Parallel Application Scheduling on Networks of Workstations

Stergios V. Anastasiadis and Kenneth C. Sevcik

Computer Systems Research Institute

University of Toronto

{stergios,kcs}@cs.toronto.edu

Abstract

Parallel applications can be executed using the idle computing capacity of workstation clusters. However, it remains unclear how to most effectively schedule the processors among different applications. Processor scheduling algorithms that were successful for shared-memory machines have proven to be inadequate for distributed memory environments due to the high costs of remote memory accesses and redistributing data.

We investigate how knowledge of system load and application characteristics can be used in scheduling decisions. We propose the new algorithm *AEP(2)* which, by properly exploiting both the information types above, performs better than other non-preemptive scheduling rules, and nearly as well as idealized versions of preemptive rules (with free preemption). We conclude that AEP(2) is suitable for use in scheduling parallel applications on networks of workstations.

1 Introduction

Advances in software and hardware technology continuously improve the performance of workstation clusters. Load balancing software which allows efficient use of otherwise underutilized resources is now commercially available. However, techniques and policies for efficiently supporting workloads

of parallel applications on such platforms are still needed.

An important question that remains is how to allocate processors among parallel applications that execute in a multiprogrammed environment. Many scheduling policies for shared-memory parallel systems were adapted from successful uniprocessor policies. However, distributed-memory machines are considerably different in both architectural design and performance behavior from shared-memory parallel systems. In particular, dynamic scheduling policies are typically very effective in shared-memory systems, but require that the applications adapt to changes in the number of allocated processors during execution. In distributed systems, the applications must dynamically redistribute their data structures. This involves considerable overhead, which can outweigh the benefits of the processor reallocation itself, as has been demonstrated both analytically and experimentally [6, 17]. Sophisticated run-time systems are required to do dynamic data redistribution automatically, and keep it from being an additional burden on the application writer [3, 8, 16].

Recently, it has been shown by Feitelson and Nitzberg [9] that application characteristics, such as execution time on a given number of processors, can be estimated. These estimates can be exploited to achieve improved scheduling. In their measurement studies, the majority of jobs run repeatedly on the same partition size by the same user exhibited a runtime distribution with low variance (i.e., coefficient of variation less than one).

Furthermore, the predictive power of the estimates can be improved by using more sophisticated job behavior models than just the mean of previous runs [5, 17, 22, 24]. In particular, Wu [24] has shown that the execution time functions of real applications, with varying sizes and structures, can be represented accurately using the model introduced by Sevcik [22]. A least-squares approximation method is applied based on the runtimes of the application for (a few) different numbers of processors.

In this paper, we investigate how different levels of information about the application characteristics and the system load can be used to improve the processor allocation decisions. We use simulation to compare the expected response times that result from using new and previously known

algorithms. Experimentation with various load levels and workload types allow us to conclude that well designed non-preemptive policies can even outperform dynamic policies that ignore estimates of application execution time.

Section 2 summarizes the assumptions we make about the software and hardware structure of the system under study. In Section 3, we define the existing and new scheduling rules that are evaluated in this paper. In Section 4, advantages of our new proposed rules are demonstrated analytically. Section 5 gives details about the simulation model of the system, the workload characteristics, and the derivation of the parameters used. Section 6 presents and analyzes the relative performance of the scheduling algorithms. In Section 7, the conclusions of our work are presented.

2 System Description

In concentrating on workstation clusters, we assume that the cost of a dynamic scheduling algorithm generally includes data transfers and data repartitioning and can be considerably more than just context switches and cache interference as in shared-memory systems. We consider a collection of interconnected nodes, each consisting of a processor and a local memory. The nodes are assumed identical, and communicate with each other with negligible latency differences. Different applications can run on separate partitions without degradation in performance due to contention in the communication medium [12]. This assumption may not be valid in the case of a heavily loaded Ethernet-based installation, but is acceptable in the context of ATM interconnects, which are expected to be prevalent soon.

We assume that the operating system can support two-level scheduling. The scheduler decides only the number of processors to allocate to each application. It is the responsibility of the application to determine how the allocated partition will be used and whether multiple threads will be interleaved on individual processors or not. The program developer makes the appropriate decision according to the computation, synchronization and communication needs of the problem [7, 20]. The

Application Characteristics
Maximum Parallelism p^{max}
Execution Time on one Processor $T(1)$
Execution Time on $p > 0$ Processors $T(p)$
Load Parameters
Number of Waiting Jobs
Number of Running Jobs

Table 1: The application characteristics and load parameters used by the scheduling policies in our study.

scheduling decisions are taken on a single node, where a central queue of job requests is maintained. Centralized algorithms that can take advantage of the clustering properties in small or large scale distributed systems have been proven successful in recent resource management environments [25] without restricting the system scalability.

In this study the parallel applications are kept apart from the sequential workload in the system. (The interaction between sequential and parallel jobs has been studied by Arpaci et al. [2]). We assume that each application is assigned a number of processors between one and a maximum number that can be used effectively, p^{max}. In reality, time constraints, memory requirements or even debugging procedures may entail minimum allocation limits for some applications though. The main objective of the scheduler is the minimization of the mean response time provided by the system, since the rationale for parallel processing is the need for decreasing the response time of large time-consuming applications. The load parameters and application characteristics used by the scheduler in the present study are given in table 1.

3 The Scheduling Policies

In this section, we describe in detail some processor allocation algorithms demonstrated to perform well in previous studies. We also introduce the new rules that we plan to compare with the others.

A classification of scheduling policies that have been proposed for different parallel architectures up to now is given by Anastasiadis [1].

3.1 Dynamic Equipartition Policy

In the *Dynamic Equipartition (DYN-EQUI)* policy, the processors are dynamically partitioned as equally as possible among the applications in the system. Special provisions are taken so that no application is given more processors that it can use. When the number of processors allocated to an application changes, the application readjusts the number of running processes accordingly [23].

Variations of this policy (under several different names) have been evaluated in many comparative studies of multiprocessor scheduling policies. The Dynamic Equipartition based rules have generally been found to yield excellent performance in shared-memory environments (where the overhead of changing processor allocations can be kept low) ([4, 11, 14, 15, 23] etc.).

3.2 Adaptive Policies

Adaptive policies assign a number of processors to each application when it is initiated, and the processors are freed only when the application completes. The number of processors allocated may depend on the current system load and on any available estimates of application characteristics. Application characteristics of interest include *maximum parallelism*, which is the largest number of processors that can be used effectively by the application, *service demand*, which is the total computation required by the application, and the *execution time function*, which is the execution time as a function of the number of processors allocated.

Adaptive Static Partitioning

Setia and Tripathi [19] introduce the *Adaptive Static Partitioning (ASP)* policy. If no processors are available when a job arrives, it joins a FIFO queue. Otherwise, it is allocated the minimum of its maximum parallelism and the number of available processors. When a job completes, the

released processors are divided as equally as possible among the waiting jobs, with no application taking more processors than its maximum parallelism. At low loads, a job is usually allocated a number of processors close to its maximum parallelism, while at high loads the partition sizes tend to be smaller.

Adaptive Policy 1

Rosti et al. [18] introduce *Adaptive Policy 1 (AP1)* specifically for distributed memory systems. The target partition size at a given time is equal to the total number of processors in the system divided by the number of waiting jobs [18]. If no processors are available when a job arrives, it joins a FIFO queue. Otherwise, it is allocated a number of processors equal to the minimum of its maximum parallelism, the target partition size, and the number of available processors. At each job completion, the same rule is used for the allocation of the released processors to jobs from the FIFO queue. The last job activated gets the remaining available processors, even if that is fewer than either of its maximum parallelism and the target partition size [1].

Adaptive Equipartition

Here we introduce a new rule called *Adaptive Equipartition (AEP)*. Intuitively, the ideal allocation is to divide the processors in the system equally among all the running and waiting jobs. By definition, this cannot be done in a non-preemptive policy. However, we can use as a target partition size the total number of processors divided by the total number of jobs in the system, both waiting and running (whereas both ASP and AP1 use a target allocation that is a number of processors divided by only the number of waiting jobs).

If no processors are available when a job arrives, it joins a FIFO queue. Otherwise, it is allocated a number of processors equal to the minimum of its maximum parallelism, the target partition size

[1] Our experiments indicated that it is always beneficial, to dispatch a job when the number of available processors is less than the target partition size. In the original definition of the rule, such a final allocation of the remaining processors was not done.

and the number of available processors. At each job completion, the same rule is used for allocating the released processors among the queued jobs, in FIFO order.

3.3 Shortest Demand First Policies

A non-preemptive policy that exhibits good performance is the *Shortest Demand First (SDF)* policy. In scheduling jobs with perfect speedup, average response time is minimized by activating jobs one at a time in SDF order and given each one all P processors [22].

The arriving jobs are kept in the queue ordered according to non-decreasing total *demand*, that is, the execution time of the job on one processor. When processors are released, jobs in the queue are activated in order by allocating each a number of processors equal to the minimum of their maximum parallelism and the number of idle processors [4, 13]. When the SDF is used with a fixed maximum allocation limit, it is called SDF-Max. Such a fixed maximum allocation limit, independent of the application characteristics, was shown to be very helpful for non-preemptive policies to keep applications from being allocated too many processors when system load is high [4].

In addition to SDF as defined above, combinations of SDF with the previously defined adaptive policies can be introduced. For example, ASP can keep the waiting jobs in non-decreasing order of their total demand. This composite policy will be referred to as ASP(1) to indicate the usage of first order knowledge of individual job characteristics, namely, service demands. Similarly, we introduce the policies AP1(1) and AEP(1), which are the AP1 and AEP respectively, with the waiting jobs queued in non-decreasing order of their demands. The pure adaptive policies use only the maximum parallelism of each application in doing processor allocation, while the above composite policies require also the total demand of each application.

3.4 Differential Allocation Policies

McCann and Zahorjan [15] introduce the *Run-To-Completion* policy, where each released processor is allocated to the waiting job for which the expected reduction in elapsed execution time is greatest,

ensuring that as many jobs as possible are activated. Wu [24] extends the Run-To-Completion policy with a Shortest Demand First queue (*RTC-SDF*).

In general, use of the execution time function for each application allows the formulation of a non-linear discrete constrained optimization problem with the objective of minimizing the total execution time of the waiting jobs [15, 17, 22, 24]. If we assume m waiting jobs, with $T_j(n)$ being the execution time function of job j, p_j^{max} the corresponding maximum parallelism, p_j the unknown processor allocation to job j, and P the number of available processors, then we have the following definition:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{m} T_j(p_j) \\ \text{subject to} \quad & \sum_{j=1}^{m} p_j \leq P \\ & p_j \in \{0, 1, 2, \ldots, p_j^{max}\}, \quad j = 1, 2, \ldots, m. \end{aligned} \tag{1}$$

The problem is tractable in the case of convex execution time functions [10]. A simple algorithm, which finds the optimal solution in a time that is exponential in the input size, has been proven correct [10]. The idea is to allocate the available processors one at a time to the waiting job that achieves the largest decrease in its response time. We use the name *Differential Allocation Policy (DIF)* to describe this aspect of allocation policies.

However, the formulation (1) presumes that all jobs are initially available and are all executed concurrently. There is no provision for jobs arriving over time or for activating only a subset of the jobs at a time. It may be beneficial for the system performance to activate a few jobs with large partitions and leave the remaining jobs in the waiting queue (at low loads), or, alternatively, to activate as many jobs as possible even with one processor each (at high loads). Thus, determination of which and how many jobs to activate requires further consideration.

According to the above definitions, RTC-SDF can be thought of as an extension of ASP with

SDF and DIF. The SDF component refers to the waiting jobs being queued in non-decreasing order of their demand. The strategy of activating as many jobs as possible coincides with allocating to each waiting job the number of processors that would be granted by ASP. After the jobs for activation have been determined, the DIF component determines the actual distribution of the free processors among them. Since it is ensured (by ASP) that each of the activated jobs will be allocated at least one processor, the difference $T(p) - T(p+1),$ for $p \geq 1$ is used by DIF to identify the job for which the execution time would be reduced the most by the allocation of one more processor.

We introduce extensions of the rules discussed earlier that include the DIF policy. To signify that they require a second level of application knowledge, namely $T_j(p_j)$ for $p_j > 0$ for each application, we call them *ASP(2), AP1(2),* and *AEP(2)*, respectively. In each of these rules, the order of the waiting jobs remains the same (SDF). However in AP1(2), the number of waiting jobs to be activated follows from the number of processors that each job would obtain according to AP1. Similarly in AEP(2), the number of waiting jobs to be activated is determined by the number of processors that would be allocated to them by AEP. The actual distribution of free processors among the jobs for activation is again decided based on the maximum differences $T(p) - T(p+1)$ for $p \geq 1$.

In table 2, a summary of the policies is presented, and the type of workload information each of them uses is shown. The *waiting jobs* and *running jobs* parameters correspond to the numbers of waiting and running jobs, respectively, while $T(p)$ corresponds to the execution time of the application with p allocated processors. The parameter p^{max} represents the maximum parallelism of the individual jobs.

4 Analytic Comparison of AEP and AP1

In order to clarify the distinctive feature of AEP, we examine the ratio of waiting time to execution time over the range of system loads for AEP and AP1. We assume that all the jobs in the system are of the same type and therefore can be described with the same execution time function, $T(n)$,

Scheduling Policy	Load Parameters	Application Characteristics
ASP	Waiting Jobs	p^{max}
AP1	Waiting Jobs	p^{max}
AEP	Waiting Jobs, Running Jobs	p^{max}
SDF		p^{max}, $T(1)$
DIF	Waiting Jobs	p^{max}, $T(p)$ for $p > 0$
DYN-EQUI	Waiting Jobs, Running Jobs	p^{max}
ASP(1)	Waiting Jobs	p^{max}, $T(1)$
AP1(1)	Waiting Jobs	p^{max}, $T(1)$
AEP(1)	Waiting Jobs, Running Jobs	p^{max}, $T(1)$
ASP(2)	Waiting jobs	p^{max}, $T(p)$ for $p > 0$
AP1(2)	Waiting Jobs	p^{max}, $T(p)$ for $p > 0$
AEP(2)	Waiting Jobs, Running Jobs	p^{max}, $T(p)$ for $p > 0$

Table 2: The policies and the information they need.

where n is the number of allocated processors.

The processor occupancy for each job is $n\,T(n)$, when the parallelization overheads are taken into account. Thus the *actual load* on the system ρ_a, if every job were allocated n processors, would be :

$$\rho_a = \frac{\lambda\, n\, T(n)}{P}, \tag{2}$$

where λ is the arrival rate and P the total number of processors in the system. By applying Little's law we have:

$$N = \lambda\, R(n),$$

where N is the total number of jobs in the system and $R(n)$ the average response time of the jobs. From these two equations we get :

$$n\,\frac{T(n)}{R(n)} = \rho_a\,\frac{P}{N}$$

In addition, $R(n) = W(n) + T(n)$, that is, the response time $R(n)$ is the sum of the waiting time $W(n)$ and execution time $T(n)$. If we set $n = \frac{P}{N}$, which is equal to the target partition size in AEP, we obtain:

$$W(n) = \frac{1 - \rho_a}{\rho_a} T(n) \tag{3}$$

Alternatively, we may combine equation (2) with the following equation:

$$L = \lambda W(n)$$

where L is the queue length of the waiting jobs:

$$n \frac{T(n)}{W(n)} = \rho_a \frac{P}{L}$$

If we set $n = \frac{P}{L}$, which is equal to the target partition size in AP1, we get:

$$W(n) = \frac{1}{\rho_a} T(n) \tag{4}$$

Thus, the difference between rules AEP and AP1 is captured in the factors $\frac{1-\rho_a}{\rho_a}$ and $\frac{1}{\rho_a}$ in equations (3) and (4) respectively that describe how the waiting time $W(n)$ changes with respect to $T(n)$ at different load levels. At high loads, where ρ_a approaches one, policy AP1 will keep $W(n)$ at a non-zero value $T(n)$, while the waiting time under AEP will become negligible with respect to $T(n)$, as follows from equations (4) and (3), respectively. The behavior of these two equations is illustrated in figure 1. The AEP policy will display a more conservative behavior than AP1 by giving fewer processors to each job at higher system loads.

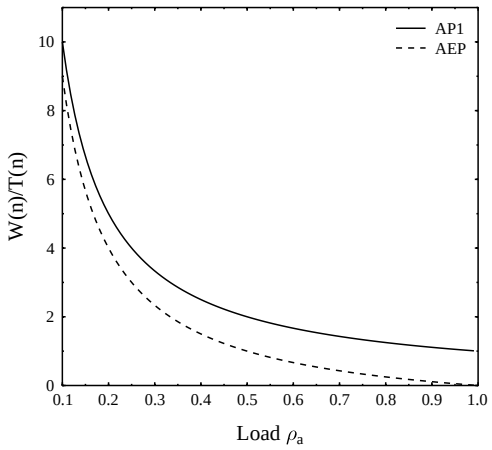


Figure 1: The ratio $\frac{W(n)}{T(n)}$ for AEP and AP1.

5 Simulation Model

We have used simulation modeling to compare the performance of the scheduling policies presented in the previous section. The experiments covered a wide range of application characteristics and arrival rates.

5.1 System Parameters

In our experiments, we concentrate on a system with $P = 32$ independent nodes and the general system features indicated in section 2. The effects of the memory requirements and the communication or synchronization latencies are not represented explicitly in the system model. Instead, they appear implicitly in the shape of the job execution time functions. By covering several different types of job execution behaviors, we expect that various architectural characteristics are captured, as well.

Preemption and scheduling overheads are ignored. In this way, the performance of the DYN-EQUI policy is ideal and thus can serve as benchmark for the remaining policies. The other algorithms are non-preemptive, and therefore the delays from processor reallocations are avoided. The computational requirements of the scheduling algorithms are assumed to be negligible. This is

valid for the number of nodes typically involved in workstation clusters.

5.2 Workload Parameters

A wide range of representative applications can be modeled by utilizing the execution time function form introduced by Sevcik [22]:

$$T(p) = \phi(p)\frac{W}{p} + \alpha + \beta p. \tag{5}$$

The parameter p is the number of processors allocated to the job and W the essential computational work. The parameter $\phi(p)$ represents the degree to which the work is not evenly spread across the p processors, and α stands for the increase of the work per processor due to parallelization. Finally, β reflects the communication and congestion delays that increase with the number of processors.

It is evident that, by choosing different values for the parameters $\phi(), W, \beta$ and α, we obtain descriptions of jobs with different characteristics and inherent structures.

Imbalance and Essential Work

The parameter $\phi(p)$ has been taken equal to one, since the real measurements conducted by Wu [24] indicate that it can be treated as constant typically with a value in the range 1.1 to 1.2.

The computational work W represents the minimum total service demand of the job.(β and α are typically smaller than W [24]). To obtain realistic workload parameters, we use the statistics gathered on a 128-node *iPSC/860* hypercube message-passing system at NASA Ames by Feitelson and Nitzberg [9]. We calculated the total demand of the jobs of each partition size, by multiplying the partition size with the respective mean job runtime. Since the jobs with 32-128 nodes use more than 90% of the system resources (node-seconds), we treated them as a separate class from those with sizes 1-16.

The average demand for the small jobs is equal to 1.3, while that for the large ones is 101

(thousands of seconds). In addition, large jobs account for $\frac{1}{8}$ of jobs in the workload with small jobs accounting for the other $\frac{7}{8}$. The generation of the W values is done by using a 2-stage hyper-exponential distribution, as in other simulation studies for the simulated workload [4]. The corresponding mean value is 13.76 and the coefficient of variation is 3.5.

Maximum Parallelism

For the execution time function (5), the maximum parallelism of a job is given by Sevcik [22]:

$$p^{max} = \begin{cases} \lceil \sqrt{\frac{W}{\beta}} \rceil & \text{if } \beta \neq 0 \\ \infty & \text{if } \beta = 0 \end{cases} \tag{6}$$

In the simulated workload, jobs had p^{max} values equal to 4, 16 or 64, each with equal probability. These values correspond to 12%, 50% and 100% of the 32 processors in the system model. Since there has been no reliable study of the actual maximum parallelism distribution of real applications, we assume it to be uniform, like the application partition size distribution observed by Feitelson and Nitzberg [9].

Job Speedup

The fundamental job characteristic that remains to be defined is the speedup function, $S(p)$, where p is the number of processors. It can be derived from equation (5), as follows:

$$S(p) = \frac{T(1)}{T(p)} = \frac{W + \beta + \alpha}{\frac{W}{p} + \beta\, p + \alpha} \tag{7}$$

By dividing both the numerator and denominator by W, and replacing $\frac{\beta}{W}$ with $\frac{1}{(p^{max})^2}$, we get:

$$S(p) = \frac{1 + \frac{1}{(p^{max})^2} + \frac{\alpha}{W}}{\frac{1}{p} + \frac{1}{(p^{max})^2} p + \frac{\alpha}{W}}. \tag{8}$$

Thus the speedup has been expressed as a function of the maximum parallelism p^{max} and the ratio $\frac{\alpha}{W}$. In order to indicate the dependence of $S(p)$ on p^{max}, we transform α into $W\left(\frac{1}{(p^{max})^2}\right)^{\mu}$, for $p^{max} > 1$:

$$S(p) = \frac{1 + \frac{1}{(p^{max})^2} + \left(\frac{1}{(p^{max})^2}\right)^{\mu}}{\frac{1}{p} + \frac{1}{(p^{max})^2}p + \left(\frac{1}{(p^{max})^2}\right)^{\mu}} \qquad \mu \in R \cup \{+\infty\} \tag{9}$$

From this equation, we see that the speedup curve is completely defined by the values of p^{max} and μ. In addition, if we let $\mu \to +\infty$ and $p = p^{max}$, assuming $\beta \neq 0$:

$$S(p) = \frac{1 + (p^{max})^2}{2p^{max}} \approx \frac{p^{max}}{2}$$

Thus, the structure of equation (5) causes speedup to be sublinear at numbers of processors close to p^{max}, even for $\alpha = 0$ when $\beta \neq 0$

Equations (6) and (9) allow us to specify the distributions of the maximum parallelism and speedup curve shape of the represented applications. In our experiments, we used four different workloads:

1. The workload WK1 consists of curves with relatively good speedup, to the degree that this is permitted by the value of p^{max}. They correspond to $\mu \to +\infty$.

2. The workload WK2 consists of jobs with $\mu = 0.4$, and speedup not as good as in WK1.

3. The workload WK3 consists of jobs with poor speedup, corresponding to $\mu = 0.2$.

4. The workload WK4 contains jobs with all three speedup types, each appearing with approximately equal frequency.

Each of the values of $\mu \in \{0.2, 0.4, +\infty\}$ and the mixed case have been combined with values of $p_{max} \in \{4, 16, 64\}$ in order to generate 12 separate job types. Each workload incorporates three of these job types, as they are defined by the respective values of μ. The results for the pure speedup

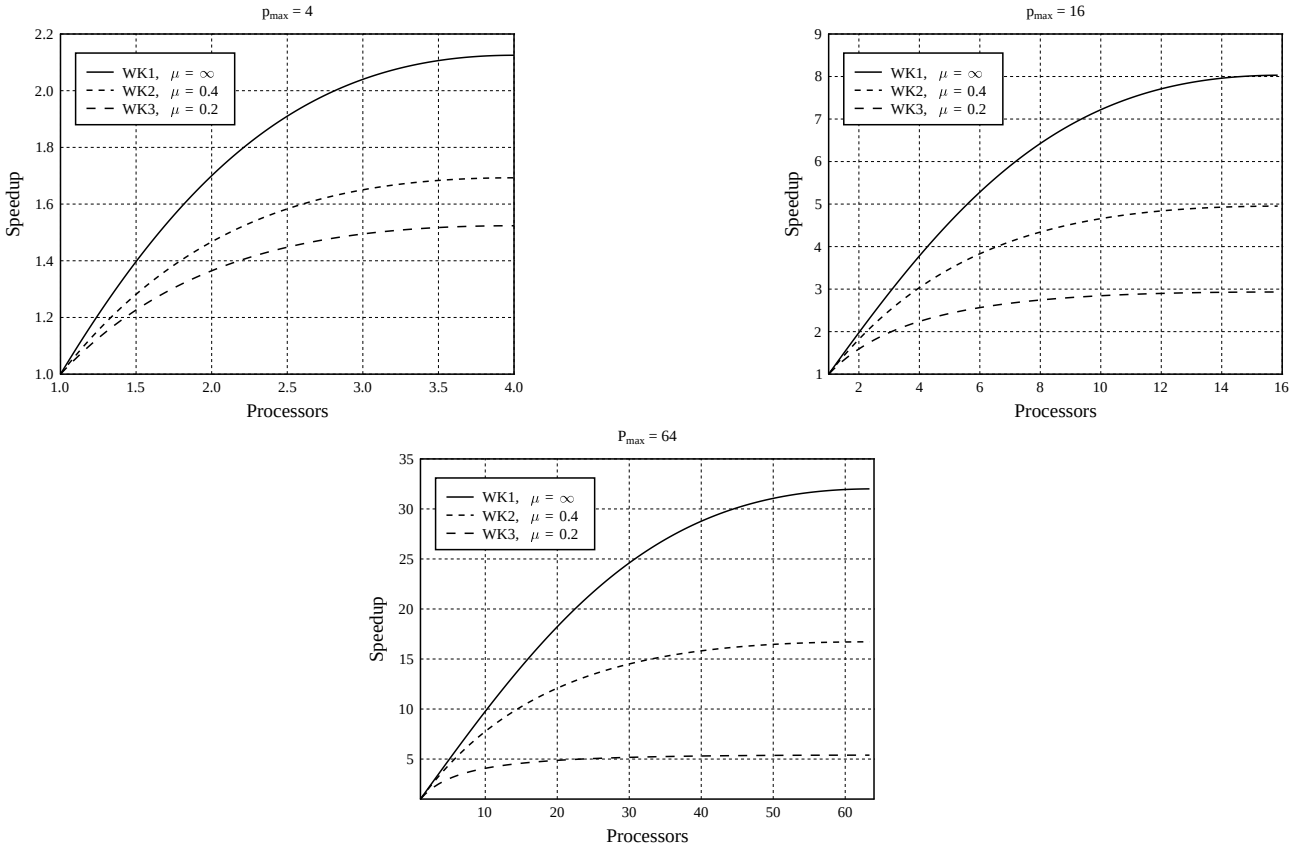


Figure 2: Speedup curves for $\mu \in \{0.2, 0.4, +\infty\}$ and $p_{max} \in \{4, 16, 64\}$.

types are shown in figure 2. These curves are intentionally similar to the speedup curves that were used in the experiments of Rosti et al. [18] and which were derived from a real application for several different inputs. However, in our study, we have three different maximum parallelism values $\{4, 16, 64\}$, instead of the one (16) that was used by Rosti et al. [18].

5.3 The Arrival Process

The *offered load* of a multiprocessor system with P servers is defined as follows :

$$Load = \frac{E(T(1))}{P \times Mean\ Interarrival\ Time} \tag{10}$$

where $E(T(1))$ is the mean total execution time of the jobs on one processor [21]. In the case of the jobs that have been used in our study, we have:

$$E(T(1)) = E(W) + E(\beta) + E(\alpha).$$

We already know that $E(W) = 13.76$. By using the theorem of total expectation across the three different values of p^{max}, we find $E(\beta) = 0.30$ and

$$E(\alpha) = \begin{cases} 0.0, & \text{if } \mu = +\infty, \\ 2.0, & \text{if } \mu = 0.4, \\ 5.0, & \text{if } \mu = 0.2, \\ 2.3, & \text{with mixed speedup.} \end{cases}$$

An exponential distribution with the Mean Interarrival Time derived from equation 10 was used for the generation of the interarrival times in the system. Table 3 summarizes the values for all the parameters that were used in our simulation model.

6 Experimental Results

As the primary performance measure for the different policies, we have used the *Response Time* normalized by the corresponding response time of the DYN-EQUI assuming zero reconfiguration overhead. Each comparison is depicted at five separate load levels, namely 10%, 30%, 50%, 70% and 90%.

To summarize the simulation procedure, in each replication, the first 500 jobs are used to warm up the system and their performance is ignored. The reported statistics correspond to the subsequent

Parameter	Description of Values
Processors P	32 of equivalent computing capacity
Execution Time $T(p)$	$\phi(p)\frac{W}{p} + \alpha + \beta p$
Essential work W	$F(W) = 0.125 \times (1 - e^{-W/101}) + 0.875 \times (1 - e^{-W/1.3})$ with $E(W) = 13.76$, and $CoV = 3.5$
Max Parallelism p_{max}	$\in \{4, 16, 64\}$ with equal frequencies
Parameter β	$\frac{W}{(p^{max})^2}$ and $E(\beta) = 0.30$
Parameter α	$W\left(\frac{1}{(p^{max})^2}\right)^{\mu}$
Speedup $S(p)$	$\left(1 + \frac{1}{(p^{max})^2} + \left(\frac{1}{(p^{max})^2}\right)^{\mu}\right) / \left(\frac{1}{p} + \frac{1}{(p^{max})^2}p + \left(\frac{1}{(p^{max})^2}\right)^{\mu}\right)$
WK1	$\mu = +\infty$, $E(\alpha) = 0.0$, and $E(T(1)) = 14.06$
WK2	$\mu = 0.4$, $E(\alpha) = 2.0$, and $E(T(1)) = 16.06$
WK3	$\mu = 0.2$, $E(\alpha) = 5.0$, and $E(T(1)) = 19.06$
WK4	$\mu \in \{0.2, 0.4, +\infty\}$ each with equal frequency $E(\alpha) = 2.34$, and $E(T(1)) = 16.40$
Interarrival Time t	$f(t) = \lambda e^{-t\lambda}$ with $\frac{1}{\lambda} = \frac{E(T(1))}{Processors \times Load}$

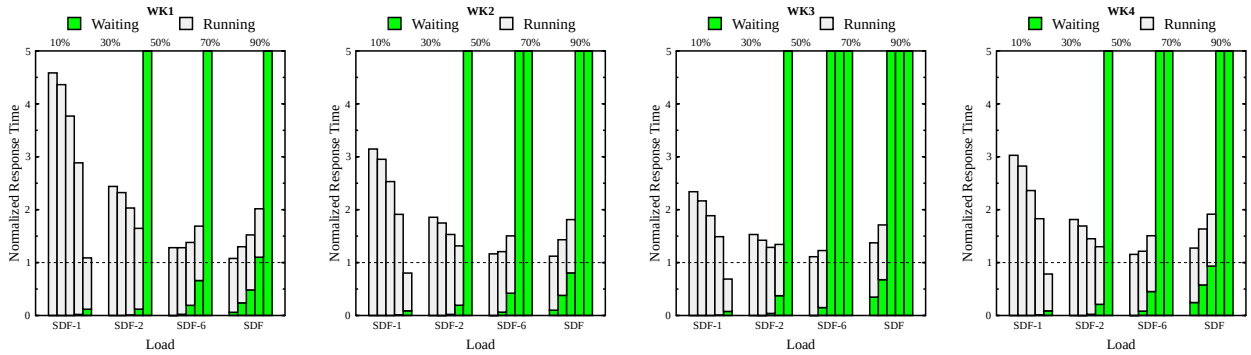
Table 3: All the system parameters as used in the experiments.

19,500 jobs. The performance of the jobs that arrive after the first 20,000 jobs is omitted. The policy is considered as saturated when the generation of 10,000 additional jobs is reported without the termination of the 19,500 intervening jobs. In this case, the mean response time is considered infinite. The number of replications is chosen to be large enough to give a 95% confidence interval with half-width no greater then 5% of the observed mean response time.

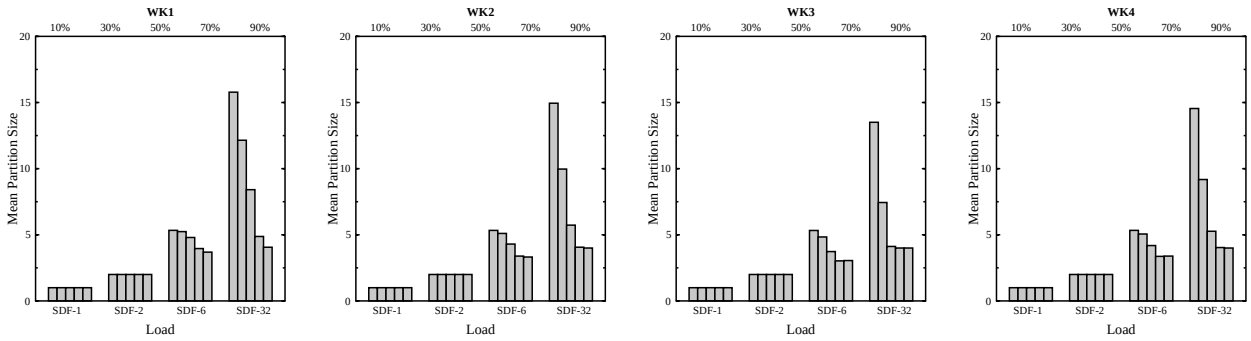
6.1 The Inadequacy of Fixed Allocation Limits

Initially, we examine four variations of the Shortest Demand First policy. One is only limited by the 32 processors in the system, while the other three are SDF-Max policies with Max=1, Max=2 and Max=6 processors. Figure 3 illustrates their normalized response times, and the respective mean partition sizes for all the four workloads. Those bars of the diagrams that reach the upper border line imply saturation of the policy for the corresponding load levels.

Our main observation is that there are particular loads at which the individual policies perform



(a) Response Time Normalized with respect to DYN-EQUI. The two components of each bar correspond to the waiting and running time of the applications.



(b) Mean Partition Size versus System Load.

Figure 3: Each group of five bars corresponds to one of the SDF-Max=1, SDF-Max=2, SDF-Max=6 and SDF from left to right. Each bar within a group corresponds to a load level between 10% and 90% from left to right, as well.

close to the DYN-EQUI. However only the SDF-Max=1 avoids saturation at at all loads, although it performs badly in most cases (except the level 90%). Larger allocation limits cannot provide the necessary adaptability to the policies to keep the response time bounded; this holds even for Max=2. In particular, for good speedup (WK1), the policies SDF, SDF-Max=2, SDF-Max=6 saturate at load 90%. For poor speedup (WK3), even a load of 50% makes the mean response time of SDF-Max=6 and SDF infinite. The excellent performance reported for SDF in a previous study [4] is illustrated there only for good speedup jobs (WK1) and load 70%. Our experiments do not contradict this specific result. In figure 3(a) notice that the higher the system load, the smaller the

Max parameter must be to obtain the best performance.

In figure 3(b), we can see the mean partition size for all the reported response times. While the SDF-Max=1 and SDF-Max=2 keep the partition size at constant values of 1 and 2, respectively, the other policies fail to lower their mean partition size at levels significantly less than four across the different speedup types and loads. The value four is the minimum p^{max} value of the jobs (the other two being 16 and 64). This proves that giving a job the minimum of its maximum parallelism and the current number of free processors, although this provides some flexibility to the policy, is not sufficiently adaptable for a wide range of workload conditions.

In general, hard limits at the maximum partition sizes inherently optimize the performance of the policy for a particular arrival rate and workload type. This means that tuning the parameter Max is always necessary. Of course, this cannot provide a solution to the scheduling problem of general purpose machines.

6.2 Comparison of the Adaptive Policies

In figure 4(a), the normalized response time of the three adaptive policies ASP, AP1 and AEP can be observed. We concentrate on the workload WK4, since the results in the other three workload are similar (see the Appendix). An important observation is that the policies ASP and AP1 have almost the same response time for all workload types [2].

In figure 4(b), the mean partition sizes as a function of load are depicted. The respective partition sizes of ASP and AP1 are the same. At low loads this could be expected, since for small queue lengths both policies tend to give all the free processors to the next arriving job. Also, it seems that at higher loads the queue length does not become large enough to make AP1 significantly different from ASP.

Another important observation from figure 4(a) is the advantage of AEP over the other two

[2]These two rules have not been compared previously with respect to mean response time. In the article that introduces AP1, *power*, which is defined as the ratio of throughput to response time, is used as performance measure [18].

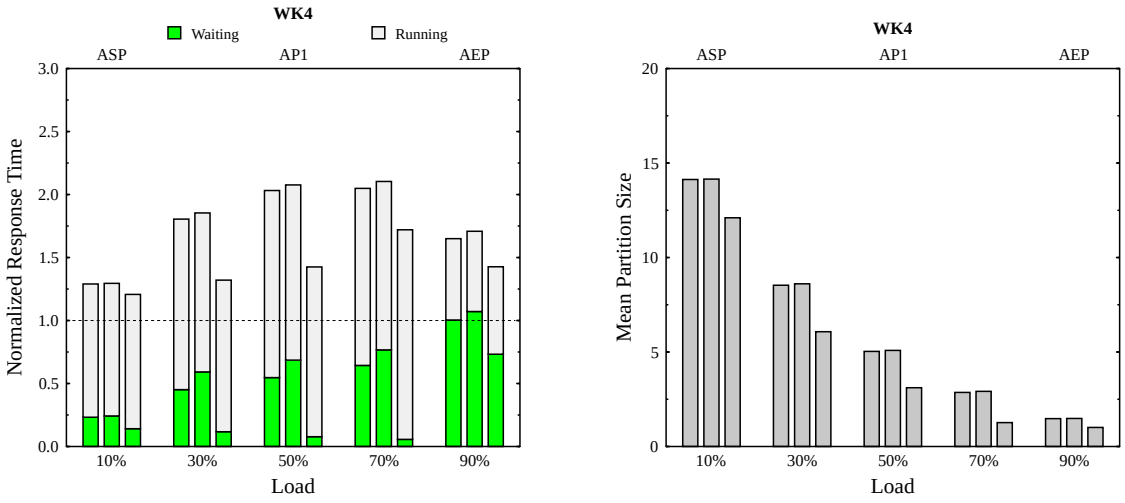


Figure 4: Each group of three bars corresponds to a load level between 10% and 90% from left to right. Each bar within a group corresponds to one of ASP, AP1, AEP. (a) Response Time Normalized with respect to DYN-EQUI. (b) Mean Partition Size versus Load.

policies. The waiting time tends to be smaller with AEP, as was predicted in Section 4. In almost all the cases, AEP correctly decides to provide smaller partition sizes relative to ASP and AP1, since this improves the mean response time of the system.

6.3 Combining the Adaptive Policies with SDF

Taking into account the very good performance that Shortest Demand First demonstrates at the different load levels with the appropriate choice of the Maximum Allocation parameter, we decided to examine SDF combined with the Adaptive Policies. The main advantage of the adaptive policies is to automatically adjust the partition sizes using the load information, as indicated by the length of the queue formed by the waiting jobs or all the jobs in the system (fig. 4(b)).

In figure 5(a), we can see the improvement in the system performance that this combination achieves. At high loads, we get a decreased response time, which in some cases is less than half the response time of the corresponding pure adaptive policies. At very low loads, there is no improve-

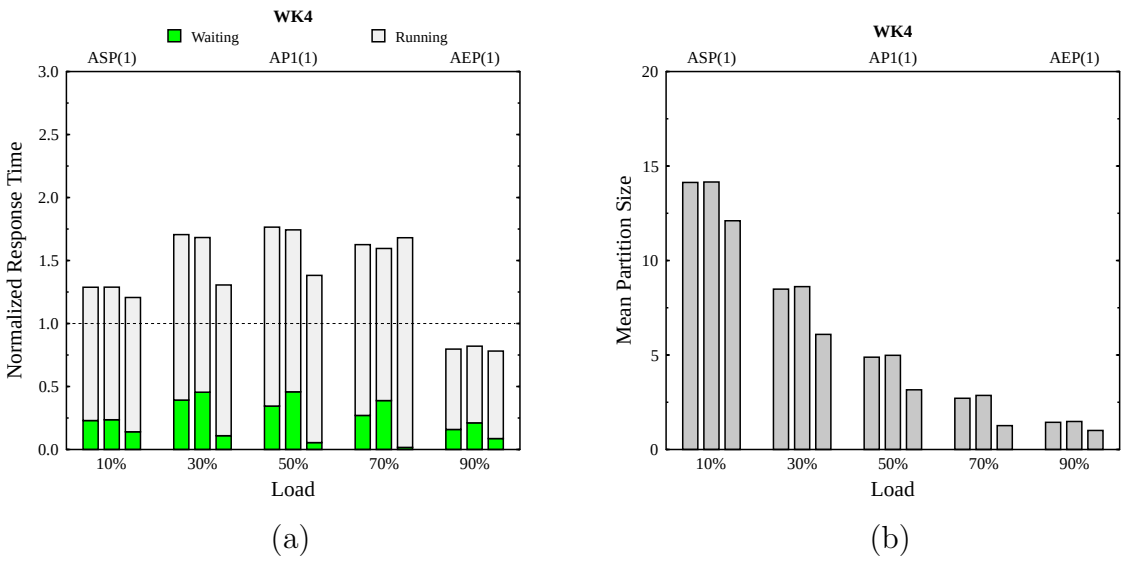


Figure 5: Each group of three bars corresponds to a load level between 10% and 90% from left to right. Each bar within a group corresponds to one of ASP, AP1, AEP combined with SDF. (a) Response Time Normalized with respect to DYN-EQUI. (b) Mean Partition Size versus Load.

ment because most arriving jobs find an almost empty system. Of course, the queue discipline can have a significant impact on the response time only if some queuing occurs.

In figure 5(b), we can see the mean partition size as function of the load in the system. Its comparison with that of the previous section (fig. 4(b)) shows that SDF does not restrict the adaptability of the policies ASP, AP1 and AEP. On the other hand, there is a remarkable decrease in the waiting time of the jobs at high loads. Thus, the desired properties of the adaptive policies are orthogonal to those of SDF. Therefore, we have a clear advantage by combining these different features, and not treating them as separate algorithms with incompatible characteristics as has been done previously [4].

An important observation from figure 5(a) is the change in the relative performance of AEP and ASP or AP1. Although the response times of the last two policies remain nearly identical to each other, it seems that the improvement previously demonstrated by AEP(1) is considerably reduced. Actually, due to the reduction in the waiting time from SDF, AEP(1) loses the advantage of minimizing the waiting time by granting fewer processors than the other two policies.

6.4 Combining the Adaptive Policies with SDF and DIF

The final enhancement that we add to the adaptive policies is the improved distribution of released processors among waiting jobs. As we noted in section 3, the greedy approach of dispatching as many waiting jobs as possible is not intuitively the best [15, 24]. As will be shown below, there is a considerable performance improvement by replacing the greedy ASP with AEP.

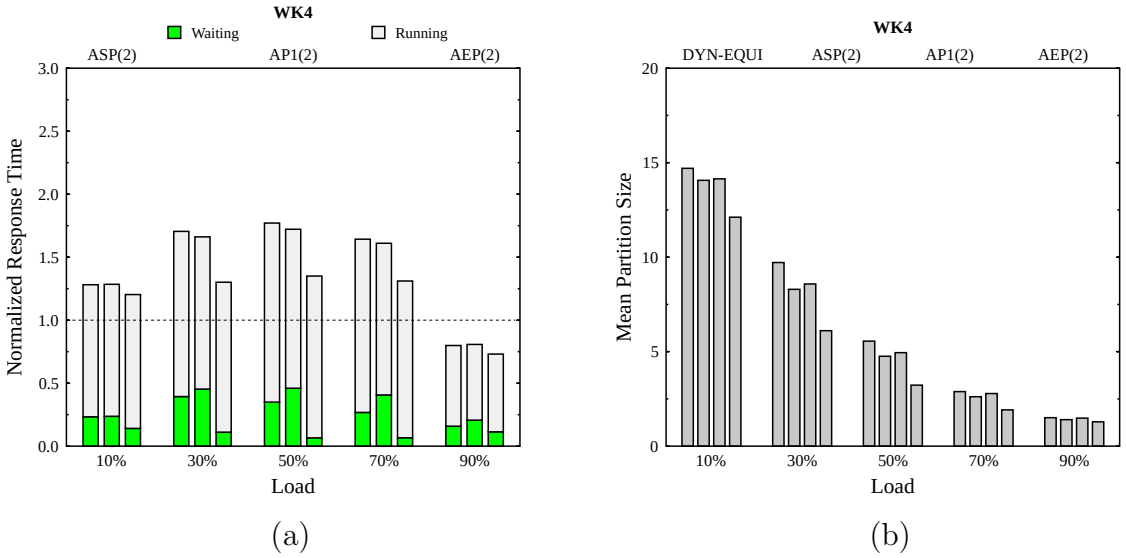


Figure 6: Each group of three(four) bars corresponds to a load level between 10% and 90% from left to right. Each bar within a group corresponds to one of (DYN-EQUI,) ASP, AP1, AEP combined with SDF and DIF. (a) Response Time Normalized with respect to DYN-EQUI. (b) Mean Partition Size versus Load.

In figure 6, we have depicted the normalized Response Time of all three combinations, ASP(2), AP1(2) and AEP(2). The most important conclusion is that AEP(2) always performs better than the other two policies. Another conclusion is that the improvement from using DIF does not affect the performance of ASP(1) and AP1(1). The main reason for this is the SDF policy itself. It seems that the allocation optimization, as described in Section 3, is most effective when it can discriminate between jobs with small and large demands. Actually, the essential computation W determines the potential of a job to reduce its response time when given additional processors. The optimization procedure is not affected by the value of α, since α does not participate in the derivative of the

execution time function [22]. Furthermore β is important but in typical workloads is much less than W [24]. Thus the ordering of the jobs according to the runtime on a single processor ($W + \beta + \alpha$) separates the jobs into groups with similar potential for processor exploitation.

DIF is effective in the case of AEP due its tendency to give few processors to each job, and therefore to be sensitive to the final processor distribution. Of course, slightly changing the allocation size to a job with five processors, is much less effective than with a job that has received only one. In the second case, an additional processor may even halve the execution time of the job. This argument does not hold in the case of ASP and AP1, due to their tendency to keep the mean partition size significantly larger than that of AEP (fig. 6(b)).

In addition, from figures (5) and (6), it is evident that both the partition size and the waiting time remain the same in the case of AEP, when compared to that without DIF. Therefore, the DIF technique affects only the execution time as was expected.

6.5 Comparison with Dynamic Equipartition

Finally, we will discuss how the final three run-to-completion algorithms, compare with the zero overhead DYN-EQUI. First, at load 90% all three policies perform better than the DYN-EQUI. Observing the waiting times, we realize that at high loads the preemptive behavior of DYN-EQUI fails to make the waiting time zero. On the other hand, the mean partition size of the adaptive policies is very close to that of DYN-EQUI (fig. 6(b)). In addition, the enhanced adaptive policies have the advantage of the Shortest Demand First discipline, which minimizes the expected waiting time.

Dynamic algorithms could also benefit from using load and job specific information [24]. However, the estimation of the execution time function for a dynamic policy is not as straightforward as in the case of static partitioning, due to the continuous change in the number of processors. But both SDF and DIF need some form of execution time representation in order to be realized. The approximation method proposed by Nguyen et al. [16] for shared-memory machines, where

the best partition size of an application is determined by sample executions on different numbers of processors, induces overhead which can become non-negligible in distributed-memory systems.

Also, by comparing our algorithm AEP(2) with ASP(2), we realize that, in the general case as is captured by workload WK4, the former manages to be within 30% of DYN-EQUI, while the latter becomes more than 75% worse at medium load (50%). A final issue is that, as was reported in the previous section, DIF is ineffective and it could even be omitted in the case of ASP and AP1. However, since even the support of the SDF rule requires knowledge of the execution time function, the DIF in the case of AEP comes almost for free, and cannot be considered to be significant additional overhead required by the algorithm for the reported improvement to be achieved.

7 Conclusions and Future Work

The goal of this work was the understanding and improvement of the most promising scheduling policies that could serve the general-purpose parallel processing requirements of workstation cluster users. Usually, different types of information are used by separate policies, and it is unclear how they interact and what benefit each yields under different system and workload conditions.

A significant step in our work was the definition of simple scheduling rules and the clarification of the type and level of information that each of them needs. With respect to system load, the policies ASP, AP1 and AEP were described. The AEP is introduced for first time and is intended to improve the robustness by using the total number of jobs in the system. With respect to application characteristics, we investigated two different approaches, the SDF and DIF. Finally, the dynamic policy DYN-EQUI was defined. In order to unify our results with previous ones, we assumed that all the policies have information about the maximum parallelism of the individual applications. Actually, this allocation upper bound is only known by those policies that have knowledge of the job execution time function.

We represented the characteristics of a wide range of applications with proper choices of param-

eters in Sevcik's execution time model. Three separate workloads were formed with applications having different speedup characteristics. A fourth workload that incorporated all the other three in equal proportions was also included.

Our first major conclusion from the simulation experiments is that, at high loads and with applications having sublinear speedup, SDF fails to complete the jobs in finite time. This is not surprising, since SDF allocates processors up to the maximum requirement of the jobs, regardless of the load in the system. In addition, fixed maximum allocation limits cannot provide general improvement in SDF in contrast to previous claims to the contrary. They just optimize the performance for some particular workload and arrival rate.

Comparison of the three adaptive policies verifies that AEP tends to allocate fewer processors and to decrease the waiting time at higher loads, as was expected from our analysis. This behavior makes AEP perform better in the general case. No differences are found between the performance of ASP and AP1.

Our next step is to compare the adaptive policies when combined with SDF, called respectively ASP(1), AP1(1) and AEP(1). It is shown that the response time of all three is considerably decreased, due to the expected reduction in the waiting time due to SDF. For the same reason, the advantage of AEP of keeping waiting time low is much less important than before. However, it is important that the properties of SDF are orthogonal to those of the adaptive policies, and the combination leads only to performance improvement.

The final step is to add DIF to the three policies, thus obtaining ASP(2), AP1(2) and AEP(2). The gain for the policies relative to ASP(1) and AP1(1) is negligible. A reason for this is that SDF causes the jobs dispatched together to have similar total demand and therefore potential for decreasing their response time with additional processors. However, the gain of AEP(2) is considerable at high loads due to its tendency to allocate fewer processors than the other two policies.

Comparison among AEP(2), ASP(2) and DYN-EQUI shows that the former two always perform

better at very high loads. The reason is that at very high loads the job partition sizes of all three policies are very close to one. With the additional advantages of SDF and DIF, it is expected that AEP(2) and ASP(2) will be better. At the other loads, DYN-EQUI performs better, due to its capability of preemption. The difference for AEP(2) with the mixed workload (WK4) is typically about 30% of the response time relative to DYN-EQUI. The corresponding differences between ASP(2) and DYN-EQUI some times exceed 75%.

We conclude, that by exploiting properly the load and application characteristics information, we have managed to improve the performance of the non-preemptive policies and to come very close to that of zero-overhead Dynamic Equipartition. Thus, we have proven that it is possible to design efficient schedulers for network clusters where dynamic policies lead to unacceptable overheads.

An implementation of the proposed algorithm on an actual workstation cluster is a necessary next step for verifying the performance predicted by simulation. In addition, machines with different uniprocessor or multiprocessor configurations must be considered, along with applications with specific hardware requirements. Also, it is necessary to investigate the impact of the applications that have minimum processor allocation requirements. Finally, We believe that the limit of non-preemptive scheduling performance has not been reached yet. It is still an open question how application characteristics and system load parameters, whether employed as in our approach or not, can be used by still better scheduling policies.

References

[1] Stergios Anastasiadis. Parallel Application Scheduling on Networks of Workstations. Master's thesis, Department of Computer Science, University of Toronto, February 1996. Also available as CSRI technical report #342.

[2] Remzi H. Arpaci, Andrea C. Dusseau, Amin M. Vahdat, Lok T. Liu, Thomas E. Anderson, and David A. Patterson. The Interaction of Parallel and Sequential Workloads on a Network

of Workstations. In *ACM SIGMETRICS/PERFORMANCE Joint Conf. on Measurement and Modeling of Computer Systems*, pages 267–278, May 1995.

[3] Nicholas Carriero, Eric Freeman, David Gelenter, and David Kaminsky. Adaptive Parallelism and Piranha. *Computer*, 28(1):40–49, January 1995.

[4] Su-Hui Chiang, Rajesh K. Mansharamani, and Mary K. Vernon. Use of Application Characteristics and Limited Preemption for Run-To-Completion Parallel Processor Scheduling Policies. In *ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, pages 33–44, May 1994.

[5] Murthy V. Devarakonda and Ravishankar K. Iyer. Predictability of Process Resource Usage: A Measurement-Based Study on Unix. *IEEE Transactions on Software Engineering*, 15(12):1579–1586, December 1989.

[6] K. Dussa, B. Carlson, L. Dowdy, and K.-H. Park. Dynamic Partitioning in a Transputer Environment. In *ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, pages 203–213, May 1990.

[7] Andrea C. Dusseau, Remzi H. Arpaci, and David E. Culler. Effective Distributed Scheduling of Parallel Workloads. In *ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, May 1996. To appear.

[8] Michael J. Feeley, Brian N. Bershad, Jeffrey S. Chase, and Henry M. Levy. Dynamic Node Reconfiguration in a Parallel-Distributed Environment. In *3rd ACM Symposium Principles and Practice of Parallel Programming*, pages 114–121, 1991.

[9] Dror G. Feitelson and Bill Nitzberg. Job Characteristics of a Production Parallel Scientific Workload on the NASA Ames iPSC/860. In *IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 215–227, April 1995.

[10] Toshihide Ibaraki and Naoki Katoh. *Resource Allocation Problems: Algorithmic Approaches*. Series in the Foundations of Computing. MIT Press, 1988.

[11] Scott T. Leutenegger and Mary K. Vernon. The Performance of Multiprogrammed Multiprocessor Scheduling Policies. In *ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, pages 226–236, May 1990.

[12] Michael R. Leuze, Lawrence W. Dowdy, and Kee-Hyun Park. Multiprogramming a Distributed-Memory Multiprocessor. *Concurrency: Practice and Experience*, 1(1):19–33, September 1989.

[13] Shikharesh Majumdar, Derek L. Eager, and Richard B. Bunt. Scheduling in Multiprogrammed Parallel Systems. In *ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, pages 104–113, May 1988.

[14] Cathy McCann, Raj Vaswani, and John Zahorjan. A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146–178, May 1993.

[15] Cathy McCann and John Zahorjan. Processor Scheduling in Shared Memory Multiprocessors. Technical Report 89-09-17, Computer Science Department, University of Washington at Seattle, 1989.

[16] Thu D. Nguyen, Raj Vaswani, and John Zahorjan. Using Runtime Measured Workload Characteristics in Parallel Processor Scheduling. Technical Report UW-CSE-95-10-01, Department of Computer Science and Engineering, University of Washington, Seattle, October 1995.

[17] Kee-Hyun Park and Lawrence W. Dowdy. Dynamic Partitioning of Multiprocessor Systems. *International Journal of Parallel Programming*, 18(2):91–120, 1989.

[18] Emilia Rosti, Evgenia Smirni, Larry Dowdy, Giuseppe Serazzi, and Brian M. Carlson. Robust Partitioning Policies of Multiprocessor Systems. *Performance Evaluation*, 19(2–3):141–165, 1994.

[19] Sanjeev Setia and Satish Tripathi. A Comparative Analysis of Static Processor Partitioning Policies for Parallel Computers. In *Int'l Workshop on Modeling and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 283–286, January 1993.

[20] Sanjeev K. Setia, Mark S. Squillante, and Satish K. Tripathi. Processor Scheduling on Multiprogrammed Distributed Memory Parallel Computers. In *ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, pages 158–170, 1993.

[21] Kenneth C. Sevcik. Characterizations of Parallelism in Applications and Their Use in Scheduling. In *ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, pages 171–180, May 1989.

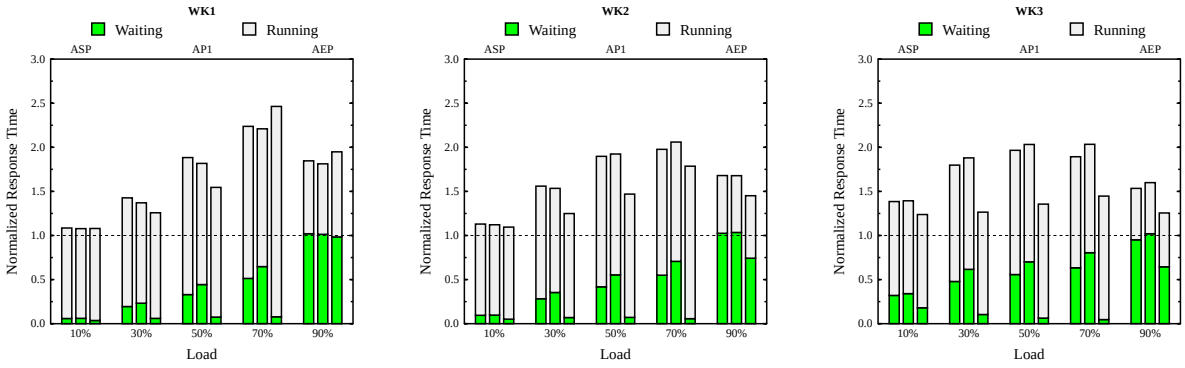
[22] Kenneth C. Sevcik. Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems. *Performance Evaluation*, 19(2–3):107–140, 1994.

[23] Andrew Tucker and Anoop Gupta. Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors. In *12th ACM Symp. Operating System Principles*, pages 159–166, December 1989.

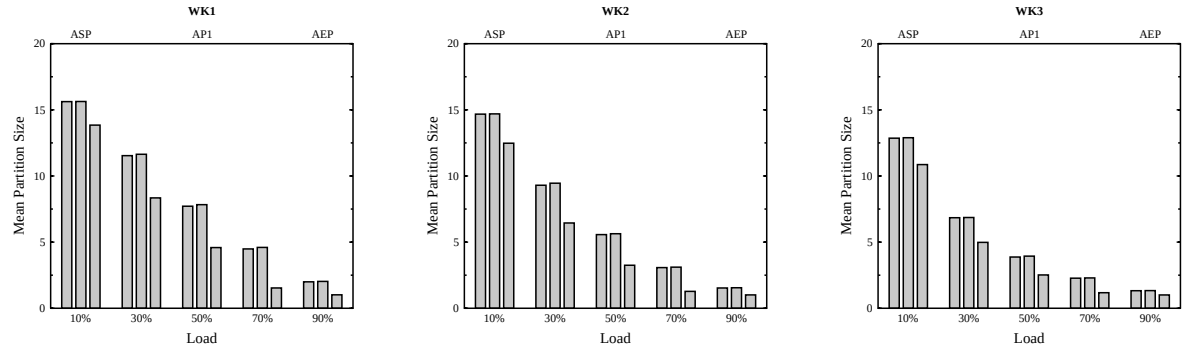
[24] Chee-Shong Wu. Processor Scheduling in Multiprogrammed Shared Memory NUMA Multiprocessors. Master's thesis, Department of Computer Science, Technical Report CSRI-341, University of Toronto, 1993.

[25] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software-Practice and Experience*, 23(12):1305–1336, December 1993.

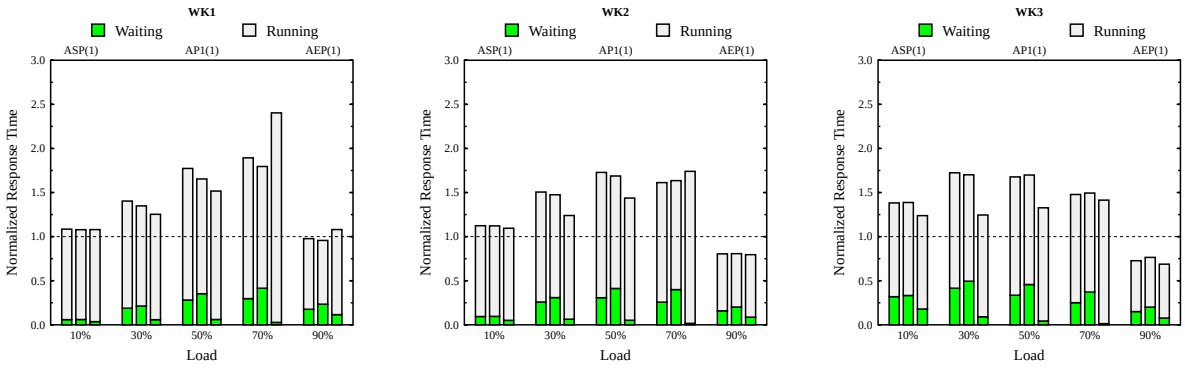
A Further Experimental Results



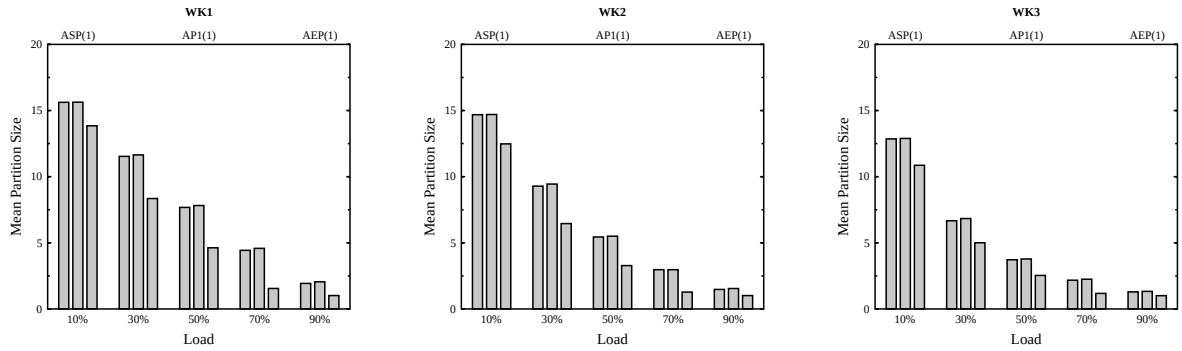
Normalized Response Time with respect to DYN-EQUI for ASP, AP1 and AEP.



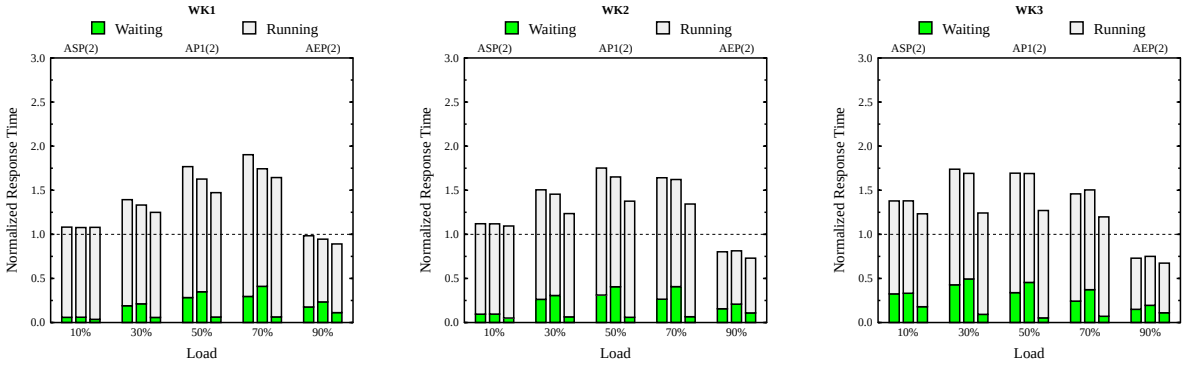
Mean Partition Size versus load for ASP, AP1, AEP.



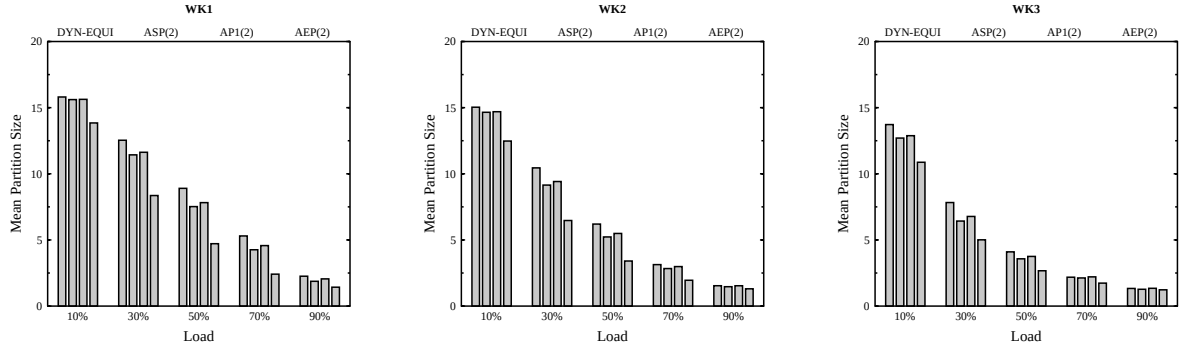
Normalized Response Time with respect to DYN-EQUI for ASP(1), AP1(1) and AEP(1).



Mean Partition Size versus Load for ASP(1), AP1(1), AEP(1).



Normalized Response Time with respect to DYN-EQUI for ASP(2), AP1(2) and AEP(2).



Mean Partition Size versus Load for DYN-EQUI, ASP(2), AP1(2), AEP(2).