Use of Application Characteristics and Limited Preemption for Run-To-Completion Parallel Processor Scheduling Policies

Su-Hui Chiang
Rajesh K. Mansharamani
Mary K. Vernon

Technical Report #1220

March 1994

Use of Application Characteristics and Limited Preemption for Run-To-Completion Parallel Processor Scheduling Policies*

Su-Hui Chiang†, Rajesh K. Mansharamani††, and Mary K. Vernon†

†Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706.

email: {suhui, vernon}@cs.wisc.edu

††TRDDC
1 Mangaldas Road
Pune 411 050, India.

email: mansha@research.trddc.ernet.in

Abstract

The performance potential of run-to-completion (RTC) parallel processor scheduling policies is investigated by examining whether (1) application execution rate characteristics such as *average parallelism (avg)* and *processor working set (pws)* and/or (2) limited preemption can be used to improve the performance of these policies. We address the first question by comparing policies (previous as well as new) that differ only in whether or not they use execution rate characteristics and by examining a wider range of the workload parameter space than previous studies. We address the second question by comparing a simple two-level queueing policy with RTC scheduling in the second level queue against RTC policies that don't allow any preemption and against dynamic equiallocation (EQ).

Using simulation to estimate mean response times we find that for promising RTC policies such as adaptive static partitioning (ASP) and shortest demand first (SDF), a maximum allocation constraint that is for all practical purposes independent of *avg* and *pws* provides greater and more consistent improvement in policy performance than using *avg* or *pws*. Also, under the assumption that job demand information is unavailable to the scheduler we show that the ASP-max policy outperforms all previous high performance RTC policies for workloads with coefficient of variation in processing requirement greater than one. Furthermore, a two-level queue that allows at most one preemption per job outperforms ASP-max but is not competitive with EQ.

1. Introduction

In this paper we consider issues related to the performance potential of *run-to-completion* (RTC) processor scheduling policies for multiprogrammed parallel systems. By RTC policies we mean the class of non-preemptive scheduling policies in which each job runs to completion without interruption on the set of processors initially allocated to it.[1] All other policies are *dynamic*.

RTC policies are attractive because they are simpler to implement and have lower overhead (i.e. process preemption and data movement) than dynamic policies. On the other hand, dynamic policies may adjust the spatial and/or temporal allocations of processing power to executing jobs when a given job completes or a new job arrives, whereas RTC policies cannot. Thus, the question of how many processors to allocate to a job is a particularly crucial issue in RTC scheduling.

The optimal parallel processor scheduling policy is unknown except for a few very specific workload and system assumptions [AMV93,Se93]. Thus, many recent studies have compared specific policies to understand, for given workloads, which parallel scheduling policies perform better than others as well as which characteristics of scheduling policies lead to improved performance. For example, several studies have shown that among dynamic policies that do not use job demand information, policies that allocate processing power approximately equally among executing jobs (i.e,. EQ policies) have high performance when the coefficient of variation, C_D, in cumulative job processing requirement is greater than one [LV90, ST91, NSS93a, NSS93b, MVZ93, MV93b]. Also, several studies have shown that particular RTC policies have poorer performance than particular dynamic policies [ZM90, MEB91, ST91, NSS93b]. As a third example, there is consensus in the literature that RTC policies have higher performance if they allocate fewer processors to a job as (instantaneous or average) system load increases [Se89, GST91, NSS93a, NSS93b, ST93]. (The data in [ZM90, MEB91, RSDS93, SRDS93] also support this conclusion.)

Questions that remain unresolved in the literature include:

(1) Should RTC policies use application execution rate characteristics, such as *average parallelism* (*avg*) or *processor working set* (*pws*) (defined in section 2) in determining processor allocations? Several studies [EZL89, Se89, MEB91, GST91] have concluded that using execution rate characteristics is beneficial, other studies have assumed that using these characteristics leads to better performance (e.g., [ZM90]), and still other studies [ST93, SRDS93] have reached the opposite conclusion.

(2) The benefit of application characteristics for dynamic policies is also an open question that to the authors' knowledge has not been studied before.

(3) Another possible approach to improving the performance of RTC policies is to use limited preemption of executing jobs. How much benefit can be obtained for RTC scheduling by the simple extension that allows at most a single preemption?

This work was partially supported by NSF grants CCR 9024144 and CDA 9024618.

*To appear in Proceedings 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.

[1]Note that we are focusing on policies for allocating processors to the active jobs in the system, rather than on how the processes or threads of a job are scheduled on the processors allocated to the job.

This paper addresses the above open questions. The results should be useful for determining the best RTC policies, interpreting and generalizing from comparisons of particular RTC and dynamic policies, and understanding more precisely the performance differential between RTC and dynamic policies. The approach pursued is as follows.

- The issue of whether *avg* and *pws* are beneficial to RTC scheduling is examined by comparing policies that differ *only* in this dimension, over a wider range of the workload parameter space than previous studies. This leads to new observations as well as explanations that unify previous results. In particular, we discover that a different characteristic that hasn't received as much attention in the previous literature is important for high performance RTC scheduling.
- We then investigate whether the performance of the smallest demand first (SDF) policy and a dynamic EQ policy, which can each be viewed as providing a target for RTC policy performance, can be improved by using the *avg* measure.
- To examine the RTC policy performance improvement achievable from a single preemption, we compare the performance of a two-level queueing policy that uses RTC scheduling in the second level queue against the performance of strict RTC policies and the EQ policies.

The remainder of this paper is organized as follows. Section 2 describes the system and workload model. Section 3 defines the processor scheduling policies evaluated in this paper. Section 4 examines whether measures such as *avg* and *pws* improve policy performance under various workload parameter values. Section 5 studies the impact of limited preemption on the performance of the best RTC policy from section 4. Finally, section 6 summarizes the conclusions of this paper.

2. System and Workload Model

To study the behavior of scheduling policies with respect to workload parameters over a wide range of workloads, we use the system and workload model of [MV93a], reviewed below.

2.1. System Model

We consider an open system with P identical processors and a central job queue. Jobs arrive to the system according to a Poisson process with rate λ as shown in Figure 2.1. The centralized queue is only conceptual; practical implementations of the policies considered may in general allow for distributed queue access.

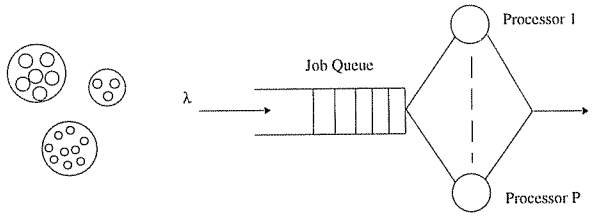

Figure 2.1: Open System Model

Scheduling and preemption overheads are highly system specific. We thus assume zero scheduling and preemption overhead and then allow evolving intuition and experience with actual system overhead to be used in interpreting the performance results. For example, the performance of an EQ policy with zero scheduling and preemption overhead can be a target for RTC performance with the caveat that the preemption overhead of the EQ policy must be qualitatively factored into the comparisons. We also note that efficient implementations of dynamic policies can reduce preemption frequency such that preemption overhead is a small fraction (i.e., <5%) of response time [NSS93a, NSS93b].

2.2. Workload Model

The following workload model attempts to capture in a few parameters the essential features of parallel applications *with respect to relative scheduling discipline performance*. These features include distributions of available parallelism and total processing requirement (demand), task synchronization and communication overheads, and correlation between parallelism and demand.

For most of the simulations in this paper all jobs are assumed to be statistically identical. Multiclass workloads that are assumed in a few experiments will be described in section 4.5. Each job is characterized by the following random variables:

(1) D, the job's total processing requirement (i.e., execution time on one processor),

(2) $N \in \{1,2,...,P\}$, the job's *available parallelism*,

(3) $E:[0,P]\rightarrow[0,P]$, the job's execution rate function (ERF) which satisfies the constraint that $E[x] = E[N]$, $x \geq N$.

The system operates as follows. Each arriving job joins the central job queue. At each time, t, processors are allocated to jobs present in the queue according to some processor allocation policy. If $a(t)$ processors (possibly fractional) are allocated to a job at time t, then its demand is satisfied at rate $E(a(t))$. The job leaves the system upon completion of its total demand, D. The available parallelism, N, of a job is the number of processors the system scheduler believes the job can productively use. For example, this may be the number of processors the user requests when submitting the job, or it may be the number of processes that the job forks at the beginning of it's execution. The model assumes that the job cannot use more than N processors productively and that N is fixed throughout the lifetime of the job.

The following specific workloads will be used in the experiments:

- Three distributions of N are considered:

(1) the *constant* distribution, i.e., $N=k$ for a fixed $k \in \{1,...,P\}$,

(2) a *2-point* distribution, where $N=1$ with probability p_1 and $N=P$ with probability $1-p_1$, and

(3) the *spread* N distribution, a bounded-geometric distribution [LV90] given by $N = \min(X,P)$, where X = Geometric(p), $0 \leq p \leq 1$.

C_N denotes the coefficient of variation of N. For a given $\overline{N}$, the constant distribution has lowest C_N (i.e., zero), the 2-point distribution has highest C_N among all distributions of N, and the spread distribution has C_N in between.

- E is derived from a *deterministic* function γ:

$$E(k) = \begin{cases} \gamma(k), & k=1,2,...,N, \\ \gamma(N), & k=N+1,...,P. \end{cases}$$

We use the following functional form derived from [Do88] for γ:

$$\gamma(k) = \frac{(1+\beta)k}{k+\beta}, \qquad k=1,2,...,P.$$

Figure 2.2 plots this nondecreasing ERF for several values of β.

- As in [MEB88], two extremes of correlation between job demand and parallelism are considered: no correlation ($r=0$), in which case D and N are independent, and full correlation ($r=1$), in which mean demand is linearly correlated with N.

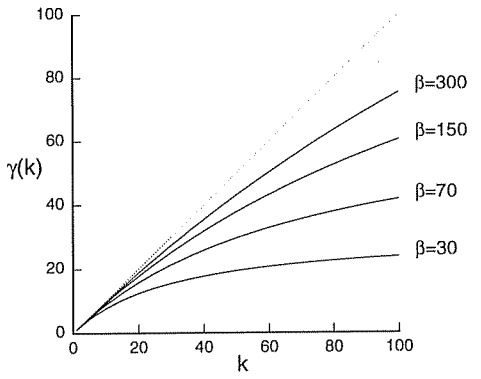

Figure 2.2: Example curves for ERF $\gamma(k)=\frac{(1+\beta)k}{k+\beta}$

More precisely, under full correlation if a job has available parallelism k then its demand is drawn from a distribution with mean $c \cdot k$ and coefficient of variation C_v, where c is a suitable constant. It can be verified that the unconditional mean demand $\overline{D} = c\overline{N}$ and thus $c = \overline{D}/\overline{N}$.

- As in [MEB88] we assume a two-stage hyperexponential distribution (H_2) for job demand with mean as explained above (for r=0 and r=1) and coefficient of variation C_v. When r=0, the overall coefficient of variation in demand C_D equals C_v. When r=1 it can be verified that

$$C_D^2 = (1+C_N^2)(1+C_v^2)-1. \qquad (2.1)$$

The hyperexponential distribution allows us to test the sensitivity of policy performance to C_D. The performance of many policies is sensitive to C_D (cf. [MEB88,LV90, MV93a]). Furthermore, computer system workloads often have $C_D > 1$. [2]

2.3. Comments on the Workload Model

The ERF is assumed to be nondecreasing since the user is unlikely to request more processors (either explicitly or by forking more processes) if this will increase program execution time. The deterministic ERF γ allows us to examine policy performance with respect to ERF sublinearity by varying a single parameter β. In section 4.5 we also consider multi-class workloads in which each class has a different ERF sublinearity (i.e., a different value for β).

The parallelism parameter N is known to the scheduler. Some schedulers are also given the average parallelism (*avg*) or processor working set (*pws*), each of which is a function of N and β as follows. Since *avg* is the speedup of the job on an unbounded number of processors [EZL89], it follows that $avg = \gamma(N)$ for $N < P$. For $N = P$ we let $avg = \gamma(P)$. The *pws* measure [GST91], coincides with the knee of the execution-time efficiency profile [EZL89,MEB91]. For the specific ERF $\gamma(k)=(1+\beta)k/(k+\beta)$ it can be verified that the $pws = \min(N, \beta)$. Note that for workloads where N varies, different jobs will have different *avg* and *pws* values even though they have the same ERF γ. Furthermore, for policies that do not use execution rate characteristics processor allocations are independent of β.

[2] C_D on one partition of our local CM-5, measured on a weekly basis for 22 weeks, ranges from approximately 2.5 to about 6, with 40% of the measures above 4.0. Moreover, measurements of Cray YMP sites have reported C_D to be in the range of 30-70 [Cr91].

3. Processor Scheduling Policies

In this section we give the definitions for several previous processor scheduling policies as well as several new policies that will be compared in section 4. The previous policies have been shown to have high performance for various workloads. The new policies are constructed to enable equitable comparisons between policies that do and do not use execution rate characteristics.

3.1. Previous Policies

ASP (Adaptive Static Partitioning): When a job arrives it is allocated the lesser of the number of idle processors in the system and its available parallelism. If no free processors are available, the job queues behind waiting jobs, if any. When a job completes, the released processors are allocated one at a time to the waiting jobs in round robin order (starting from the first waiting job), under the constraint that no job is allocated more processors than its available parallelism. This policy was first defined in [ST91].

PWS: A FCFS policy in which each job is allocated the minimum of *pws* and the number of idle processors. This is equivalent to the FF+FIFO policy defined in [GST91] and renamed PWS in [ST93], with window size equal to one.

AVG: The same as PWS except the measure *avg* is used in place of *pws*. We also consider a smallest-average-parallelism-first scheduling policy, denoted by SAVG, because of its potential for high performance under correlated workloads and because it allows for more jobs to be in service simultaneously.

A+&mM: Jobs are scheduled in FCFS order. A job is allocated the minimum of α and the number of free processors if there is any. The parameter α is a function of the system load and the job's average (*avg*), minimum (m), and maximum ($M = N$) parallelism [Se89]. If a job's execution profile contains only sequential ($m = 1$) and fully parallel phases, α is computed as follows :

$$\alpha = \begin{cases} N - \rho\frac{(N-avg)}{\pi} & \text{if } \rho < \pi \\ 1 + (1-\rho)(\frac{avg-1}{1-\pi}) & \text{if } \rho \geq \pi \end{cases} \qquad (3.1)$$

where π is set to be 0.25 in [Se89].

SDF (Smallest Demand First): Jobs are scheduled in order of increasing demand (D), without preemption. A job is allocated the minimum of its parallelism (N) and the number of idle processors.

EQ: Dynamic EQuiallocation policies allocate an equal fraction of processing power to each job in the system unless a job has smaller available parallelism than the equipartition value, in which case each such job is allocated as many processors as its available parallelism, and the equipartition value is recursively recomputed for the remaining jobs. The EQ policy simulations spatially partition the processors (with temporal sharing for the fractional part of processing power if any). Spatial equi-allocation policies have been considered in [TG89, MVZ93] and temporal versions in [LV90, LN91,MV93b].

3.2. New Policies

Adaptive-AVG: Upon an arrival, the job is allocated the lesser of its *avg* and the number of idle processors in the system. If no free processors are available, the job joins the end of the queue. When a job completes, the released processors are allocated to waiting jobs approximately in proportion to their *avg* (*pws*) such that each job is allocated an integral number of processors that is at least one but no more than *avg* (*pws*). As before we use the prefix S (e.g., Adaptive-SAVG) to indicate smallest *avg* first scheduling.

SDF-AVG: Jobs are scheduled as under SDF except that *avg* processors are allocated rather than N.

EQ-AVG: Processor allocation proceeds in two phases. In the first phase processors are assigned to jobs as per EQ except that *avg* is used instead of available parallelism (N). If there are idle processors left after phase one, the idle processors are allocated to jobs in an EQ fashion using $N - avg$ instead of N.

4. Use of *avg* and *pws* in Processor Scheduling

In this section we examine the use of *avg* and *pws* in processor scheduling. The ASP policy, which has been studied in several papers [ST91, ST93, NSS93a, NSS93b, RSDS93][3] and is among the most promising RTC policies to date [ST93], decreases processor allocations as instantaneous load increases but does not use job demand or execution rate characteristics. The AVG and PWS policies, which have been widely studied under specific workload assumptions [Se89, MEB91, GST91, ST93, SRDS93], serve as starting points for understanding whether execution rate characteristics are beneficial.

In section 4.1 we compare the performance of ASP against AVG and PWS policies under similar workload assumptions as in [ST93], as well as under new workloads. We then modify AVG to decrease processor allocations as queue length increases (i.e., Adaptive-AVG), in order to compare policies that do and do not use execution rate characteristics on a more equitable basis. Section 4.2 compares ASP with Adaptive-AVG. Finally, to enable an unbiased comparison of ASP and Adaptive-AVG we extend ASP to include a maximum allocation constraint as in [RSDS93]. Sections 4.3 and 4.4 evaluate whether the maximum allocation constraint improves the performance of ASP and provide a simple rule of thumb for the "max" value. A policy that uses a load-adaptive formula for "max", ASP-max+, is also considered and is compared against the A+&mM policy [Se89].

Section 4.5 provides results for additional workloads not considered in sections 4.1-4.4. Section 4.6 discusses how the results in sections 4.1-4.5 can be used to explain apparent discrepancies in the previous literature. Finally, in section 4.7 and 4.8 we investigate the potential benefit of *avg* or a maximum allocation constraint for SDF, an RTC policy that uses job demand information, and the dynamic EQ policy, which is used throughout the section as a target for RTC policy performance.

The mean system response times reported in this section were estimated using discrete event simulation. All simulation estimates have 95% confidence intervals with half-widths typically less than 5%. The regenerative method was used to compute confidence intervals whenever possible. Otherwise, the method of batch means was used.

Most experiments consider systems with P=100, although systems with P=16 and P=64 are also considered when the impact of P is relevant. We set $\bar{D} = P$ so that offered load $\rho \equiv \lambda\bar{D}/P = \lambda$. Since parallel systems often have high load and since distinctions among policies are more pronounced at high load, ρ is typically held fixed at 0.7 when studying the sensitivity of mean response time to other workload characteristics. Other loads are also considered when ρ is varied between 0 and 0.9. In most experiments we set C_v=5. Initially results are provided for exponential demands (C_v=1), whereas section 4.5 provides results for $C_v = 30$. Two ERFs in figure 2.2 are used in most of the experiments: the highly sublinear ERF $\beta = 70$, or the more nearly linear execution rate $\beta = 300$. Workloads that allow different classes of jobs to have different ERFs are defined in section 4.5.

[3]Under the workload assumptions of [RSDS93] their EPM policy with maximum allocation equal to number of system processors reduces to the ASP policy.

4.1. ASP versus PWS and AVG

Setia and Tripathi [ST93] show that ASP outperforms PWS at moderate to high loads for a two-class workload with exponential demands per class, constant available parallelism N=P=7 in each class, with one class having close to linear ERF and the other class having a highly sublinear ERF. The overall coefficient of variation in demand, C_D, in their experiments was close to 1. We obtain similar results using our model for a similar workload except that $N = P = 100$ and all jobs have highly sublinear ERFs ($\beta = 70$). We also compare the policies at a higher C_D. In addition to ASP and PWS, we also consider the AVG policy.

Figures 4.1a and b plot the mean response times of PWS, AVG,

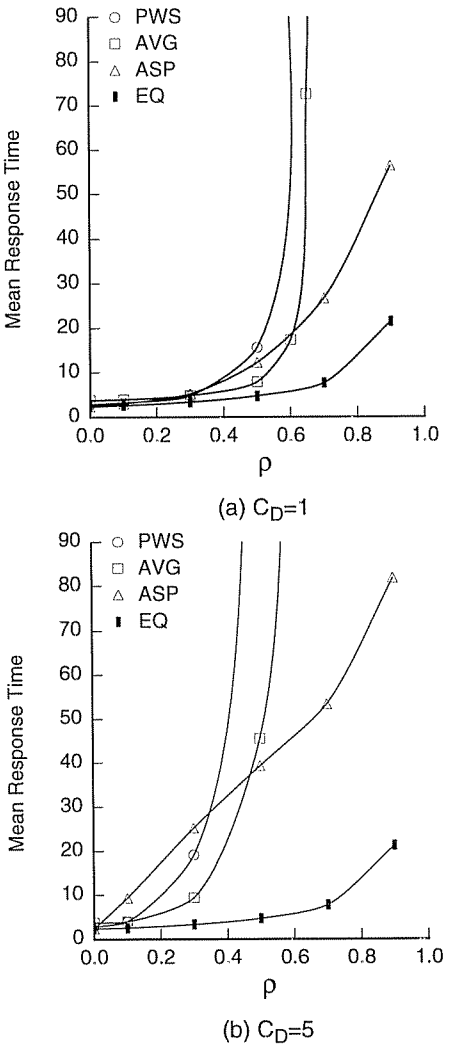

Figure 4.1: Comparison of PWS,AVG and ASP
P=100; $\bar{D}$=P;
N=P; β=70

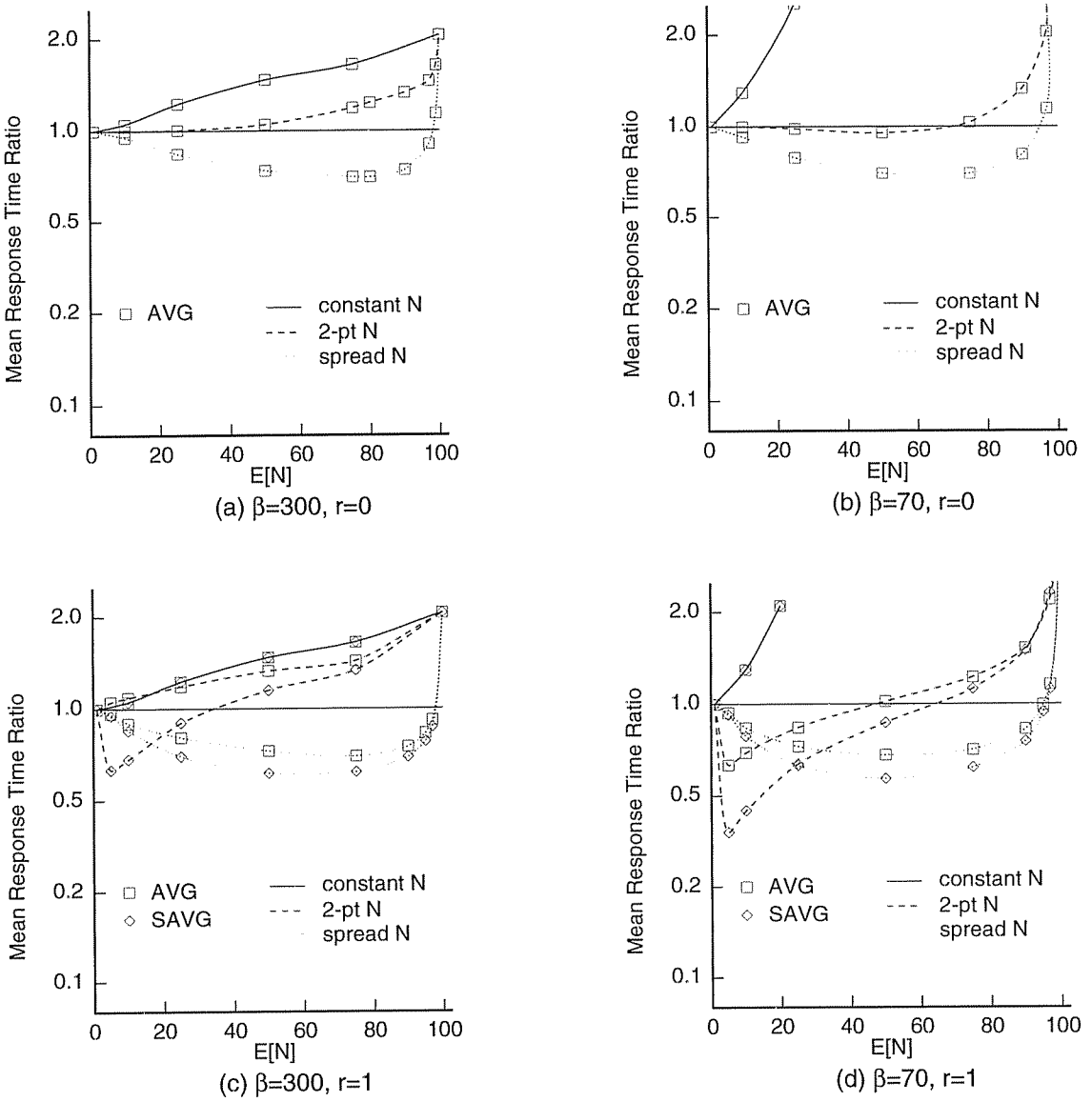

Figure 4.2: Mean Response Time Ratios of AVG,SAVG to ASP
P=100; $\overline{D}=P$
ρ=0.7; C_v=5

ASP, and EQ versus offered load for a 100-processor system with $N=P$, β=70, and C_D=1 and 5 respectively. The data in figure 4.1a is in agreement with the results in [ST93], namely that for a workload with C_D close to 1, $N=P$, and a sublinear ERF, ASP outperforms PWS at moderate to high load, and the PWS system saturates before ASP. We also note that AVG outperforms PWS for the given workload, but has similar saturation behavior. Finally, we observe that the mean response times of all three RTC policies are substantially higher than R_{EQ} when ρ > 0.5. In figure 4.1b we see similar trends for C_D=5, but the response time differences are more pronounced since the performance of the RTC policies has degraded with C_D but that of EQ has not.

We now examine the sensitivity of the relative performance of PWS, AVG, and ASP to the distribution of N, ERF sublinearity (β), and workload correlation. We consider constant, 2-point, and spread N distributions with $\overline{N}$ varying from 1 to P. We consider β = 70,300 and both uncorrelated (r=0) as well as fully correlated (r=1) workloads, and we set ρ=0.7 and C_v=5. Figures 4.2a and b plot the *ratio* of mean response times of AVG to ASP versus $\overline{N}$ for the uncorrelated workload. Figures 4.2c and d give the same for the correlated workload. For the correlated workload the SAVG policy performance is shown as well. The curves for PWS are not given since they depict trends similar to the AVG curves, and since AVG outperforms PWS for all the given parameter settings.

Figures 4.2a and b show that the relative performance of AVG and ASP is sensitive to the distribution of N. In fact, for the constant N distribution (considered in [ST93]), ASP outperforms AVG and PWS for all $\overline{N}$. However for the 2-point N workload ASP and AVG are roughly equivalent for $\overline{N} < 0.5P$, and for the spread N distribution AVG has higher performance than ASP for a large range of $\overline{N}$. Figures 4.2c and d show that the relative performance of AVG and ASP under r=1 is similar to that under r=0, except for the 2-point N distribution at low $\overline{N}$.

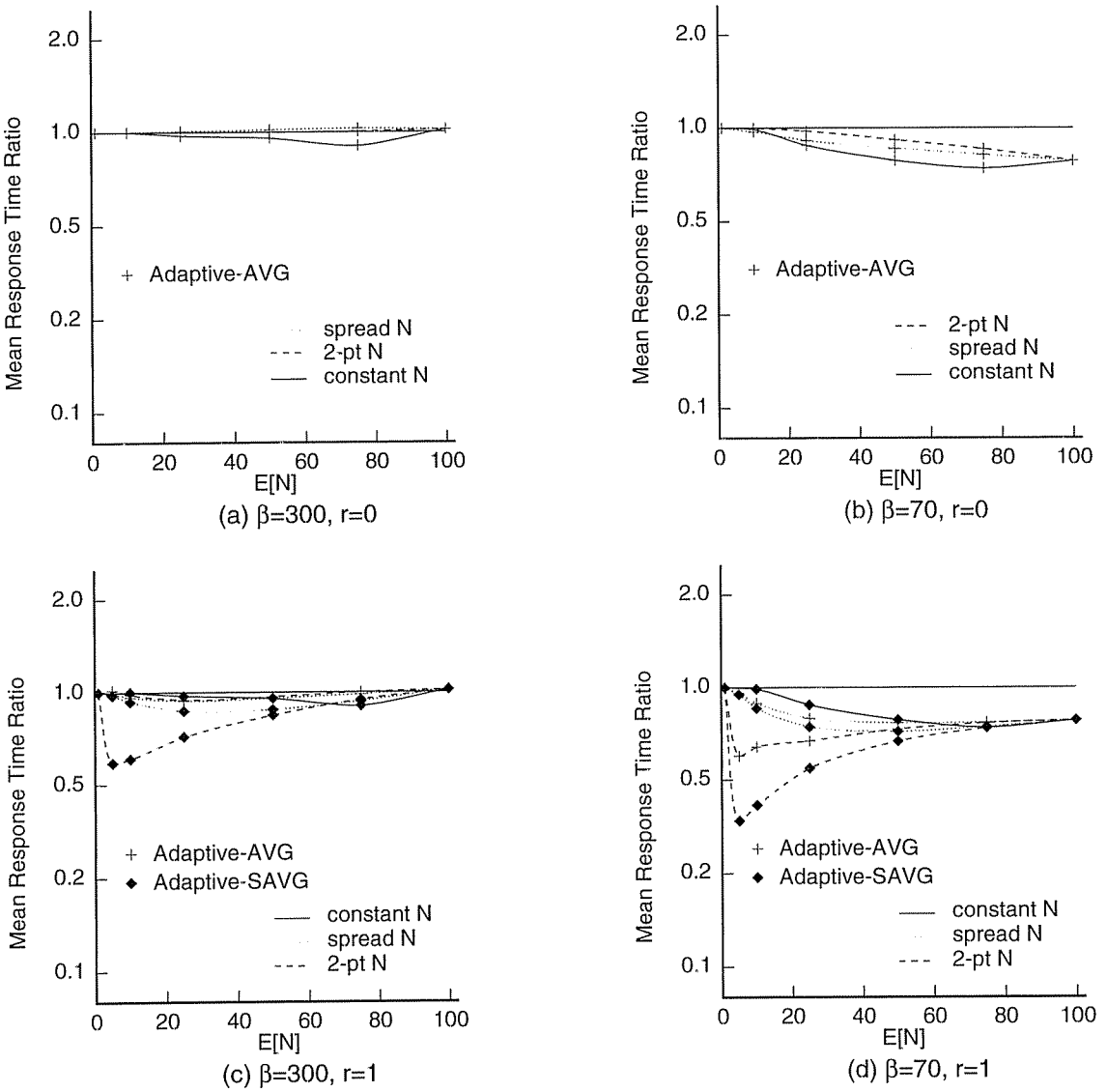

Figure 4.3: Mean Response Time Ratios of Adaptive-AVG,Adaptive-SAVG to ASP

P=100; $\bar{D}=P$
ρ=0.7; $C_v=5$

When r=0 AVG and SAVG have roughly equal mean response times (not shown). However when r=1, SAVG performs noticeably better than AVG. Furthermore, the performance improvement of SAVG compared with AVG increases with C_N, as SAVG more accurately differentiates small and large demand jobs.

The curves in figure 4.2 indicate that the system saturation point for AVG and SAVG decreases with increase in $\bar{N}$, decrease in β, and decrease in the spread of N (i.e., as we move from spread N to 2-point N to constant N).

4.2. ASP versus Adaptive AVG

AVG and ASP differ both in whether or not they use application execution rate characteristics and in whether or not they decrease processor allocation as queue length increases. (The latter difference is the reason that the AVG system saturates much sooner than ASP, which in turn is a major factor in its poorer performance.) To more accurately assess the benefits of using execution rate characteristics in scheduling, we next compare ASP against the Adaptive-AVG policy, both of which adapt to queue length. (As before, Adaptive-PWS performs somewhat worse than Adaptive-AVG under the assumed workload conditions, and thus is not shown.) Figure 4.3 plots the ratio of $\bar{R}_{Adaptive-AVG}$ to $\bar{R}_{ASP}$, for the same workloads as in figure 4.2. Comparing figure 4.3 with figure 4.2, the relative performance of Adaptive-AVG is much less sensitive to the distribution of N than that of AVG, and has qualitatively similar or higher performance than AVG *except* for spread N workloads with β=300. More importantly, Adaptive-AVG has roughly the same stability as ASP and generally performs as well as or better than ASP for all the given workloads. In particular, the performance of Adaptive-AVG is noticeably better than that of ASP for more sublinear ERFs (β=70). As in SAVG versus AVG, Adaptive-SAVG has higher performance than Adaptive-AVG when r=1, especially as C_N (and C_D) increases.

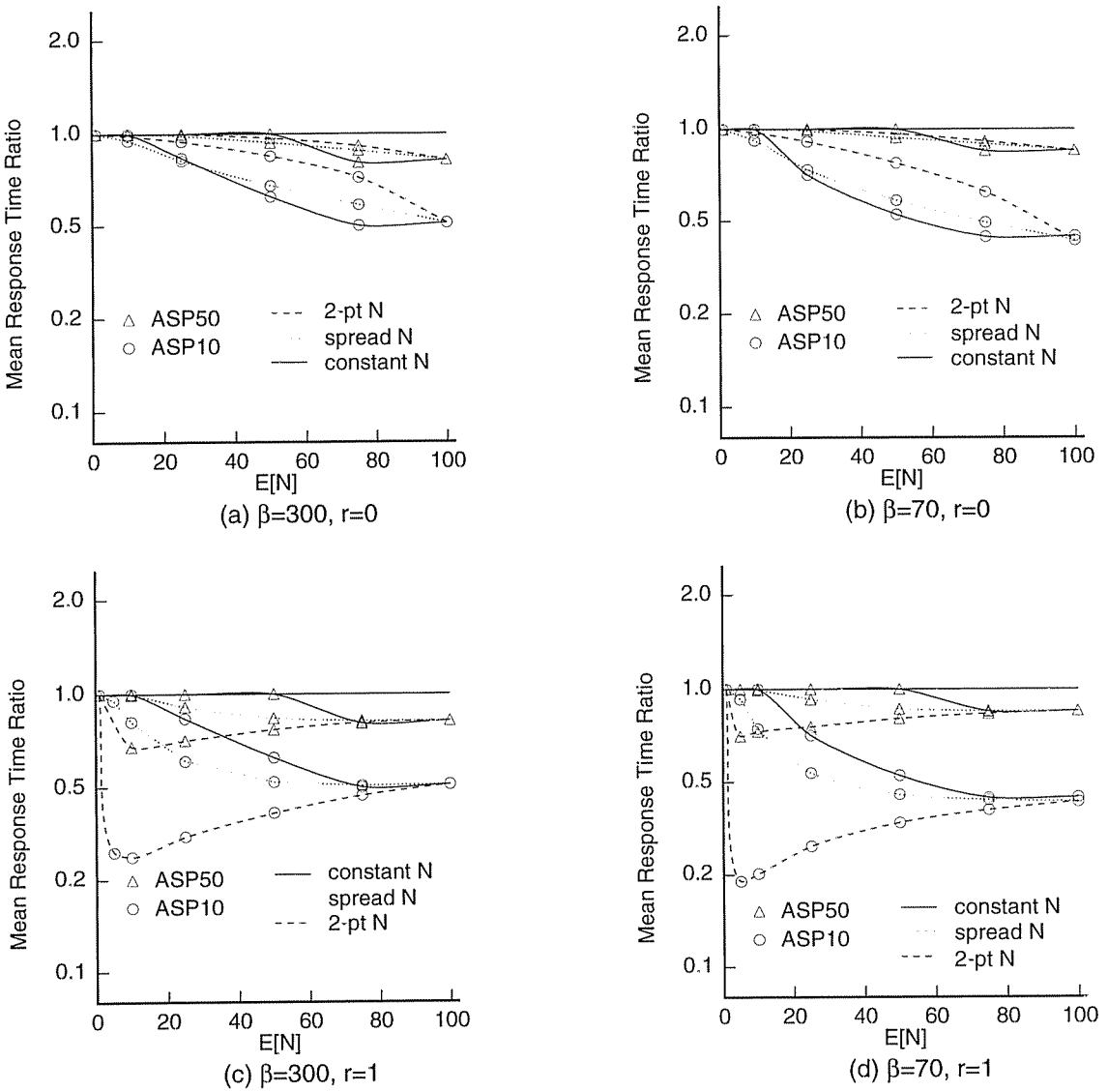

Figure 4.4: Mean Response Time Ratios of ASP-max to ASP
P=100; $\overline{D}=P$
ρ=0.7; $C_v=5$

4.3. Performance of ASP-max

The comparison in section 4.2 suggests that using application characteristics such as *avg* improves the performance of processor scheduling. However, we note that Adaptive-AVG contains a maximum allocation constraint i.e., *avg*, whereas ASP contains no such constraint. This motivates adding a maximum allocation constraint to ASP (possibly independent of execution rate characteristics) in order to perform a still more careful examination of this issue.

The ASP-max family of policies has a maximum allocation constraint equal to "max" and schedules jobs just like ASP except that for a job with available parallelism N it does not allocate more than *min* (*N*,*max*) to the job. The EPM policy in [RSDS93] is very similar and has the same mean response time as ASP-max.[5]

Figure 4.4 plots the ratio of $\overline{R}_{ASP10}$ and $\overline{R}_{ASP50}$ to $\overline{R}_{ASP}$ for a 100 processor system and the same workloads as in figures 4.2 and 4.3. We observe that (1) the performance of each ASP-max policy relative to ASP is insensitive to β, (2) ASP50 has very similar performance to the Adaptive-AVG policy at both β=300 and β=70, and (3) ASP10 significantly outperforms ASP, Adaptive-AVG, and Adaptive-SAVG for the entire parameter space in the figure. We also ran experiments with a fixed maximum allocation constraint on Adaptive-AVG but found no improvement over the performance of ASP-max. Hence, for these policies that differ only in whether or not job execution rate characteristics are used in deciding how many processors to allocate to a job, it appears that using the job execution rate information does not improve policy performance.

[4]The difference between EPM and ASP-max arises when a job completes and the number of free processors is not a multiple of the number of waiting jobs. In this case ASP-max allocates the excess processors to jobs that arrived earlier whereas EPM does so for jobs that arrived later.

For the uncorrelated workloads in figure 4.4, the performance of ASP-max relative to ASP improves gradually with $\bar{N}$ for all three distributions of N. When r=0, the relative performance of ASP-max improves with decrease in C_N. When r=1, the trend is reversed since as seen from equation (2.1) when r=1 a high C_N implies a high C_D.

4.4. Selecting the value of max

A key point from Figures 4.4a-d is that the relative order of ASP10, ASP50, and ASP is insensitive to the distribution of N, the correlation (r) between demand and parallelism, and β in the range (70,300). In this section we further explore the extent to which the optimal value of "max" is sensitive to these parameters, as well as to load (ρ) and C_v which were held fixed in Figure 4.4.

The solid curves in Figures 4.5(a) depict the ratio of $\bar{R}_{ASP-max}$ to $\bar{R}_{ASP}$ versus ρ for various values of "max", assuming a 2-point distribution of N, $\bar{N}=75$, $P=100$, $\beta=300$, and $C_v=5$. Figure 4.5(b) shows the corresponding curves for the spread N distribution. As expected, the optimal value of "max" decreases as ρ increases. However, the performance of the ASP-max policies for a given load in Figure 4.5 is not highly sensitive to variations in "max" near the optimal value, and thus the ASP20 policy performs very well for $\rho \geq 0.3$. This conclusion holds, and the curves are qualitatively very similar, for other values of $\bar{N}$ ($\bar{N}$=25,50), a constant distribution of N ($N=100$), *higher sublinearity in the ERF* ($\beta=70$), as well as correlated workloads ($r=1$). We also ran experiments for a system with $P=64$ processors and $C_v=5$, and found that the ASP12 policy performed very well for all experiments conducted, which included constant, 2-point and spread N distributions, and β in the range of 50 to 200. Finally, we ran a small number of experiments for $P=16$ under constant N as in [RSDS93] and found that when $C_v=5$, the ASP3 and ASP4 policies have high performance. Thus, for $C_v=5$ and $\rho \geq 0.3$, a very simple rule of thumb, "max" $=0.2P$, appears to give near-optimal performance unless the ERF is extremely sublinear (e.g., $\beta=20$ for $P=100$).[5] Experiments for other values of C_v show that the value of max should increase for lower C_v (results omitted to conserve space) and decrease for higher C_v (as will be shown in section 4.5).

An alternative to the simple rule of thumb, "max" $=0.2P$, is to let the value of "max" vary with the measured average system load. To investigate this approach, the ASP-max+ policy uses equation (3.1) with *avg* replaced by $0.2P$ (and $\pi=0.5$) to compute a load-dependent value for "max". Note that this policy can be compared against the A+&mM policy [Se89] reviewed in section 3.1 to determine the benefits of using application execution rate characteristics in a new context. The performance of the ASP20+ and the A+&mM policies are shown with dashed and dotted curves, respectively in Figure 4.5. Both policies perform nearly as well as the optimal ASP-max policy at all values of ρ, although the A+&mM policy performs somewhat less well for the 2-point distribution of N. In any case, the ASP20+ policy which allocates processors independently of β, performs at least as well as the A+&mM policy which requires knowledge of *avg* (i.e., β) and m.

[5]The experimental data in [RSDS93] (figure 4) show that the EPM4 policy (max $=0.25P$) is close to the optimal EPM policy at moderate to high loads for sublinear ERFs and at high loads for close to linear ERFs. Thus their data corroborates the stated rule of thumb regarding the optimal value of "max".

4.5. Comments on C_v=30 and Multi-class workloads

The experiments in previous sections assume a fixed value of β and $C_v=5$. In this section we consider multi-class workloads where each class has the same distribution of N and D, but a different value for β. We have observed that for a wide range of uncorrelated 2-class workloads with C_v=5, the relative performance of various policies (not shown) is qualitatively the same as for the corresponding single-class workloads.

Figure 4.6 gives the mean response time ratios of AVG, Adaptive-AVG, A+&mM, ASP10, ASP20, ASP20+, and EQ to ASP as a function of ρ for $C_v=30$, $P=100$, the spread N distribution with $\bar{N}$=75, and β defined by the following pmf: P(70)=0.15,

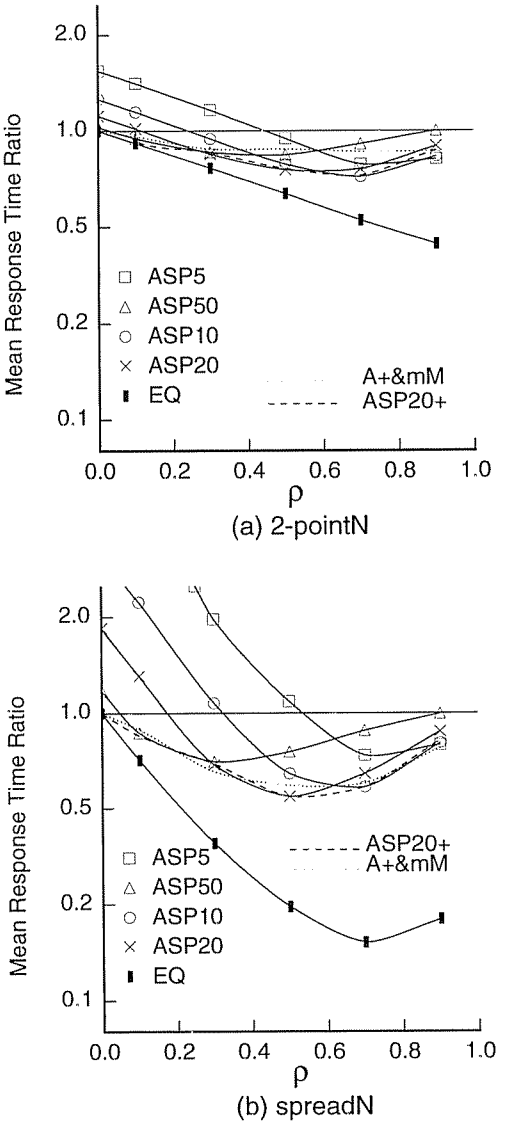

Figure 4.5: Selecting "max" for ASP-max

($\bar{R}_\Psi/\bar{R}_{ASP}$, $\Psi \in \{ASP-max, ASP20+, A+\&mM\}$)

P=100; $\bar{D}=P$

$\bar{N}$=75; β=300; C_v=5; r=0

P(200)=0.2, P(300)=0.45, and P(10000)=0.2. (Note the new scale on y-axis in this figure.) Due to the time consumed in the simulations most curves are plotted only for $\rho \leq 0.7$. The results show essentially the same relative policy performance as for the single-class workloads with $C_v = 5$. In particular, ASP-max significantly outperforms Adaptive-AVG which outperforms ASP, and EQ significantly outperforms all of the RTC policies. We conclude that a maximum allocation constraint that depends only weakly on average system load and/or C_v is an important characteristic of high performance RTC scheduling. We also observe that the simple rule of thumb for "max" should be somewhat smaller (i.e., less than 20% of P) for larger C_v, as mentioned in section 4.4.

The *A+&mM* policy outperforms the Adaptive-AVG policy, but is significantly poorer than the ASP-max and ASP20+ policies, perhaps because it allocates proportionately more processors to the class of jobs with nearly linear execution rates yet some of these jobs will have very long service times. We have not investigated this issue in detail.

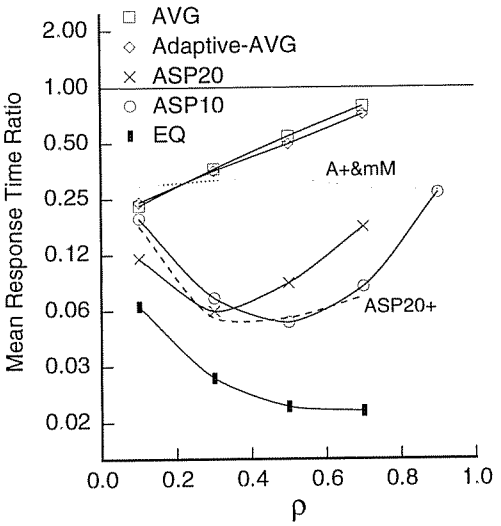

Figure 4.6: Performance Comparison at C_v=30
($\bar{R}_\Psi/\bar{R}_{ASP}$, $\Psi \in \{AVG, AAVG, A+\&mM, ASP-max\}$)
P=100; $\bar{D}=P$
spread N; $\bar{N}$=75; $\beta \in \{70,200,300,10000\}$; r=0

4.6. Unification of Previous Results

The experiments in sections 4.1-4.5 indicate that, for $C_v > 1$ application execution rate characteristics are not very useful for determining processor allocations in RTC policies. This is also true for exponential demands [AMV93]. In retrospect, the intuition behind this result is that policy performance degrades when jobs with long execution times are allocated too many processors. Since execution rate characteristics have not been shown or assumed to be correlated with demand, they are poor predictors of this behavior.

As shown in sections 4.1-4.2, [ST93] reached similar conclusions by comparing the PWS and ASP policies for workloads with constant available parallelism and exponential job demands. Four studies have reached the opposite conclusions [EZL89, Se89, MEB91, and GST91], based on the following detailed results. Sevcik [Se89] shows that policies that use application characteristics *and adapt to load* (i.e., A+&mM and a simpler policy A+) are superior to policies that do neither. Sevcik [Se89] and Majumdar et. al. [MEB91] show that AVG is nearly as good as the load-dependent optimal fixed partitioning policy at low to moderate load. Note that in this case neither policy decreases the processor allocations when instantaneous load increases, but both policies have a maximum allocation constraint. Finally, Ghosal et. al. [GST91] compare four *pws*-based policies, and show that the two policies that have a maximum allocation constraint equal to *pws* outperform the two policies that have no maximum. Each of these detailed results makes sense in the context of the experiments presented in sections 4.1-4.5.

Rosti et al. [RSDS93] give experimental data under exponential demands (C_v=1) to show that an adaptive partitioning policy named AP that does not have a maximum allocation constraint performs almost as well as the optimal EPM (i.e., ASP-max) policy. However, their data for non-exponential demands ($C_v = 3$) shows that the AP policy performs worse than ASP-max (EPM in their paper) as C_v increases.

We note the A+ policy of [Se89] was shown to have poor performance by Smirni et al. [SRDS93]. Under their workload assumption that the maximium and minimum parallelisms are P and 1, respectively, for all jobs, A+ is equivalent to A+&mM. We remark that our implementation of A+&mM is work-conserving, while their implementation of A+ is not.

Smirni et al. proposed a heuristic estimate of the lower bound on the performance of RTC policies [SRDS93]. The lower bound estimate is derived by using the best fixed partitioning policy with equal size partitions and FCFS scheduling at each value of system load. We have verified that the performance of ASP-max lies within this envelope for exponential demands. However, for C_v=5 the ASP-max policy has better performance than the envelope at moderate to high loads ($\rho \geq 0.5$). Also, Naik et al. [NSS93b] have observed that for a workload with high C_v, ASP performs as well as the envelope of best fixed partitioning policy performance. Thus, of all the RTC policies proposed to date that do not use job demand information, ASP-max appears to have highest performance for $C_v \geq 1$.

4.7. Use of *avg* and max for SDF

We now investigate the use of *avg* or an independent maximum allocation constraint for the RTC policy known as shortest demand first (SDF). We compare the SDF policy with the SDF-AVG policy that allocates at most *avg* processors to a job, the SDF-max policy that allocates at most "max" processors to a job, and the ASP-max policies which do not make use of demand information.

Figure 4.7 plots the ratios of mean response time for SDF, SDF-AVG, SDF20, ASP20, and EQ to the mean response time for ASP as a function of $\bar{N}$ for 2-point and spread N distributions, assuming a 100 processor system, $\rho = 0.7$, $C_v = 5$, and $\beta = 300$. For constant N (not shown) the relative performance of policies was similar to figure 4.7b. Note that for the 2-point N workloads, SDF and SDF-AVG have poorer performance than ASP-max over a wide range of $\bar{N}$. However, for all the given workloads, SDF20 performs appreciably better than SDF-AVG and SDF, and is always competitive with or better than ASP-max. Thus, for the SDF policy, using a fixed value of "max" such as 20% of P for the maximum allocation provides better performance than using *avg*, and is critical to the policy's competitiveness. We found this to be true for a range of values of β (50 to 500), for correlated workloads, and at higher system loads as well. For extremely sublinear ERFs (i.e., $\beta < 50$) SDF10 outperforms SDF20. Thus, the optimal value of "max" is weakly correlated with *avg*, but this is only significant for highly sublinear ERFs. We also ran experiments for C_v=1 and β=300 and found that, as with the ASP policies, the optimal value for "max" increases as C_v decreases if the ERF is close to linear.

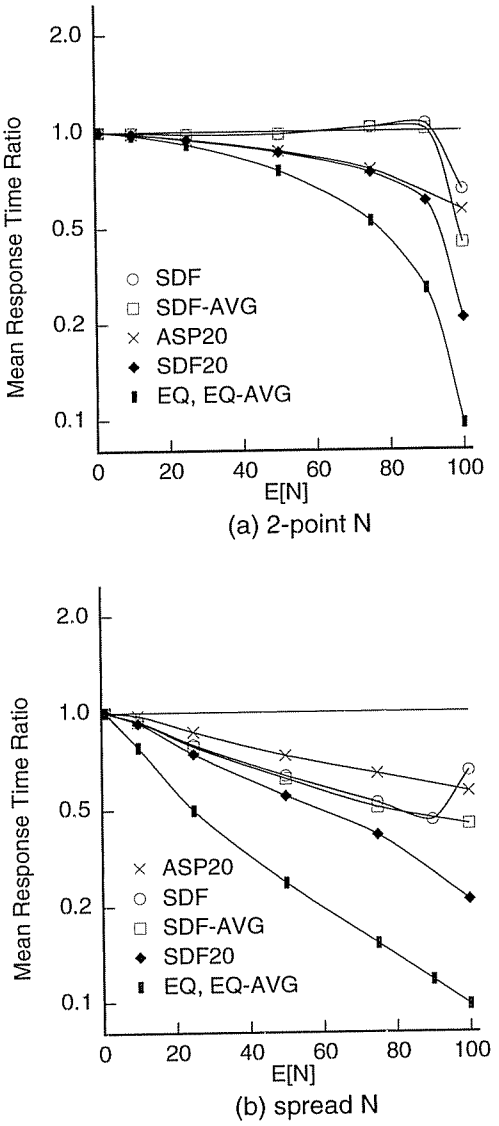

Figure 4.7: Performance of SDF-AVG and SDF-max
(R_Ψ/R_{ASP}, $\Psi \in \{SDF, SDF\text{-}AVG, SDF20, ASP20, ASP\}$)
P=100; $\overline{D}$=P
ρ=0.7; C_v=5; β=300; r=0

4.8. Use of *avg* and max for EQ

In this section we report on experiments that examined using *avg* and/or a maximum allocation constraint to improve the performance of EQ. Related work shows that limiting the maximum allocation under EQ degrades performance [MV93b]. This motivated us to use *avg* for EQ in a way that the allocation to jobs should increase beyond their *avg* if processors are idle, which led to the EQ-AVG policy defined in section 3.2. We however, found that for a variety or workloads with very sublinear as well as close to linear ERFs $\overline{R}_{EQ-AVG} \approx \overline{R}_{EQ}$.[6] Thus, as for ASP and SDF, we have not found *avg* to be useful for the EQ policy.

[6]We also observed similar behavior for a two class workload with class dependent ERFs.

5. Benefits of Limited Preemption

In section 4 we found that SDF-max and ASP-max+ are among the highest performance RTC policies that use and don't use job demand information, respectively. However, neither of these policies is competitive with the idealized EQ policy, as was shown in Figures 4.5-4.7. One reason is that under RTC policies processor allocation cannot be increased in the middle of a job's execution to make use of idle processors. Another reason is that large demand jobs cannot be preempted in favor of small demand jobs. As a result the performance of RTC policies degrades with increase in coefficient of variation of service demand. This motivates the use of a single preemption to improve the performance of RTC policies. In this section we investigate this alternative by defining an appropriate class of policies, discussing choice of policy parameters, and evaluating policy performance with respect to EQ.

5.1. The Two-level, FCFS/ASP-max Policy

One way to implement limited preemption is to use a multilevel (ML) queueing discipline [Kl76] in which arriving jobs join the highest priority queue and thereby quickly receive a small amount of service. Leutenegger [Le90] has studied the behavior of two-level queues for specific dynamic parallel processor policies. Here we consider a two level queue with FCFS scheduling in the first queue and ASP-max scheduling in the second level queue. Thus, each job is preempted at most once during it's execution.

The two-level (TL) system operates as follows. P_1 processors are reserved for the first level queue. (We denote this set of processors as the "first" partition). The remaining $P_2 \equiv P - P_1$ processors (i.e., the "second" partition) can serve either queue, with the first level having non-preemptive priority over the second. An arriving job joins the first level FCFS queue. When it reaches the head of the queue, it is scheduled on the partition with the greater number of available processors and is allocated the maximum of its available parallelism N and the number of available processors in the partition. The job receives uninterrupted service up to a quantum of Q units of cumulative demand. If the job requires further service it is preempted and placed in the second level queue where the scheduling policy is ASP-max on the second partition. Once scheduled from the second level queue, the job runs to completion.

5.2. Guidelines for Choosing Q and P_1

The key issues in obtaining high performance for the TL system are: the choice of the quantum size Q, and the choice of partition size P_1 (or equivalently P_2). Below we propose a specific method to choose Q and P_1. Further investigation is needed to fully evaluate the viability of the approach. For the following discussion, let D_1 be the service requirement of jobs in the first level queue and likewise D_2 in the second level queue. (Note that D_2 is undefined if total processing demand $D<Q$.)

Since scheduling is non-preemptive in each level of the TL queue it is desirable to keep the coefficients of variation of each of D_1 and D_2 as low as possible. If Q is close to zero then C_{D_1} is close to zero and C_{D_2} is close to C_D. As Q increases C_{D_1} increases until it reaches the limiting value of C_D. By increasing Q we likewise expect C_{D_2} to decrease until it reaches a limiting value.

Due to the opposing trends of C_{D_1} and C_{D_2} with increase in Q it is not possible to minimize both of them simultaneously. A possible heuristic, and the one used in the reported experiments, is to choose that value of Q at which C_{D_1} and C_{D_2} cross over. This balances the coefficients of variation across both phases of service in

the TL system, but results in a value of Q that can be very sensitive to the distribution of demand. Alternative approaches are possible, e.g., choosing a Q that is biased towards keeping C_{D_1} low or towards keeping C_{D_2} low. These did not result in better performance for the specific demand distributions that we examined.

Choosing P_2 appropriately generally requires experimental results for a range of possible values. The search can be reduced by deriving a lower bound on P_2 for a given Q. The job arrival rate for the second phase of service is $\lambda_2=\lambda \cdot P[D>Q]$. Thus, the minimum requirement for stability in the second partition is that $\lambda_2 \overline{D}_2/P_2<1$, or $P_2>\lambda_2 \overline{D}_2$. To determine a suitable choice for P_2, we first computed the lower bound for P_2 at the maximum value for λ and then simulated the system for a range of values of P_2 between the lower bound and P.

5.3. Performance of TL-FCFS/ASP-max

We provide experimental data comparing the performance of the TL-FCFS/ASP-max policy against that of ASP, ASP-max, and EQ for a system with P=100, $\overline{D}=P$ and $C_v=5$. We use the 20% rule of thumb for the value of "max" when $\rho \geq 0.3$ and use max=P (i.e., ASP) to obtain higher performance at lower loads. For the specific H_2 distribution our heuristic for Q yielded Q=175 and using simulation we found a good choice of P_2 to be 80 for β=70 and β=300, r=0 and r=1, across different values of $\overline{N}$ and for all three distributions of N. As a result we considered the use of ASP16 in the second level queue at moderate to high loads and ASP (i.e., "max" $=P_2$) at low loads.

Figure 5.1 plots the mean response time ratios of ASP-20, TL-FCFS/ASP-16, TL-FCFS/ASP, and EQ to ASP, as a function of ρ, for the spread distribution of N. Similar results were obtained for the other two distributions of N. We observe that the TL-FCFS/ASP-max policy (combination of TL-FCFS/ASP at $\rho < 0.5$ and TL-FCFS/ASP16 at $\rho \geq 0.5$) has higher performance than ASP-max. More specifically, if $\overline{R}_{EQ}$ is viewed as the target mean response time for RTC policies, the relative gain of ASP-max over ASP is at most 1/2 the relative gain of EQ over ASP, and the maximum relative improvement is at $\rho = 0.3 - 0.5$. On the other hand, the relative gain of the TL-FCFS/ASP-max policy is *at least* 1/2 of the relative gain of EQ for all $\rho \leq 0.9$.

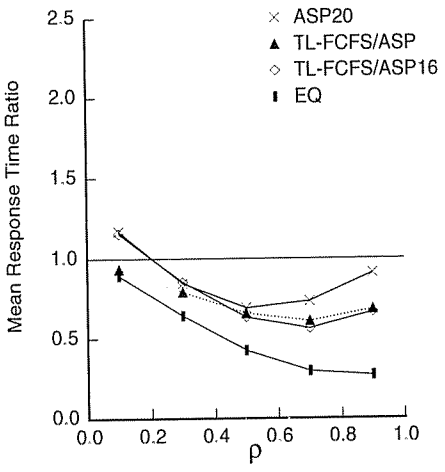

Figure 5.1: Performance of Single Preemption
($\overline{R}_\Psi/\overline{R}_{ASP}$, $\Psi \in \{TL\text{-}FCFS/ASP\text{-}max, ASP\text{-}max, EQ\}$)
$\underline{P}$=100; $\overline{D}=P$
spread N; $\overline{N}$=50; β=300; C_v=5; r=0

In spite of the improved performance of the TL-FCFS/ASP-max policy relative to ASP-max, this simple limited preemption policy is not competitive with EQ. Thus, it appears likely that the EQ policy with a lower bound on the time between preemptions [NSS93a, NSS93b] has higher performance than is achievable with policies that allow at most one preemption per job. Further experimentation is needed to confirm this conjecture.

6. Conclusions

This paper has examined the issues of whether application execution rate characteristics and/or limited preemption can be used to improve RTC policy performance, and has compared the performance of RTC policies to that of the idealized dynamic EQ policy. The experiments have included a wide range of workload characteristics, including $C_D = 5$ as well as $C_D = 1$ and $C_D = 30$, various distributions of job parallelism, zero as well as full correlation between job demand and parallelism, and highly sublinear as well as nearly linear execution rates. Note that most previous studies of RTC policies have assumed workloads where $C_D = 1$ and all jobs have parallelism equal to the number of processors.

Wherever applicable, our results are in agreement with and serve to unify previous results in the literature. In particular, apparent contradictions in the literature were shown to be due to the particular policies and/or workloads examined. We have also provided new observations and reached different overall conclusions in some cases, due to the additional workloads and policies we have considered. For example, we have shown that the AVG and PWS policies can outperform ASP (even at moderate to high load) and that the relative performance of these policies is sensitive to the distribution of parallelism. We have also shown that when $C_D = 5$, Adaptive-AVG consistently outperforms ASP, but ASP-max and ASP-max+ outperform Adaptive-AVG and appear to have highest performance among all RTC policies that do not use job demand information to date. Likewise, results were obtained that show SDF-AVG performs only slightly better than SDF (at high $\overline{N}$), but SDF-max consistently outperforms SDF-AVG.

We conclude that a maximum allocation constraint that depends very weakly on system load and/or on C_v, rather than the use of application execution rate characteristics, is an important characteristic of high-performance RTC scheduling. Experimental results for the EQ policy also showed that a particular approach to using the *avg* measure to improve policy performance did not in fact lead to improved performance. The intuitive explanation for why the use of application execution rate characteristics does not improve policy performance at $C_D > 1$ is that system performance degrades when jobs with long execution times are allocated too many processors and job execution rate characteristics are poor predictors of this behavior. Thus, for example, when processors are allocated in proportion to *avg*, large demand jobs with high *avg* can consume an unreasonable fraction of processing power, outweighing the benefit of *avg* being an efficient operating point.

Concerning limited preemption, we studied a relatively simple two-level queueing discipline, with non-preemptive scheduling at each level (i.e., FCFS and a load-adaptive ASP-max, respectively). Preliminary results show that this policy has better mean response time than the ASP-max policy over the workloads we have considered, but is not competitive with dynamic equiallocation policies.

Acknowledgments

The authors thank Ken Sevcik for several discussions and suggestions that improved the quality of this paper.

References

[AMV93] R. Agrawal, R. Mansharamani, M. Vernon. Response Time Bounds for Parallel Processor Allocation Policies. Technical Report 1152, Computer Sciences Dept., University of Wisconsin–Madison, June 1993.

[Cr91] Cray Research, private communication, 1991.

[Do88] L. Dowdy. On the Partitioning of Multiprocessor Systems. Technical Report, Vanderbilt University, July 1988.

[EZL89] D. Eager, J. Zahorjan, and E. Lazowska. Speedup Versus Efficiency in Parallel Systems. *IEEE Transactions on Computers, 38*, 3 (Mar. 1989), 408-423.

[GST91] D. Ghosal, G. Serazzi, and S. Tripathi. The Processor Working Set and Its Use in Scheduling Multiprocessor Systems. *IEEE Transactions on Software Engineering* 17, 5 (May 1991), 443-453.

[Kl76] L. Kleinrock. *Queueing Systems, Volume II: Computer Applications*. Wiley, New York, 1976.

[Le90] S. Leutenegger. Issues in Multiprogrammed Multiprocessor Sharing. Ph.D. Thesis, Technical Report #954, Computer Sciences Dept., University of Wisconsin-Madison, Aug. 1990.

[LN91] S. Leutenegger, and R. Nelson. Analysis of Spatial and Temporal Scheduling Policies for Semi-Static and Dynamic Multiprocessor Environments. IBM RC #75594, IBM T.J. Watson Research Center, Aug. 1991.

[LV90] S. Leutenegger, and M. Vernon. The Performance of Multiprogrammed Multiprocessor Scheduling Policies. *Proc. 1990 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Boulder, CO, (May 1990), 226-236.

[MEB88] S. Majumdar, D. Eager, and R. Bunt. Scheduling in multiprogrammed parallel systems. *Proc. 1988 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Santa Fe, NM. (May 1988), 104-113.

[MEB91] S. Majumdar, D. Eager, and R. Bunt. Characterisation of Programs for Scheduling in Multiprogrammed Parallel Systems. *Performance Evaluation 13*, (1991), 109-130.

[MV93a] R. Mansharamani, and M. Vernon. Approximate Analysis of Parallel Processor Allocation Policies. Technical Report 1187, Computer Sciences Dept., University of Wisconsin–Madison, Nov. 1993.

[MV93b] R. Mansharamani, and M. Vernon. Properties of the EQS Parallel Processor Allocation Policy. Technical Report 1192, Computer Sciences Dept., University of Wisconsin–Madison, Nov. 1993.

[MVZ93] C. McCann, R. Vaswani, and J. Zahorjan. A Dynamic Processor Allocation Policy for Multiprogrammed, Shared Memory Multiprocessors. *ACM Transactions on Computer Systems 11*, 2 (May 1993), 146-178.

[NSS93a] V. Naik, S. Setia, and M. Squillante. Scheduling of Large Scientific Applications on Distributed Memory Multiprocessor Systems. *Proc. of the 6th SIAM Conference on Parallel Processing for Scientific Computation*, 1993.

[NSS93b] V.Naik, S. Setia, and M. Squillante. Performance Analysis of Job Scheduling Policies in Parallel Supercomputing Environments. *Proc. of Supercomputing'93*, November 1993.

[RSDS93] E. Rosti, E. Smirni, L. Dowdy, G. Serazzi, and B. Carlson. Robust Partitioning Policies of Multiprocessor Systems. To appear in *Performance Evaluation*, special issue on the performance modeling of parallel processing systems.

[Se89] K. Sevcik. Characterization of Parallelism in Applications and Their Use in Scheduling. *Proc. 1989 ACM SIGMETRICS and Peroformance '89 Int'l. Conf. on Measurement and Modeling of Computer Systems*, Berkeley, CA, (May 1989), 171-180.

[Se93] K. Sevcik. Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems. To appear in *Performance Evaluation*, special issue on the performance modeling of parallel processing systems.

[SRDS93] E. Smirni, E. Rosti, L. Dowdy, and G. Serazzi. Evaluation of Multiprocessor Allocation Policies. Technical Report, Vanderbilt University, 1993.

[ST91] S. Setia and S. Tripathi. An Analysis of Several Processor Partitioning Policies for Parallel Computers. Technical Report CS-TR-2684, University of Maryland, May 1991.

[ST93] S. Setia and S. Tripathi. A Comparative Analysis of Static Processor Partitioning Policies for Parallel Computers. *Proc. of the International Workshop on Modeling and Simulation of Computer and Telecommunication Systems (MASCOTS)*, January 1993.

[TG89] A. Tucker and A. Gupta. Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors. *Proc. of the 12th ACM Symposium on Operating System Principles*, Dec. 1989, 159-166.

[ZM90] J. Zahorjan and C. McCann. Processor Scheduling in Shared Memory Multiprocessors. *Proc. 1990 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Boulder, CO, (May 1990), 214-225.