Dynamic Partitioning in Different Distributed-Memory Environments

Nayeem Islam[1], Andreas Prodromidis[2], Mark S. Squillante[1]

[1] IBM T. J. Watson Research Center, Yorktown Heights NY 10598, USA
[2] Columbia University, New York NY 10027, USA

Abstract. In this paper we present a detailed analysis of dynamic partitioning in different distributed-memory parallel environments based on experimental and analytical methods. We develop an experimental testbed for the IBM SP2 and a network of workstations, and we apply a general analytic model of dynamic partitioning. This experimental and analytical framework is then used to explore a number of fundamental performance issues and tradeoffs concerning dynamic partitioning in different distributed-memory computing environments. Our results demonstrate and quantify how the performance benefits of dynamic partitioning are heavily dependent upon several system variables, including workload characteristics, system architecture, and system load.

1 Introduction

Parallel computer systems, consisting of numerous tightly- or loosely-coupled nodes, represent an increasingly important class of high-performance computing environments that make it possible to solve large and complex problems. Fundamental to realizing these performance benefits is the design of scheduling policies which allocate nodes among the parallel jobs submitted for execution in a manner that tends to minimize job response time and maximize system throughput. A number of scheduling strategies have been proposed for such parallel environments, each differing in the way nodes are shared among the jobs. One particularly important class of scheduling policies is based on *space sharing* where the nodes are partitioned among different parallel jobs.

There are three basic types of space-sharing policies. The *static partitioning* of the nodes into a fixed number of disjoint sets, each of which are allocated to individual jobs, is a space-sharing strategy that has often been employed in a number of commercial systems. This is due in part to its low system overhead and its simplicity from both the system and application viewpoints. The static scheduling approach, however, can lead to relatively low system throughputs and resource utilizations under nonuniform workloads [33, 26, 27, 30, 34], which can be common in scientific and engineering computing environments. *Adaptive partitioning* policies, where the number of nodes allocated to a job is determined when jobs enter and leave based on the current system state, have also been considered in a number of research studies [21, 38, 16, 26, 27, 32, 7, 30]. This approach tends to outperform its static counterparts by adapting partition sizes to the current load. On the other hand, the performance benefits of adaptive partitioning can be limited due to its inability to adjust scheduling decisions in response to subsequent workload changes. These potential problems are alleviated under *dynamic partitioning*, where the size of the partition allocated to a job can be modified during its execution, at the expense of increased overhead [37, 9, 21, 38, 17, 23, 26, 27, 34].

The relative runtime costs of a dynamic partitioning policy are heavily dependent upon the parallel architecture and application workload. In uniform-access, shared-memory systems, these overheads tend to be relatively small and thus the benefits of dynamic partitioning outweigh its associated costs. A number of research studies have made this quite clear, showing that dynamic partitioning outperforms all other space-sharing strategies in such environments [37, 21, 38, 17, 23]. In more distributed parallel environments, however, the overheads of a dynamic partitioning policy can be significant due to factors such as data/job migration, node preemption/coordination and, in some cases, reconfiguration of the application [9, 26, 27, 31].

There are several fundamental issues that must be considered in order to effectively exploit dynamic partitioning in distributed computing environments. First, the applications must be capable of executing on variable numbers of nodes and must be capable of reconfiguring the number of nodes on which it executes. In this paper we develop a system approach to provide this functionality for an important class of parallel applications. Moreover, our approach provides the system structure to extend this functionality to applications beyond those considered herein.

Another important issue concerns the overheads of dynamic partitioning in distributed computing environments, where a better understanding of these fundamental scheduling costs is needed to determine the manner in which such policies can be effectively employed in different distributed-memory systems. To complement and extend previous studies of repartitioning overheads for certain distributed-memory environments [26, 27, 31], we conduct a detailed measurement-based analysis of dynamic partitioning overheads in computing environments based on the IBM SP2 and a network of workstations. An experimental testbed is developed on both system architectures, which we use to obtain measurement data for distinct workloads composed of an important class of parallel applications. In our analysis we identify two different types of overheads: one is due to the system, such as process management, and the other is attributable to application reconfiguration.

We also use our experimental testbed together with an analytic model to analyze the impact of these overheads on the system performance characteristics of dynamic partitioning strategies in various distributed-memory environments. System measurements are used to parameterize, validate and complement the model, whereas the computational efficiency of the model allows us to examine a large design space. Our detailed analysis of dynamic partitioning and its associated overheads combine these experimental and analytical methods to yield fundamental insights into the performance characteristics of real parallel systems. Such coupling of experimental and analytical work is rare, and we believe it has proven to be an effective tool for parallel system design and analysis.

Our results show that the benefits of dynamic partitioning in distributed computing environments depend heavily upon the application workload as well as the reconfiguration overhead. We show that dynamic partitioning provides significant improvements in performance over other forms of space sharing under many workloads when the costs of repartitioning are fairly small relative to the workload execution requirements, and our results quantify these considerable performance gains. Under certain workload conditions, however, the costs associated with dynamic partitioning tend to outweigh its benefits, particularly at light to moderate system loads for the class of parallel applications considered.

In Section 2 we describe the scheduling policies considered. Section 3 presents various aspects of the experimental testbed used in this study, including the system hardware and software architectures, and the parallel application workload. We then briefly describe the analytic models used in this study. Sections 5 and 6 present some of the results of our experiments and quantitative analysis. Our concluding remarks are presented in Section 7.

2 Scheduling Policies

We now define the two main policies considered in this paper: a dynamic equi-partitioning (DEP) scheme and a static partitioning (SP) policy. Throughout this paper, we use P to denote the number of nodes in the system and we use M to denote the minimum number of nodes allocated to any job (i.e., the nodes are allocated in units of M). The maximum number of node partitions under each policy is therefore given by $N \equiv P/M$. Jobs that have not been allocated nodes wait in a first-come first-served (FCFS) *system queue*.

2.1 Dynamic Equi-Partitioning

A DEP policy basically attempts to equally divide the nodes among the jobs in the system, up to a maximum of the first N jobs. If a job arrives to the system when $i-1$ jobs are being executed, $1 \leq i \leq N$, then the nodes are repartitioned among the i jobs such that each job is allocated (on average) P/i nodes. A job arrival that finds $i \geq N$ jobs in the system is placed in the FCFS system queue to wait until a node partition becomes available. When one of the $i+1$ jobs in execution departs, $0 \leq i < N$, the system reconfigures the nodes allocations so that each job receives (on average) P/i nodes. A job departure when $i > N$ simply causes the job at the head of the system queue to be allocated the available partition, and no repartitioning is performed.

We consider a particular form of DEP in which the number of applications repartitioned upon a job arrival or departure is minimized. To better illustrate our policy, we present in Table 1 the various node allocation changes that occur in response to these events for an 8-node system with $M = 1$ (hence, $N = P = 8$).

Initial System State	State after arrival event	State after departure event
{ }	{8}	
{8}	{4,4}	{ }
{4,4}	{3,3,2}	{8}
{3,3,2}	{2,2,2,2}	{4,4}
{2,2,2,2}	{2,2,2,1,1}	{3,3,2}
{2,2,2,1,1}	{2,2,1,1,1,1}	{2,2,2,2}
{2,2,1,1,1,1}	{2,1,1,1,1,1,1}	{2,2,2,1,1}
{2,1,1,1,1,1,1}	{1,1,1,1,1,1,1,1}	{2,2,1,1,1,1}
{1,1,1,1,1,1,1,1}	{1,1,1,1,1,1,1,1}	{2,1,1,1,1,1,1}

Table 1. State transitions when applications enter and leave the system. The transitions are for an eight-node system.

2.2 Static (Adaptive) Partitioning

The system nodes are statically divided into K partitions each of size $S \equiv (N/K)\,M$, where we only consider values of K that evenly divide N; i.e., $K \in \{1, 2, \ldots, N/2, N\}$. A job arrival is allocated S nodes if one of the K partitions is available, otherwise the job waits in the FCFS system queue until a partition becomes free. Each parallel job is executed to completion without interruption and all S nodes are reserved by the application throughout this duration. Upon a job departure, the available partition is allocated to the job at the head of the system queue, if any. Since the node partitions cannot be modified, jobs do not incur any reconfiguration overhead. The only overhead incurred by each job is the cost to set up the job for execution on the S nodes allocated to it.

Our decision to consider equal-sized node partitions is motivated by the results of several studies (e.g., [30, 22]) showing that adaptive/static strategies in which the system is divided into equal-sized partitions outperform other adaptive/static policies when job service time requirements are not used in scheduling decisions. A number of research studies, under different workload assumptions, have also shown that adaptive partitioning yields steady-state performance comparable to that of the best static partitioning policy for a given system load [26, 27, 32]. Hence, when this relation holds, the mean job response time under adaptive partitioning is accurately approximated by the static policy that provides the lowest response time for a given load, and the results of Section 6 are also representative of a comparison between adaptive and dynamic partitioning policies.

3 Experimental System Platform and Applications

In this section we describe four aspects of our experimental platform: the hardware, the system software, the parallel applications and the workloads studied. Our focus is on distributed-memory systems where there are a set of independent nodes that do not share memory. Each node runs independent operating system images that communicate through message passing. The operating system runs a special distributed scheduler (DS) that interacts with applications to perform distributed space-sharing.

3.1 System Hardware and Operating System Configurations

We experiment with two different distributed-memory environments, namely a network of workstations (NOW) and the IBM SP2 machine.

NOW. A group of workstations connected by a token ring that has a bandwidth of 16 megabits/sec. The workstations are Model 980F machines which use 62 MIPS PowerPC processors. These machines run AIX 3.2.5.

SP2. The IBM SP2 is a distributed-memory multicomputer that is connected by a high-speed switch. We use TCP/IP to communicate over the switch to make the port of the software easy. The nodes run 133 MIPS PowerPC processors. These machine run AIX 4.1. The applications do not use the fast user-level communications of the switch.

Distributed Scheduler Architecture. A parallel application is submitted by a *launcher*, such as a shell, to a distributed scheduler (DS). The DS allocates a partition for the application or places it in the FCFS system queue to wait for an available partition, as described in Section 2. Under dynamic partitioning, the nodes controlled by the DS are divided into multiple, dynamically-created and dynamically-changing non-overlapping partitions, whereas the node partitions are fixed under static partitioning. Each partition is comprised of a group of nodes, and each application runs in its own dedicated partition until it completes. The DS informs the application of node allocation changes at runtime. The application then reconfigures itself based on the new set of nodes available to it. The DS implements both DEP and SP policies, the performance characteristics of which are examined in this paper. We refer the interested reader to [14] for more details on the architecture of our distributed scheduler.

3.2 Parallel Application Structure

Many parallel applications are written such that the number of nodes allocated to them can only be set when they start. However, it is desirable for a parallel application (if it runs under a scheduler that supports dynamic partitioning) to be able to handle, at any time during its execution, fewer or more nodes than it was initially allocated. We refer to applications that are able to react to such changes as *reconfigurable*. There are a variety of ways to structure parallel applications to make them reconfigurable. We present one such approach.

We assume that each application can use all of the nodes allocated to it during its lifetime. We also assume that these applications can be decomposed into the structure depicted in Fig. 1. This structure has been variously called bag-of-tasks[1, 10], master-slave parallelism[25] and task-queue model [15, 5].

Each application consists of a coordinator process along with a set of worker processes as shown in Fig. 1. When an application starts it spawns a set of worker processes and the logically centralized coordinator. Each worker process is given a set of tasks to work on. When a worker is finished with its tasks it sends the results back to the coordinator, and waits for more tasks from it. The worker processes may also communicate with each other.

The coordinator is the point of contact between the system and the application. The DS notifies an application of node allocation changes via the coordinator. Under dynamic partitioning, each partition may shrink or grow with time, and a reconfigurable application must be able to handle such node allocation changes. For example, the scheduler may notify the coordinator of an application that it has lost a node. It does so by sending a reconfiguration message to the coordinator. The coordinator must then work with

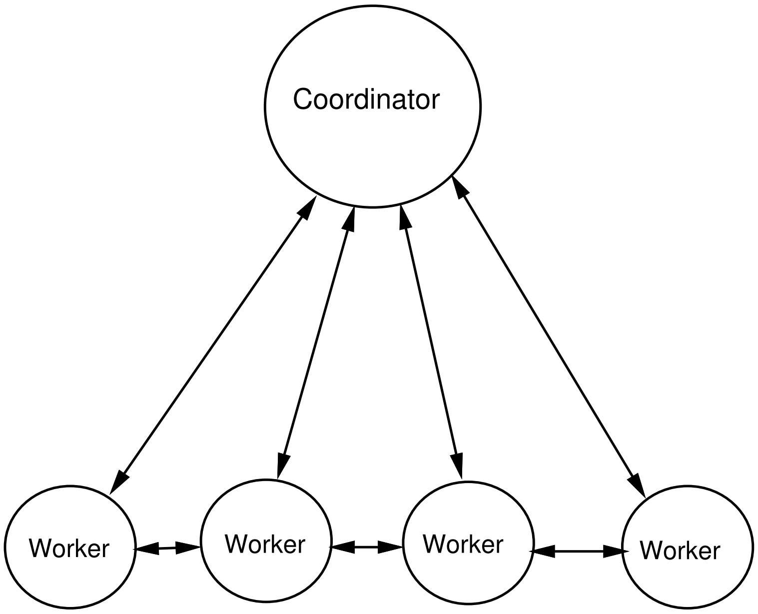

Fig. 1. Structure of an Adaptive Parallel Application

the worker processes to handle the lost node. This requires sending a reconfigure message to each of the worker processes. The worker processes checkpoint their data with the coordinator, and then wait for further instructions from it.[3] The worker that resides on the lost node is terminated. The other workers are sent more work and then they continue their normal processing. The applications we chose (see Section 3.2) are large-grained applications for which checkpointing at the coordinator is not a bottleneck.

We assume that reconfiguration does not involve process migration but only data migration. Under this scenario when an application loses a node, it must checkpoint its data and gracefully terminate the process running on that node. The system then starts the new application's processes on that node.

The parallel programming model we have chosen is a popular one, and it is easy to make adaptive. It is possible to structure a large class of applications in this manner, including Adaptive Quadrature [1] (a method for performing numerical integration), AtEarth [3] (a simulation of the flight of neutrinos from the sun towards the earth), DNA parallel sequence generation programs [3] and Computational Fluid Dynamics applications [26, 27]. Furthermore, many Linda programs are inherently structured in this manner [4, 3].

The Applications. We examine the performance of two applications in this paper: Adaptive Quadrature and AtEarth.

AtEarth. AtEarth simulates the flight of neutrinos from the sun toward the earth. The simulation consists of many trials where each trial simulates a neutrino's flight with given characteristics (e.g, energy and direction of flight). The trials are independent. The coordinator generates tasks, and the workers execute the trial simulations and then return the results back to the coordinator.

Adaptive quadrature. Adaptive Quadrature is an algorithm for numerical integration. It is an approximation algorithm where the area under the function to be calculated is approximated with a parallelogram. If the approximation is above a certain threshold the process is recursively refined. In the parallel version of the

[3] An alternative would be to throw away the current task set on the node which is being preempted. We will explore this alternative in our future work.

algorithm the region to be integrated is split into Z parts, where Z is the total number of regions in the problem. The running time is a function of the desired accuracy of the intergration, the interval over which the computation is being performed, and the function(s) to be computed.

Reconfiguration. When the scheduler changes the size of a partition, the corresponding application (i.e., its coordinator) is notified of its new partition size through a special reconfiguration message, which contains a list with the lost (if the partition is about to shrink) or new (if the application is about to expand) nodes.

The application worker processes checkpoint their work with the coordinator, which is now free to choose a totally new parallelization. The coordinator informs the scheduler that it has checkpointed. If a partition shrinks, the processes on the nodes being reassigned are gracefully terminated. If a partition expands, new worker processes are launched on the additional nodes. The coordinator must reset the communication links with all the worker processes, and set up the appropriate data structures so that it can communicate with the workers and the individual processes can communicate with each other.

3.3 Workloads

Current and expected workloads for large-scale parallel computing environments consist of a mixture of applications with very different resource requirements, often resulting in a highly variable workload [26, 27, 13, 18, 19]. We therefore use a simple probability distribution to control and vary different mixtures of instances of the two applications discussed in Section 3.2, where an instance is determined by both the application and its input data set. In other words, we model the system workload by probabilistically determining which instance we submit on each job arrival.

The results presented in Section 6 are based on three of the workloads considered in our study. These workloads were chosen because they are representative of the trends we have observed in our investigations of different parallel systems. These three instances can be characterized by their execution times: very small, small, medium and large. Table 2 summarizes the different probabilities used for these parallel workloads for each of the three workloads. As a specific example, we present in Fig. 2 the speedup curve of workload 2 in the NOW environment.

Job Size	Very Small	Small	Medium	Large
Workload 1	0.0	0.2339	0.274	0.491
Workload 2	0.0	0.66	0.34	0.0
Workload 3	1.0	0.0	0.0	0.0

Table 2. Three workloads used in the experiments.

4 Analytical System Models

In this section we summarize an analytic model of the distributed-memory, dynamic partitioning system described in Section 3, as well as an analytic model of the corresponding parallel system under static/adaptive partitioning. The technical details of our models and their solutions are beyond the scope of this paper. We refer the interested reader to [35] for derivations of an exact solution of each model, including expressions for performance measures of interest. Additional details on the models can be found in [35, 36].

4.1 Dynamic Partitioning

We model a parallel computer system consisting of P identical nodes that are scheduled according to the (basic) DEP policy defined in Section 2.1. Recall that the node allocations are reconfigured whenever a job

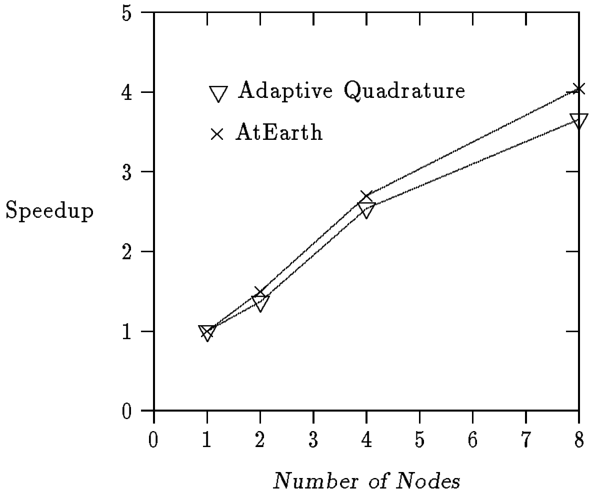

Fig. 2. Speedup of the Applications in Workload 2 on NOW-cluster.

arrives to a system with $0 \leq i < N$ jobs and whenever a job departs from a system with $1 \leq i \leq N$ jobs. The exact details of the node allocation decisions made by the scheduler in each case, as well as the overheads of making these decisions and of reconfiguring the applications involved, are reflected in the model parameter distributions and the analysis of the corresponding stochastic process [35]. In this manner we can model various types of dynamic partitioning strategies, although our focus in this paper is on DEP.

The interarrival times of jobs are modeled as independent and identically distributed (i.i.d.) random variables with a phase-type probability distribution $\mathcal{A}(\cdot)$ and mean interarrival time $1/\lambda$. When the system is executing i jobs, the service times of each of these jobs are modeled as i.i.d. random variables with a phase-type distribution $\mathcal{B}_i(\cdot)$ and mean execution time $1/\mu_i$, $1 \leq i \leq N$. The times required to repartition the nodes among the i jobs being executed (either due to a departure when the system contains $i+1$ jobs or an arrival when the system contains $i-1$ jobs) are i.i.d. random variables having a phase-type distribution $\mathcal{C}_i(\cdot)$ with mean repartitioning overhead $1/\gamma_i$, $1 \leq i \leq N$. The use of phase-type distributions [28] for our model parameters is motivated in part by their important mathematical properties, which can be exploited to obtain a tractable analytic model while capturing the fundamental aspects of dynamic partitioning. Just as important, however, is the fact that any real distribution can in principle be represented arbitrarily close by a phase-type distribution, and a number of algorithms have been developed for fitting phase-type distributions to empirical data [2, 12, 20]. As our measurements confirm, this results in an extremely accurate modeling analysis of dynamic partitioning in real parallel systems.

This DEP model is solved for relative system loads $\widehat{U}$ in the range 0.02 to 0.98 in increments of 0.02. The corresponding mean job response times, $\overline{T}_{\mathrm{DP}}(\widehat{U})$, are then computed from these model solutions.

4.2 Static (Adaptive) Partitioning

We also model the above parallel system under the SP policy defined in Section 2.2. Recall that the nodes are statically divided into K partitions each of size $S \equiv (N/K)\,M$, where we only consider values of K that evenly divide N.

This static system model is solved for each value of $K \in \{1, 2, \ldots, N/2, N\}$ and the corresponding mean response times, $\overline{T}_{\mathrm{SP(K)}}$, are computed. The mean job response time under the best SP policy, for a given relative load $\widehat{U}$, is then given by the minimum of these response times. That is,

$$\overline{T}_{\mathrm{SP}^*}(\widehat{U}) \;=\; \min_{1 \leq K \leq N} \{\, \overline{T}_{\mathrm{SP(K)}}(\widehat{U}) \,\}.$$

As previously noted, the value of $\overline{T}_{\mathrm{SP}^*}$ is representative of the mean response time under adaptive partitioning in many system environments.

5 Overheads Associated with Dynamic Equi-Partitioning

In this section we present some of the results from our experimental testbed on the overheads associated with implementing dynamic space-sharing strategies in the NOW and the SP2 environments described in Section 3. There are two types of overheads associated with the dynamic partitioning schemes: those that are experienced by the system and those that slow down the application. For the applications we have considered, reconfiguration overheads are independent of the size of the data sets. To ensure the accuracy of these measurements, each set of experiments were repeated 50 times and we present the average of these runs.

System Overhead. For each new application entering the system, the DS first makes sure that a new partition can be created as explained in Section 2.1. Once the DS determines that it can create a new partition for an application it goes through the following steps:

– Determine which of the nodes (we call them *moving nodes*) will be assigned to the new partition.
– Send a reconfiguration notification only to the applications (i.e., to the appropriate coordinators for each application) that may use the *moving nodes.* The reconfiguration message includes information on how many and which nodes are to be preempted.
– Wait until it receives from each coordinator an acknowledgement that reconfiguration has completed. At this point it issues a kill message which gracefully terminates all the application's processes that run on the *moving nodes.* (The processes do not exit after a checkpoint; this is discussed in more detail in the paragraph on application overhead). Now, all of these *moving nodes* are free and ready to be used.
– Update its data structures to reflect the changes (e.g., partition sizes, and which nodes belong to which partitions).
– Initialize the new data structures with the *moving nodes.*

At this point, the new partition is initialized and ready, so the DS launches the new application to run on it.

The DS follows similar steps upon the termination (normal or abnormal) of an application. The difference is that now the DS divides the available nodes among the remaining applications, instead of "squeezing" the applications to use fewer nodes. Here are the steps taken by the DS:

– Determine which of the remaining partitions (we call them expanding partitions) will be assigned the moving nodes.
– Send a reconfiguration notification to each application (i.e., to its coordinator) that runs on an expanding partition. The reconfiguration message includes information on how many and which nodes are added.
– Wait until it receives an acknowledgement from each coordinator that reconfiguration has completed.
– Update its data structures to reflect the changes (e.g., partition sizes, which nodes belong to which partitions).
– Deallocate the data structures associated with the partition that was eliminated.

At this point, for each expanding partition, the DS starts executing the application program on the newly available nodes.

Application Overhead. A component of the reconfiguration overhead actually occurs in the communication library linked with the application. The break down of these overheads are as follows:

– When an coordinator receives a reconfiguration notification (the message contains information on the new size and set of nodes) it sends a checkpoint message to all of its worker processes.
– Once the workers complete their current phase, they checkpoint and they send an acknowledgement to the coordinator that they are ready. Then they cut their old connection with the coordinator and they try to establish a new one and start from the beginning.
– Meanwhile, the coordinator waits to receive all of the acknowledgements. Once this happens, it sends an acknowledgement back to the DS that checkpointing has completed.

– Then it starts accepting connection requests but only from the workers of valid nodes (nodes that belong in the partition). A new node that has just joined the partition is obviously considered valid. If a worker of an invalid node (i.e., one that has been allocated to another partition) tries to reconnect, the coordinator refuses the connection.

When all connections are reestablished, the coordinator assigns work to its new set of workers and the application resumes.

5.1 Overheads for Dynamic Equi-Partitioning

We present both the overheads of the system and the total (system plus application) overheads associated with DEP. Partitions shrink in size when an application enters the system (if the current number of jobs is less than the maximum number of partitions) and expand when an application exits.

In the following we present the overheads of starting with a system with no application, and then measure the overheads as more and more application enter the system until the number of applications is equal to the maximum number of partitions. We call this the *shrinking phase* since the partitions keeps shrinking in size. We then reduce the number of applications one by one until there are none left, which we call the *expanding phase*.

The overheads of shrinking partitions as an increasing number of applications enter the system is given in Fig. 3 for eight nodes, and in Fig. 4 for four nodes. The transitions and the individual partition sizes for each of the points on the graph are shown in Table 1, which describes the overheads when $P = 8$ and $M = 1$. The overheads include those of the system and the application reconfiguration. The corresponding overheads for shrinking partitions in a 12-node NOW system are provided in Fig. 5.

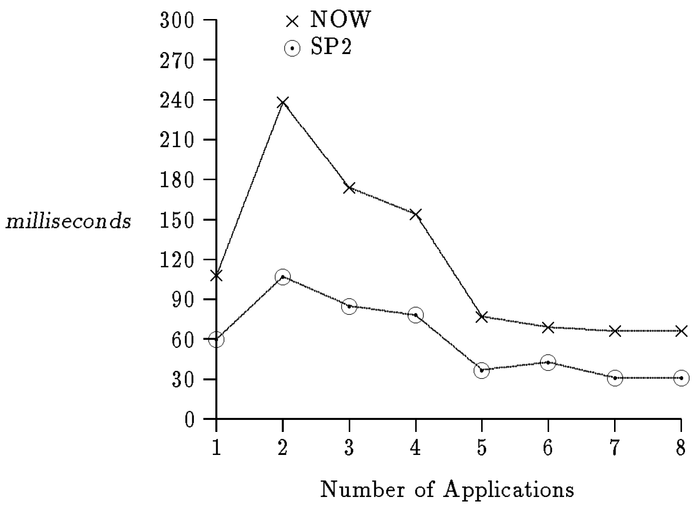

Fig. 3. Overheads associated with Shrinking a partition as number of applications is increased. The policy is DEP.

The overheads of expanding partitions starting with 8 applications, each executing on a partition of 1 node, is shown in Fig. 6. Similarly, the overheads of starting with 4 applications and 4 nodes is given in Fig. 7. The corresponding overheads for expanding partitions in a 12-node NOW system are also provided in Fig. 5.

From these figures we observe several trends. First, an expanding event is more expensive than a comparable shrinking event. For example, the transition $\{1,1\} \rightarrow \{2\}$ is more expensive than the transition $\{2\} \rightarrow \{1,1\}$. The system overheads are the same in both cases, however, in the $\{1,1\} \rightarrow \{2\}$ case the application overheads for the new partition of $\{2\}$ are greater than the new partitions $\{1,1\}$ in the transition $\{2\} \rightarrow \{1,1\}$. This is due to the fact that in the former case (expanding) the coordinator reconnects with two

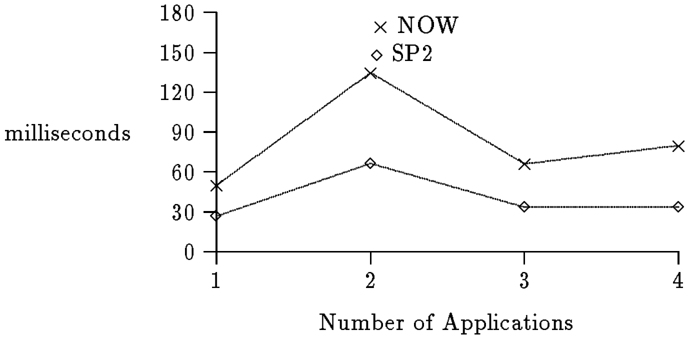

Fig. 4. Overheads associated with Shrinking a partition as number of applications is increased. The total number of nodes is 4.The policy is DEP.

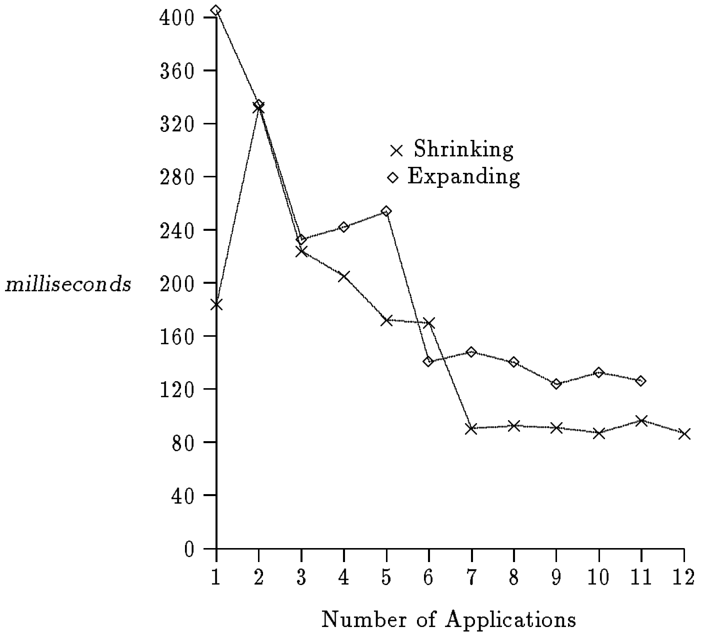

Fig. 5. Overheads associated with Shrinking and Expanding with 12 nodes. The policy is DEP.

workers, while in the later (shrinking) each worker reconnects with one (different) coordinator. This effect is more pronounced as the number of nodes is increased (there is more congestion in the network). Second, the SP2 overheads are lower due to the faster CPU and the fast SP2 interconnect, as expected. The total overheads on the SP2 are about 2 to 4 times lower than the comparable overheads on the NOW.[4]

The overheads are greater when few large applications (many nodes per application) are reconfigured. This is expected since creating a new partition while keeping the sizes balanced implicates applications, specially if they are large. When there are many jobs in the system, fewer applications are interrupted and the number of nodes per partition involved in a reconfiguration is small. As the number of applications approaches the maximum number allowed into the system, the reconfiguration cost becomes a fixed overhead as only one partition, of size 1 (the minimum partition size) in our example, is reconfigured.

[4] On the SP2, we believe many further optimizations are possible particularly from the perspective of communication.

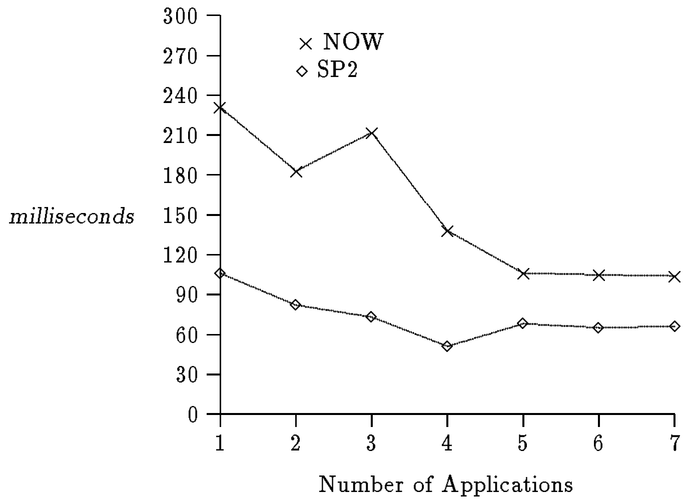

Fig. 6. Overheads associated with Expanding partitions as number of applications is decreased from 8 to 1.The policy is DEP.

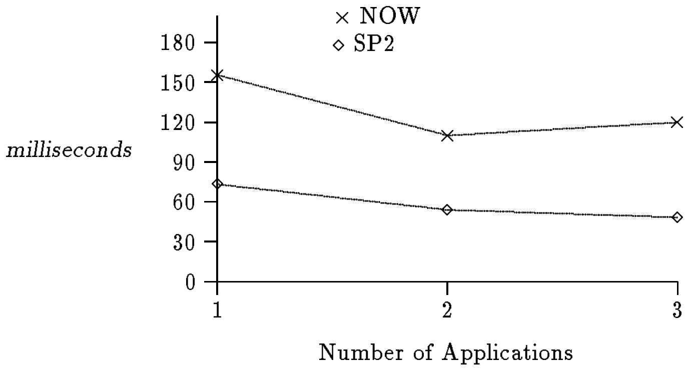

Fig. 7. Overheads associated with Expanding partitions as number of applications is decreased from 4 to 1.The policy is DEP.

6 System Performance

In this section we present some of the results of our detailed analysis of dynamic partitioning in the NOW and SP2 system environments. The measurement data presented in Sections 3 and 5 were used to parametrize the analytic models of Section 4. Several of the response time estimates predicted by our models were then compared against corresponding measurements from our experimental testbed. These results show that our analytic estimates are in excellent agreement with the performance measurements of dynamic partitioning in both parallel systems executing real scientific/engineering workloads, and thus validate our analytic models. We note that the model results are computed in an extremely efficient manner, requiring less than a few seconds to obtain each set of results presented below. This makes it possible to analyze various performance characteristics of dynamic partitioning across a large parallel system design space. Based on the insights gained from this analysis, we then use our experimental testbed to analyze additional performance characteristics of dynamic space-sharing strategies, including different allocation methods to reduce the impact of reconfiguration overheads and highly variable job arrivals. Our overall objective is to effectively combine our experimental and analytical approaches to quantitatively evaluate the benefits and limitations of dynamic partitioning in distinct distributed-memory environments.

The results in this section are for system workloads consisting of the application mixes described in Section 3.3, together with a probability distribution for the times between job arrivals to the system. While most previous parallel scheduling studies have assumed a Poisson arrival process (i.e., exponential interarrival times), recent measurements of real scientific and engineering workloads demonstrate that the job interarrival times in such high-performance computing environments tend to be significantly more variable [13, 18, 19]. We therefore consider hyperexponential interarrival times that statistically match the workload measurements presented in [13, 18, 19], and we compare these results with those obtained under the exponential interarrival assumptions of previous work.

The characteristics of the different scheduling policies, together with their corresponding overheads, cause each of the various parallel systems considered in our study to saturate (i.e., the response times become unbounded) at different job arrival rates. The best possible service rate, or *capacity*, for the DEP system under a particular workload is bounded above by the value of $N\mu_N$[5] for that workload, and saturation is guaranteed for all arrival rates $\lambda \geq N\mu_N$. Although this capacity $N\mu_N$ is not actually achievable due to the overheads incurred under DEP, we use this capacity value to define a relative measure of system utilization as the basis for all of our performance comparisons. The capacity values of the SP2 system are chosen for this purpose since this environment has lower service times and overheads (hence, higher capacities) than the NOW environment under each application workload considered. We therefore use *relative system load* to refer to the ratio $\widehat{U} \equiv \lambda / N\mu_{N,SP2}$. The results that follow for each system are all plotted as functions of the relative system load over the interval $(0, 1)$. We note that the corresponding curves for the NOW environment will span a smaller region of this $\widehat{U}$ interval due to its lower capacities.

Mean Response Times. Our first set of results considers the performance characteristics of DEP for each workload and system environment. In Fig. 8 we plot mean response times for application workload 1 (W1) in 8-node NOW and SP2 system environments under hyperexponential job interarrival times as a function of relative system load (labeled W1(Ahyp,Bcv)). For the purpose of comparison, Fig. 8 also includes response time results for W1 under Poisson arrival times (labeled W1(Aexp,Bcv)), as well as results for the case where the coefficient of variation[6] of the workload service times is doubled (labeled W1(Aexp,B2cv)). The corresponding curves for application workload 2 (W2) and workload 3 (W3) are presented in Figs. 9 and 10, respectively.

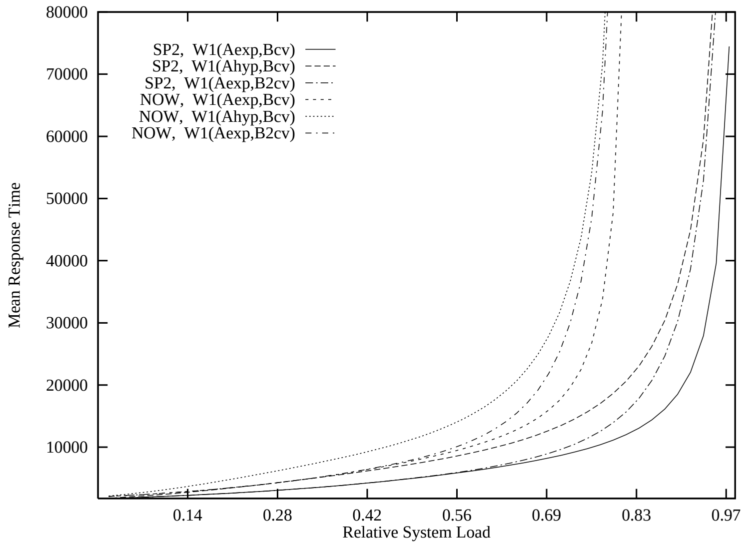

Fig. 8. Mean Response Times under DEP, for Workload 1 and $P = 8$

[5] $1/\mu_N$ is the mean service time, excluding overhead, of a generic job when the system contains at least N jobs.
[6] The coefficient of variation is the ratio of the standard deviation to the mean.

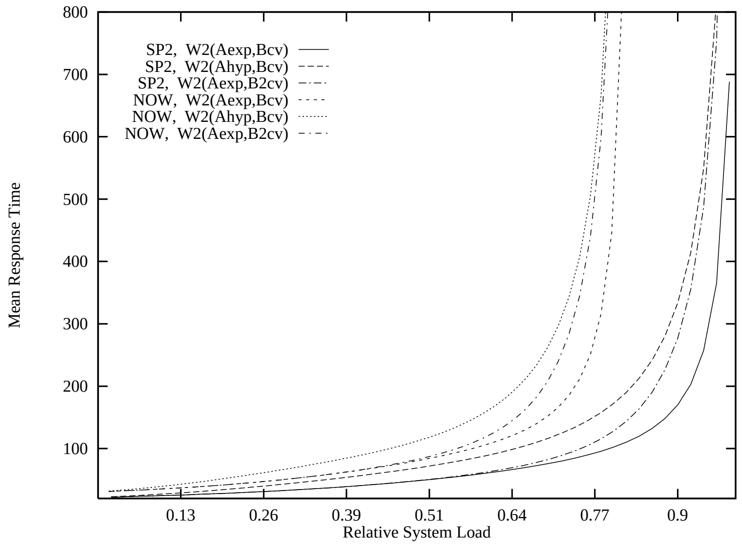

Fig. 9. Mean Response Times under DEP, for Workload 2 and $P = 8$

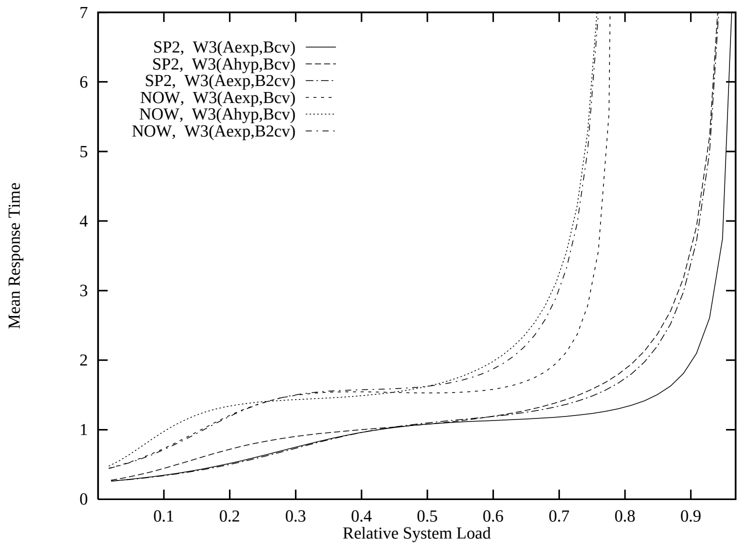

Fig. 10. Mean Response Times under DEP, for Workload 3 and $P = 8$

We observe that the response times under DEP are consistently worse in the NOW environment than those realized in the SP2 environment. This may be as expected due to the larger service times and reconfiguration overheads measured for the NOW environment (see Section 5). We also observe from these results that the more realistic hyperexponential interarrival times yield significantly higher response times under DEP than those obtained under Poisson arrivals in both system environments. This is due to the fact that, under the more variable hyperexponential distribution, a considerable amount of time is spent repartitioning applications only to be interrupted before completion by a subsequent arrival, thus causing a new repartitioning without making any progress on behalf of the jobs involved. This suggests that the mean response times under DEP in distributed-memory systems may be considerably larger than those predicted by previous studies, at least within the context of the systems and workloads considered in this paper. We further observe that the mean response times of DEP tend to also increase under more variable workload execution times, although to a considerably smaller degree than under more variable interarrival times at light to fairly heavy system loads.

Each of the above performance characteristics are similarly observed for the DEP system under W2 and W3, as illustrated in Figs. 9 and 10. We note, however, that there is a reduction in the relative performance differences among the curves going from W1 to W2, which is reduced even further going from W2 to W3. This can be explained in part by observing that a larger fraction of the workload is comprised of jobs with smaller execution times upon moving from W1 to W2 to W3.

Relative Response Times. Our next set of results quantifies the performance benefits of DEP with respect to static/adaptive partitioning. Taking the ratio of the mean response time under the best static policy to that achieved by the dynamic policy, we obtain mean response time ratios as a function of $\widehat{U}$. The corresponding results for W1, W2 and W3 under both parallel system environments are plotted in Figs. 11, 12 and 13, respectively. While the mean response time trends observed above were quite similar, here we see very different performance characteristics in comparison to static/adaptive partitioning for the various workloads.

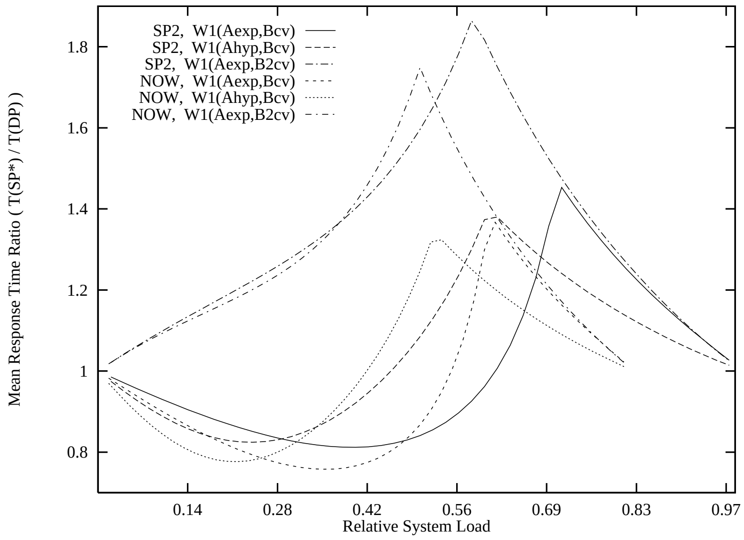

Fig. 11. Mean Response Time Ratios ($\overline{T}_{\mathrm{SP}^*}/\overline{T}_{\mathrm{DP}}$), for Workload 1 and $P = 8$

For the base W1 execution times, DEP provides poorer response times relative to those obtained under the static/adaptive policy at light to moderate loads, independent of the interarrival distribution. The larger performance degradations and the smallest performance improvements are observed for the NOW environment when compared against those for the SP2 system. The overheads of repartitioning the nodes and of the allocation decisions of the dynamic scheme tends to outweigh its benefits relative to the static/adaptive policy under W1 at these system utilizations. This is particularly true at light loads since the overheads for reconfiguration are greater at these utilizations (see Figs. 3 and 6). These reconfiguration overheads and the aggressive repartitioning decisions degrade the relative performance even though the long run percentage of reconfigurations is low (see Fig. 14). Moreover, larger relative performance degradations are observed for the system under Poisson arrivals than under the more variable hyperexponential interarrival times in both system environments. The relative performance degradations (and benefits) for the NOW environment appear at smaller system loads than those in the SP2 system, and within each of these environments the

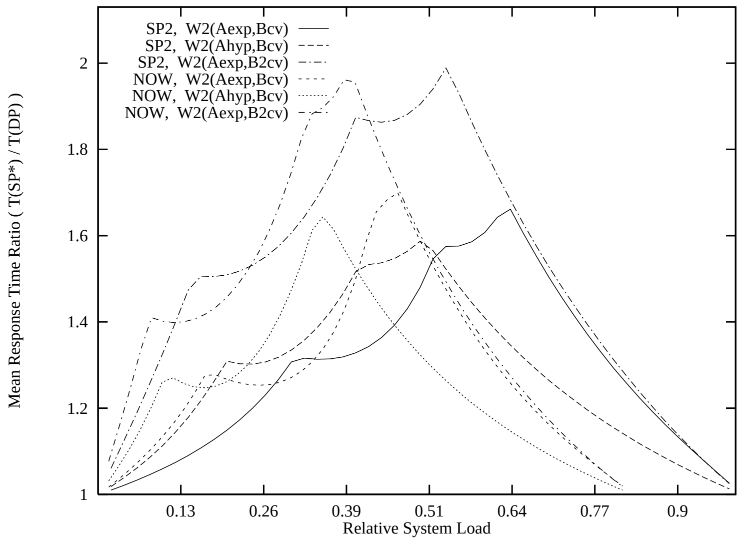

Fig. 12. Mean Response Time Ratios ($\overline{T}_{\mathrm{SP}^*}/\overline{T}_{\mathrm{DP}}$), for Workload 2 and $P = 8$

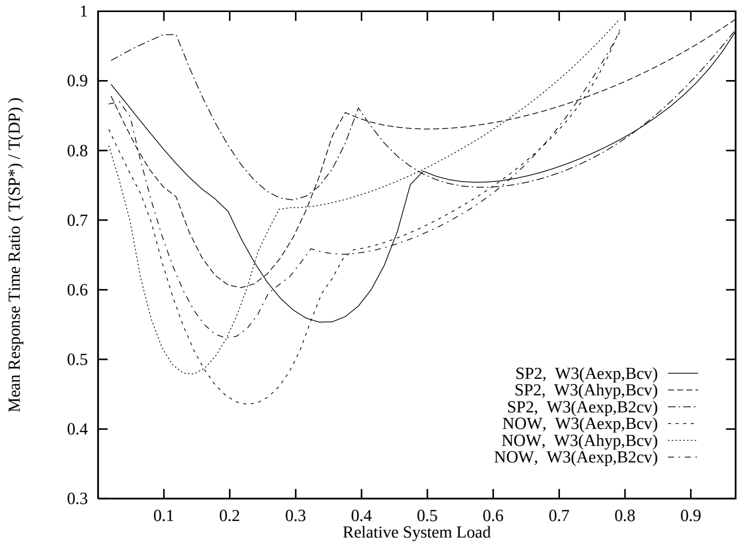

Fig. 13. Mean Response Time Ratios ($\overline{T}_{\mathrm{SP}^*}/\overline{T}_{\mathrm{DP}}$), for Workload 3 and $P = 8$

relative performance degradations (and benefits) for hyperexponential interarrivals appear at lighter loads than those under Poisson arrivals.

Interestingly, DEP provides the largest performance benefits (and no degradation) relative to static/adaptive partitioning under the more variable W1 execution times. By adjusting scheduling decisions in response to changes in the highly variable workload, the DEP policy provides more efficient utilization of the nodes when compared against the static/adaptive policy, which results in better response time ratios. These relative performance benefits tend to increase as $\widehat{U}$ rises from small to moderate values since workload changes are more frequent and DEP adjusts its node allocations accordingly to achieve superior steady-state performance.

As the system load increases, we observe considerable performance benefits under DEP relative to static/adaptive partitioning, with the largest response time ratios appearing in the SP2 system. We also note that the case of Poisson arrivals yields larger maximum (relative) performance benefits than the hyperexponential interarrival times in both system environments. Once again, by adjusting scheduling decisions in response to workload changes, the DEP policy provides a more efficient utilization of the nodes in comparison to static/adaptive partitioning for moderate to heavy loads under W1. These relative performance benefits tend to increase as $\widehat{U}$ rises, since workload changes are more frequent and DEP adjusts its node allocations accordingly to achieve superior steady-state performance. In the limit as the system approaches saturation, the probability that the system repartitions the nodes tends toward 0, i.e., the frequency of reconfigurations decreases to 0 as the system spends essentially all of its time with N or more jobs. It therefore follows that the DEP system converges toward the static policy with N partitions in the limit as the system approaches saturation.

Turning to W2, we observe that DEP provides the best space-sharing performance characteristics and that these relative benefits are even larger than those shown for W1. Here we see that the largest relative improvements in performance are generally obtained for the NOW environment. We again find that hyperexponential interarrival assumptions yield smaller maximum (relative) performance benefits than the corresponding Poisson arrival case. The response time ratios tend to increase as $\widehat{U}$ rises because scheduling decisions are being adjusted in response to workload changes, resulting in very efficient utilization of the nodes, and workload changes are more frequent with these increasing system loads. As noted above, the system under DEP eventually converges toward the static policy with N partitions in the limit as the system approaches saturation.

Conversely, the DEP policy under W3 yields significant performance degradations relative to static/adaptive partitioning across all system utilizations. We further observe that the response time ratios for the case of Poisson arrivals are worse than those obtained under hyperexponential interarrival times at all but light loads in both parallel system environments. These results are primarily due to the large repartitioning overheads relative to the job service times comprising the workload, where the ratio of execution time to overhead is roughly 6 to 1. It is for exactly these reasons that an adaptive partitioning strategy should be used for jobs with relatively small processing demands [26, 27]. Such information can be successfully given by users [18, 19] provided that countermeasures are taken by the system [8], it can be estimated with performance tools and run-time systems [11], and/or determined via standard methods such as multi-level feedback queues.

We should point out that the scalloped shape of the response time ratio curves for both workloads are the result of the response time behavior of the best static partitioning policy. Specifically, each of the points where the response time ratio reaches a local maxima (within a particular $\widehat{U}$ region) is due to a change in the number of partitions employed under the static/adaptive policy. This in turn causes the response time under DEP to be compared with a different static partitioning response time curve, which is further from saturation than the response time curve for the previous system load.

Reconfigurations. To better understand the system performance impact of the overheads of repartitioning, our next set of results considers the long run proportion of time that the system spends reconfiguring its node allocations, i.e., the steady-state probability p_r that the system is executing a reconfiguration. The corresponding results for W1, W2 and W3 are plotted as a function of $\widehat{U}$ in Figs. 14, 15 and 16, respectively. We observe a sharp initial increase in p_r as the relative load rises, and that this initial increase corresponds to the performance degradation under DEP for W1 and W3. Similarly, we observe that the shape of the p_r curves and the loads over which these characteristics are found, both correspond to the performance benefits exhibited in the response time ratio curves; e.g., compare Figs. 11 and 12 with Figs. 14 and 15. The NOW environment spends a larger percentage of its time repartitioning nodes than the corresponding SP2 system due to the larger reconfiguration overheads experienced in this environment (see Section 5). Moreover, the maximum values of p_r appear at smaller $\widehat{U}$ in the NOW system than in the SP2 environment. A Poisson arrival process tends to increase p_r over that observed under hyperexponential interarrivals, and this trend appears in both system environments.

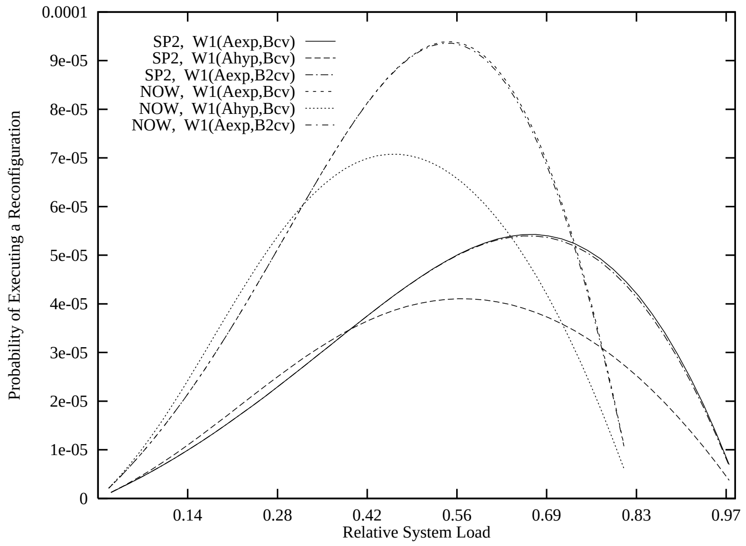

Fig. 14. Probability of Repartitioning the Nodes in Steady State, for Workload 1 and $P = 8$

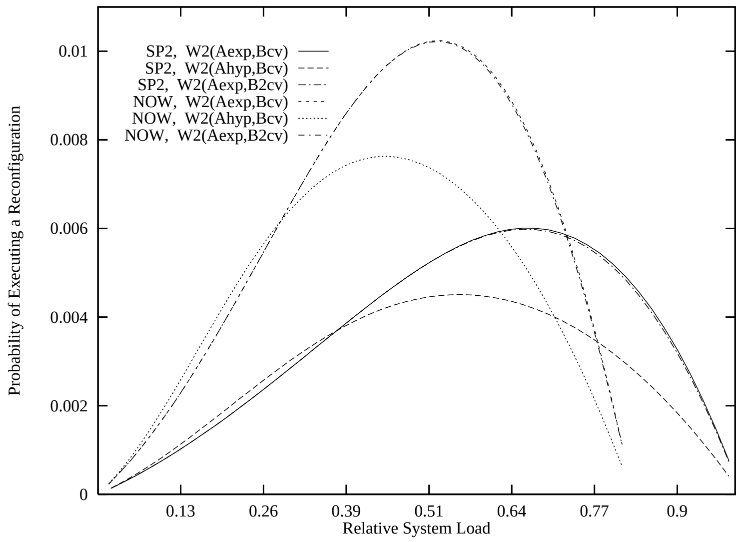

Fig. 15. Probability of Repartitioning the Nodes in Steady State, for Workload 2 and $P = 8$

7 Conclusions and Future Work

In this paper we examined the benefits and limitations of dynamic partitioning with respect to other space-sharing strategies in different parallel system environments. We developed and used an experimental testbed in computing environments based on networks of workstations and on the IBM SP2 distributed-memory computer. We also used an analytic model of dynamic partitioning, which was fitted to measurement data obtained from our experimental testbed running various parallel applications. The computational efficiency of this model allowed us to explore the large parallel system design space.

Our results show that the performance benefits of dynamic partitioning are heavily dependent upon its associated costs, the system load and the workload characteristics. When the reconfiguration overhead is small relative to the processing requirements, the performance benefits of dynamic partitioning can be quite

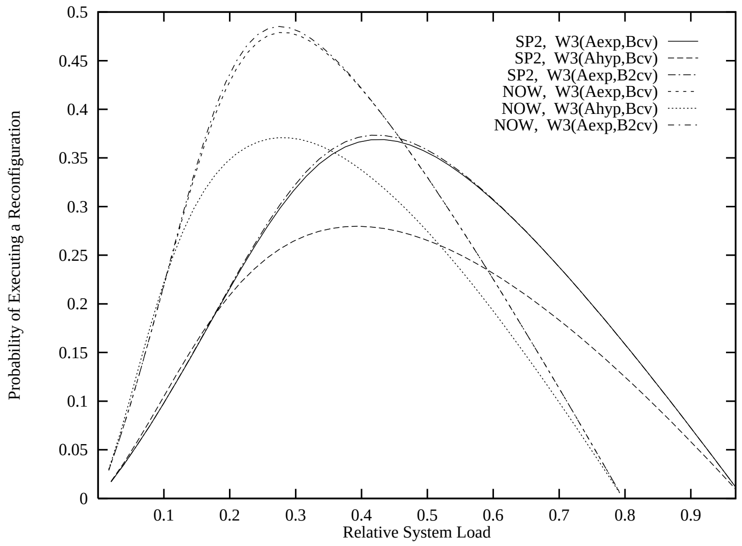

Fig. 16. Probability of Repartitioning the Nodes in Steady State, for Workload 3 and $P = 8$

significant for most of the workloads considered. In these cases, the dynamic partitioning policy provides the most efficient utilization of the nodes among the various space-sharing strategies by adjusting scheduling decisions in response to workload changes. These performance benefits tend to increase with rising traffic intensities, since workload changes are more frequent and dynamic partitioning adjusts its node allocations accordingly to achieve the best steady-state, space-sharing performance. When the reconfiguration costs are sufficiently large, however, this overhead tends to outweigh the benefits of dynamic partitioning, particularly at light to moderate system loads for the workloads studied.

Within the context of the parallel systems and application workloads considered in this paper, our results suggest that:

- Jobs with small resource requirements should not be dynamically reconfigured in distributed-memory environments. A workload consisting of a majority of such small jobs can suffer from a form of thrashing where the system spends a large percentage of its time reconfiguring node allocations.
- On the other hand, the measurements from our experimental testbed show that, in the majority of application instances expected for large-scale parallel computing, the repartitioning overheads tend to be small relative to the execution times of these application instances.
- Dynamic partitioning appears to be viable in a variety of different distributed-memory environments, provided that the applications are capable of executing on variable numbers of nodes and are capable of reconfiguring the number of nodes on which it executes. Our system approach provides the structure to extend this functionality to applications beyond those considered herein.

Furthermore, our results clearly demonstrate that the overheads of dynamic equi-partitioning in distributed-memory environments must be considered by the scheduling algorithms employed in practice, otherwise these reconfiguration costs can in general limit and/or eliminate the potential system performance benefits. We have been exploring several variants of dynamic partitioning to address these issues. One strategy for decreasing the overheads associated with dynamic equi-partitioning is to use the folding approach found in [24], which reduces the number reconfigurations performed under the greedy dynamic policy, at the expense of a less equitable allocation of the nodes among the competing jobs. Another approach consists of using the equi-partitioning method to equally divide the nodes among the jobs in the system whenever a repartition is performed, while placing a minimum period of time (which can be dynamically adjusted) between when the system can repartition its node allocations. In addition to reducing the number of reconfigurations, this approach tends to reduce the performance effects of job arrival variability by effectively smoothing the arrival process [6].

There is a fundamental tradeoff between these two dynamic partitioning approaches. Folding provides the advantage of immediately responding to workload changes, but it reduces the repartitioning overhead by interrupting fewer jobs to yield somewhat less equitable node allocations. Dynamic partitioning with smoothing, on the other hand, reduces the repartitioning overhead by reacting less quickly to workload changes, but it provides the advantage of dividing the system resources equally among the running jobs. The best solution to this performance tradeoff depends upon a number of factors, and it is particularly sensitive to the application workload characteristics and the job arrival process. For long running applications, it may be just as important or even more important to equitably allocate the nodes among the running jobs as it is to reduces the number reconfigurations performed.

Several preliminary experiments with these policies under workload 1 of Section 3.3 have consistently demonstrated that dynamic partitioning with smoothing exhibits lower mean response times, as well as a smaller variance in the execution times, than that observed for dynamic equi-partitioning and folding, with equi-partitioning consistently providing better response times than folding. The latter result is in contrast to those of Padhye and Dowdy [29] which show that folding generally outperforms equi-partitioning in a distributed-memory environment under a workload based on scientific matrix computation programs. The differences between the results of these two studies are primarily due to the differences in the respective workloads, where the workload used in our experiments consists of applications with larger execution times than those studied in [29]. This further highlights the fundamental tradeoff between the two above dynamic partitioning approaches for reducing the repartitioning overheads in distributed-memory environments. We are continuing to examine these and related scheduling issues in distributed-memory parallel systems.

References

1. G. R. Andrews. Paradigms for process interaction in distributed programs. *ACM Computing Surveys*, 23(1):49–90, Mar. 1991.
2. S. Asmussen, O. Nerman, and M. Olsson. Fitting phase type distributions via the EM algorithm. Technical Report 1994:23, Department of Mathematics, Chalmers University of Technology, May 1994.
3. Carriero, Freeman, Gelernter, and Kaminsky. Adaptive Parallelism in Piranha. *IEEE Computer*, 28(1):40–49, Jan. 1995.
4. N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(4):444–458, Apr. 1989.
5. R. Chandra, A. Gupta, and J. Henessey. COOL: A Language for parallel programming. In *Proceedings of the Second Workshop on Programming Languages, and Compilers for Parallel Computing*, Aug. 1989.
6. C. S. Chang. Smoothing point processes as a means to increase throughput. Technical Report RC 16866, IBM Research Division, May 1991.
7. S.-H. Chiang, R. K. Mansharamani, and M. K. Vernon. Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 33–44, May 1994.
8. E. G. Coffman, Jr. and L. Kleinrock. Computer scheduling methods and their countermeasures. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 32, pages 11–21, April 1968.
9. K. Dussa, B. Carlson, L. Dowdy, and K.-H. Park. Dynamic partitioning in transputer environments. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 203–213, 1990.
10. J. D. L. Eager and J. Zahorjan. Chores:Enhanced Run-Time support for shared-memory parallel computing. *ACM Transactions on Computer Systems*, 11(1):1–32, Feb. 1993.
11. K. Ekanadham, V. K. Naik, and M. S. Squillante. PET: A parallel performance estimation tool. In *Proceedings Seventh SIAM Conference on Parallel Processing for Scientific Computing*, February 1995.
12. M. J. Faddy. Fitting structured phase-type distributions. Technical report, Department of Mathematics, University of Queensland, Australia, April 1994. To appear, *Applied Stochastic Models and Data Analysis.*
13. D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In *Job Scheduling Strategies for Parallel Processing,* D. G. Feitelson and L. Rudolph (eds.). Springer-Verlag, 1995. Lecture Notes in Computer Science Vol. 949.
14. L. L. Fong, A. S. Gopal, N. Islam, A. Prodromidis, and M. S. Squillante. Extensible resource management for cluster computing. Technical report, IBM Research Division, May 1996.
15. R. Gabriel. Queue based multiprocessing lisp. In *ACM Symposium on Lips and Functional Programming*, pages 25–43, 1984.
16. D. Ghosal, G. Serazzi, and S. K. Tripathi. The processor working set and its use in scheduling multiprocessor systems. *IEEE Transactions on Software Engineering*, 17:443–453, May 1991.
17. A. Gupta, A. Tucker, and S. Urushibara. The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1991.
18. S. G. Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. In *Proceedings of the 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, April 1996.
19. S. G. Hotovy, D. J. Schneider, and T. O'Donnell. Analysis of the early workload on the Cornell Theory Center IBM SP2. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1996. Poster.
20. A. Lang. Parameter estimation for phase-type distributions, part I: Fundamentals and existing methods. Technical Report 159, Department of Statistics, Oregon State University, 1994.
21. S. T. Leutenegger and M. K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 226–236, May 1990.
22. R. K. Mansharamani and M. K. Vernon. Properties of the EQS parallel processor allocation policy. Technical Report 1192, Computer Sciences Department, University of Wisconsin–Madison, November 1993.
23. C. McCann, R. Vaswani, and J. Zahorjan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146–178, May 1993.
24. C. McCann and J. Zahorjan. Processor allocation policies for message-passing parallel computers. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 19–32, May 1994.

25. J. Mohan. *Performance of Parallel Programs: Models and Analyses.* PhD thesis, Carnegie Mellon University, July 1984.
26. V. K. Naik, S. K. Setia, and M. S. Squillante. Performance analysis of job scheduling policies in parallel supercomputing environments. In *Proceedings of Supercomputing '93*, pages 824–833, November 1993.
27. V. K. Naik, S. K. Setia, and M. S. Squillante. Scheduling of large scientific applications on distributed memory multiprocessor systems. In *Proceedings Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 913–922, March 1993.
28. M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach.* The Johns Hopkins University Press, 1981.
29. J. D. Padhye and L. W. Dowdy. Preemptive versus non-preemptive processor allocation policies for message passing parallel computers: An empirical comparison. In *Proceedings of the 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, April 1996.
30. E. Rosti, E. Smirni, L. W. Dowdy, G. Serazzi, and B. M. Carlson. Robust partitioning policies of multiprocessor systems. *Performance Evaluation*, 19:141–165, 1994.
31. S. K. Setia. *Scheduling on Multiprogrammed, Distributed Memory Parallel Computers.* PhD thesis, Department of Computer Science, University of Maryland, College Park, MD, August 1993.
32. S. K. Setia and S. K. Tripathi. A comparative analysis of static processor partitioning policies for parallel computers. In *Proceedings of MASCOTS '93*, January 1993.
33. K. C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 171–180, May 1989.
34. K. C. Sevcik. Application scheduling and processor allocation in multiprogrammed parallel processing systems. *Performance Evaluation*, 19:107–140, 1994.
35. M. S. Squillante. Analysis of dynamic partitioning in parallel systems. Technical Report RC 19950, IBM Research Division, February 1995.
36. M. S. Squillante. On the benefits and limitations of dynamic partitioning in parallel computer systems. In *Job Scheduling Strategies for Parallel Processing,* D. G. Feitelson and L. Rudolph (eds.). Springer-Verlag, 1995. Lecture Notes in Computer Science Vol. 949.
37. A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 159–166, December 1989.
38. J. Zahorjan and C. McCann. Processor scheduling in shared memory multiprocessors. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 214–225, May 1990.