Multiprocessor Scheduling for High-Variability Service Time Distributions

Eric W. Parsons and Kenneth C. Sevcik

Computer Systems Research Institute
University of Toronto

{eparsons,kcs}@cs.toronto.edu

Abstract. Many disciplines have been proposed for scheduling and processor allocation in multiprogrammed multiprocessors for parallel processing. These have been, for the most part, designed and evaluated for workloads having relatively low variability in service demand. But with reports that variability in service demands at high performance computing centers can actually be quite high, these disciplines must be reevaluated. In this paper, we examine the performance of two well-known static scheduling disciplines, and propose preemptive versions of these that offer much better mean response times when the variability in service demand is high. We argue that, in systems in which dynamic repartitioning in applications is expensive or impossible, these preemptive disciplines are well suited for handling high variability in service demand.

1 Introduction

There have been numerous scheduling disciplines proposed for multiprogrammed multiprocessor systems, the evaluation of which has, for the most part, been based on workloads having a relatively low variability in the processing requirements of jobs. However, high performance computing centers have reported that variability in service demands can in fact be quite high. In a detailed study of the anticipated workload of its "Numerical Aerodynamic Simulation" (NAS) facility [NAS80], NASA specified a workload consisting of eight types of computational tasks with expected mean service requirements differing by as much as a factor 3500. Assuming that the service requirements within each class are exponentially distributed, the coefficient of variation of service times (C_d), which is the ratio of the standard deviation of service time to its mean, in the overall workload is 7.23. This high degree of variability is further supported by a recent workload characterization study of parallel applications at the same NAS facility [FN95]. In another study, Chiang, Mansharamani, and Vernon report that the coefficient of variation observed on a weekly basis on the CM-5 at the University of Wisconsin ranges from 2.5 to 6, with 40% of them being above 4 [CMV94]. They also report that some measurements from Cray YMP sites range from 30 to 70 [CMV94, Ver94].

In this paper, we consider two well-known static scheduling disciplines, showing how they behave under high variability in service demand, and propose ways in which they can be adapted to better handle this condition. These enhancements make no additional assumptions about the information available at a job's arrival, other than what is required by the original discipline. However, they do use preemption to effectively time-share sets of processors among jobs. Each job always executes on the same number of processors, as determined upon arrival, but is not necessarily run to completion when first activated.

Focusing on C_d in the range of 5 to 70, our goals are (1) to compare the performance of the existing disciplines and (2) to propose enhancements to these disciplines that make them perform better over this range of C_d. For comparison purposes, we also consider the performance of an ideal form of equipartitioning (IEQ) in which each job receives an equal share of the processors [ZM90]. This discipline is very different from the static disciplines in that a job's processor allocation changes at each job arrival or completion. If the overhead of adapting to the altered allocation is neglected (hence "ideal" equipartition), then IEQ is known to perform very well for high values of C_d.

The principle underlying the enhanced disciplines is based on the knowledge that, in uniprocessor scheduling, the variability in service demand plays a large role in determining the best scheduling discipline when exact knowledge of service demands is absent. For $C_d > 1$, a discipline that favours jobs that have the least acquired service can greatly reduce the mean response time (MRT) of the system. Our enhanced disciplines generalize the uniprocessor multilevel feedback (FB) approach to a multiprocessor setting.

The technique used to evaluate these disciplines is experimentation using a synthetic workload simulation. In studies such as this, one is usually required to use synthetic workloads because real workloads cannot be simulated efficiently enough and real systems with actual workloads are not available for experimentation. Also, useful analytic models are difficult to derive because the subtleties between various disciplines are difficult to model and because the workload model is quite complex.

The results in this paper are based on three synthetic workloads that differ in the amount of speedup attained by jobs. In the first workload, all jobs have near-perfect speedup. In this case, the results are consistent with what is known from the study of uniprocessor systems. As the coefficient of variation of service times increases, so does the mean response time for the run-to-completion disciplines. The enhanced disciplines have worse performance when $C_d < 1$, but provide improved performance as C_d increases beyond one. In the second workload, small jobs have very poor speedup, while large ones have relatively good speedup. With this workload, the enhanced disciplines are still superior, but to a lesser degree. Finally, we consider a workload derived from the NASA study mentioned above, with an intermediate speedup characterization, in order to show how the disciplines behave under what we believe is a more realistic workload. In this study, we used a speedup characterization that explicitly accounts for contention and overhead as a function of the number of processors. Thus, jobs which are allocated too many processors can experience a reduction in overall performance [Sev94]. Such a characterization is more realistic than those that simply consider the parallel and sequential components of a job.

The structure of the paper is as follows. In the next section, we survey scheduling disciplines that have previously been proposed and evaluated. Then, we describe in detail the disciplines from which we derive the new versions, and present our modifications. In Sec. 4, we specify system and workload models, and describe the simulation experiments upon which we base our conclusions. The results of the simulation experiments are presented and analyzed in Sec. 5, and conclusions are presented in Sec. 6.

2 Background

2.1 Uniprocessor Scheduling

As mentioned earlier, the performance of uniprocessor scheduling disciplines depends to a great extent on the distribution of service times. If $C_d < 1$, a first-come first-served (FCFS) discipline gives the best mean response time, since the job with the most acquired service is expected to be the closest to completion. On the other hand, if $C_d > 1$, then the multilevel feedback policy (FB) tends to perform best since the job with the least acquired service is the one that has the least expected remaining service time[1]. If C_d is close to one, or no information is available about C_d, then the round-robin discipline (RR) offers a good compromise as it yields a mean response time that is insensitive to the service time distribution.

Figure 1 illustrates how mean response times depend on the coefficient of variation of service times for FCFS, RR, and FB, along with two other disciplines that require advance knowledge of the service time of each job. Shortest Processing First (SPT) schedules the job with the smallest service time whenever the processor is free. The preemptive version, Shortest Remaining Processing Time (SRPT), always assigns the processor to the available job that is closest to completion. The value of service time knowledge can be seen for SPT and SRPT (particularly for $C_d < 2$) and the value of preemption can be seen for RR, FB, and SRPT (for $C_d > 2$).

2.2 Multiprocessor Scheduling

Multiprocessor scheduling algorithms can be classified according to (1) the amount of information they require about jobs or classes of jobs, and (2) the extent to which they preempt and reallocate processors among jobs.

The levels of information used by various algorithms can range from none, to some knowledge about the workload or classes of jobs, to knowledge of the characteristics of individual jobs, to full knowledge about all jobs. In this paper, we consider the third case where some job characteristics, such as the maximum parallelism, are known. In a realistic environment, such characteristics would not be known precisely, but understanding how to schedule with exact job-specific information is an important step in understanding how to schedule effectively when this information is only approximate.

The scheduling of threads of a job may be either *independent* or *coordinated*. In the independent case, there is a single (logical) queue of individual threads. Any free processor takes a thread off the queue and executes it either to completion or for a specified quantum. The queue may be ordered by arrival time, expected service required, number of threads in the job, or some other criterion.

In the coordinated case, processors are allocated to and perhaps preempted from jobs in groups. This approach renders it possible to exploit cache affinity and to make an appropriate processor allocation to each job in light of whatever is known about its characteristics.

[1] The performance of FB actually depends on the service time distribution. It has been shown that for some distributions (e.g., a 3-point distribution with high C_d), the performance of FB can be quite poor [Sch70]. For the hyperexponential class of distributions, as are used in this study, FB is markedly better than RR.

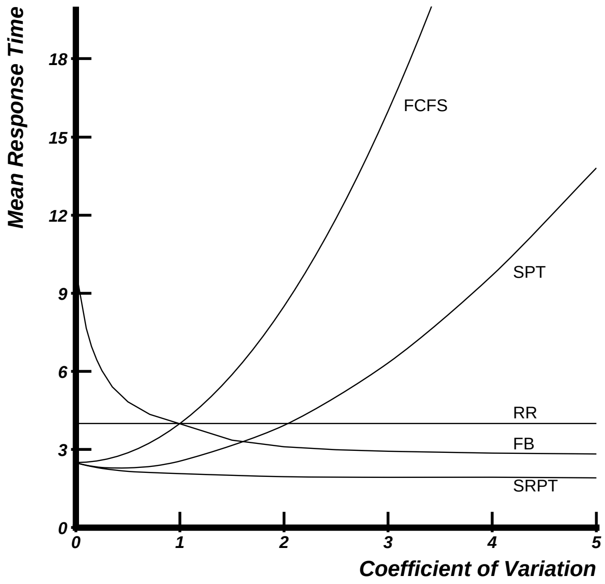

Fig. 1. Mean response time as a function of coefficient of variation for uniprocessor scheduling disciplines at a loading factor of 0.75. The FB curve was obtained from simulations; all others were obtained analytically.

The styles of preemption that we will distinguish for coordinated scheduling are:

- Static Run-To-Completion (RTC) — Some number of processors is assigned to a job when it is activated, and it retains exclusive use of all those processors until its service is completed.
- Dynamic — The number of processors allocated to a job may change during its execution. There may be a significant overhead for the job to reconfigure itself to effectively use the new number of processors.
- Static Quantum-Based — When a job is first activated, it is assigned a number of processors (as in Static Run-To-Completion), and it uses that number of processors whenever it is active. However, its execution may be suspended, either at the termination of a quantum or when an arrival or departure occurs, and then resumed repeatedly until its service requirement is satisfied.

Independent Scheduling The first independent multiprocessor scheduling strategies that were proposed were simple extensions of the uniprocessor FCFS, RR, SPT, and SRPT disciplines [MEB88]. In FCFS and RR, threads are ordered according to the time at which they were placed on the queue, while in SPT and SRPT, threads are ordered according to increasing cumulative service demand remaining for the job. Implicit in most models used to evaluate independent scheduling is the assumption that a job's service requirement is independent of the number of processors it is allocated. In this case, the disciplines yield performance that is in relative agreement with uniprocessor results. The importance of preemption increases with the variability in service demand, as does explicit knowledge of a job's service requirement.

But because of the multiprocessor aspect, some anomalies can occur. In particular, RR will tend to give proportionately more processing time to jobs having a larger number of threads. RRJob avoids this by timeslicing equally among jobs as well as among the threads of a job [LV90]. Both RR and RRJob have been shown to perform well for values of C_d ranging from 3 to 5 [LV90].

Static Run-To-Completion Scheduling Early work in static RTC scheduling showed that a good number of processors to allocate to a job is the value that leads to the smallest ratio of execution time to efficiency (i.e., the value at the knee of the execution time-efficiency profile), as this is a point that maximizes the ratio of benefit to cost [EZL89]. If the number of processors allocated at the knee of the curve is not known, allocating a number of processors equal to the average parallelism has been shown to be a good alternative. The average parallelism (A_j) of a job j is the average degree of parallelism exhibited by the job over its lifetime. If there are no overheads due to parallelism, A_j is equivalent to the speedup on an unlimited number of processors [EZL89].

The overall workload volume, when known, should also be taken into account in the scheduling decision [Sev89]. By reducing the number of processors allocated to jobs as the system load increases, the mean response time can be greatly improved. In the limit, where the system load is near one, jobs should (it can be argued) be allocated no more than one processor since this leads to the highest possible efficiency of the system, assuming there are no other considerations such as large memory requirements. Conversely, under light overall load, each job can be allocated as many processors as it needs to attain its maximum execution rate. Most multiprocessor disciplines take the system load into account in some way to avoid the problem of early saturation that may be caused by running jobs with too many processors.

An interesting generalization of efficiency is to consider the ratio of a job's speedup to an arbitrary cost function instead of to the number of processors. This ratio, called the efficacy of a job, is used to determine a job's processor working set (*pws*) [GST91] (described in more detail below). With the specific cost function chosen for use, the *pws* is equivalent to the number of processors at the knee of the execution time-efficiency profile. A variety of disciplines using knowledge of a job's *pws* have been examined. It has been concluded that an important consideration is to not leave any processors idle while work is available. The best of these disciplines, referred to as FF+FIFO by Ghosal et al. [GST91], but more commonly referred to as PWS, is investigated in this paper.

A policy that does not make use of any job characteristics other than the maximum parallelism is Adaptive Static Partitioning (ASP) [ST93]. Through analytic models, it has been shown that ASP is in general superior to PWS, but the results are only known for a 2-point service time distribution where $C_d = 0.31$. Subsequent simulation studies provide further evidence of the benefits of a variant of ASP where the maximum allocation to a job is bounded [CMV94]. Although most of the results relate to $C_d = 5$, one experiment demonstrates that bounded ASP also performs well when $C_d = 30$. In this paper, we only use ASP as originally defined [ST93], with no bound on the maximum allocation to a job.

Dynamic Scheduling Most research in dynamic scheduling has been directed towards reducing or eliminating the cost of processor reallocations. In the process control approach, jobs and the operating system cooperate in processor reallocations, allowing a job to dynamically match its degree of parallelism to the current allocation [TG89, GTS91]. In this approach, the goal is to avoid altogether the problems of losing cache context and the blocking of critical threads.

Two important disciplines that have been studied in the context of process control are equipartition and demand-driven scheduling (called "dynamic scheduling" originally). In equipartition, each job is allocated an equal fraction of the processors, up to their maximum parallelism [ZM90]. This assumes that all jobs have a degree of parallelism that is constant throughout their lifetime. In order to handle jobs with varying degrees of parallelism, demand-driven scheduling uses process control both to allow the system to take into account changes in a job's parallelism and to allow the job to adapt to changes in processor allocation [ZM90, MVZ93].

Equipartition has been shown to be effective over a wide range of workloads and a wide range of distributions in service demand [LV90, CMV94]. When system loads increase, allocations to jobs decreases allowing them to operate at a more favourable point on their efficiency curve. Also, because equipartition is effectively the analog of RR in uniprocessing, it is relatively insensitive to variability in service requirement. However, not all applications can dynamically change their degree of parallelism according to the current processor allocation, and those that can may incur a high overhead. As a result, we begin by considering equipartition in its ideal form (ideal equipartition or IEQ) in which no overheads exist, and later introduce overheads in the context of the NASA-based workload.

Several techniques have been used with dynamic scheduling in an attempt to reduce problems associated with having fewer processors than threads:

- cache-affinity scheduling: preference is given to threads that have previously run on a given processor in order to limit lost cache context;
- spin-block locking: threads requesting a lock spin for a short duration, after which point they block and yield their processor to another thread;
- coordinated descheduling: the system ensures that threads holding locks do not have their processor taken way.

The use of a combination of these techniques and others can be very effective in reducing the penalty associated with having fewer processors than threads in centralized shared-memory systems [GTU91].

Static Quantum-Based In this paper, we propose coordinated scheduling disciplines that are static quantum-based. In this category, the only discipline that has been previously studied is gang scheduling, in which sets of jobs are actively timesliced [Ous82, LV90, MVZ93]. Research into this topic has been primarily focussed on the mechanism, such as how it should be implemented [FR90] and how important it is for fine-grained applications [FR92]. In our work, we consider issues such as how many processors to assign to an arriving job and when that job should be scheduled to run.

3 Disciplines

This section describes in detail the two disciplines used in our experiments, PWS and ASP, both before and after the enhancements made to handle high variability in service demand. These two disciplines differ primarily in the amount of information given to the scheduler; in PWS, characteristics of the speedup curve is known while in ASP only the maximum parallelism is known. We also experimented with other disciplines (in particular AVG [LV90] and AP [RSD$^+$94]), but as these did not offer much in terms of additional insight, we omit the results.

3.1 Standard Disciplines

The *pws* of a job is the minimum number of processors that maximizes the ratio of its speedup $S_j(n)$ to a cost function $C_j(n) = n/S_j(n)$. This cost function expresses the notion that the cost of a processor depends on how efficiently it is being utilized. Ghosal et al. explore several different discipline that make use of the *pws*, and conclude that the following discipline (called FF+FIFO by them) performs best [GST91]:

PWS When a job arrives in the system, and there are free processors, it is allocated the lesser of the number of free processors and its *pws*. When a job leaves, the scheduler repeatedly examines the jobs in the queue and selects the first one whose *pws* fits in the available processors; if none fit, then the first job is given the remaining processors.

PWS as originally defined limits its search of the queue to just the first w jobs, but in this study, we set no such limit in order to maximize the chance of finding a job for which the *pws* fits in the available processors.

The ASP discipline differs from PWS in that it spreads free processors evenly among all waiting jobs instead of allocating the first job as many processors as it requires:

ASP When a job arrives to the system, it is given the lesser of its maximum parallelism and the number of free processors. When a job is completed, the processors are allocated evenly among jobs that are waiting.

We assume that a job's maximum parallelism is the number or processors for which its speedup function is maximized.

As the baseline policy, we define ideal equipartition as follows:

IEQ When a job arrival or departure occurs, the processors are dynamically reallocated to the current set of jobs in such a way that $(P \bmod J_{total})$ jobs are allocated $\lfloor P/J_{total} + 1 \rfloor$ processors and the rest one less, where J_{total} is the total number of jobs in the system. Periodically, the scheduler rotates the jobs, placing the last job (in a run queue) at the front, thereby evening out any imbalances in processor allocation. In particular, if there are more jobs than processors, all jobs receive some fraction of the system's processing capacity. Once again, a job is never allocated more processors than its maximum parallelism.

As in previous studies [LV90], we assume that a job adapts instantaneously to the number of processors allocated to it after each reallocation.

3.2 Multiprocessor Feedback Disciplines

The feedback scheduling disciplines derived from the static disciplines all follow a similar pattern. When a job arrives to the system, it is configured for a certain number of processors that depends on the known characteristics of the job and the other jobs currently available for execution. At the start of each time slice, all active jobs are examined, and those having the least acquired processing time are scheduled to run. Acquired processing time is just the number of processor-seconds allocated to the job so far, and thus can differ from the actual work accomplished by the computation since the latter is influenced by the job's speedup properties. The scheduler repeatedly selects the next job which has the least acquired processing time and which fits within the remaining set of processors (as configured upon arrival), and schedules it to run.

The two variants of the static quantum-based disciplines that we present are:

FB-PWS An arriving job is configured for a partition size of

$$\min\left\{M_j, \frac{\min\left\{pws_j, P\right\}}{R+\min\left\{pws_j, P\right\}} \times P\right\}$$

processors, where M_j is the job's maximum parallelism and R is the sum of the processor allocations for all jobs currently in the system. In general, each job is allocated a fraction of the P processors that corresponds its share of the total anticipated number of processors allocated in the system. Under light load, a job will allocated more processors than its *pws* and under heavy load it will be allocated fewer.

FB-ASP An arriving job is configured with $\mathrm{round}(P/J_{total})$ processors, except that if there are twice as many available processors as the computed partition size, then one more processor is given (to account for uneven partition sizes). This is different from the static ASP in that the partition size is not based solely on the number of waiting jobs (which would not make sense in this case.)

When there are fewer processors left than any of the remaining jobs' configurations, the scheduler runs the next job anyway using the remaining processors. For this, we assume that we have a thread scheduler that, at the very least, avoids blocking critical threads and implements some form of cache-affinity scheduling. But when a job with a static allocation of p processors is activated with only q ($q < p$) processors, its execution rate will be less that q/p times its full execution rate due to the mismatch of threads and processors. Gupta et al. studied the effect of combined thread scheduling features, including those just described, and showed that for a set of four applications, the processor utilization dropped by just under 9% over batch scheduling (see Fig. 6 in Gupta et al. [GTU91]). Adopting this result, we assume that a job that is running with fewer processors than its static configuration progresses 9% slower than q/p times its full execution rate. Experimentation indicates, however, that these two disciplines, particularly FB-ASP, tolerate higher slowdown values reasonably well. When the slowdown value was increased to 100%, the increase in mean response times for FB-PWS ranged from 0% at low loads to 24% at high loads, while for FB-ASP, the increase was less than 2% throughout.

4 Definition of Model

We use a discrete event simulation to evaluate the different scheduling disciplines. Input parameters to the simulator included the arrival rate, average service demand, coefficient of variation of the cumulative service demand, job speedup characteristics, and the scheduling discipline of interest. The specification of and results from the various simulation runs are given in Sec. 5.

4.1 System Model

The system model consists of 100 functionally equivalent processors. No details of the interconnection network or the memory system are modeled. The cost of descheduling and rescheduling a job in the static quantum-based disciplines is modeled explicitly, and is assumed to be 2.5% of the length of a timeslice (a conservative estimate). The cost of reallocating processors in IEQ is assumed to be zero.

Jobs are assumed to arrive according to a Poisson process, and have a service time distribution that is Erlang, exponential, or hyperexponential, depending on the specified coefficient of variation.

4.2 Workload Model

Our choice of job characterization explicitly allows for overheads in a parallel computation [Sev94]. The execution time of a job j is defined by:

$$T_j(n) = \phi \frac{W_j}{n} + \alpha + \beta n$$

where n is the number of processors allocated to the job and W_j is the amount of work (cumulative service demand) for the job. ϕ represents the load imbalance in the threads of the computation. α represents the amount of sequential computation and the amount of per-processor work required for the parallelization of the computation. Finally, β represents the communication and congestion delays that increase with the number of processors. It has been shown by Wu [Wu93] that actual measured speedup functions can be represented with this functional form if ϕ, α, and β are chosen to yield the best fit (with respect to least-square error). Our choice of values for ϕ, α, and β are based on this work.

Given this execution-time function, a job's maximum parallelism is:

$$M_j = \begin{cases} \infty & \text{if } \beta = 0 \\ \sqrt{\frac{\phi W_j}{\beta}} & \text{otherwise} \end{cases}$$

and its *pws* is:

$$pws_j = \begin{cases} \frac{\phi W_j}{\alpha} & \text{if } \beta = 0 \\ \frac{\alpha - \sqrt{\alpha^2 + 12\beta\phi W_j}}{-6\beta} & \text{otherwise} \end{cases}$$

Two characterizations were used to represent workloads having quite distinct parallelization overheads. In the first, all jobs, irrespective of size, have nearly perfect speedup, whereas in the second, small jobs experience poor speedup and large jobs experience relatively good speedup. Illustrated in Fig. 2 are representative speedup curves for the two workloads for job sizes ranging from 500 to 5000000. The parameters used for characterizing the two workloads (as well the NASA workload) are shown in Table 1.

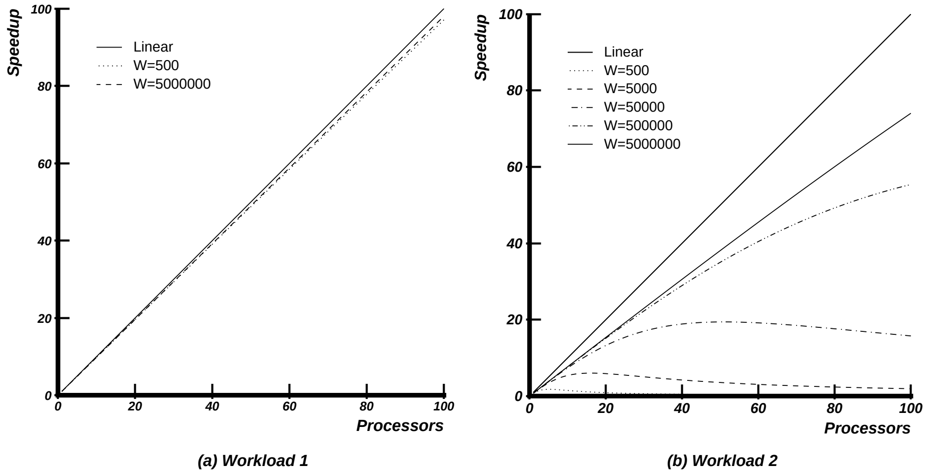

Fig. 2. Speedup curves for workloads 1 and 2 used in this paper.

Workload	Mean Service Requirement	Parameters
1	1000	$\phi = 1.02$, $\alpha = 0.05$, $\beta = 0.0$
2	1000	$\phi = 1.3$, $\alpha = 25$, $\beta = 25$
NASA	92371	$\phi = 1.15$, $\alpha = 1000$, $\beta = 600$

Table 1. Parameters used for the workloads used in this paper.

We also experimented with other workloads and found that, qualitatively, their performance was between the two we chose. In particular, we considered mixed workloads in which jobs had varying values of ϕ, α, and β, as might be found in actual workloads.

As such, we feel that the two workloads chosen are sufficient to explore the behaviour of the various scheduling disciplines.

We study a third specific workload, which is chosen to represent the NASA workload described earlier. The service time distribution is an 8-stage hyperexponential distribution with one stage corresponding to each distinct workload component [NAS80]. In the absence of speedup information about the jobs, we chose to use a characterization that fell in between our endpoints.

5 Analysis of Simulation Results

First, we examine the performance of the various scheduling disciplines under workloads 1 and 2 as a function of the coefficient of variation in service demand. Next, we examine the performance of all the disciplines under the NASA workload as a function of system load.

For the most part, a sufficient number of independent trials were done to obtain a 95% confidence interval that was within 5% of the mean for each data point in our simulation results. Because of the instability of distributions having high coefficient of variation, however, the data points for $C_d = 70$ sometimes have a confidence interval greater than 5% of the mean for the higher system load values. Each trial had a warm-up period in which the first 20 jobs were discarded. A trial terminated when the subsequent 100000 jobs (twice that for $C_d = 30$ and four times that for $C_d = 70$) to arrive left the system. The simulation results of a run are based only on the response times of these 100000 jobs.

5.1 Workload 1

Figure 3 plots the performance of the original disciplines in comparison to their FB counterparts as a function C_d. Curves are shown for each of four arrival rates for each discipline. The solid lines represent the non-FB disciplines, while the dotted lines represent their FB counterparts. The mean service required per job was 1000, so the mean interarrival times of 50, 20, 15, and 12.5 correspond to system loading factors of 20%, 50%, 67%, and 80%, respectively. The performance of ideal equipartitioning is shown separately.

In either case, the FB variant outperforms the non-FB variant as the coefficient of variation increases beyond one. At high load and $C_d = 70$, the response times of the non-FB variants of PWS and ASP are more than one hundred times worse than their FB counterparts. Consistent with results from uniprocessor scheduling, the FB variants have, in general, decreasing mean response time with increasing C_d, while the opposite holds for the static RTC disciplines.

Figure 4 shows the relative performance of the various disciplines at light and heavy loads in comparison to ideal equipartitioning. At light load, the performance difference between the static quantum-based disciplines and IEQ is very small, and at heavy load, FB-PWS performs equally well as IEQ for $C_d > 1$. The reason why FB-ASP does not perform as well as FB-PWS in this workload is that large jobs receive the same share of processors as small jobs, leading to a lower processor utilization when the small jobs leave the system.

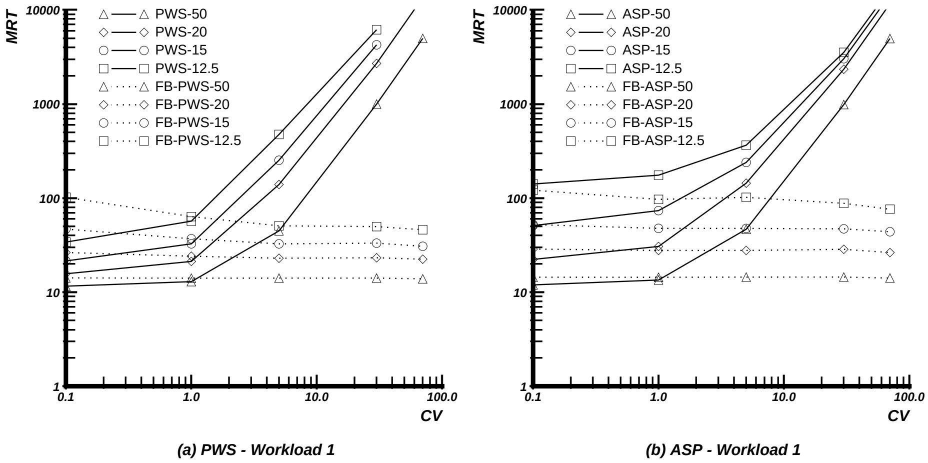

Fig. 3. Performance of scheduling disciplines under workload 1.

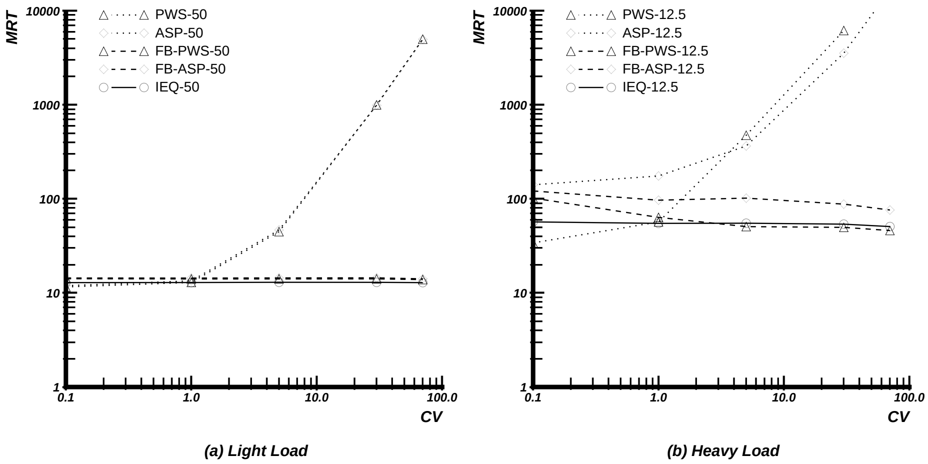

Fig. 4. Relative performance of scheduling disciplines under workload 1 at light and heavy loads.

5.2 Workload 2

As workload 1 is studied in Figs. 3 and 4, the corresponding graphs for workload 2 are shown in Figs. 5 and 6. Recall that, in workload 2, large jobs (with $W_j \geq 500000$) attain nearly linear (but not unitary) speedup out to 100 processors, but the speedup for small jobs (with $W_j = 500$) reaches a maximum by the point at which five processors are assigned. Although the graphs display similar tendencies as with the first workload, one

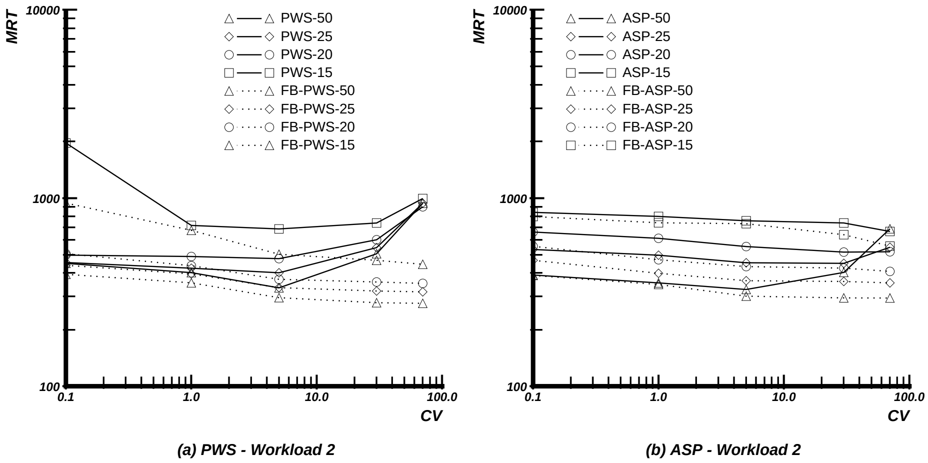

Fig. 5. Performance of scheduling disciplines under workload 2.

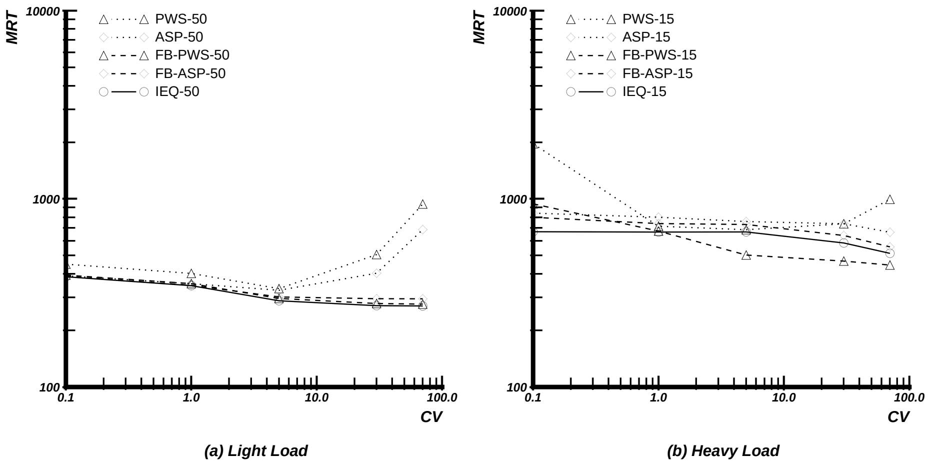

Fig. 6. Relative performance of scheduling disciplines under workload 2 at light and heavy loads.

can observe a number of important differences, primarily due to the different speedup characteristics exhibited by differently sized jobs.

In the graph for PWS, the non-FB version shows a much smaller degradation for high C_d as compared to that with workload 1. In fact, at high values of C_d, a crossover takes place and mean response time is slightly lower at higher loads than at lower loads. The problem with FCFS policies in general is that long jobs delay short jobs for the duration of their execution. What happens in PWS, however, is that large jobs tend to receive smaller and smaller partitions as load increases, reducing their negative impact on mean response time.

The reason for this is that PWS allocates a job the lesser of its *pws* and the number of free processors. As the load increases, the pending queue gets larger, and processors freed by a departing job are immediately allocated to another. The size of the new partition is no greater than that of the departing job and, as a result, partition sizes tend to only get smaller as time goes on. Under light load, it is quite possible for a large job to arrive in a relatively quiet period and monopolize a large proportion of the processors for an extended period of time, but as load increases, this becomes less and less likely. The performance of PWS is poor at $C_d = 0.1$ since all jobs are roughly the same size (and thus have the same *pws*); partition sizes never get a chance to decrease in size.

The FB variant of PWS, in this workload, does not offer as great an improvement as with the previous workload. At low loads, PWS has response times more than three times worse than FB-PWS. This ratio drops to just less than 2 under heavier system load. At lighter load, FB-PWS shows better performance because it can assign a job more processors than can PWS in order to make use of the entire machine.

ASP is quite different from PWS in this workload. Its response times still increase at low loads, where, for high values of C_d, the response times significantly exceed those at higher loads. At higher system loads, response times are insensitive to C_d. Since jobs are given what amounts to an equal fraction of processors, ASP behaves much like a round robin system would in this case. Nonetheless, FB-ASP performs better than ASP across all job variations. (Note that the curve for FB-ASP at an arrival rate of 20 is indistinguishable from the curve for ASP at a rate of 25.) One reason for this is that ASP partitions processors freed by a departing job quite aggressively. For example, if a job which has 10 processors is completed, and three jobs are pending, the processors are partitioned as 3-3-4. FB-ASP takes a more gradual approach, giving each process a fraction of the processors based on the total number of jobs in the system. Thus, ASP often ends up not allocating enough processors to each job, leaving many processors idle.

The comparison between the scheduling disciplines in Fig. 6 again shows little performance difference between the static quantum-based disciplines and IEQ under light load, but also shows better performance by FB-PWS than IEQ as the load increases for $C_d > 1$. One can observe that IEQ has a decreasing mean response time with increasing C_d. This is due to the fact that small jobs are greatly restricted in the number of processors they can acquire (e.g., a job of size 500 has a maximum parallelism of 6), resulting in large jobs acquiring proportionately more processors than they would without such restrictions. Since large jobs have better speedup characteristics, the overall efficiency of the system increases, thus reducing the mean response time as C_d increases.

5.3 NASA Workload

To obtain a better understanding of how the various disciplines perform under a more realistic workload, we evaluated the disciplines under the NASA workload described earlier. Since this workload has a fixed coefficient of variation of 7.23, we plot the mean response time as a function of load in Fig. 7.

As can be seen, FB-PWS and IEQ once again perform very well at all load levels. The benefit of preemption as used by the FB disciplines and IEQ is quite apparent, espe-

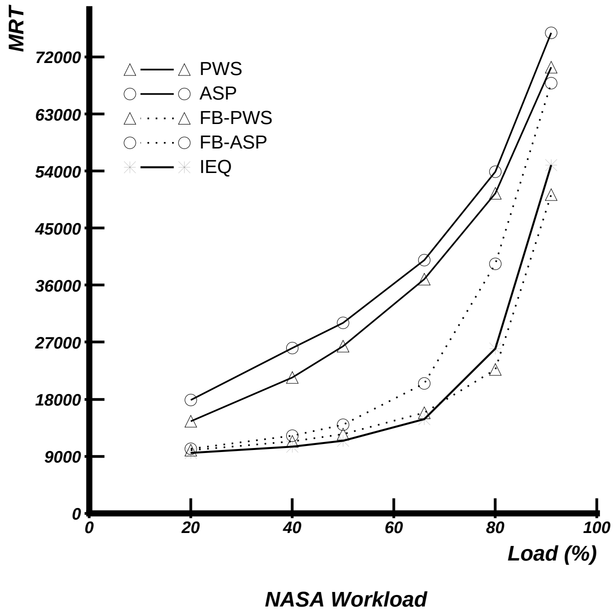

Fig. 7. Performance of the scheduling disciplines under the NASA workload as a function of system load.

cially at lower load levels. At about 50% load, the mean response time for PWS is twice that of FB-PWS. Similarly, ASP has mean response time a little more than double that of FB-ASP at the same load level.

5.4 Considerations for Distributed-Memory Systems

We are particularly interested in studying static quantum-based scheduling schemes because dynamic repartitioning schemes, such as equipartition, may either not be available or may incur substantially greater costs on distributed shared-memory systems. On such architectures, remote data accesses can be an order of magnitude more expensive than local requests. As a result, moving a thread from one processor to another, for load balancing purposes, can be quite costly. Similarly, reconfiguring a job to run on a different number of allocated processors can require substantial amounts of data movement. For example, a matrix that has been stored in an interleaved fashion may need to be completely redistributed if the original processor allocation is not a multiple of the new one.

Recall that in our implementation, we had jobs that were run in leftover partitions that were smaller that the degree of parallelism configured for the job (to avoid having processors remain idle). In the centralized shared-memory architecture, the overhead of doing this was a small factor of 9%; in a distributed shared-memory system, the overhead is likely to be much larger. We consider the case where a job runs half as fast as it would be expected to (i.e., a slowdown factor of 100%), which is much more conservative than the case where threads can be multiplexed (possibly unevenly) on the remaining processors. We also consider the case where the overhead for equipartitioning is non-zero. Since we do not have any previous work on which we can base a value for the overhead, we consider the cases where the repartitioning overhead is 0.25%, 0.5%, and 0.75% of the mean service requirement for jobs[2].

Figures 8 shows the effect of these modifications for the NASA workload, now omitting the lines for the static RTC disciplines. FB-ASP seems to be relatively unaffected by the increased slowdown. The degradation in performance relative to the previous model is at most 1.8% over the range in loads shown. FB-PWS was affected by this change to a greater extent, ranging from 4% at low load to roughly 24% at high load. The reason for this is that FB-PWS allocates much larger partition sizes to large jobs than FB-ASP does, which causes these jobs to run more frequently in "leftover partitions" at half the speed. But IEQ is affected quite severely by repartitioning overhead, especially at the two higher levels. For an overhead of 0.75%, equipartitioning has a mean response time up to almost twice that of FB-PWS.

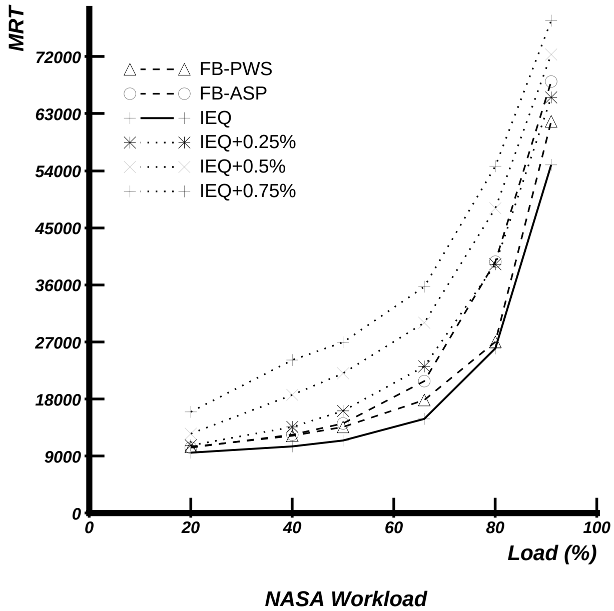

Fig. 8. Effect of increasing scheduling overheads on the NASA workload, both for the static quantum-based and the IEQ disciplines.

[2] This is not intended to be representative of real overheads, but merely a way to examine how overheads can affect IEQ.

6 Conclusions

In this paper, we have examined the sensitivity of various scheduling disciplines to high variability in job service demand and proposed new preemptive disciplines that have much better performance characteristics when the service time coefficient of variation is large. Our primary focus is where the coefficient of variation of service demand (C_d) ranges from 5 to 70, as has been observed at various high performance computing centers.

When jobs have nearly perfect speedup characteristics (i.e., workload 1), the behaviour of the various disciplines are relatively consistent with the corresponding uniprocessor results. The performance of static run-to-completion (RTC) disciplines degrades severely as C_d increases, while the performance of preemptive multilevel feedback (FB) disciplines improves. As expected, the RTC and FB disciplines yield comparable response times at $C_d = 1$, but at $C_d = 70$, their response times differ by two orders of magnitude.

When, on the other hand, small jobs are characterized as having poor speedup and large ones good speedup (i.e., workload 2), there are a number of interesting differences. The performance of the RTC disciplines, because of the way in which they deal with leftover processors, become more round-robin in nature, displaying a fair amount of insensitivity to C_d at higher loads. The improvements gained by using FB variants of these is substantially reduced relative to the near-perfect speedup case, from a factor of 100 to a factor of about four, but the improvements are still quite significant. In all workloads, the static quantum-based disciplines (in particular FB-PWS) proved to be competitive with ideal equipartitioning (IEQ).

Our results can be summarized as follows:

- IEQ does very well in all our experiments with high C_d. The corresponding practical discipline, EQ, is a good discipline to use if it can be implemented without incurring excessive overhead from (1) frequent preemptions, (2) loss of cache affinity when threads are moved from one processor to another, and (3) restructuring of jobs to adapt to changes in processor allocations. However, in large multiprocessors with physically distributed memory modules, coordinated placement of data and threads is critical, so overheads (2) and (3) are likely to be large. In this case, the static quantum-based disciplines are preferable.
- If estimates of the *pws* for each job are available for use in scheduling, then FB-PWS does as well as IEQ at keeping response times low for high C_d. This is somewhat surprising since FB-PWS must commit to a static partition size for each job when it is activated, while IEQ does not.
- If only an estimate of each job's maximum parallelism is available for use in scheduling, then FB-ASP is the best rule to use, although the availability of added knowledge can make a significant difference (as shown by FB-PWS).

There are some important implications in our study for scheduling real multiprogrammed multiprocessor systems. Currently, a user is usually expected to choose a processor allocation (in selecting a partition of the system) and/or to indicate the expected execution time of the job (in selecting a batch queue). With FB-ASP scheduling, a user is not required to provide either of these, except for perhaps a maximum parallelism to avoid allocating too many processors to a job (necessary only in lightly-loaded systems). If a system is being used for a well-defined set of applications, it is reasonable to expect more speedup information to be available, either collected automatically by the system or prepared off-line. In this case, a discipline like FB-PWS will give the best performance.

One issue that the study does not consider is the resource requirements of jobs. In particular, none of the disciplines examined, particularly IEQ, will perform well if memory is overcommitted, as paging overhead can significantly impact the progress of a computation [BHMW94]. Clearly, future scheduling disciplines for multiprogrammed multiprocessor systems must take into account all requirements of jobs, including processor, memory, and I/O.

References

[BHMW94] Douglas C. Burger, Rahmat S. Hyder, Barton P. Miller, and David A. Wood. Paging tradeoffs in distributed-shared-memory multiprocessors. In *Proceedings Supercomputing '94*, November 1994.

[CMV94] Su-Hui Chiang, Rajesh K. Mansharamani, and Mary 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 Modelling of Computer Systems*, pages 33–44, 1994.

[EZL89] Derek L. Eager, John Zahorjan, and Edward D. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3):408–423, March 1989.

[FN95] Dror G. Feitelson and Bill Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph (eds.), Lecture Notes in Computer Science Vol. 949. Springer-Verlag, 1995.

[FR90] Dror G. Feitelson and Larry Rudolph. Distributed hierarchical control for parallel processing. *Computer*, 23(5):65–77, May 1990.

[FR92] D. G. Feitelson and L. Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, 16:306–318, 1992.

[GST91] Dipak Ghosal, Guiseppe Serazzi, and Satish K. Tripathi. The processor working set and its use in scheduling multiprocessor systems. *IEEE Transactions on Software Engineering*, 17(5):443–453, May 1991.

[GTS91] Anoop Gupta, Andrew Tucker, and Luis Stevens. Making effective use of shared-memory multiprocessors: The process control approach. Technical Report CSL-TR-91-475A, Computer Systems Laboratory, Stanford University, July 1991.

[GTU91] Anoop Gupta, Andrew Tucker, and Shigeru 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, 1991.

[LV90] Scott T. Leutenegger and Mary K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, pages 226–236, 1990.

[MEB88] S. Majumdar, D. L. Eager, and R. B. Bunt. Scheduling in multiprogrammed parallel systems. In *Proceedings of the 1988 ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, pages 104–113, May 1988.

[MVZ93] Cathy McCann, Raj Vaswani, and John Zahorjan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146–178, May 1993.

[NAS80] Numerical aerodynamic simulator processing system. Technical Report PC320-02, NASA Ames Research Center, September 1980.

[Ous82] John K. Ousterhout. Scheduling techniques for concurrent systems. In *Proceedings of the 3rd International Conference on Distributed Computing (ICDCS)*, pages 22–30, October 1982.

[RSD+94] E. Rosti, E. Smirni, L. W. Dowdy, G. Serazzi, and B. M. Carlson. Robust partitioning policies of multiprocessor systems. *Performance Evaluation*, 19:141–165, 1994.

[Sch70] Linus E. Schrage. Optimal scheduling rules for information systems. *Operations Research*, 26, August 1970.

[Sev89] Kenneth C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. In *Proceedings of the 1988 ACM SIGMETRICS International Conference on Measurement and Modelling of Computer Systems*, pages 171–180, May 1989.

[Sev94] K. C. Sevcik. Application scheduling and processor allocation in multiprogrammed parallel processing systems. *Performance Evaluation*, 19:107–140, 1994.

[ST93] Sanjeev Setia and Satish Tripathi. A comparative analysis of static processor partitioning policies for parallel computers. In *Proceedings of the International Workshop on Modeling and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 283–286, January 1993.

[TG89] Andrew Tucker and Anoop Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 159–166, 1989.

[Ver94] Mary K. Vernon. Private communication, September 1994.

[Wu93] Chee-Shong Wu. Processor scheduling in multiprogrammed shared memory NUMA multiprocessors. Master's thesis, University of Toronto, 1993.

[ZM90] John Zahorjan and Cathy McCann. Processor scheduling in shared memory multiprocessors. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, pages 214–225, 1990.