Dynamic vs. Static Quantum-Based Parallel Processor Allocation *

Su-Hui Chiang and Mary K. Vernon
suhui@cs.wisc.edu *vernon@cs.wisc.edu*

Computer Sciences Department
University of Wisconsin
Madison, WI 53706 USA

Abstract. This paper improves upon previous synthetic workload models and compares the performance of dynamic spatial equipartitioning (EQS) and the semi-static quantum-based FB-PWS processor allocation defined in [23], under synthetic workloads that have not previously been considered. These new workloads include realistic repartitioning overheads and job characteristics that are consistent with system measurement, anticipated trends, and experience. The overall conclusion from the results is that the EQS policy is generally superior to the FB-PWS policy *even under realistic repartitioning overheads*. We find cases where the EQS system saturates earlier than the FB-PWS system, and vice versa. This leads to the definition of a modified EQS policy, called EQS-PWS, which has performance equal to or better than EQS and FB-PWS for all workloads examined in this paper.

1 Introduction

In static quantum-based parallel processor allocation policies, each job is configured for a static number of processors and timesharing is used to share the processors among the jobs. Ousterhout's coscheduling policies [21] are examples of this class of policies. Semi-static quantum-based policies allow limited changes in processor allocations as system load changes. Such policies greatly reduce the frequency of job reconfiguration (which can involve significant data repartitioning overheads) as compared with dynamic policies such as the spatial equipartitioning (EQS) policy.

In a recent paper [23] Parsons and Sevcik have proposed a new semi-static quantum-based parallel processor allocation policy, FB-PWS, that has the following characteristics:

load-adaptability: the number of processors allocated to a newly arriving job decreases as the number of jobs in the system increases,

processor working set (pws): as load increases, the allocation for a newly arriving job is proportional to it's *pws* measure [9], where *pws* is the num-

*This research was partially supported by the National Science Foundation under grants CCR-9024144, CDA-9024618, and GER-9550429.

ber of processors that permit the job to run at approximately the knee of its execution-time vs efficiency profile [15][1],

Multilevel-Feedback (FB): in each quantum, priority is given to the jobs that have so far received least service,

infrequent repartitioning: In each quantum, at most one job runs on a smaller number processors than it's initial allocation (and those processors would otherwise be idle).

They also define another semi-static quantum-based policy called FB-ASP that is similar to FB-PWS but does not use the *pws* measure.

Parsons and Sevcik show that, under particular workloads with job characteristics that have been observed in practice, the FB-PWS policy is competitive with EQS *even when repartitioning is assumed to have zero cost.* This is an impressive result since FB-PWS commits to a processor allocation at job arrival time. Previous static allocation policies have generally not been competitive with dynamic policies such as EQS under zero repartitioning cost [13, 29, 19, 4]. They also show that FB-PWS and FB-ASP can substantially outperform EQS under an ad hoc model of repartitioning costs that is intended to illustrate the possible impact of repartitioning overheads on relative policy performance.

In this paper, we further investigate the relative performance of EQS and FB-PWS. First, we examine how idealized EQS and FB-PWS compare for synthetic workloads that are not considered in [23], but that are designed to represent parallel workloads encountered in practice. An improved approach to representing application speedup characteristics is developed as part of this effort (see section 3). Second, we assess whether specific features of the FB-PWS policy might be incorporated in the EQS policy to improve performance. In particular, we consider a specific use of the *pws* measure to modify the processor allocations computed by the EQS policy, leading to a new policy called EQS-PWS. Finally, we consider how processor repartitioning might be handled in a practical implementation of EQS or EQS-PWS. We compare the EQS, EQS-PWS, and FB-PWS policies under workload models and repartitioning overheads that are based on recent system measurements ([8, 6, 7, 1, 10]) extrapolated to future production parallel systems.

We are primarily interested in evaluating relative policy performance for daytime workloads, and thus the principal measure of interest is mean job turnaround time. We note that results in the previous literature suggest that the EQS and FB policies provide good performance for small ("interactive") jobs. For example, the EQS policy like the *processor sharing* policy for uniprocessor systems, has the key property that expected response time is *proportional to* the job's service requirement. This property is called "fairness" in [12]. We also note that the FB-PWS policy has reduced potential for starvation of large jobs as compared with the FB policy for uniprocessor systems, due to it's load-adaptive space-sharing property. Further investigation of these and other more detailed measures is left for future work.

[1] Specifically, *pws* is the minimum number of processors that maximizes the ratio of speedup, $S(n)$, to the cost function $n/S(n)$.

The synthetic workloads that are used to evaluate the processor allocation policies in this paper are based on measured characteristics of parallel workloads, but do not include memory and I/O resource requirements. In fact, neither the FB-PWS nor the EQS policy can be directly applied to real workloads in which memory or I/O requirements are significant. For example, in the case of memory requirements, each policy must be modified to ensure that jobs are allocated enough memory to execute reasonably efficiently, resulting in reduced space-sharing and greater time-sharing of the processing power. The degree to which the performance of each policy will change depends on the specific memory requirements in a workload of interest. The goal of the policy evaluations in the absence of memory and I/O requirements is to provide a baseline of policy performance comparisons, as well as some understanding of the relative policy strengths and weaknesses that may usefully guide the design and evaluation of more complex high performance policies for real workloads.

The remainder of this paper is organized as follows. Section 2 provides definitions of the FB-PWS, FB-ASP, and EQS policies. Section 3 defines our system assumptions and synthetic workload models, including a revised method for modeling application speedups. Section 4 provides the policy comparisons for both the idealized case (where repartitioning cost is assumed to be zero) and for repartitioning overheads that are estimated from system measurement. Section 5 contains the conclusions of this work.

2 Policy Definitions

The FB-PWS, FB-ASP, and EQS policies considered initially in this paper are each defined below.

2.1 FB-PWS and FB-ASP

The FB-PWS policy is proposed and clearly defined in [23]. The brief definition is repeated here for the sake of reader convenience.

An arriving job, j, is configured for the following partition size:

$$min\left\{N_j, \frac{min(pws_j, P)}{S + min(pws_j, P)} \times P\right\},$$

where N_j is the job's maximum parallelism, pws_j is the *processor working set* measure for the job (defined in section 1), S is the sum of the processor allocations for all jobs currently in the system, and P is the number of processors in the system. At the start of each time slice, jobs are examined in order of least acquired processing time, where acquired processing time is the number of processor-seconds so far allocated to the job. Each job in turn is scheduled to run on the number of processors it is configured for, until there are fewer processors left than any of the remaining jobs' configurations. In this case the scheduler runs the highest priority unscheduled job on the remaining processors.

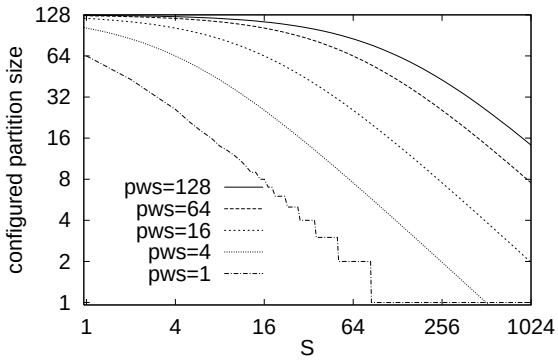


Figure 1: Configured partition size vs. S for FB-PWS
$N_j \geq$ P; P = 128

Thus, only the last job scheduled for execution in each quantum may run on a different number of processors than its initial configuration.

The FB-ASP policy is identical to the FB-PWS policy, except that an arriving job is configured for $min(N_j, P/J)$ processors, where J is the number of jobs in the system including the new arrival.

Note that processor allocations decrease as system load (S) increases. For the FB-PWS policy, Figure 1 illustrates the relationship between allocation size and S for several values of pws_j, assuming $P = 128$ and $N_j \geq P$ for each job j. Note that the allocations approach being proportional to pws as S increases, and the allocations are generous unless pws is small and S is large. Even when S is quite large (e.g., greater than 128), the allocation for a newly arriving job with $pws = P$ is still a significant fraction of P. This allows efficient jobs with long service times to execute on a reasonable number of processors after shorter jobs have departed.

2.2 EQS

The spatial equipartitioning policy, EQS, was initially proposed by Tucker and Gupta in [27] and has been evaluated and/or refined in many subsequent studies (e.g., [13, 18, 16]). When a job arrival or departure occurs, the processors are dynamically reallocated so that each job has an equal fraction of the processors unless a job has smaller maximum parallelism than the equipartition value. In the latter case, each such job is allocated a number of processors equal to its maximum parallelism, and the equipartition value is recursively computed for the remaining processors and jobs. The precise processor allocations may differ from the equipartition value by small adjustments to avoid non-integer allocations. In this paper, we adjust up for jobs that have less acquired processing time, and adjust down for jobs that have more acquired processing time.

Note that for the EQS policy as defined above, repartitioning can occur quite frequently. In section 4 we examine the impact of realistic repartitioning

overheads and we consider a "practical implementation" of the EQS policy in which full repartitioning only occurs once every 500 seconds.

3 Model Definition

In this section we define the system and synthetic workloads that will be simulated to evaluate scheduling policies in section 4. To date, traces of production parallel workloads do not include job speedup functions, which are needed to evaluate the EQS and FB-PWS policies. Thus, we formulate a synthetic workload model that mimics the variation in job parallelism and service demands in the traces, and can represent a variety of speedup behaviors that are observed in practice.

3.1 System Assumptions

The system is assumed to contain P processing nodes, each functionally and performance-equivalent with respect to applications in the workload. Communication costs are represented in the synthetic workload model; otherwise, details of the interconnection network and memory system are left unspecified.

We assume that jobs are capable of adapting to changes in the number of processors that are allocated to them. Adaptive programming techniques and runtime support for program restructuring are active areas of research and appear to be feasible for both shared memory and message passing systems (e.g., [27, 20, 6]). Although job reconfiguration can involve substantial cost, particularly if massive data movement is required, the results in [6, 11] show that the benefit of better processor scheduling can outweigh the associated cost. This key issue is explored further for EQS and FB-PWS in section 4.

We assume that the system knows the maximum number of processing nodes that each job can make productive use of, either because this information is specified when the job is submitted or because the system is capable of determining this information at runtime using methods such as the self-tuning approach recently proposed by Nguyen et. al. [20]. Similarly, for the FB-PWS policy (or the EQS-PWS policy yet to be defined), we assume that the system is capable of knowing the *pws* measure for each job, perhaps from runtime estimation techniques similar to those described in [20].

In the remainder of the paper we assume the set of processing nodes are dedicated to servicing a parallel workload. One might also imagine that the nodes are a set of currently idle nodes in a non-dedicated network of workstations (NOW) that is available for serving large (parallel) jobs. Such a system requires an effective policy for recruiting idle nodes as well as efficient mechanisms for migrating the processes of parallel jobs away from nodes that are preempted by a higher priority user [2, 30, 1]. Although we do not consider the impact of node interruptions nor particular policy customizations that might be needed, we consider synthetic workloads and repartitioning overheads that are relevant to such environments. Repartitioning overheads are discussed further in section 4.

3.2 Synthetic Workload Model

Job arrivals occur at rate λ, and are modeled as a Poisson process except in one set of experiments where we investigate whether higher variability in inter-arrival times changes the impact of repartitioning overhead on relative policy performance.

The set of characteristics that define each job are:

W_j - the work (total cpu service requirement) that the job must perform,

N_j - the maximum number of nodes that the job can productively use,

and a yet-to-be-specified set of parameters that correspond to the communication and other execution overheads in a particular model of job speedup. These job characteristics are discussed below. Section 3.2.1 develops a model of job execution overheads suitable for the goals of this study. We then describe the characterization of job parallelism (section 3.2.2), job service requirement (section 3.2.3), and correlation among the various model parameters (section 3.2.4). The workload model is summarized in section 3.2.5.

3.2.1 Job Execution Overheads

One possible functional form for job execution time, proposed in [26], is defined as follows:

$$T_j(n) = \frac{\phi W_j}{n} + \alpha + n\beta, \qquad \phi \geq 1, \qquad \alpha, \beta \geq 0, \tag{1}$$

where

- n is the number of processors allocated to the job,
- ϕ is an inflation factor that models load imbalance in the computation,
- α represents fixed overheads such as per-processor initialization, and
- β represents communication overhead, which increases with n.

The speedup function, $\frac{W_j}{T_j(n)}$, for the above execution time function is:

$$S_j(n) = \frac{1}{\frac{\phi}{n} + \frac{\alpha}{W_j} + \frac{n\beta}{W_j}}, \qquad \phi \geq 1, \qquad \alpha, \beta \geq 0, \tag{2}$$

and the point at which this speedup function is maximized, M_j, is:

$$M_j = \begin{cases} \sqrt{\frac{W_j \phi}{\beta}} & \text{if} \quad \beta > 0 \\ \infty & \text{if } \beta = 0. \end{cases} \tag{3}$$

The parameters of equation (1) correspond to overheads that are observed in practice, and the equation has been shown to match well with measured speedup functions if ϕ, α, and β are adjusted to yield best fit [28]. Below we propose a few modifications to the equation that improve its intuitive appeal and are needed for our study. We then point out some important characteristics of the revised speedup function.

To derive the new speedup function, we first observe that one minor deficiency in equation (1) is that load imbalance and communication overhead costs are incurred even for $n = 1$. Another minor deficiency is that ϕ, the inflation factor that represents total idle time due to load imbalance, is independent of n, whereas in practice load imbalance generally increases as n increases. We further observe that for any given job it is equally valid to define a new communication overhead parameter, β', such that $\beta = \beta' W_j$. That is, for the given job, the communication overhead is expressed as a fraction of W_j; of course, this fraction may vary among different jobs. With these motivations in mind, we make the following small modifications to equation (1):

$$T'_j(n) = \frac{(1+(n-1)\phi')W_j}{n} + \alpha + (n-1)\beta' W_j, \quad \alpha \geq 0, \quad 0 \leq \phi', \beta' \leq 1. \tag{4}$$

Note that the above *linear* increase in load imbalance with n may overestimate the increases that are typically observed in real workloads. However, with suitable controls and/or conservative estimates for ϕ', defined later in this paper, the simple linear dependence is adequate for the present purposes.

As will be discussed further in section 3.2.2, equation (2) cannot represent particular workloads of interest because the speedup depends directly on W_j. We fix this problem in the speedup function corresponding to equation (4) by assuming the fixed overhead, α, is negligible.[2] We justify this approximation as follows. First, typically only the jobs with small processing requirement have non-negligible fixed overhead, and these jobs tend to account for negligible amounts of total processor usage in parallel systems [8]. Second, the approximation will be imperceptible even for these jobs if the other parallelism overheads are non-negligible. Finally, due to the assumed linear increase in load imbalance with n in equation (4), α has approximately the same impact on the shape of the speedup function as does ϕ'. For these reasons, the approximation that $\alpha = 0$ shouldn't affect the policy comparisons in this paper. Making this change in equation (4), yields the following speedup function:

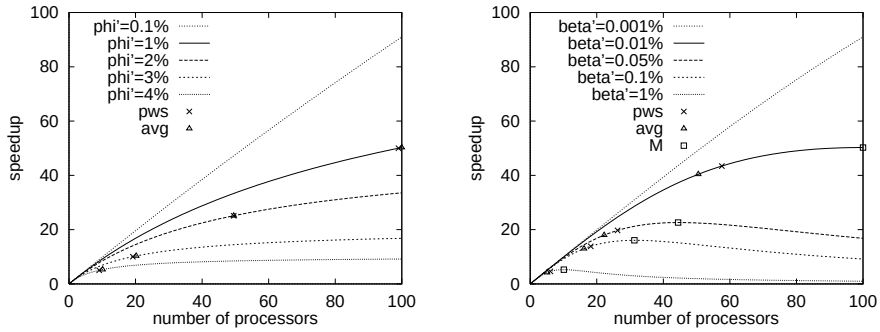
$$S'_j(n) = \frac{1}{\frac{1}{n} + \frac{(n-1)\phi'}{n} + (n-1)\beta'}, \quad \alpha = 0, \quad 0 \leq \phi', \beta' \leq 1, \tag{5}$$

that has the following maximum, M'_j:

$$M'_j = \begin{cases} \sqrt{\frac{1-\phi'}{\beta'}} & \text{if } \beta' > 0 \\ \infty & \text{if } \beta' = 0. \end{cases} \tag{6}$$

The revised speedup function has two parameters: ϕ' and β'. The impact of varying ϕ' and β' on the shape of the speedup curve is shown in Figures 2(a) and (b), respectively. For each curve, the value of *pws* is shown, as is $M \equiv M'_j$

[2] In this case, the speedup could still be correlated with W_j if we specify a correlation between any of the execution overhead parameters and W_j, as discussed in section 3.2.4.



(a) $\beta' = 0$ $(M_j = \infty)$ (b) $\phi' = 0$

Figure 2: Curves generated by the modified speedup function (equation 5)

if $M'_j \leq P$ (see figure 2(b). Note that the increase in speedup between $n = pws$ and $n = \min(M'_j, P)$ is small. This is the motivation for allocating processors in proportion to pws at high load, as in the FB-PWS policy. Also note that the new speedup function is capable of representing a variety of curves that match those that are observed in practice, similar to the speedup function in equation (2).

If execution efficiency on N_j processors, $S_j(N_j)/N_j$, is equal to c, it is straightforward to show from equation (5) that

$$(N_j - 1)\phi' + N_j(N_j - 1)\beta' = \frac{1}{c} - 1. \tag{7}$$

We will use the above equation to define particular overhead characteristics in the experiments in section 4.3.

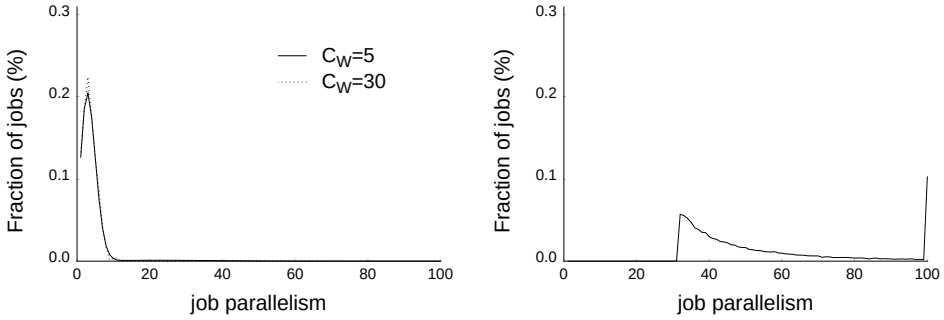
Finally, we consider an alternate speedup model [5, 16], that has been used widely in studies of scheduling policy performance [13, 17, 4, 20]:

$$S_j(n) = \frac{(\delta + 1)n}{\delta + n}. \tag{8}$$

We note that this function is a special case of equation (5) in which $\beta' = 0$ and

$$\phi' = \frac{1}{1 + \delta}. \tag{9}$$

Thus, the curves in Figure 2(a) are also examples of the speedup function in equation (8). Furthermore, $pws = \delta$ for this speedup function; thus several of the curves are labeled with the determining parameter δ. Since efficiency *increases* as δ increases, we will find it convenient to produce a positive correlation between efficiency and W_j in some of the synthetic workloads by setting $\beta' = 0$ and specifying a distribution for δ that is positively correlated with W_j. For $\delta \geq P$ (i.e., $\phi' \leq \frac{1}{P+1}$), note that the efficiency on $n \leq P$ processors is greater than or equal to 50%, as shown in equation (8) and illustrated in Figure 2(a).



(a) Speedup Function of Equation (2) $W_j \sim \text{Hyperexp}_2(1000, C_W)$, ϕ=1.3, α=25, β=25

(b) Speedup Function of Equation (5) $\phi' \sim \text{Uniform}(0, 1\%)$, $\beta' \sim \text{Uniform}(0, 0.1\%)$

Figure 3: Distributions of $N_j = M_j$ (P=100)

3.2.2 Job Parallelism

One approach to generating a synthetic workload [23] is to specify the distributions of the parameters that characterize the job speedups, and then to let job parallelism, N_j, be equal to the point at which the speedup function is maximized, M_j. This can be done using either the speedup function in equation (2) or equation (5). Below we discuss the disadvantages of this approach and then define the approach and distributions of N_j that will be used in our policy performance comparisons.

One drawback of setting $N_j = M_j$ is that the complex relationship between M_j and the parameters of the speedup function may lead to an unanticipated or undesirable distribution of job parallelism. For example, Figures 3(a) and (b), give the distributions of $N_j = M_j$ for the speedup functions in equation (2) and equation (5), respectively. In each case, the speedup parameters are set at particular (reasonable) values. In Figure 3(a), M_j depends directly on W_j (see equation (3)); thus the hyperexponential distribution of W_j, or any other realistic distributions of W_j, leads to a skewed distribution toward very low job parallelism. This distribution or the distribution in Figure 3(b) may not be the desired parallelism distribution for the synthetic workload.

The approach of setting $N_j = M_j$ also limits the types of correlations that can be specified among work, parallelism, and execution efficiency. For example, one cannot model both high correlation between work and job parallelism (as observed in [8]) but weak correlation between work and efficiency. This may be desirable because, for example, both small program development runs of highly efficient parallel codes, as well as large jobs with moderate communication overheads may be expected in a parallel system of interest.

We solve these problems by taking a different approach. First, we explicitly specify the distribution of job parallelism. Next, we explicitly specify the distributions of W_j, ϕ', and β', and possibly correlations among these parameters,

subject only to the following constraint:

$$\beta' \leq \frac{1-\phi'}{N_j^2}. \tag{10}$$

This constraint guarantees $N_j \leq M_j$ for the speedup function in equation (5).[3] Note that if $\beta' = 0$ then equation (10) is trivially satisfied due to the practical restriction that $0 \leq \phi' \leq 1$.

The distributions of job parallelism that will be used to compare policy performance are illustrated in Figure 4. These distributions are motivated by the variation in job parallelism reported for daytime user jobs on the iPSC/860 machine at NASA Ames [8], and also on the SP/2 machine at the Cornell Theory Center [10]. The distributions in Figure 4 were generated from a parameterized *bounded geometric distribution* of job parallelism that is adapted from prior work [13, 17] and has the following four parameters:

N_{max} - the maximum value for job parallelism,

$P_{N_{max}}$ - the probability that an arriving job has parallelism equal to N_{max},

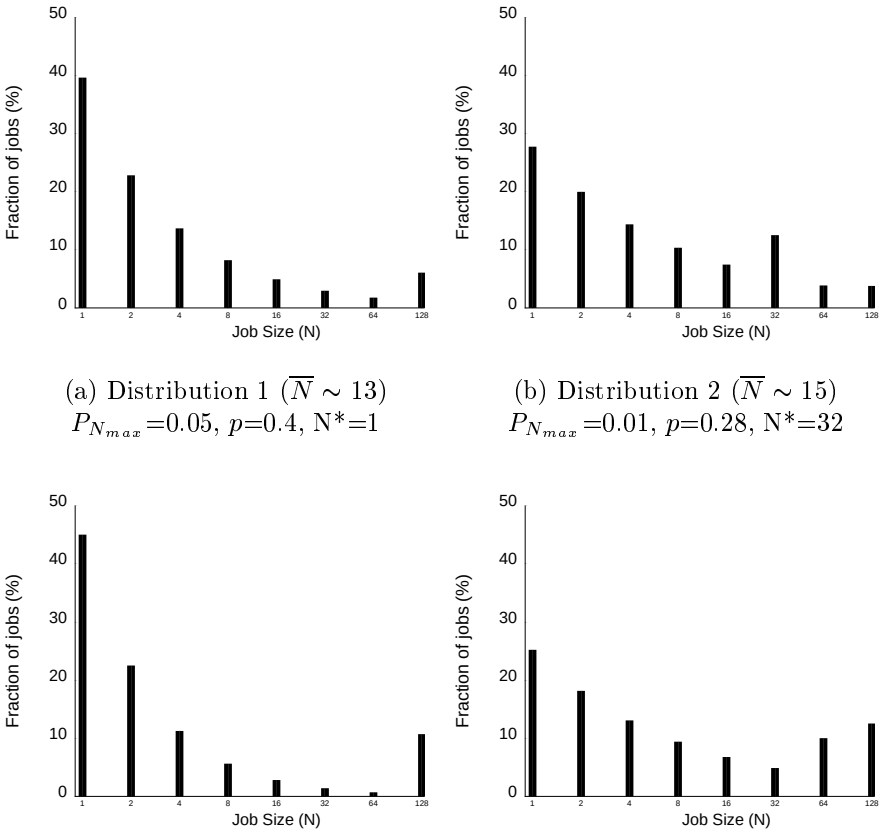
p - the parameter of the bounded geometric distribution of parallelism for all other jobs, and

$N^* \leq N_{max}$ - the value of parallelism whose probability will increase by the probability for parallelism greater than N_{max} in the geometric distribution.

An arriving job has parallelism N_{max} with probability $P_{N_{max}}$. Otherwise, the parallelism of the job is chosen from a geometric distribution with parameter p. If the selected parallelism is larger than N_{max}, the job is assigned parallelism N^*.

One rationale for the bounded geometric distribution is that one can expect a significant number of highly parallel jobs; i.e., all of the production jobs that can run fairly efficiently on as many processors or nearly as many processors as are available in the system. Another rationale is that there is another class of jobs made up of program-development runs and codes that cannot run efficiently on P or close to P processors. In this class of jobs, one can perhaps expect the probability to decrease as the parallelism increases. We note that one discrepancy between this model and the data in [8] is that parallelism equal to two has lower probability in the measured system than suggested by the bounded geometric. This type of discrepancy shouldn't have great impact on the relative policy performance comparisons in this paper. Finally, the parameter N^* is included to model the preferred parallelism equal to 32 in the measured iPSC/860 and SP/2 workloads [8, 10].

[3] Note that the restriction $N_j \leq M_j$ assumes that either users are sophisticated enough not to request more than M_j processors (because the job execution time will be longer), or self-tuning [20] is used to achieve same result. Allowing $M_j > N_j$ assumes that users sometimes configure their jobs to run on at most $N_j < M_j$ processors, for convenience or because the speedup model doesn't accurately reflect a sharp decline in the actual speedup function beyond N_j.



(a) Distribution 1 ($\overline{N} \sim 13$)
$P_{N_{max}}$=0.05, p=0.4, N*=1

(b) Distribution 2 ($\overline{N} \sim 15$)
$P_{N_{max}}$=0.01, p=0.28, N*=32

(c) Distribution 3 ($\overline{N} \sim 17$)
$P_{N_{max}}$=0.1, p=0.5, N*=128

(d) Distribution 4 ($\overline{N} \sim 27$)
$P_{N_{max}}$=0.1, p=0.28, N*=64

Figure 4: Example Bounded-Geometric Distributions for Job Parallelism $N_{max} = 128$

3.2.3 Job Service Requirement

In some workloads, we model job service requirement, W_j, by a two-stage hyperexponential distribution with mean $\overline{W}$ and coefficient of variation, C_W. We use the notation $W_j \sim \text{Hyperexp}_2(\overline{W}, C_W)$ to denote this distribution.

For most experiments, mean job service requirement will be proportional to either job parallelism or the square of job parallelism. Let $\overline{N}$ represent mean job parallelism and $\overline{W}$ represent the overall mean job service requirement. Also let $W_{j|n}$ be the service requirement for a given job, j, that has parallelism $N_j = n$. To specify mean service requirement proportional to job parallelism, $W_{j|n}$ has a two-stage hyperexponential distribution with mean $\frac{n}{\overline{N}}\overline{W}$, and coefficient of

variation called $C_{W|n}$; *i.e.,*

$$W_{j|n} \sim \text{Hyperexp}_2(\frac{n}{\overline{N}}\overline{W}, C_{W|n}). \tag{11}$$

This model was proposed in [14] and formalized in [16]. If mean service requirement is correlated with the square of job parallelism, then the multiplier for $\overline{W}$ is replaced by $\frac{n^2}{\overline{N^2}}$. The reported measures of $W_{j|n}$ vs. n for the iPSC/860 workloads at NASA Ames [8] lie between these two cases.

3.2.4 Correlation Between Workload Parameters

In some experiments, we will assume that execution overheads are on average lower for jobs with larger total service requirement. In these cases, we will use the same notation as in equation (11). For example,

$$\delta_{j|w} \sim \text{uniform}(\frac{w}{\overline{W}}100, \frac{w}{\overline{W}}200) \tag{12}$$

specifies that the values of δ are selected from a uniform distribution with a lower bound and upper bound that are each proportional to the job service requirement ($W_j = w$), with overall mean value for δ equal to 150.

3.2.5 Workload Model Summary

In summary, the synthetic workloads used to evaluate relative policy performance in this paper have four parameters: N_j (job parallelism), W_j (total service requirement), ϕ' (load imbalance), and β'(communication overhead, as a fraction of total work). These parameters have the following characteristics: (1) a bounded geometric distribution of job parallelism (N_j) as illustrated in Figure 4, (2) a two-stage hyperexponential distribution of W_j, in most cases with mean proportional to N_j or ${N_j}^2$ as defined in section 3.2.3, and (3) load imbalance overhead ($\phi' = \frac{1}{1+\delta}$) and communication overhead (β') that are either fixed values or are selected from a specified distribution. In some experiments, the *average* execution overhead will be inversely proportional to service requirement, as defined in section 3.2.4. The execution overhead parameters must also satisfy the constraint in equation (10), which guarantees that the job's speedup function is non-decreasing up to N_j processors. This workload model is very similar to the previous workload model in [17]. The new features are the communication overhead parameter, β', the N^* parameter in the bounded geometric distribution for N_j, and the distributions for the execution overhead that will be specified in the next section.

4 Policy Comparisons

In this section we present the results of policy comparison experiments that are based on simulations with a variety of synthetic workloads. The discussion is focussed on comparisons between the FB-PWS, EQS, and EQS-PWS policies,

although results for the FB-ASP policy [23] are also provided in the figures for the sake of completeness.

The four parameters that characterize each job in the synthetic workloads are summarized in section 3.2.5. The distribution of job parallelism, N_j, will be one of the four distributions depicted and numbered in Figure 4, depending on the experiment. The distributions for the execution overhead parameters will be explained as each experiment is introduced; these distributions are motivated by comparisons with earlier results [23] or by speedups that are encountered in practice (*e.g.,* [24, 22]). Arrivals are assumed to be Poisson unless otherwise specified, and the system size (P) is 128 processing nodes in all experiments.

The simulations were performed using the batch means method of generating confidence intervals, with batch size ranging from 100,000 to 200,000 job departures, depending on the particular experiment. Except as noted, reported results have 90% confidence intervals that are within 5% or less of the given value.

Section 4.1 compares the EQS and FB-PWS policies for several workloads, assuming zero repartitioning cost for both policies and zero swapping cost for FB-PWS. Section 4.2 introduces the EQS-PWS policy and compares this policy against EQS and FB-PWS, again assuming zero cost for swapping and repartitioning. Finally, Section 4.3 compares the three policies under a case of realistic partitioning overheads.

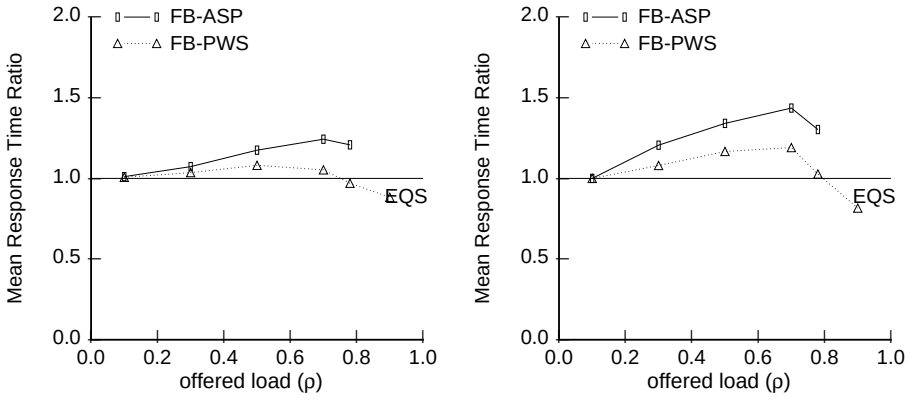
4.1 Comparisons under Zero Repartitioning Cost

We first compare the EQS and FB-PWS under a workload that is nearly identical to Workload 2 in [23]. That is, we use the speedup function in equation 5, set the overhead parameters and distribution of W_j as given in Figure 5, and let $N_j = M_j$. Note that because we have modified the execution time function so that load imbalance increases linearly in the processor allocation, we have chosen $\phi' = 0.003$ such that the load imbalance on 100 processors is the same as the fixed overhead in Workload 2.

Figure 5 gives the ratio of mean response time for the FB-PWS policy to the mean response time of EQS, as a function of offered load, $\rho = \lambda\overline{W}/P$, for both $C_W = 5$ and $C_W = 30$. When $\rho = 0.9$, system utilization for FB-PWS or EQS is in the range of 92% - 99%. The response time ratio for FB-ASP is also given for completeness, as noted above. Similar to the results reported in [23], these results show that FB-PWS is competitive with EQS throughout the range of offered load, and also that the system with the EQS policy saturates at a slightly earlier point than the FB-PWS system.[4]

The workload in Figure 5 has a distribution for N_j that is very similar to the distribution in Figure 3(a). Thus, nearly all of the jobs have parallelism < 10, and a negligible fraction of jobs have parallelism greater than 50. Furthermore, in this workload, larger W_j implies larger *pws* and larger *pws* implies higher effi-

[4]We also reproduced several of the graphs in [23], not shown in this paper, to validate that we have correctly implemented the policy simulations.



(a) $C_W = 5$ (b) $C_W = 30$

Figure 5: FB-PWS vs. EQS under workload 2 in [23]
$W_j \sim \text{Hyperexp}_2(1000, C_W),\ N_j = M_j,\ \ \phi' = 0.003, \alpha = 25, \beta = 25$

ciency on any given processor allocation. This workload thus might be favorable for the FB-PWS policy that uses the *pws* measure to determine allocations.

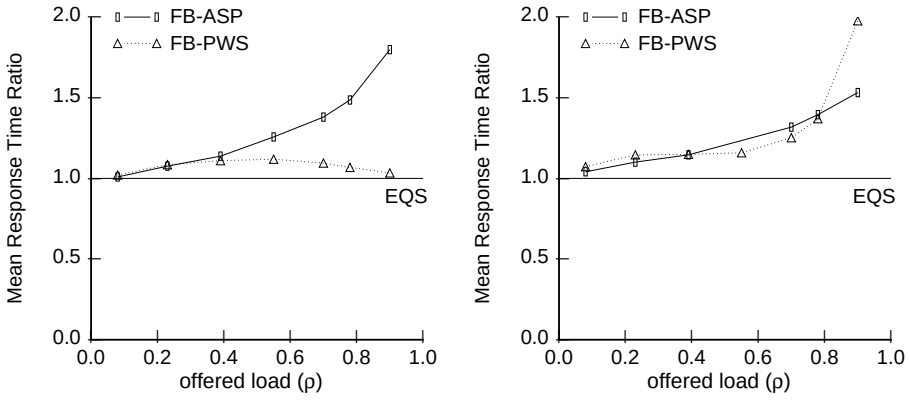
Figure 6 shows relative policy performance for two workloads that have the following characteristics that differ from Figure 5:

- Distribution of job parallelism that is more consistent with observed workloads; i.e., Distribution 1 in Figure 4.
- Moderate sublinearity in the speedups of large jobs. ϕ' is fixed and β' is *inversely proportional* to N_j^2.[5] Thus, larger N_j implies higher *pws* and higher *pws* implies higher efficiency on any given processor allocation. Also, W_j has *mean* proportional to ${N_j}^2$ and coefficient of variation approximately equal to 36. However, *larger* W_j *does not necessarily imply larger pws or higher efficiency.*

Due to the correlation between mean service requirement and ${N_j}^2$, jobs with $N_j \geq 32$ account for 98% of the system resource usage by this workload, in agreement with system measurements in [8].

For the workload in Figure 6(a) ($\phi' = 0$), FB-PWS is comparable to the EQS policy throughout the entire range of offered load. When load imbalance is more significant (i.e., $\phi' = 0.01$, which implies that efficiency loss due to load imbalance is 50% on 100 processors), FB-PWS saturates sooner than EQS. The reason FB-PWS does less well for this workload is that jobs with large N_j (and large *pws* due to low communication overhead) will, on average, experience less space-sharing under FB-PWS than under EQS (see Figure 1). Since these jobs dominate system resource usage and have modest speedups due to load imbalance, the FB-PWS system saturates sooner. We note that the results in

[5] In fact, β' is defined such that N_j is the point where the speedup curve is maximized.



(a) $\phi' = 0$ (b) $\phi' = 1\%$

Figure 6: The impact of large jobs with moderate execution overhead
$N_j \sim$ Distribution 1, $W_{j|n} \sim \text{Hyperexp}_2(\frac{n^2}{N^2}1000, 10)$, $\beta' = \frac{1-\phi'}{N^2}$
($C_W = 36$)

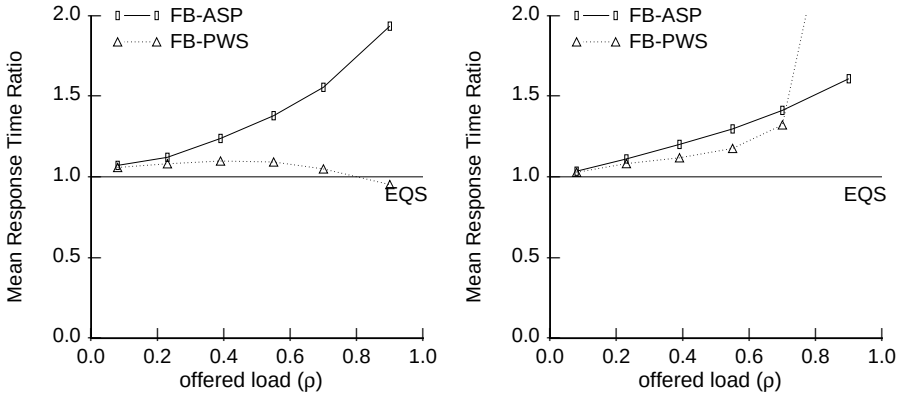
Figure 6(a) and (b) are largely the same if the workloads are changed to have $C_{W|n} = 2$ ($C_W = 6.6$), except that the FB-ASP policy has better performance at high load in the case that $\phi' = 0$.

As a final set of comparisons of FB-PWS and EQS under zero repartitioning cost, Figure 7 shows relative policy performance for workloads with similar parallelism (Distribution 3 in Figure 4) and the same distribution of W_j as in Figure 6. However, the communication overhead (β') is assumed to be zero and the overhead due to load imbalance ($\phi' = \frac{1}{1+\delta}$) is selected from a *uniform distribution* for δ. Recall that $pws = \delta$, and that Figure 2 shows that if $\delta > 100$ the job will have efficiency greater than approximately 50% on any processor allocation. Since δ is nondeterministic, large N_j does not imply large pws nor high efficiency on a given feasible processor allocation.

In Figure 7(a), efficiency is positively correlated with W_j; in Figure 7(b), efficiency is independent of W_j. The results are very similar to the results in Figure 6(a) and (b), respectively. Thus, in these cases, the weaker correlation between N_j and efficiency has not affected the relative performance of the policies.

4.2 The EQS-PWS Policy

Recall the four characteristics of FB-PWS that are possibly beneficial to policy performance from section 1. The load adaptive property is also a characteristic of the EQS, and repartitioning issues will be considered in section 4.3. Furthermore, we have seen that multilevel feedback can lead to earlier system saturation (Figures 6-7) if a significant fraction of the jobs with large W_j and large parallelism have only moderate speedups. On the other hand, the EQS



(a) $\delta_{j|w} \sim$ uniform($100\frac{w}{W}$, $200\frac{w}{W}$) (b) $\delta_j \sim$ uniform(100,200)

Figure 7: Policy comparison with variable load imbalance overhead
$N_j \sim$ Distribution 3, $W_{j|n} \sim \text{Hyperexp}_2(\frac{n^2}{N^2}1000, 2)$, $\beta' = 0$
($C_W = 6.6$)

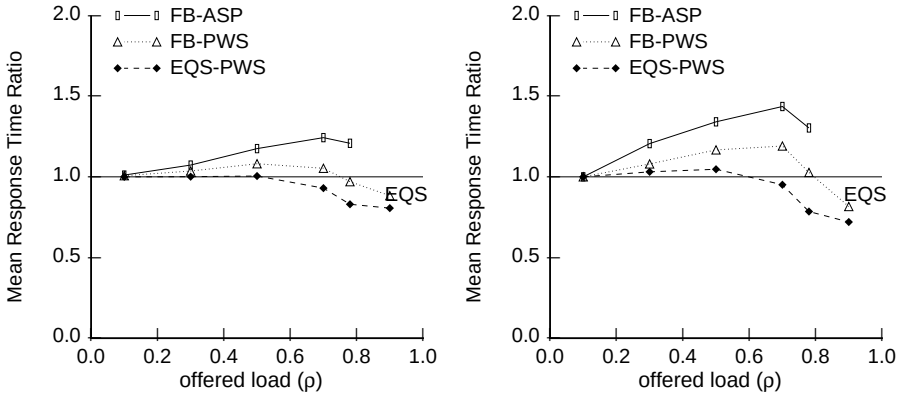
policy saturates earlier than the FB-PWS policy in Figure 5, perhaps indicating that the EQS policy could be improved by using the *pws* measure to adjust processor allocations for this (and possibly other) workloads. This motivates the following new policy that we call EQS-PWS.

In the EQS-PWS policy, processor allocation proceeds in two phases. In the first phase processors are assigned to jobs as in the EQS policy, except that each job's processor allocation is bounded by $\min(pws_j, N_j)$ instead of N_j. If there are idle processors left after phase one, the idle processors are equipartitioned among the jobs, using $N_j - \min(pws_j, N_j)$ as the upper bound on the additional processors given to job j.

The EQS-PWS policy is identical to the EQS-AVG policy defined in [4], except that the *pws* measure is used in place of average parallelism (*avg*). In [4], the EQS-AVG was found to have approximately the same performance as EQS, but new workload parameters are considered in this paper. Furthermore, figure 2(b) shows that eliminating the allocations above *pws* processors at high load may be more favorable than eliminating allocations above *avg* processors.

For the workloads in Figures 6-7, EQS-PWS has *identical performance* to EQS, indicating that there is not much benefit to using the *pws* measure for those workloads. Note that this is another reason why FB-PWS doesn't perform as well at high loads in figures 6(b) and 7(b).

Figures 8 and 9 show the relative performance of the EQS-PWS policy for the workload in Figure 5 and a new workload, respectively. The workload in Figure 9 is similar to the workload in Figure 7, except that the execution overhead parameter, $\delta = pws$, has a hyperexponential distribution instead of a uniform distribution, leading to much greater speedup sublinearity, and perhaps



(a) $C_W = 5$ (b) $C_W = 30$

Figure 8: Performance of EQS-PWS under workload 2 in [23]
$W_j \sim \text{Hyperexp}_2(1000, C_W)$, $N_j = M_j$, $\phi' = 0.003, \alpha = 25, \beta = 25$

unrealistically poor speedups for the majority of the jobs. However, in this case there is more potential for the EQS-PWS and FB-PWS policies to outperform EQS.

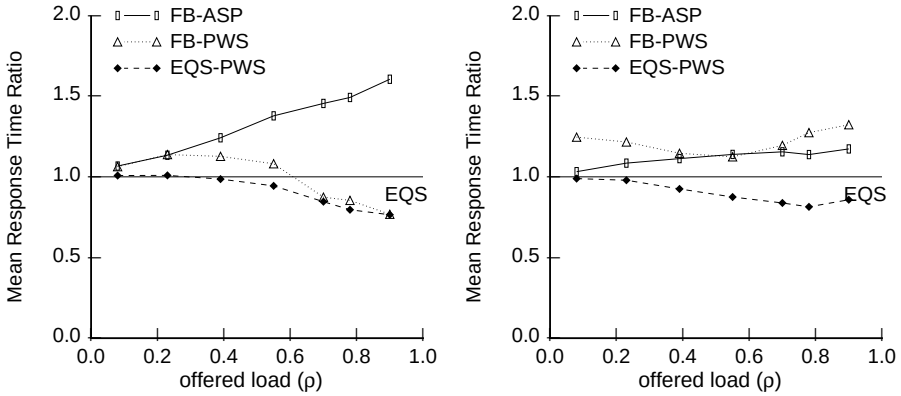
In Figures 8 and 9, EQS-PWS has performance equal to or better than EQS and FB-PWS over the entire range of offered load. In particular, the EQS-PWS policy does not saturate earlier than FB-PWS in Figure 8 or Figure 9(a). Results for EQS-AVG, not given in the figures, show that for the workloads in figures 5 through 9, EQS-AVG has very nearly the same performance as EQS-PWS.

4.3 Comparisons with Repartitioning Overheads

Our final set of experiments are aimed at determining the impact of realistic repartitioning overhead on the relative performance of the EQS, EQS-PWS, and FB-PWS policies. In this case we use a new synthetic workload – one that corresponds closely with the workload measurement data in [8]. Below we explain the workload, the repartitioning overheads, and the results.

The key features of the workload are:

- Distribution 4 of Figure 4 is used for job parallelism. This was derived from the measured distribution in [8], by shifting some of the probability mass for $n = 32$ to higher values of parallelism. The shift corrects for the incentives in the measured system for parallelism equal to 32 during the daytime. In a system with EQS or FB-PWS scheduling, such incentives would not be necessary.
- The service requirement is determined by a set of values for $\overline{W_{j|n}}$ and $C_{W|n}$, $n = 1, 2, 4, 8, ..., 128$, that are computed from Table 2 in [8]. Specifically, $\overline{W_{j|n}}$ is initially set to $n\times$ the average runtime for job size n, and $C_{W|n}$ is



(a) $\delta_{j|w} \sim \text{Hyperexp}_2(\frac{w}{\overline{W}}100, 5)$ (b) $\delta_j \sim \text{Hyperexp}_2\ (100,\ 5)$

Figure 9: Policy comparison for workloads with very sublinear speedup
$N_j \sim$ Distribution 1, $W_{j|n} \sim \text{Hyperexp}_2(\frac{n^2}{\overline{N^2}}1000, 10)$, $\beta' = 0$
$(C_W = 36)$

set to the coefficient of variation in runtime for n, and then the mean values were adjusted proportionately downward (to remove execution overhead) to get the measured system load.[6] Note the large value of average total service requirement ($\overline{W}$ = 10566 seconds, or approximately 2.9 node-hours) for this measured iPSC/860 workload. The measured average node-hours of running time per application (with overhead), is approximately double that value.

- For the parallelism overheads, we let

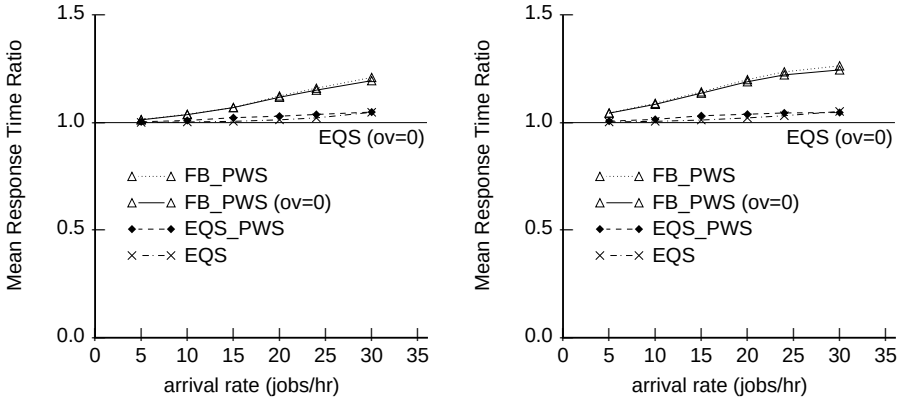
$$\phi' \sim \text{uniform}(0, \frac{1/c_1 - 1}{N_j - 1}) \tag{13}$$

and then

$$\beta' \sim \text{uniform}(0, \min(\frac{1/c_2 - 1}{N_j(N_j - 1)} - \frac{\phi'}{N_j}, \frac{1 - \phi'}{N^2})), \tag{14}$$

where $c_1 = 0.75$ and $c_2 = 0.5$ (see equation (7)). For this workload, these distributions yield first an execution efficiency on N_j processors, E, that is approximately uniformly distributed between 75% and 100%, and then an efficiency on N_j processors that is approximately uniformly distributed between 50% and E. This is a somewhat arbitrary, but well-specified model of the spread of execution efficiencies that are encountered in practice. In

[6] These calculations are necessarily approximate since runtime includes execution overhead whereas total processing requirement does not. Processing requirements (without overhead) are not given in the measured data. However, we anticipate that the computed values give approximately the correct relative magnitudes of the average work as a function of job parallelism, and this is more important than quantitative accuracy of the individual values.



(a) Poisson arrivals

(b) C_V of interarrival time = 3

Figure 10: Policy Comparison with Repartitioning Overhead
$N_j \sim$ Distribution 4, $\overline{W} = 10566$ seconds, $C_W = 4.26$,
$c_1 = 0.75, c_2 = 0.5$

the absence of data in the literature, we have relied on a variety of informal information about parallel job speedups in developing this model.

The workload defined above provides one more context for comparing policy performance, irrespective of repartitioning overhead. Note also that the overall mean $\overline{W}$ for the measured system is larger than assumed in the synthetic workloads for our previous experiments.

For repartitioning overhead, we assume that *each time* the processor allocation changes for a job, the *entire job will stall* for 5 seconds. This estimate was arrived at by computing the time to fetch 32 megabytes of data from a remote memory, either in a network of workstations that runs the GMS global memory management system [7] or in the KSR or DASH memory systems [20, 3]. In GMS, each remote fetch of an 8-kilobyte page requires 2 milliseconds. In KSR, it takes 30 milliseconds to fill a 256KB cache from remote memory [20]. In DASH, each remote fetch of a 16-byte cache block requires approximately 170 cycles on a 33 MHz processor. Thus, the transfer of 32 megabytes requires approximately 4-10 seconds in these systems. Anticipating continued improvements in network latencies, we conservatively select 5 seconds for the repartitioning overhead.

Figure 10(a) shows the mean response time ratios of FB-PWS, EQS-PWS and EQS with repartitioning overhead with respect to an EQS system with zero repartitioning cost. The ratio for FB-PWS with zero repartitioning overhead is also given. Job arrival rate is varied up to 30 jobs/hour, which is higher than observed on the NASA Ames iPSC/860 (Figure 12 of [8]) or the Cornell Theory Center SP/2 [10]. The system utilization at arrival rate of 30 jobs/hour is 82%-85%.

The results in Figure 10(a) show that relative policy performance is un-

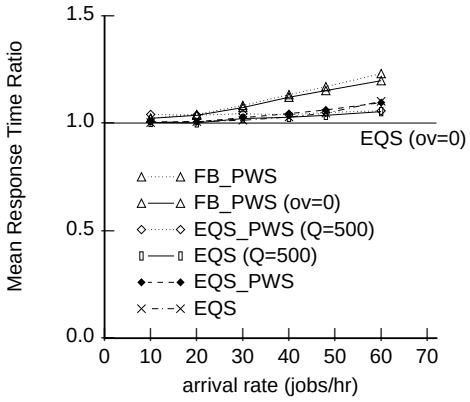


Figure 11: Policy Comparison with Repartitioning Overhead
Poisson arrivals, $N_j \sim$ Distribution 4, $\overline{W} = 5283$ seconds, $C_W = 4.25$, $c_1 = 0.75, c_2 = 0.5$

changed for the given workload, with or without repartitioning overhead, even though processor repartitioning occurs on every arrival and departure in the EQS system. Figure 10(b) shows that this result holds even if the coefficient of variation of interarrival times is increased to three by using a two-stage hyper-exponential distribution of interarrival times, reflecting the measured coefficient of variation in [8].

To see what would happen if arrival rate is doubled to 60/hour, we halved each of the values of $\overline{W_{j|n}}$ and re-ran the experiment. The results are shown in Figure 11. For the EQS and EQS-PWS policies, we include a new case where the system performs full repartitioning at most once per every 500 second quantum. In this case, the system gives immediate service to an arriving job by judiciously stealing processors from a job that is already executing; jobs with largest service so far received, or with allocations greater than the equipartition value, have highest priority for relinquishing some of their processors to a newly arriving job. Repartitioning overhead is charged for each job reconfiguration that occurs between or at quantum boundaries.

The relative policy performance is unchanged for the higher arrival rates in Figure 11, but the EQS and EQS-PWS policies that only perform full repartitioning at the beginning of every 500 sec quantum have perceptibly better performance at high load than the policies that do repartitioning at every job arrival or departure..

Overall, the experiments in this section provide evidence that EQS or EQS-PWS provides superior performance *even when realistic data repartitioning overheads are considered*, yet the FB-PWS policy is still a remarkably competitive alternative over a wide range of workloads.

5 Conclusions

In this paper, we have compared the EQS and FB-PWS policies under synthetic workloads that have not previously been considered, yet have realistic job characteristics [8, 10] and repartitioning overheads. As part of this effort, we have improved the previous workload models in [13, 17, 23] and we have shown how the different speedup functions used in the previous models are related. Finally, we have defined a new policy, EQS-PWS, which has what appear to be the most promising characteristics of both EQS and FB-PWS.

A key feature of our realistic workloads is that job service requirements are substantial enough to warrant execution on a parallel system, and thus job arrival rate is at most 30−60 jobs/hour [8, 10]. The principal conclusions that we reach from the experiments performed in section 4, are:

- The EQS policy is generally superior to the FB-PWS policy *even when realistic repartitioning overheads are considered.*
- If a reasonable fraction of the jobs with large parallelism and large total service requirement have moderate execution overheads (e.g., 50% - 75% efficiency on P processors), then a system with FB-PWS scheduling saturates before a system with EQS scheduling (Figures 6(b), 7(b)), due to less effective space sharing.
- If a large fraction of the jobs are very inefficient; that is, they have *pws* significantly smaller than their maximum parallelism, then EQS saturates before FB-PWS (Figures 8, 9(a)).
- For the workloads examined, EQS-PWS always performs as well as or better than EQS and FB-PWS. In particular, EQS-PWS avoids the early saturation of EQS in systems with a large fraction of very inefficient jobs.
- Although the above differences are worthy of consideration in future policy design, the overall differences in performance among the EQS, EQS-PWS, and FB-PWS policies are perhaps surprisingly small.

Given the results in this paper, we would argue that the simple EQS policy, which does not require knowledge of the *pws* measure for each job, may be the preferred policy. However, the ultimate choice of policy will also depend on at least two factors: (1) the significance of the cases where EQS-PWS outperforms EQS, and (2) how well the *pws* measure can be estimated in practice. Fruitful areas for further investigation include: (1) quantifying the workload characteristics that lead to differences in relative mean response times of the policies, (2) examination of more detailed measures such as expected response time conditioned on job service requirement, (3) how well the *pws* measure can be estimated at runtime using techniques similar to those in [20], and (4) suitable modifications to the policies to support jobs with large memory requirements.

Acknowledgements

The authors gratefully acknowledge comments and suggestions by Thu Nguyen, John Zahorjan, other workshop participants, and the anonymous reviewers,

which helped to improve this paper.

References

[1] R. H. Arpaci, A. C. Dusseau, A. M. Vahdat, L. T. Liu, T. E. Anderson, D. A. Patterson, The Interactions of Parallel and Sequential Workloads on a Network of Workstations. *Proc. 1995 ACM Sigmetrics Joint Int'l. Conf. on Measurement and Modeling of Computer Systems*, Ottawa, pp. 267-278, May 1995.

[2] A. Bricker, M. Litzkow, M. Livny, Condor Technical Summary. Technical Report TR 1069, Computer Sciences Dept., University of Wisconsin, Madison, WI, January 1992.

[3] R. Chandra, S. Devine, B. Verghese, A. Gupta, M. Rosenblum, Scheduling and Page Migration for Multiprocessor Compute Servers. *Proc. 6th Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI)*, San Jose, CA, pp. 12-24, October 1994.

[4] S.-H. Chiang, R. K. Mansharamani, M. K. Vernon, Use of Application Characteristics and Limited Preemption for Run-to-Completion Parallel Processor Scheduling Policies. *Proc. 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Nashville, TN, pp. 33-44, June 1994.

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

[6] G. Edjlali, G. Agrawal, A. Sussman, J. Saltz, Data Parallel Programming in an Adaptive Environment. *Proc. 9th Int'l. Parallel Processing Symposium* Santa Barbara, CA, April 1995.

[7] M. J. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, H. M. Levy, C. A. Thekkath, Implementing Global Memory Management in a Workstation Cluster. *Proc. Symp. on Operating Systems Principles*, Copper Mountain, CO, pp. 201-212, December, 1995.

[8] D. G. Feitelson, B. Nitzberg, Job Characteristics of a Production Parallel Scientific Workload on the NASA Ames iPSC/860. *Proc. IPPS '95 Workshop on Job Scheduling Strategies for Parallel Systems*, Santa Barbara, CA, pp. 337-360, April 1995.

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

[10] S. Hotovy, Workload Evolution on the Cornell Theory Center IBM SP2. *Proc. IPPS '96 Workshop on Job Scheduling Strategies for Parallel Systems*, Honolulu, Hawaii, April 1996.

[11] N. Islam, A. Prodromidis, M. S. Squillante, Dynamic Partitioning in Different Distributed-Memory Environments. *Proc. IPPS '96 Workshop on Job Scheduling Strategies for Parallel Systems*, Honolulu, Hawaii, April 1996.

[12] L. Kleinrock. *Queueing Systems, Vol II: Applications.* John Wiley & Sons, 1976.

[13] S. T. Leutenegger, M. K. Vernon, The Performance of Multiprogrammed Multiprocessor Scheduling Policies. *Proceedings of the ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, Boulder, CO, pp. 226-236, May 1990.

[14] S. Majumdar, D. L. Eager, R. B. Bunt, Scheduling in Multiprogrammed Parallel Systems. *Proc. 1988 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Santa Fe, NM, pp. 104-113, May 1988.

[15] S. Majumdar, D. Eager, and R. Bunt. Characterisation of programs for scheduling in multiprogrammed parallel systems. *Performance Evaluation*, Vol. 13, pp. 109-130, 1991.

[16] R. Mansharamani. Efficient Analysis of Parallel Processor Scheduling Policies. Ph.D. Thesis, Computer Sciences Dept., University of Wisconsin, Madison, WI, November 1993.

[17] R. K. Mansharamani, M. K. Vernon, Properties of the EQS Parallel Processor Allocation Policy. Technical Report #1192, Univ. of Wisconsin - Madison Computer Sciences Dept., November 1993.

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

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

[20] T. D. Nguyen, R. Vaswani, J. Zahorjan, Using Runtime Measured Workload Characteristics in Parallel Processor Scheduling. *Proc. IPPS '96 Workshop on Job Scheduling Strategies for Parallel Systems*, Honolulu, Hawaii, April 1996.

[21] J. K. Ousterhout, Scheduling Techniques for Concurrent Systems, *Proc. 3rd Int'l. Conf. on Distributed Computing Systems*. pp. 22-30, October 1982.

[22] J. D. Padhye, L. W. Dowdy, Dynamic versus Adaptive Processor Allocation Policies for Message Passing Parallel Computers: An Empirical Comparison. *Proc. IPPS '96 Workshop on Job Scheduling Strategies for Parallel Systems*, Honolulu, Hawaii, April 1996.

[23] E. W. Parsons, K. C. Sevcik, Multiprocessor Scheduling for High-Variability Service Time Distributions. *Proc. IPPS '95 Workshop on Job Scheduling Strategies for Parallel Systems* Santa Barbara, CA, pp. 127-145, April 1995.

[24] V. G. J. Peris, M. S. Squillante, V. K. Naik, Analysis of the Impact of Memory in Distributed Parallel Processing Systems. *Proc. 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Nashville, TN, pp. 5-18, June 1994.

[25] K. C. Sevcik, Characterizations of Parallelism in Applications and Their Use in Scheduling. *Proc. 1989 ACM SIGMETRICS/Performance '89 Int'l. Conf. on Measurement and Modeling of Computer Systems*, Berkeley, CA, pp. 171-180, May 1989.

[26] K. C. Sevcik, Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems. *Performance Evaluation*, Vol. 19, No. 2/3, pp. 107-140, March 1994.

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

[28] C.-S. Wu, Processor Scheduling in Multiprogrammed Shared Memory NUMA Multiprocessors, Master's thesis, University of Toronto, 1993.

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

[30] S. Zhou, J. Wang, X. Zheng, P. Delisle, Utopia: A Load Sharing Facility for Large Heterogeneous Distributed Computing Systems. Technical Report, University of Toronto, 1992.