Using Parallel Program Characteristics in Dynamic Processor Allocation Policies *

Timothy B. Brecht and Kaushik Guha
Department of Computer Science, York University
4700 Keele Street North York, Ontario CANADA M3J 1P3
email: brecht@cs.yorku.ca / guha@cs.yorku.ca

Abstract

In multiprocessors a parallel program's execution time is directly influenced by the number of processors it is allocated. The problem of scheduling parallel programs in a multiprogrammed environment becomes one of determining how to best allocate processors to the different simultaneously executing programs in order to minimize mean response time.

In this paper we address the problem of how many processors to allocate to each of the executing parallel jobs by examining the following questions:

1. Is allocating processors equally among all jobs (equipartitioning) a desirable property of a scheduling algorithm?
2. Does using information about the service demand of parallel jobs significantly reduce mean response time?
3. Does using information about the efficiency with which parallel jobs execute significantly reduce mean response time?
4. Does allocating each job a number of processors corresponding to the knee of the execution time – efficiency curve significantly reduce mean response time?
5. What are the desirable properties of a scheduler that is designed to minimize mean response time?

The main contributions of this paper are: a first-order understanding of how processing power should be dynamically allocated to jobs, a new family of algorithms for dynamic processor allocation and a rough quantification of the benefits that may be realized by properly utilizing job characteristics when making processor allocation decisions. We believe that these new algorithms can be combined with recently demonstrated techniques for obtaining sufficiently accurate runtime estimates of job efficiencies to improve scheduler implementations for multiprogrammed multiprocessors.

*This paper was presented at *PERFORMANCE '96: International Conference on Performance Theory, Measurement and Evaluation of Computer and Communication Systems*, Lausanne, Switzerland, October, 1996. A version of this paper appears in **Performance Evaluation**, Volume 27 and 28, pp. 519-539, October, 1996.

1 Introduction

When trying to minimize the mean response time of parallel programs in a 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 if one job is not efficiently utilizing the processors it has been allocated, another job might be able to make better use of them. The problem of minimizing mean response time is further complicated by other factors: the amount of work to be executed by each job (its service demand) and the arrival of new jobs.

Understanding that the optimal uniprocessor scheduling policy executes the job with the Shortest Remaining Processor Time (SRPT) first has led to significant improvements in many uniprocessor scheduler implementations, even though the execution time of jobs is not known *a priori* . For example, one of the objectives of multi-level feedback queue schedulers [6, 2, 15], popular in many UNIX systems, is to try to approximate the SRPT algorithm by decreasing the amount of CPU time (the number of quantums) allocated to long running jobs. This approach helps to complete shorter jobs first and reduces mean response time when compared with a Round Robin algorithm [16]. Policies that attempt to approximate a SRPT approach have been proposed and studied in multiprocessor scheduling contexts [20, 32, 3]. However, we believe that such strategies must also consider the efficiency with which parallel programs execute.

Previous work has suggested approaches for obtaining estimates of a job's efficiency during runtime [3]. Recently, Nguyen, Vaswani, and Zahorjan [25, 24] have experimentally demonstrated that such estimates can in fact be determined by monitoring the job's execution during run time. They also find that the performance of scheduling policies that use these estimates achieve performance surprisingly close to that possible when perfect *a priori* information is used.

We seek foundational insights into the properties of effective scheduling algorithms for multiprogrammed multiprocessors by examining policies that assume *a priori* knowledge of job characteristics (work and efficiency) to improve processor allocation decisions. We believe that these insights and resulting algorithms, when combined with the recently demonstrated techniques for obtaining job characteristics at runtime, can form the basis of future multiprocessor scheduler implementations.

In this paper we use simulations to evaluate a number of different processor allocation policies that are designed to reduce mean response time when compared with an equipartition policy. In Section 2 we discuss related work. This is followed by descriptions of the job, workload and system models, in Section 3. In Section 4 we describe a generalization of a family of processor allocation policies. We then use this generalization to explore a spectrum of policies that make processor allocation decisions based on the work a job executes (in Section 5), the efficiency with which jobs execute (in Section 6) and the knee of the execution time – efficiency profile (in Section 7). The results of these experiments show that while each approach is capable of reducing mean response time, the reductions are either relatively small or limited to a relatively small subset of possible workloads. In Section 8 we use the insights gained from these experiments to propose and evaluate a new strategy that incorporates

characteristics of both a job's work and its efficiency. Our evaluation of this technique shows that it can be used to obtain significantly lower mean response times than equipartition under a variety of workloads. We present our conclusions in Section 9.

2 Related Work

A topic of interest and some debate in recent studies has been how to use job characteristics in multiprocessor scheduling algorithms and whether or not their use offers significant benefits (mainly, reductions in mean response time).

An algorithm that has been derived from a technique called Process Control [33, 13], and has been widely studied and extensively compared with other processor allocation policies [36, 19, 18, 22, 5] is called equipartition. The equipartition algorithm simply allocates an equal number of processors to each job in the system by dynamically reallocating processors whenever jobs arrive or depart. It is a very practical algorithm in the sense that it can be easily implemented since no job characteristics are required for making allocation decisions. Moreover, it is robust in that it provides acceptable mean response times over a wide variety of workloads. Studies by some researchers have concluded that a desirable property of a good scheduler is that it share processing power equally among all jobs in the system [19, 18]. It is our hypothesis that a good scheduler should be able to produce mean response times that are significantly lower than equipartition, if given information about the execution characteristics of jobs.

Majumdar, Eager, and Bunt [20] compare policies having accurate knowledge or an estimate of a job's cumulative service demand with policies having no such information. They report that policies using knowledge of service demands improve mean response times significantly over Round Robin and First Come First Served (policies that have no information about service demand). They make no comparison with equipartition since it did not exist at the time.

In order to analytically compare the performance of equipartition with optimal algorithms Brecht [3] examines a number of scheduling algorithms in restricted environments. By assuming that there are no new job arrivals, that all jobs execute with perfect efficiency and considering a batch scheduling environment (i.e., all jobs are available to be scheduled at time zero), equipartition is shown to produce mean response times that are guaranteed to be within a factor of two of the optimal algorithm (LWF). Also, when new arrivals are considered (i.e., no longer using batch scheduling) the mean response time of equipartition can (under pessimal workloads) be a factor of N times (where N is the number of jobs executed) worse than the optimal algorithm Least Remaining Work First (LRWF).

When developing and comparing static scheduling algorithms a number of researchers have considered characteristics related to the efficiency of a job's execution such as a job's average, minimum, maximum and variation in parallelism, the knee of the execution time – efficiency profile, and the processor working set [8, 31, 11, 21]. Eager, Zahorjan, and Lazowska [8] suggest that the number of processors to allocate to each job in a multiprogrammed environment could be made based on the point at which the ratio of efficiency to execution time, $E(p_i)/T(p_i)$, is maximized (p_i is the number of processors allocated to job J_i, $E(p_i)$ is the job's efficiency and $T(p_i)$ is its execution time, when allocated p_i processors). Ghosal,

Serazzi, and Tripathi [11] introduce another characterization of a parallel program called the processor working set *(pws)*. *Pws* has been defined as the minimum number of processors that maximizes the speedup per unit of cost associated with the allocation of a processor. The *pws* is also shown to be equivalent to the knee for a linear cost function. They study a number of static scheduling algorithms that use information about a job's *pws* and show that they perform well under a varying system load. Majumdar, Eager and Bunt [21] also report that in a static scheduling environment, allocating a number of processors near the knee was effective in producing low mean response times over a broad range of system loads.

Chiang, Mansharamani, and Vernon [5] study a number of static (run-to-completion) algorithms that use information about a job's execution rate characteristics (average parallelism and *pws*). They assert that policies using 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 (because jobs with long execution times are allocated too many processors).

Sevcik [31] finds that static policies that consider 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. Sevcik [32] also identifies a number of useful parameters for characterizing parallel applications and examines the problem of statically allocating processors to a batch of parallel applications given *a priori* knowledge of these parameters.

Zahorjan and McCann [36] and McCann, Vaswani, and Zahorjan [22] use simple information about a 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 repartitioning processors strictly upon job arrivals and departures [33, 13]. The advantage of this dynamic scheduling policy is that it may adjust to a job's changing demands for processors over time.

Recent studies examine scheduling algorithms that consider another important characteristic, the memory requirements of jobs [37, 4, 29, 23, 30, 1, 27, 28]. These studies identify memory (as well as processors) as being a critical resource for the effective execution of parallel programs and devise and investigate scheduling policies that ensure minimum processor allocations for jobs in order to secure sufficient memory resources. Our work does not explicitly model a job's memory requirements but rather relies on the execution signature of a job to implicitly indicate its memory requirements.

3 The Job, Workload, and System Models

We intentionally adopt a fairly high-level approach to the scheduling problem by using a relatively simple model of jobs, workloads, and the system. This is done in order to gain a first-order understanding of how processing power should be allocated to jobs in order to minimize mean response time. These models are used in this paper to simulate a number of scheduling policies executing under a variety of workloads.

3.1 The Job Model

In a uniprogrammed multiprocessor environment the execution time of a parallel program is mainly influenced the amount of basic work to be executed by the job. However, the execution time is also prolonged by a number of overheads incurred through the parallel execution of the job. These overheads include: the degree to which the work can be divided and balanced among multiple processors, the amount and type of communication among the cooperating processes, the amount and type of synchronization, as well as the costs incurred when creating and executing multiple processes on different processors. These different overheads can all be considered to be characteristics of the job being executed and can be modelled as inefficiencies in the ability of parallel programs to fully utilize the assigned processors. We use an execution rate function to model these inefficiencies.

In our job model two parameters are used to characterize a job, J_i.

1. W_i — is the amount of basic work executed. This corresponds to the execution time required to execute the program serially (i.e., the service demand).

2. β_i — is a parameter to an execution rate function and is used to characterize the rate at which the work, W_i, is executed when allocated a specified number of processors, p_i. This models the efficiency of the parallel job.

A number of models of parallel system and parallel program performance have been proposed and studied [10, 35, 7, 32]. We use the following execution rate function, used in a number of previous studies [5, 23, 30], which has been derived from an execution rate function (also called an *execution signature*) proposed by Dowdy [7]:

$$F \;=\; S(p_i) \;=\; \frac{(1 \;+\; \beta_i)\; p_i}{\beta_i \;+\; p_i}.$$

In this equation $S(p_i)$ is the speedup obtained when the job is executed on p_i processors and β_i is the parameter that is used to determine the efficiency of the job. If the number of processors allocated to a job, p_i, is fixed for the duration of the job's execution its execution time is therefore:

$$T(p_i) \;=\; \frac{W_i\;(\beta_i \;+\; p_i)}{(1 \;+\; \beta_i)\; p_i}.$$

When working with the execution rate function, F, we found it difficult to think in terms of β_i values for the execution rate. For example, does $\beta_i = 300$ imply that the job attains good speedup? The answer depends on the number of processors in the system, P. Therefore, we use a term called *effective efficiency*, ϵ_i, which expresses the speedup attained as a percentage of the number of processors used if all P processors would be allocated to the job. Therefore, $\epsilon_i = 90\%$ means that if the job was executed in isolation using all of the processors its speedup would be 90% of P. Alternately, $E(P) = 0.90$.

$$\epsilon_i \;=\; \frac{100\; S(P)}{P} \;=\; \frac{100\;(1 \;+\; \beta_i)}{(P + \beta_i)}, \text{ and given } \;\epsilon_i, \quad \beta_i \;=\; \frac{P\;\epsilon_i \;-\; 100}{100 \;-\; \epsilon_i},$$

$$\text{assuming} \quad P \;>\; 1 \text{ and } \frac{100}{P} \;\leq\; \epsilon_i \;\leq\; 100.$$

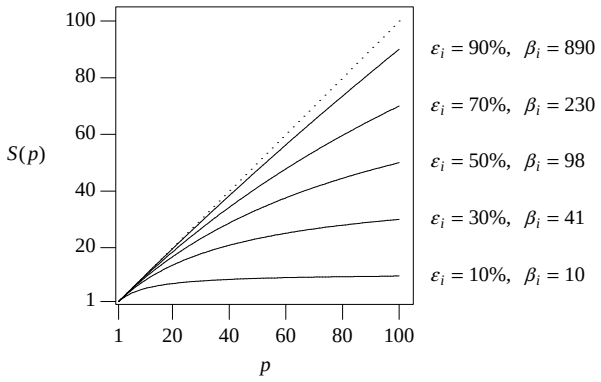

Figure 1: Execution rate function, speedup ($S(p)$) versus the number of processors (p) for various β_i

Note that if $\epsilon_i = 100$, $\beta_i = \infty$ and that a lower bound on ϵ_i is $\frac{100}{P}$, since the parameter β_i, is bounded by zero.

Using ϵ_i (rather than β_i) we gain an intuitive feeling for possible values of, and perhaps even representative distributions of, ϵ_i. Therefore, we can think of and perform experiments in terms of ϵ_i values while the simulator internally uses the corresponding β_i values. Figure 1 plots the execution rate function using a number of ϵ_i values (corresponding β_i values are also shown), assuming a system of 100 processors.

We recognize that this function does not accurately model the efficiency of all parallel applications because some applications may limit their parallelism (since they are unable to support P threads of execution efficiently) and because some applications will actually experience decreases in speedup when too many processors are allocated (due to increases in communication overhead). However, we chose not to use such models in this paper, because it could produce what might be considered an unfair bias against the equipartition policy. Since workloads can be easily generated on which the equipartition algorithm will perform poorly (because it will unknowingly allocate too many processors to jobs), developing algorithms that use job characteristics to improve mean response times would be trivial. Additionally, any algorithms developed using the model used here will only produce greater advantages when used with models of efficiency that consider jobs with limited parallelism and jobs that experience "slowdown" when too many processors are allocated. Note that more detailed and accurate models of efficiency, including a model that addresses both of these issues, are examined by Guha [12].

3.2 The Workload Model

We assume that the work a job has to execute, W_i, is drawn from a Hypergeometric distribution with mean $\bar{W} = 1000$ and coefficient of variation C_W (1, 5, and 30 are considered). This is consistent with variations in service demands used in previous studies [5, 26] and those observed at one supercomputer installation [9]. We also assume that there is no correlation

between the amount of work a job executes and the efficiency with which it is executed. Although this might not be true for all applications, we want to separately examine the effects of service demand and efficiency on the mean response time obtained with different allocation policies. This is not possible if efficiency and work are correlated.

We model the fact that jobs execute with different efficiency by using the execution rate function, F, for all jobs and choosing ϵ_i uniformly between ϵ_{low} and ϵ_{high}. This distribution is similar to that used in previous studies [23, 30] except that we ensure that the distribution is uniformly distributed in ϵ_i rather than β_i. We believe that this is what was actually intended in the previous studies.

Each workload executes M jobs, whose arrival follows a Poisson distribution. Each experiment is repeated a number of times using different random seeds in order to compute confidence intervals. The number of jobs and repetitions used for each experiment was chosen in order to achieve 90% confidence intervals that are within 5% of the mean. We use the bootstrap method for computing confidence intervals since it is robust for small numbers of repetitions and for non-normal distributions of observed means [34].

3.3 The System Model

In order to determine how different allocations of processing power affect the mean response time we assume that allocation decisions and processor reallocations can be performed instantaneously with negligible overhead. This is not unreasonable since we are only comparing dynamic scheduling algorithms (i.e., we are not comparing with static policies). Furthermore, experimental studies have demonstrated that although overhead is required to reallocate processors, the overhead does not significantly impact dynamic scheduling algorithms [25, 24]. For the purposes of simplifying the implementation of our simulator, we also assume that the number of processors allocated to a job may be fractional. Such an assumption does not affect the qualitative results of this work. As well, the effects of this assumption on the quantitative results are insignificant, since a large number of processors are used. The simulation results reported in this paper all assume a multiprocessor system containing $P = 100$ processors. For implementation purposes, the algorithms studied here could be easily modified to use an integral number of processors.

4 The Adaptive Algorithms

In order to explore different methods of allocating processing power to jobs we employ a slightly modified version of the generalization of processor allocation policies proposed by Brecht [3]. In this generalization, X_i is the characteristic of job J_i being used to make the allocation decision, 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. The generalization is defined as:

$$p_i = \frac{P\ {X_i}^{\alpha}}{\sum_{j=1}^{N} {X_j}^{\alpha}}\ .$$

Different values of α represent various points on a spectrum of processor allocation strategies. The α chosen acts as both a means of selecting the degree of processor sharing and as a control on which jobs should be given larger portions of the processors. Different values of X_i can be used to make allocation decisions based on different job characteristics. For example, using $X_i = W_i$, we have policies that base allocation decisions on work. In this case 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). Specific values of α worth noting (when using $X_i = W_i$) are:

1. $\alpha = 1$, which allocates processors in direct proportion to the work being executed by each application [3].

2. $\alpha = 0.5$, which allocates processors in proportion to the square root of the work being executed by each application. This policy has been examined by Sevcik [32] and Brecht [3] in the context of static scheduling algorithms.

3. $\alpha = 0$, which allocates processors equally among all jobs (i.e., independently of the work being executed by each job). This is the much studied equipartition policy [33, 19, 36, 5].

We consider dynamic scheduling policies that allocate or reallocate processors at job arrival and departures (if required). The scheduling algorithm limits the number of jobs active at any time, N, to be less than or equal to the number of processors, P.

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 according to

$$p_i \quad = \quad \frac{P \; {X_i}^{\alpha}}{\sum_{\mathrm{j}=1}^{N} \; {X_j}^{\alpha}} \; .$$

  }
  else add J_i to the inactive queue

- Upon the departure of job J_i :
  if the inactive queue is not empty {
  move a job from the inactive queue to the active list
  }
  now repartition processors according to

$$p_i \quad = \quad \frac{P \; {X_i}^{\alpha}}{\sum_{\mathrm{j}=1}^{N} \; {X_j}^{\alpha}} \; .$$

Under this scheme at most P jobs can be active at one time. Note, however, that although a job is considered active (under this definition) it may be allocated zero processors by the partitioning scheme. Once a job is added to the active list it is never moved back into the inactive queue. Obvious variations on this algorithm exist. However, during our simulated experiments we found that the inactive queue was almost always empty. Therefore, we have not yet evaluated different techniques for maintaining the active list and the inactive queue.

5 Considering Work

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). This is done by 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 equipartition (i.e., $\alpha = 0$). Since jobs execute with perfect efficiency we use $X_i = W_i$ in order to determine processor allocations and examine a range of α values.

The results shown in Figure 2 have been obtained by simulating a system with $P = 100$ processors, using a mean work requirement of $\bar{W} = 1000$, and the coefficient of variation of work being $C_W = 1$. The graph plots the mean response time as a function of the scheduling policy (α value) and shows results for loads (mean processor utilizations) of 30%, 50%, 70%, and 90%.

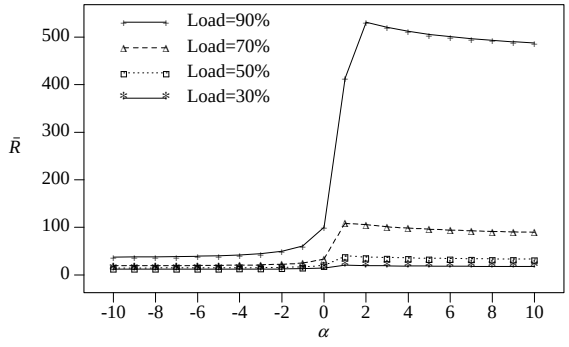

Figure 2: Mean response time vs. scheduling algorithm (α); $C_W = 1$, perfectly efficient jobs

We first note, as expected, that the mean response times, $\bar{R}$, obtained with positive values of α are very large when compared with values of α less than or equal to zero. This is because positive values of α will assign a larger portion of processors to larger jobs (i.e., jobs

with larger W_i). 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. We include positive α values in this experiment mainly to point out that MRWF is not pessimal. We see that in this experiment, with a load of 90%, $\bar{R}$ is maximized (there is a peak in the curve) when $\alpha = 2$. This is because the mean response time of a MRWF first policy can be increased by introducing processor sharing among the largest jobs, thus increasing the mean response time.

We also note that the difference in mean response time across different scheduling policies (α values) decreases as the load decreases. This is because as the load gets lighter the degree of multiprogramming decreases. 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).

In Figure 3 we more closely examine the range $-10 \leq \alpha \leq 0$. This graph has been produced by running experiments using a load of 90% and increasing the coefficient of variation of work. We consider $C_W = 1, 5,$ and 30. In each case, as α increases in the negative direction the mean response time decreases. This is not surprising since (as mentioned previously) all jobs execute with perfect efficiency and a Least Remaining Work First (LRWF) policy is optimal under these conditions [32, 3]. As well, as the coefficient of variation increases so does the difference between $\alpha = 0$ and $\alpha = -10$, demonstrating the increased benefits of knowing and appropriately using the job characteristic W_i. We also point out that the shape of the curves becomes sharper with increases in C_W. This is because with a small variation in W_i, higher values of α are required in order to approximate a LRWF policy, while with larger variations in W_i smaller values of α can be used to approximate LRWF.

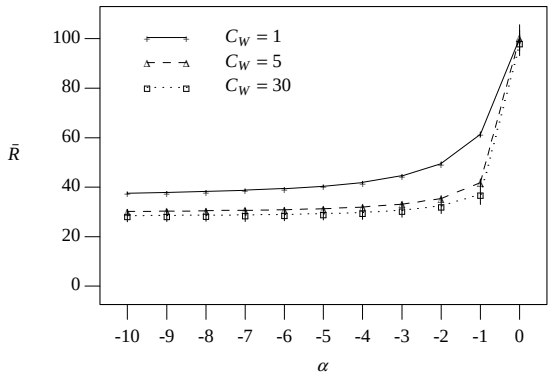

Figure 3: Mean response time vs. scheduling algorithm (α); load=90%, perfectly efficient jobs

In Table 1 we provide a rough quantification of the performance gains that can be achieved by knowing and utilizing W_i when jobs are perfectly efficient and the load is 90%. This table contains one column for the mean response times and 90% confidence intervals for the two policies $\alpha = 0$ and $\alpha = -10$, as well as one column showing the percentage improvement that is obtained by using $\alpha = -10$ instead of $\alpha = 0$. It shows that for $C_W = 1$ an improvement

of roughly 64% can be expected and that the improvements likely increase with C_W (note the larger confidence intervals for larger C_W).

This rough quantification is of interest because it provides an approximate bound on the reduction in mean response time that can be obtained by using W_i (or β_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), thus reducing the difference between an equipartition and an optimal policy. As well, previous analytic comparisons between LRWF and equipartition policies [3] show that the reduction in mean response time obtained when all jobs arrive simultaneously can be as large as 50% and that the difference can grow with the number of jobs (i.e., it is not competitive) when new arrivals are considered. Since this previous result assumes that arrival times and job sizes are designed to demonstrate maximum differences between the two policies, our comparison is important because it provides more reasonable bounds on the performance improvements that might be expected.

Table 1: Comparing mean response times for $\alpha = 0$ and $\alpha = -10$ for different C_W; load = 90%

C_W	R ($\alpha = 0$)	R ($\alpha = -10$)	%Impr
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

These same experiments have also been conducted under the assumption that the work, W_i, is not known *a priori* (see [12] for details). By keeping track of each job's accumulated execution time and using that as an estimate of a job's remaining work, influenced by studies performed by Leland and Ott on uniprocessor systems [17] (a portion of this study was recently confirmed for modern workloads by Harchol–Balter and Downey [14]), the response time is significantly reduced when compared with equipartition, although the reductions are not as large as when W_i is known.

We now explore the consequences of using only information about a job's remaining work, W_i, if jobs do not execute with perfect efficiency. In this experiment we again choose W_i with a mean of $\bar{W} = 1000$ and $C_W = 1, 5,$ and 30. Now instead of having all jobs execute with perfect efficiency, each job executes work at the rate defined by the execution rate function and the parameter β_i (ϵ_i). In the following experiments we generate ϵ_i values uniformly between ϵ_{low} and ϵ_{high} and consider different workloads with different ϵ_{low} and ϵ_{high} values.

The graph on the left side of Figure 4 shows that for workloads with relatively low average efficiency ($\bar{\epsilon} = 50\%$), using only W_i to make scheduling decisions leads to increased mean response times (versus equipartitioning processors, $\alpha = 0$). This is because these policies ($-10 \leq \alpha \leq 1$) are allocating larger portions of the processors to small jobs even though they may not be capable of utilizing them effectively, and are therefore essentially wasting processing power. However, the graph on the right side of Figure 4 shows that for workloads with high average efficiency ($\bar{\epsilon} = 87\%$), using only W_i to make scheduling decisions

can decrease 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 of allocating large portions of the processors to these jobs (reduced response time) outweigh the costs (under-utilized processors).

Figure 4: Mean response time vs. scheduling algorithm (α); load=90%, jobs are not perfectly efficient

6 Considering Efficiency

As we have seen in the previous section, if jobs do not execute with high average efficiency, the mean response times obtained using scheduling algorithms that make allocation decisions based only on W_i are quite high when compared with a scheduling algorithm that makes no use of job characteristics. This is because jobs are being allocated processors that they are not able to use effectively. In this section, we consider scheduling policies which only use information about a job's efficiency.

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). Since under our workload model the efficiency of each job is characterized by β_i, we set $X_i = \beta_i$ and use:

$$p_i = \frac{P \ {\beta_i}^{\alpha}}{\sum_{\mathrm{j}=1}^{N} \ {\beta_j}^{\alpha}} .$$

The graph in Figure 5 plots the mean response time against different α values (representing different scheduling policies) and shows results for $C_W = 1$ and 5. ($C_W = 30$ yields such a pronounced v-shape that it is not included because it hides the distinction between $C_W = 1$ and $C_W = 5$.) This graph shows that as α increases, in both positive and negative directions, the mean response time also increases. These experiments were conducted using $\epsilon_{\mathrm{low}} = 50\%$ and $\epsilon_{\mathrm{high}} = 99\%$ and a low target load (30%). Even under this light load we find that with $C_W = 5$ performance degrades significantly for $\alpha < 0$ and $\alpha > 1$. We use this light load to demonstrate the general shape of the response time curves since at higher loads

the differences in mean response times for the various algorithms change drastically (i.e., the v-shape becomes more pronounced more quickly). Under this scheme, positive α values imply that more processors are given to jobs with higher efficiency. Significantly increasing the number of processors allocated to efficient jobs, while improving the degree to which processors are efficiently utilized, 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.

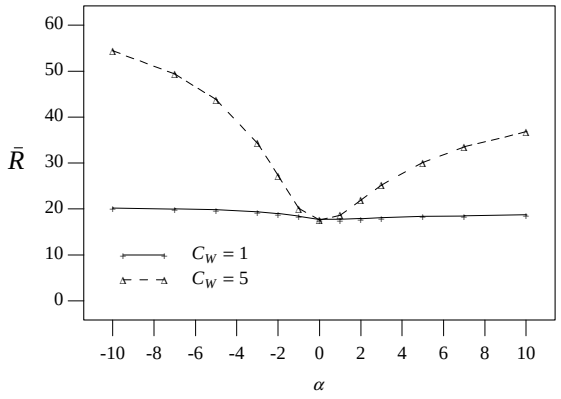

Figure 5: Mean response time versus scheduling policy (α); $\epsilon_{\text{low}} = 50\%, \epsilon_{\text{high}} = 99\%, load = 30$

Note that the mean response time obtained when $\alpha = 1$ is relatively close to that obtained when $\alpha = 0$. In fact in most cases it yields mean response times that are statistically equal to those obtained with $\alpha = 0$. Because the $\alpha = 0$ and $\alpha = 1$ values are so similar we now more closely examine these algorithms and also consider another point on the spectrum of algorithms ($\alpha = 0.5$).

Table 2 contains the mean response times and 90% confidence intervals for $\alpha = 0$, 0.5, and 1.0 for workloads with $C_W = 1$, 5, and 30 and ranges of effective efficiency of 1-50%, 1-99%, and 50-99%. The column labelled "%Impr" gives the percent by which the mean response time of $\alpha = 0$ is reduced if $\alpha = 0.5$ is used. (A negative value indicates the mean response time has increased, however, these negative differences can not be considered to be statistically significant.) Each experiment was run using an arrival rate that produces observed loads of approximately 90%. Interestingly, when average efficiency is relatively low, the algorithm obtained when using $\alpha = 0.5$ produces mean response times that are lower than those obtained when $\alpha = 0$ or $\alpha = 1$.

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 determines the efficiency of each job we expect that the improved mean response times obtained for $\alpha = 0.5$ are more indicative of how to schedule using β_i than how to schedule using other characteristics of a job's efficiency. (For example, we expect that if ϵ_i was used $\alpha = 1$ might perform better than $\alpha = 0.5$.) However, these results do demonstrate that a characteristic of each job's efficiency, β_i in this case, can be used to make processor allocation decisions and produce

Table 2: Comparison of mean response times for different policies based on efficiency

$\epsilon_{\text{low}} - \epsilon_{\text{high}}$	C_W	R ($\alpha = 0.0$)	R ($\alpha = 0.5$)	R ($\alpha = 1.0$)	%Impr
01-50%	1	98.9 +/- 0.1	95.3 +/- 0.1	97.0 +/- 0.1	3.64
	5	98.8 +/- 0.3	95.4 +/- 0.3	97.0 +/- 0.7	3.44
	30	99.5 +/- 1.6	96.1 +/- 1.5	98.1 +/- 1.5	3.42
01-99%	1	62.4 +/- 0.1	58.3 +/- 0.1	60.6 +/- 0.1	6.57
	5	62.3 +/- 0.3	58.5 +/- 0.2	62.4 +/- 0.2	6.10
	30	62.9 +/- 1.4	59.3 +/- 1.2	66.5 +/- 1.3	5.72
50-99%	1	37.6 +/- 0.1	37.0 +/- 0.1	37.6 +/- 0.1	1.60
	5	37.4 +/- 0.2	38.0 +/- 0.2	42.6 +/- 0.2	-1.60
	30	36.6 +/- 1.0	37.3 +/- 1.0	44.5 +/- 1.2	-1.91

mean response times that are statistically lower than an equipartition strategy. However, these differences are quite small.

7 Considering the Knee

Eager, Zahorjan, and Lazowska [8] point out that where speedup is a measure of the "benefit" of using some number of processors for the parallel execution of a job, the efficiency is a measure of the "cost" of using those processors. The point at which the ratio of efficiency to execution time, $E(p_i)/T(p_i)$, is maximized is called the knee of the execution time – efficiency profile and may be useful in determining effective processor allocations in multiprogrammed environments. Given our function that describes the execution rate of parallel jobs, the knee can be easily derived and is $k_i = \beta_i$. Chiang, Mansharamani, and Vernon [5] also point out that the knee, k_i, for job J_i can be shown to be equal to β_i.

In the previous section we based processor allocations on the efficiency of jobs and used β_i to characterize the efficiency of an application. Since the knee of the execution time – efficiency profile for the execution rate function we have chosen is β_i, we have already, indirectly, examined the affects of using the knee, k_i, to make scheduling decisions (in the previous section). To reiterate, the problem with such an approach is that while small benefits can be obtained by utilizing the knee, those benefits are limited because the approach does not directly consider the amount of work executed by each job.

8 Combining Work and Efficiency

We now propose a new scheduling strategy that is designed to take into account both the efficiency of a job and its work. This algorithm 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 are unable to significantly reduce mean response times over the wide range of potential workloads considered. While it may be desirable to give less processing power to jobs that execute large amounts of work, in order to prevent them from delaying jobs that execute smaller amounts of work, it is also necessary to consider the efficiency with which each job can utilize the allocated processors.

We use a two-phased algorithm to determine the number of processors, p_i, to allocate to each job J_i. Our strategy is to maintain a sorted list of jobs in the system (sorted by increasing W_i). This ordered list is used to prevent short jobs from being stuck behind large jobs. 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 they execute. During this first phase jobs are assigned processors 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 which assigns the remaining processors to activated jobs. A number of different assignment policies are possible for the first and second phases. In this paper we focus our investigation 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 — assuming the average efficiency is not unreasonably low). The policy used for the second phase divides the remaining processors equally among the active jobs (since this is a safe approach).

An obvious strategy to use in the first phase is to assign processors to each job according to the knee of the execution time – efficiency profile ($f_i = k_i = \beta_i$). Unfortunately, this approach has an important drawback: β_i is greater than P for jobs with relatively low efficiency. In fact, for $P = 100$ processors, $\beta_i \geq P$ for all $\epsilon_i \geq 50.5\%$. This may be due to the execution rate function used. 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 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 $\leq P$. In future work we plan to consider alternative execution rate functions and to investigate the importance of appropriate mappings. One way to avoid this problem is to simply assign $min(\beta_i, P)$ processors, rather than β_i. However, this approach will allocate all P processors to any job for which $\epsilon_i \geq 50.5\%$. As expected, experiments showed that this approach did not perform well for workloads with high average efficiency, $\bar{\epsilon}$. For example, using this policy to execute a workload with $C_W = 1$, ϵ_i = 50-99%, and a load of 90% yields a mean response time that is 27% higher than obtained using equipartition. Therefore, we require a function that more evenly maps the range, β_i, to the domain, f_i.

We start by considering an alternate, simple and obvious mapping, $f_i = e_i$. The advantage of this 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). We call the general form of our algorithm $W\&E$ (Work and Efficiency). The form that uses $f_i = k_i = \beta_i$ we call $W\&\beta_i$, and the form that uses $f_i = \epsilon_i$ we call $W\&\epsilon_i$.

We have conducted a series of experiments to compare $W\&\epsilon_i$ with equipartition. The

results of these experiments are shown in Figure 6. These experiments were conducted using arrival rates that produced loads of approximately 30% and 90% respectively. An arrival rate was determined using $W\&\epsilon_i$ and was then used for equipartition. Each bar shown in these graphs represents the mean response time of $W\&\epsilon_i$, normalized with respect to the mean response time of equipartition (represented by the dotted line, EQUI). The workload for each experiment is specified below the x-axis by displaying the parameters C_W and the range $\epsilon_{\text{low}} - \epsilon_{\text{high}}$.

As expected under low loads (30%), the reductions in mean response time obtained by using $W\&\epsilon_i$ 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. Under this load equipartition is relatively robust since the maximum observed improvement is about 15%. As seen by the results obtained when a load of 90% is used, if the degree of multiprogramming is increased, the size of the reductions in mean response times increases.

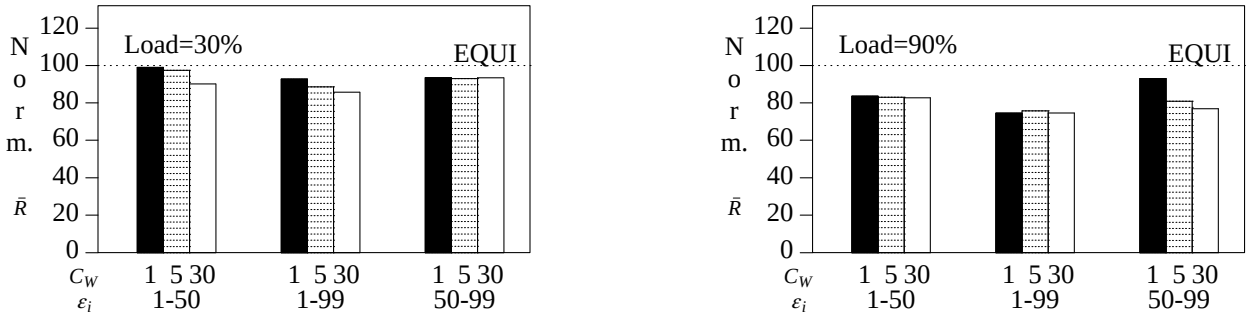

Figure 6: Mean response time of $W\&\epsilon_i$ for different workloads, normalized with respect to EQUI

In the next series of experiments (shown in Figure 7) we consider additional ranges of efficiency values by also considering ϵ_i = 75-99% and 99-99%. These values are considered to emphasize that as the average efficiency of the jobs increases, the behaviour of our $W\&\epsilon_i$ policy approaches that of an optimal algorithm. This can be seen by comparing the improvements obtained in these graphs with those obtained in Section 4. These ranges demonstrate the substantial reductions in mean response time that are possible when the average efficiency of the jobs is high and illustrate how the size of these reductions grows with increases in average efficiency.

Additionally we study the affect that the variation in job efficiencies, C_{ϵ_i}, 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 our distribution and compare the ranges 1-99%, 30-70%, and 50-50% (which produce C_{ϵ_i} of 0.57, 0.23, and 0.0 respectively). The results in Figure 7 show that as the variation in efficiencies decreases, the mean response time obtained using $W\&\epsilon_i$ increases substantially when compared with equipartition.

Next we conducted a series of experiments in order to determine if this increase is because

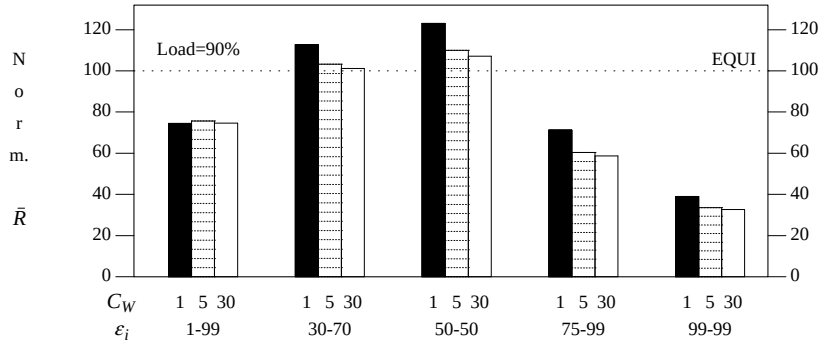

Figure 7: Mean response time of $W\&\epsilon_i$ for different workloads, normalized with respect to EQUI

the general method, $W\&E$, is unable to handle workloads with low variation in efficiency or because the number of processors allocated during the first phase could be improved. We started by experimentally testing whether or not a value of f_i could be found that improved the mean response time 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 found that for low and high efficiency ranges (e.g., 1-20% and 80-99%) allocating ϵ_i processors produced response times lower than for equipartition. Using these results we developed a simple assignment policy that we hoped would uniformly produce mean response times equal to or lower than equipartition.

This new policy (or mapping) was derived from our observations that $f_i = \epsilon_i$ performed reasonably well for $1\% \leq \epsilon_i \leq 20\%$ and for $80\% \leq \epsilon_i \leq 99\%$ and that $f_i = 30$ performed well for the workload with the efficiency range 50-50%. Therefore, we (naively) chose a piecewise linear function as the basis for a new variation of our algorithm. This variation is called $W\&F(\epsilon_i)$ and is described below. The different forms of the $W\&E$ algorithm are shown in Figure 8. Algorithmically this function is described (for $P = 100$) as follows:

$$f_i = \begin{cases} \epsilon_i & : \; if \; 1\% \leq \epsilon_i \leq 20\% \\ (\epsilon_i - 20)\,(10/30) + 20 & : \; if \; 20\% \leq \epsilon_i \leq 50\% \\ (\epsilon_i - 50)\,(50/30) + 30 & : \; if \; 50\% \leq \epsilon_i \leq 80\% \\ \epsilon_i & : \; if \; 80\% \leq \epsilon_i \leq 99\% \end{cases}$$

As shown in Figure 8, when $\epsilon_i = 50\%$, there is a large difference between the number of processors allocated using $W\&\beta_i$ (i.e., using the knee) and using either $W\&\epsilon_i$ or $W\&F(\epsilon_i)$. This seems to indicate that under these job and workload models, processor allocations in a dynamic scheduling environment might be better made at a point much lower than the knee.

The results of using this policy, $W\&F(\epsilon_i)$, for a wide variety of workloads are shown in Figure 9. These experiments show that this policy does produce mean response times that are at least as good as, and are in many cases substantially better than, those obtained with the equipartition policy. This policy therefore performs better across the variety of workloads

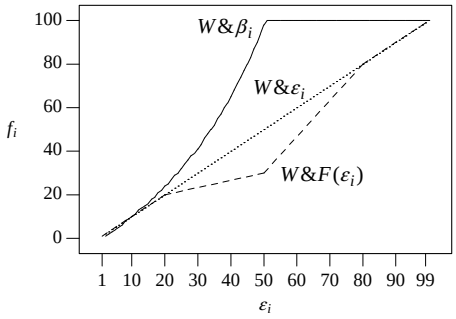

Figure 8: Different allocation policies considered for $W\&E$

considered than the $W\&\beta_i$, the $W\&\epsilon_i$ 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 (especially if a wide variety of workloads are possible). (We plan to further investigate other factors, such as the load, the execution rate function, and the distribution used to generate the efficiency for each job.)

In the case when a load of 90% is used, the largest improvement seen is approximately 70%. A general trend that can be seen by examining the ranges 1-50%, 50-99%, 75-99% and 99-99% in this graph, is that the size of the reductions obtained by using $W\&F(\epsilon_i)$ 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. We also point out again 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.

Although the $W\&F(\epsilon_i)$ algorithm can very likely be improved, since it was chosen naively, these experiments demonstrate that the $W\&E$ strategy is an attractive approach to using job characteristics to make processor allocation decisions in a dynamic scheduling environment. If the number of processors to assign 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 the importance of using job characteristics correctly as well as the need for allocation policies that effectively use characteristics of both work and efficiency.

9 Conclusions and Future Research

In this paper we examine the problem of processor allocation in multiprogrammed multiprocessors. We intentionally use relatively simple models of a parallel program's execution, the workload, and the system, in order to gain a first-order understanding of how job characteristics might be used to make better processor allocation decisions. The experiments include

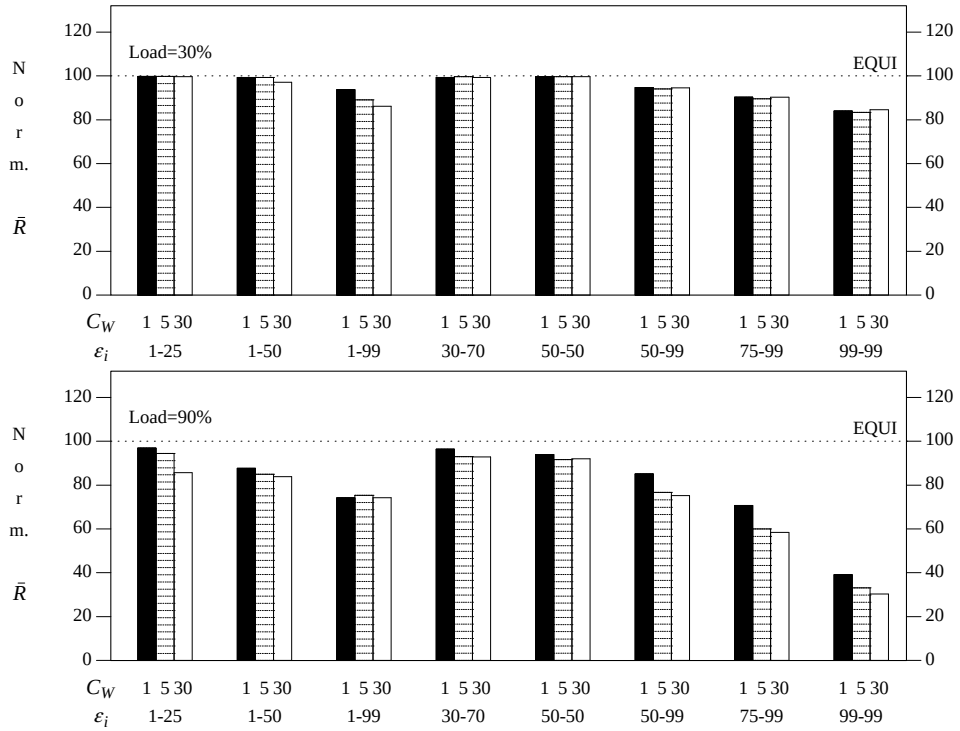

Figure 9: Mean response time of $W\&F(\epsilon_i)$ for different workloads, normalized with respect to EQUI

a wide range of workload characteristics, including large variations in the amount of work jobs execute as well as jobs that execute work at different rates.

Using our job, system, and workload models we make the following observations:

1. Policies that use only information about the amount of work each job executes do not reduce mean response time when compared with an equipartition policy unless the average efficiency of the jobs is relatively high. 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. Improvements are maximized and are substantial when all jobs are perfectly efficient.

2. Policies that use only information about the efficiency with which a job executes can provide small but statistically significant reductions in mean response time when compared with equipartition, if extreme partitioning policies are avoided. With the particular workload used in these experiments we found that using a value of $\alpha = 0.5$ yielded reductions of up to 14% in some instances. However, these policies are unable to produce further improvements because they fail to consider the amount of remaining work each job has to execute. At first this conclusion might appear to conflict with results from recent work by Nguyen, Vaswani, and Zahorjan [25]. However, their significant performance gains are obtained under workloads for which equipartition can naively allocate too many processors (thus actually increasing the execution time of applications). If similar workload models were used here we would see similarly large improvements.

3. Under the execution rate function used in this paper, policies that make processor partitioning decisions using only information about the knee of the execution time – efficiency profile of each job provide the same advantages and disadvantages of policies that consider only the efficiency of a job.

4. Policies that consider both a job's remaining work and a job's efficiency when making processor allocation decisions 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 average efficiency with which jobs execute their work, and the system load. Our new algorithm is based on using characteristics about the amount of work a job executes and the efficiency with which it can be executed. Although such information 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, practical scheduling algorithms.

5. We demonstrate that it is not desirable to blindly equipartition processors, and that considerable improvement in mean response time can be obtained by using information about a job's work and efficiency. However, the performance benefits obtained by using partitioning techniques based on job characteristics is likely bounded and 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.

While true evidence of the applicability of our results requires an experimental evaluation, we believe that the insights from our work can be used to guide the implementation of scheduling algorithms in production multiprogrammed multiprocessors. Nguyen, Vaswani and Zahorjan [25, 24] have shown experimentally that estimates of a job's efficiency can be obtained and used effectively at run-time, and that the cost of reallocating processor in response to changes in job and workload characteristics is not prohibitive. Leland and Ott [17] point out, and Harchol–Balter and Downey confirm their study using modern UNIX workloads [14], that under sequential UNIX workloads there is a strong correlation between the length of time a job has been executing and its expected remaining execution time. This observation is part of the inspiration behind the multilevel feedback queue schedulers currently used on many UNIX systems [2] and we expect that similar observations will hold for multiprocessor workloads.

These previous studies demonstrate that it is possible to obtain estimates of a job's efficiency and of its remaining work. They also provide evidence that these estimates are reasonably accurate and that the estimates can be applied efficiently. We believe that such estimates can be combined with the scheduling techniques outlined in our algorithms and that this combination can be used to reduce mean response time when compared with existing multiprogrammed multiprocessor scheduling algorithms.

9.1 Future Work

Now that we have a better understanding of the conditions under which characteristics of parallel programs can be used to reduce mean response time and a better grasp of the size of the reductions that can be expected, we plan to examine the importance of these job characteristics and the relative performance of the algorithms investigated here while considering different job models. (A modified model that includes limitations on a job's maximum parallelism as well as the maximum number of processors a job can use efficiently has already been considered [12].) We hope to more fully explore variations on our $W\&E$ algorithm and to try to determine the optimal allocation given the job, workload and system models used in this paper. We are also interested in trying to apply our $W\&E$ technique in a static scheduling environment.

10 Acknowledgments

We wish to thank Tom Fairgrieve for influential discussions related to the generalization of the allocation policies and Ken Sevcik for helping us to arrive at a proper formulation. Nian Gu and Minas Spetsakis provided help in solving various equations and with using Maple. Mary Vernon encouraged us to consider job models that also have limited degrees of parallelism and whose speedup can decrease when allocated more processors (detailed results obtained using such models are presented in [12]). We wish to thank the anonymous referees for their

suggestions for improving the presentation of this paper. We gratefully acknowledge the efforts of Raul Flores who quickly and accurately converted this document from troff to latex format and Lucia Dow and Nian Gu who read and provided helpful comments on various drafts of this paper. Finally, both authors thank the Natural Sciences and Engineering Research Council (NSERC) for a grant which partially supported this research.

References

[1] G. Alverson, S. Kahan, R. Korry, C. McCann, and B. Smith. Scheduling on the Tera MTA. *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, 949:19–44, 1995.

[2] M. J. Bach. *The Design of the UNIX Operating System.* Prentice Hall Inc., 1986.

[3] T. Brecht. Multiprogrammed parallel application scheduling in NUMA multiprocessors. Ph.D. Thesis, Technical Report CSRI-303, University of Toronto, Toronto, Ontario, June 1994.

[4] T. B. Brecht. On the importance of parallel application placement in NUMA multiprocessors. In *Proceedings of the Fourth Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS IV)*, pages 1–18, San Diego, CA, September 1993.

[5] S. Chiang, R. K. Mansharamani, and M. K. Vernon. Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. In *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 33–44, Nashville, TN, May 1994.

[6] E. Coffman and L. Kleinrock. Feedback queueing models for time-shared systems. *Journal of the ACM*, 15(4):549–576, October 1968.

[7] L. Dowdy. On the partitioning of multiprocessor systems. In *Performance 1990: An Internationional Conference on the Performance of Computers and Computer Networks*, pages 99–129, March 1990.

[8] D. L. Eager, J. Zahorjan, and E. D. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3):408–423, March 1989.

[9] D. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, 949:337–360, 1995.

[10] H. P. Flatt and K. Kennedy. Performance of parallel processors. 12(1):1–20, 1989.

[11] D. Ghosal, G. Serazzi, and S. K. Tripathi. The processor working set and its use in scheduling multiprocessor systems. *IEEE Transactions on Software Engineering*, 17(5):443–453, May 1991.

[12] K. Guha. Using parallel program characteristics in dynamic multiprocessor allocation policies. M.Sc. Thesis, Technical Report CS-95-03, York University, North York, Ontario, May 1995.

[13] A. Gupta, A. Tucker, and S. Urushibara. The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications. In *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 120–132, San Diego, CA, May 1991.

[14] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. In *Proceedings of the 1996 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 13–24, Philadelphia, PA, May 1996.

[15] J. L. Hellerstein. Achieving service rate objectives with decay usage scheduling. *IEEE Transactions on Software Engineering*, 19(8):812–825, August 1993.

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

[17] W. E. Leland and T. J. Ott. Load-balancing heuristics and process behavior. In *Proceedings of the 1986 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 54–69, Raleigh, NC, 1986.

[18] S. T. Leutenegger and R. D. Nelson. Analysis of spatial and temporal scheduling policies for semi-static and dynamic multiprocessor environments. Technical Report RC 17086 (No .75594), IBM Research Division, August, 1 1991.

[19] S. T. Leutenegger and M. K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 226–236, Boulder, CO, May 1990.

[20] S. Majumdar, D. Eager, and R. B. Bunt. Scheduling in multiprogrammed parallel systems. In *Proceedings of the 1988 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 104–113, May 1988.

[21] S. Majumdar, D. Eager, and R. B. Bunt. Characterisation of programs for scheduling in multiprogrammed parallel systems. *Performance Evaluation*, 13:109–130, 1991.

[22] C. McCann, R. Vaswani, and J. Zahorjan. A dynamic processor allocation policy for multiprogrammed, shared memory multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146–178, May 1993.

[23] C. McCann and J. Zahorjan. Scheduling memory constrained jobs on distributed memory parallel computers. In *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 208–219, Ottawa, ON, May 1995.

[24] T. Nguyen, R. Vaswani, and J. Zahorjan. Maximizing speedup through self-tuning of processor allocation. In *Proceedings of the 10th International Parallel Processing Symposium*, pages 463–468, Waikiki, HI, April 1996.

[25] T. Nguyen, R. Vaswani, and J. Zahorjan. Using runtime measured workload characteristics in parallel processor scheduling. In *Proceedings of the IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processor Scheduling*, pages 93–104, Waikiki, HI, April 1996.

[26] E. Parsons and K. Sevcik. Multiprocessor scheduling for high-variability service time distributions. *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, 949:127–145, 1995.

[27] E. Parsons and K. Sevcik. Coordinated allocation of memory and processors in multiprocessors. In *Proceedings of the 1996 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 57–67, Philadelphia, PA, May 1996.

[28] E. W. Parsons and K. C. Sevcik. Benefits of speedup knowledge in memory-constrained multiprocessor scheduling. In *Proceedings of the 1996 Conference on Performance Theory, Measurement and Evaluation of Computer and Communication Systems*, October 1996.

[29] V. G. J. Peris, M. S. Squillante, and V. K. Naik. Analysis of the impact of memory in distributed parallel processing systems. In *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 5–18, Nashville, TN, May 1994.

[30] S. Setia. The interaction between memory allocations and adaptive partitioning in message-passing multiprocessors,. *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, 949:146–164, 1995.

[31] K. C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. In *Proceedings of the 1989 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 171–180, 1989.

[32] K. C. Sevcik. Application scheduling and processor allocation in multiprogrammed multiprocessors. *Performance Evaluation*, 9(2-3):107–140, 1994.

[33] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 159–166, 1989.

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

[35] Chee-Shong Wu. Processor scheduling in multiprogrammed shared memory NUMA multiprocessors. M.Sc. Thesis, University of Toronto, Toronto, Ontario, October 1993.

[36] J. Zahorjan and C. McCann. Processor scheduling in shared memory multiprocessors. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 214–225, Boulder, CO, May 1990.

[37] S. Zhou and T. B. Brecht. Processor pool-based scheduling for large-scale NUMA multiprocessors. In *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 133–142, San Diego, CA, May 1991.