and R. D. Nelson, ‘‘Analysis of Spatial and Temporal Scheduling Policies for Semi-Static and Dynamic Multiprocessor Environments’’, IBM Research Report RC 17086 (No. 75594), August 1, 1991.

[12] S. T. Leutenegger and M. K. Vernon, ‘‘The Performance of Multiprogrammed Multiprocessor Scheduling Policies’’, *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 226-236, Boulder, CO, May, 1990.

[13] S. Majumdar, D. Eager, and R. B. Bunt, ‘‘Scheduling in Multiprogrammed Parallel Systems’’, *Proceedings of the 1988 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 104-113, May, 1988.

[14] S. Majumdar, D. L. Eager, and R. B. Bunt, ‘‘Characterisation of Programs for Scheduling in Multiprogrammed Parallel Systems’’, *Performance Evaluation*, *Vol*. 13, *No*. 2, pp. 109-130, 1991.

[15] M. A. Marsan, G. Balbo, and G. Conte, ‘‘A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems’’, *ACM Transactions on Computer Systems*, *Vol*. 2, *No*. 2, pp. 93-122, May, 1984.

[16] C. McCann, R. Vaswani, and J. Zahorjan, ‘‘A Dynamic Processor Allocation Policy for Multiprogrammed, Shared-Memory Multiprocessors’’, *ACM Transactions on Computer Systems*, *Vol*. 11, *No*. 2, pp. 146-178, May, 1993.

[17] V. Naik, S. Setia, and M. Squillante, ‘‘Scheduling of Large Scientific Applications on Distributed Memory Multiprocessor Systems’’, *Proceedings of the the 6th SIAM Conference on Parallel Processing for Scientific Computation*, pp. 913-922, 1993.

[18] V. Naik, S. Setia, and M. Squillante, ‘‘Analysis of Job Scheduling Policies in Parallel Supercomputing Environments’’, *Proceedings of Supercomputing’93*, pp. 824-833, November, 1993.

[19] E. Rosti, E. Smirni, L. Dowdy, G. Serazzi, and B. Carlson, ‘‘Robust Partitioning Policies of Multiprocessor Systems’’, *Performance Evaluation*, *Vol*. 19, *No*. 2-3, pp. 141-165, 1994.

[20] S. Setia, ‘‘The interaction between Memory Allocation and Adaptive Partitioning in Message-Passing Multicomputers’’, *IPPS ’95 Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 89-100, Santa Barbara, CA, April 25, 1995.

[21] S. Setia and S. Tripathi, ‘‘An Analysis of Several Processor Partitioning Policies for Parallel Computers’’, *CS-TR-2684, Dept. of Computer Science, Univ. of Maryland*, May, 1991.

[22] S. Setia and S. Tripathi, ‘‘A Comparative Analysis of Static Processor Partitioning Policies for Parallel Computers’’, *Proceedings of the International Workshop on Modeling and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 283-287, January 1993.

[23] S. K. Setia, M. S. Squillante, and S. K. Tripathi, "Analysis of Processor Allocation in Multiprogrammed, Distributed Memory Parallel Processing Systems", *IEEE Transactions on Parallel and Distributed Systems*, *Vol*. 5, *No*. 4, pp. 401-420, April, 1994.

[24] K. C. Sevcik, "Characterizations of Parallelism in Applications and Their Use In Scheduling", *Proceedings of the 1989 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 171-180, May, 1989.

[25] K. C. Sevcik, "Application Scheduling and Processor Allocation in Multiprogrammed Multiprocessors", *Performance Evaluation*, *Vol*. 9, *No*. 2-3, pp. 107-140, 1994.

[26] K. C. Sevcik and E. Parsons, "Multiprocessor Scheduling for High-Variability Service Time Distributions", *IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 76-88, Santa Barbara, CA, April 25, 1995.

[27] A. Tucker and A. Gupta, "Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors", *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pp. 159-166, 1989.

[28] A. H. Veen, "Dataflow machine architectures", *ACM Computing Survey*, *Vol*. 18, *No*. 4, pp. 365-396, December 1986.

[29] T. H. Wonnacott and R. J. Wonnacott, **Introductory Statistics**, John Wiley and Sons, Inc., p. 277, 1990.

[30] Chee-Shong Wu, Processor Scheduling in Multiprogrammed Shared Memory NUMA Multiprocessors, M.Sc. Thesis, University of Toronto, Toronto, Ontario, October, 1993.

[31] J. Zahorjan and C. McCann, "Processor Scheduling in Shared Memory Multiprocessors", *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 214-225, Boulder, CO, May, 1990.

[32] J. Zahorjan and C. McCann, "Scheduling Memory Constrained Jobs on Distributed Memory Parallel Computers", *Proceedings of the 1995 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 208-219, Ottawa, May, 1995.

[33] S. Zhou and T. B. Brecht, "Processor Pool-Based Scheduling for Large-Scale NUMA Multiprocessors", *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 133-142, May, 1991.