Application-Assisted Dynamic Scheduling on Large-Scale Multi-Computer Systems

Ravi B. Konuru José E. Moreira Vijay K. Naik
{ravik,moreira,vkn}@watson.ibm.com

IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598-0218

Abstract. On multi-user large-scale multi-computers, application workload is highly variable and typically unpredictable. In this paper, we present and analyze the performance of three scheduling policies for such systems. Two of these are static scheduling policies that assign resources at job startup time, but make no subsequent changes in allocated resources. The third policy allocates resources to jobs dynamically taking into account resource requirements of all jobs in the system. We compare the performance of these three policies using the resource reconfiguration infrastructure provided by the Distributed Resource Management System (DRMS). On a variety of workloads we tested, our results indicate that, among the three policies, the reconfigurable policy provided the lowest response time for any given utilization.

1 Introduction

Maximizing throughput while maintaining fairness in resource allocation to jobs is an important resource management issue in high-performance computing systems. In the past, on many large-scale parallel systems, jobs were scheduled on disjoint sets of *statically* carved out processor partitions. Such static policies result in poor system utilization and throughput. Current generation large-scale parallel systems have incorporated scheduling policies that allocate resources at job initiation. A drawback of these policies is that once resources are committed to a job, they cannot be reallocated until the job terminates. Hence, the resources cannot be adjusted quickly to changes in demands, which in turn affects the performance adversely. Dynamic policies, that can reshuffle allocated resources among new and running jobs, can adjust more quickly to such changes and, thus, have the potential to deliver better system performance. To be viable, execution of such dynamic policies should have low overheads and also the cost of developing reconfigurable applications should not be high. Keeping this in mind, we have implemented Distributed Resource Management System (DRMS) that provides an environment for developing reconfigurable applications and provides the necessary infrastructure to reconfigure such applications at run-time.

In this paper, we present and analyze the performance of three scheduling policies: two that allocate resources when starting up a job, but make no subsequent changes to these allocations, and a third that can dynamically reallocate resources among new and executing jobs. We then describe the DRMS architecture and some details on the

infrastructure for application reconfiguration. Dynamic scheduling policies, such as the one described in this paper, can make use of the DRMS infrastructure for dynamic resource management. Our results show that, for a variety of workloads, the performance of the system, in terms of job response time, is superior with the dynamic scheduling policy. The rest of this paper is organized as follows. Section 2 presents the applications used in our study. In section 3, the three policies implemented within the system are described. Section 4 presents relevant aspects of the DRMS infrastructure. Section 5, provides details regarding our performance experiments and analysis of the results. Related research is discussed in section 6 and we finally conclude in section 7. The full version of this paper is available as [6].

2 Applications

In this study, we used a job mix of 21 different computational fluid dynamics (CFD) applications. These 21 applications were obtained by varying the problem size and number of iterations of the three NAS Parallel Benchmarks [2] pseudo applications: appbt, applu, and appsp. Table 1 lists our 21 applications, identified by the notation *application*<*size, iterations*>. The applications have been grouped into three categories (I,II,III) qualitatively representing small, medium, and large jobs. In Table 1, under the job category, it is indicated the range of processor partition sizes allowed for the execution of an application of that category. Also listed in that table are the execution times, in seconds, when each application is run on its minimum and maximum allowed processor partition. The memory-requirement and execution-time ranges represented by these 21 applications is typical of supercomputing centers where both application development (small data sets and fewer iterations) and production runs (larger data sets and many iterations) are carried out simultaneously.

Table 1. Applications used in the experiments.

Job category (PE range)	Application <size,iters>	Time on min PEs (s)	Time on max PEs (s)
I (1,2,4)	appsp<64,200>	677	184
	appsp<64,400>	1271	344
	appsp<80,100>	827	203
	appsp<80,200>	1419	361
II (4,8,12,16)	appsp<64,5000>	4034	1327
	appsp<80,2500>	3995	1221
	appsp<102,1250>	3906	1176
	appbt<64,1600>	3554	1045
	appbt<80,800>	3435	979
	applu<64,4000>	7135	2169
	applu<80,2000>	3227	1036
III (8,12,16,20, 24,28,32)	appsp<64,40000>	18332	6324
	appsp<80,20000>	18244	5348
	appsp<102,10000>	17248	5195
	appsp<126,5000>	16639	4746
	appbt<64,12000>	13811	4132
	appbt<80,6000>	13399	3961
	appbt<102,3000>	13990	3702
	applu<64,32000>	29800	9538
	applu<80,16000>	13790	5237
	applu<102,8000>	16990	5082

All of our applications go through three phases during the course of their execution: (i) *setup* phase, (ii) *solution* phase, and (iii) *summary* phase. The solution phase, which represents the bulk of the execution time, consists of several iterations, each performing the same number of operations. For the experiments reported in this paper, we started with SPMD versions of the applications that were tuned for the IBM SP2 (see [11] and references there in). We modified these by inserting DRMS annotations (see Section 4)

at the beginning of each iteration so that the applications can be potentially reconfigured to execute on a different set of processors every fifth iteration.

3 Scheduling Policies

We consider three scheduling policies for this study: (i) lazy scheduling (LS), (ii) adaptive scheduling (AS), and (iii) reconfigurable scheduling (RS). In all cases, the scheduler maintains arriving jobs in a queue prioritized on the arrival time. Each job defines a list of processor partition sizes over which it can run. The policy allocates to a job one of its eligible partitions.

The LS policy: Under the LS policy, the scheduler continuously scans the job arrival queue and schedules the earliest job that can run on all or on a subset of the available processors, giving it as many processors as it can take. LS is a *first to fit* policy, with an allocation preference towards maximum eligible number of processors. Once a job is scheduled to run on a set of processors, that job runs till completion on those processors. When the load is light, LS policy tends to schedule jobs in the order in which they arrive (lower average queueing delays) and the number of processors allocated to a job is towards the maximum requested by that job (smaller service time). When the load is heavy, jobs requesting larger number of processors get lower priority (higher queueing delays) and the number of processors allocated to a job tends to be towards its minimum requested number of processors (longer service time).

The AS policy: Under the AS policy, whenever processors are available to schedule jobs, the scheduler tries to schedule jobs in the order in which they arrived, but instead of scheduling the earliest job on the maximum possible number of processors, it tries to schedule as many of the currently waiting jobs in the arrival queue as possible. AS is a *maximum to fit with priority* policy. Once a job is scheduled to run on a set of processors, that job runs till completion on those processors. When load is light, this policy behaves similar to LS, as each job is allocated closer to its maximum number of processors. When load is heavy, the number of processors allocated to each job tends to be closer to its minimum. This helps in reducing "fragmentation" of the processor space at moderate to heavy loads and thus improves overall processor utilization.

The RS policy: Under the RS policy, when there are processors available, jobs are scheduled in the same way as in the case of the AS policy. In addition, when not enough free processors are available and there are jobs waiting to run, it tries to free up processors from jobs that are currently running on more than their minimum number of processors. Similarly, when there are no jobs waiting to be scheduled and free processors are available, RS tries to expand one or more of the running jobs to run on a larger set of processors. The policy incorporates parameters to determine the minimum time interval between same job reconfigurations. The RS policy adapts processor partition sizes of new and existing jobs to dynamic changes in the load. This policy performs the best when jobs can be reconfigured as frequently as necessary and to any desired number of processors. This flexibility is achieved at the cost of reconfiguring jobs that are already

running. The lower the reconfiguration costs and the more frequently and at a finer granularity a job can be reconfigured, the better is the performance of the RS policy.

4 DRMS Infrastructure for Reconfigurable Scheduling

Figure 1 shows the main functional components of DRMS and the primary interactions among these components. Resource scheduling is performed by the Resource Coordinator (*RC*) and the Job Scheduler and Analyzer (*JSA*). The run-time management and coordination of user applications is accomplished by the User Interface Coordinator (*UIC*), *RC*, and the Task Coordinator and Run-time Monitor (*TC*). A run-time system (*RTS*), linked to each application, coordinates with the external environment and propagates resource requests from the program to the resource management components of DRMS. It also implements the data distribution operations to support reconfigurable applications. The user submits jobs and can interact with the system throughout the course of the program execution via the *UIC*. For a more comprehensive description of DRMS we refer the reader to [6]. In the rest of this section we discuss in some detail the implementation of the job reconfiguration mechanism using the DRMS infrastructure.

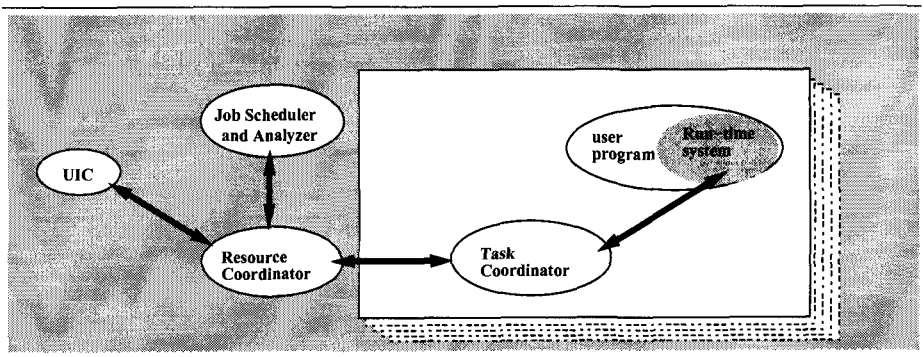

Fig. 1. DRMS Architecture

The system-level allocation and scheduling decisions are made by *JSA* based on the enforced scheduling policies. These decisions may use information such as application supplied resource requests, job priorities, and current and expected workload. Policies for making such decisions can be supplied and modified by system administrators. The *JSA* communicates its decisions to *RC*, which interfaces with the applications.

Each user application has an associated *TC*, which consists of multiple agents, one per processor on which the user application is scheduled for execution. A job interacts with the resource management system via its designated *TC*. These interactions occur only at user-specified points in the application execution called Schedulable and Observable Points (SOPs)[3]. However, the *TC* coordinates external interactions throughout the course of job execution. When the *JSA* decides to reconfigure a job during the course of its execution, this decision is asynchronously delivered to the *RTS* via *RC* and *TC*. At the next SOP, the *RTS* interprets the reconfiguration message, causes

synchronization among the tasks of the application, and communicates with the *JSA*. After the initialization of the application's new processor partition, the *RTS* is informed of the new partition information. Then the *RTS*, in conjunction with the *TC*, rearranges the application data so that the program can run on the new set of processors. Thus, the reconfiguration cost as seen by the application is the time elapsed between the initiation of a resource request at an SOP to the time at which the application resumes execution on the new set of allocated processors. Note that an application can also voluntarily initiate a reconfiguration of its allocated set of processors. For this paper, we consider only *JSA*-directed application reconfiguration.

For implementing reconfiguring policies such as `RS`, and also to allow applications to modify their resources dynamically, applications must be *reconfigurable*, that is, they must be able to accommodate changes in their processor partition during execution. DRMS provides a set of *language extensions* and *library functions* for SPMD Fortran programs that facilitate the task of designing reconfigurable applications. The extensions are in the form of annotations (source-level comments) that are translated into executable code by the DRMS compiler. These extensions implement the DRMS programming model, a modified SPMD model. In the classical SPMD model, a group of processors all execute the same code, with each processor applying the code to a different section of the data set. In the DRMS model, a parallel program execution consists of the consecutive execution of a number of SPMD *stages*, each on a (potentially) independent set of processors.

We define a stage as a unit with its own resource requirements, data distribution, and execution code. In our model, a stage consists of four sections: *resource*, *data*, *control*, and *computation* sections. The *resource section* specifies the type and quantity of each resource necessary for the execution of a stage. The *data section* specifies how the global data structures should be distributed among the processors executing the stage. The *control section* specifies values for variables that control the execution of code inside a stage. Usually, the flow of this execution depends on resource allocation and data partitioning. The *computation section* specifies the computations to be performed in each processor, as well as the communication that occurs between processors.

5 Performance Analysis

The objective of this study is to evaluate the performance of the three scheduling policies as implemented in the context of DRMS. We measured several performance parameters of DRMS and used them in a DRMS system-level simulator, to evaluate the three policies. In this section, we first present the results for performance parameters of DRMS, followed by the steady-state performance characteristics of the three scheduling policies, as obtained from simulation.

Performance results from DRMS: Performance parameters were measured from the DRMS system running on a 32-node partition of the IBM SP2 [1] at NASA Ames Research Center. We measured both the total reconfiguration time, as seen by each application, and one of its components: the redistribution time. The redistribution time is the time it takes to rearrange application data on the new set of processors. This

time is dependent on the problem size. The other components of reconfiguration, which include acquiring and releasing processors, partition reconfiguration, and restarting the application, depend not on the problem size but on the number of processors involved.

Table 2. Measured reconfiguration and redistribute costs for applications under DRMS.

Application	Reconfiguration time		Redistribute time	
	min (s)	max (s)	min (s)	max (s)
appsp<64>	4.51	7.17	1.15	1.94
appsp<80>	5.35	5.35	1.69	1.69
appsp<102>	6.42	20.40	3.52	6.66
appbt<64>	4.07	12.30	1.31	2.74
appbt<80>	6.07	7.21	2.37	4.90
appbt<102>	7.80	9.31	4.08	5.94
applu<102>	5.35	11.60	1.42	3.44

Table 2 summarizes the reconfiguration and redistribute times measured for the applications considered. Each application was reconfigured several times between partitions ranging in size from 4 to 32 processors. The table shows the maximum and minimum reconfiguration and redistribution times observed for each application. For purpose of simulation, we adopted times dependent only on the problem size and not the particular application or partition sizes involved. For problem sizes of 64, 80, 102, and 126 the reconfiguration times adopted were 8s, 11s, 17s, and 29s, respectively. The redistribution times were 3s, 6s, 12s, and 24s, respectively.

The DRMS System Simulator: Applications are modeled in the simulator at a very high level. Each application type shown in Table 1 is characterized by its setup time, its summary time, its iteration time, its number of iterations, and its reconfiguration and redistribute times. The setup, summary, and iteration times for each processor partition sizes were obtained through direct measurement. The central point of the simulator is the job scheduler, which mimics the actions of the *JSA* in DRMS. The simulator is event-driven, with 6 different types of events that trigger the job scheduler. At each of these six events, the job scheduler is invoked in order to schedule jobs for execution and decide which jobs need to be reconfigured.

Job arrival times are generated using an exponential distribution, with different mean inter-arrival times for each job type. The mean inter-arrival time for a job type is computed in order to deliver the desired system utilization. Measured performance parameters are the response times and service times for each job type and for the job mix as a whole. We measure both mean (μ) and standard deviation (σ) of n samples. To reduce the correlation between successive samples, we use the batch means method as described in [5]. After the system achieves steady state, the simulation runs until the standard error ($\delta = \sigma/\sqrt{n}$) of the response time is less than $\xi\mu$ for each job type and also for the whole job mix. In our experiments, we use $\xi = 0.01$. The simulator was validated by comparing its steady-state performance results to analytical results for $M/M/m$ queues [5], and also by comparing to some observed results in the real DRMS system.

Performance comparison of the three policies: We present simulation results comparing the average response times under the three scheduling policies. Let E[$R_{(p)}$] denote mean response time under policy p, $p \in \{\mathtt{AS}, \mathtt{LS}, \mathtt{RS}\}$, when all jobs are considered together. Similarly, let E[$R_{(p)}$, k] denote the mean response time under policy p, for jobs belonging to category k, $k \in \{\mathrm{I}, \mathrm{II}, \mathrm{III}\}$. For the ease of comparison, we use the ratios E[$R_{(\mathtt{AS})}$]/E[$R_{(\mathtt{RS})}$] and E[$R_{(\mathtt{LS})}$]/E[$R_{(\mathtt{RS})}$]. We define utilization as the job arrival rate times the ideal service time of the average job when executed on all processors. The ideal service time of a job when executed on all processors is determined as the execution time on a single processor divided by the total number of processors. The workloads we use for our experiments are defined by the fraction of the utilization produced by jobs from categories I, II, and III. For our experiments, we used six workloads. We varied the fraction of utilization from category I from 5% to 55% in steps of 10%, we kept the fraction from category II fixed at 25%, and we varied the fraction from category III accordingly, from 70% to 20%. In our notation, a workload of 05:25:70 means that 5%, 25%, and 70% of the utilization are from category I, II, and III jobs, respectively. Within a category, applications arrive with the same rate. We did not allow jobs from category I to be reconfigured.

Shown in Figures 2(a) and 2(b) are the response time ratios E[$R_{(\mathtt{LS})}$]/E[$R_{(\mathtt{RS})}$] and E[$R_{(\mathtt{AS})}$]/E[$R_{(\mathtt{RS})}$], respectively, as function of utilization and workload. The response time with RS policy is always better than with the other two policies, and AS performs better than LS. For a given workload, the performance of RS first gets better with increasing utilization and then the advantage diminishes at high utilization. Similarly, as the proportion of small jobs in the workload is increased, the advantage of RS policy initially increases and then it decreases.

We plot the average response times for the three policies in Figures 2(c)–(f), for all jobs combined and for each category individually. For these experiments, we have used a workload of 25:25:50. We observe that jobs from all categories benefit with RS policy, but category I jobs benefit the most and category III jobs the least. Under RS policy, any excess over a minimum number of processors allocated to medium and large jobs (i.e., category II and III jobs) can be taken away to schedule new jobs. This benefits the small jobs the most since they can be accommodated more often, thus reducing their queueing delays. AS policy performs significantly better than the LS policy for category I jobs. For the other two categories, with a workload of 25:25:50, performance of AS is not significantly different from LS. Thus, much of the gains obtained with RS policy are because of its ability to reconfigure running jobs.

We conducted a series of four additional experiments to determine the sensitivity of the RS policy to various parameters. For brevity, we only report here some general observations. First, we repeated our simulations under an ideal condition in which reconfiguration takes zero time. We found that the average response time, for all jobs combined, decreases by at most 10% as compared to the results reported in Figure 2. In the second additional experiment, we allowed jobs of category I to reconfigure and we found no significant difference in results. In the third additional experiment, we restricted the range of valid partition sizes for each category. Valid sizes for category I were { 1, 4 }, for category II { 4, 8, 16 }, and for category III { 16, 32 } (compare to Table 1). We found a significant negative impact on the average response time, especially

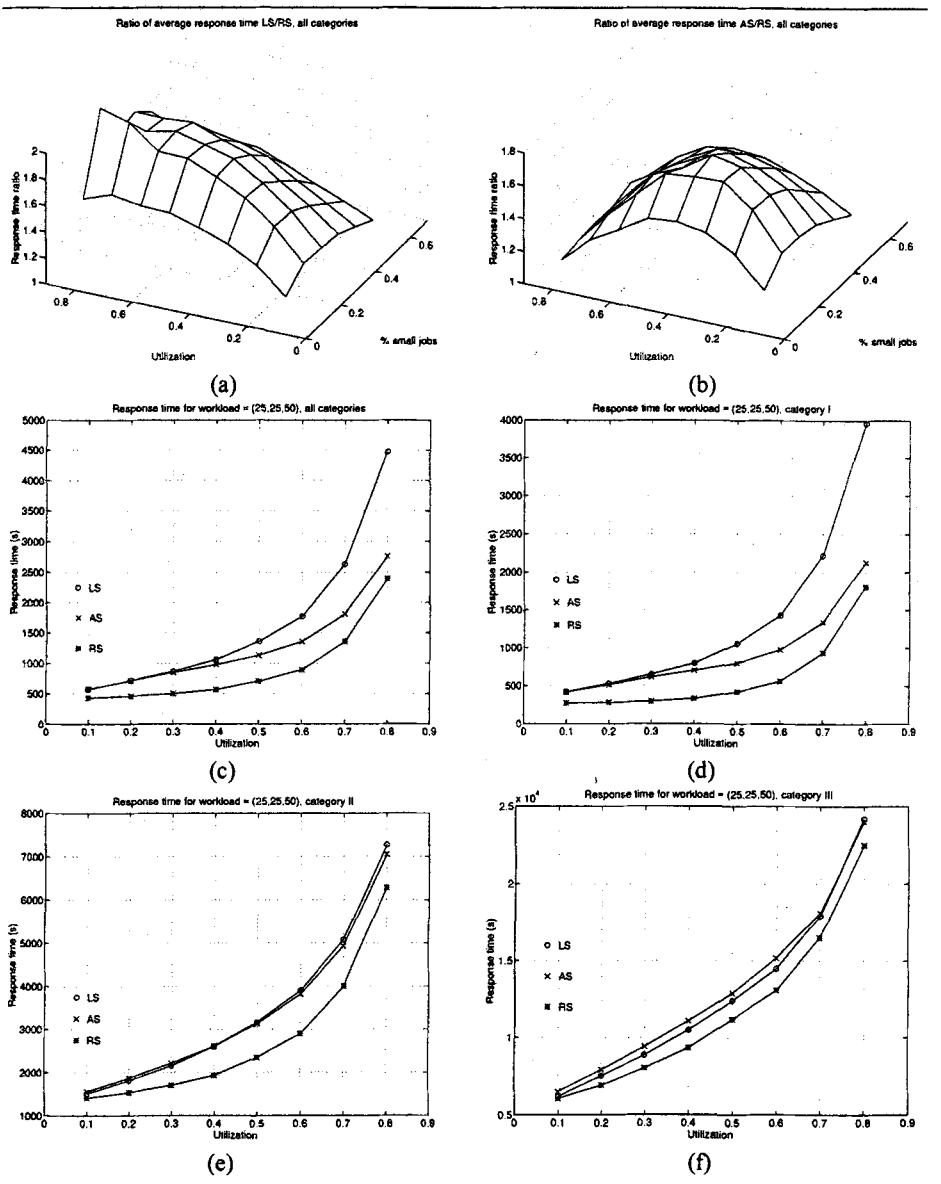

Fig. 2. Average response times with AS, LS, and RS policies.

at higher utilization. This impact can be explained by the diminished reconfiguration flexibility of the applications. Finally, we performed an experiment where we varied the minimum interval between reconfigurations for each application. A smaller reconfiguration interval implies quicker response to load changes, but also implies potentially more reconfiguration events and thus overhead. We varied the minimum interval between reconfigurations from 2 to 20 times the reconfiguration time of the application, with little impact on the average response time.

6 Related Work

Many studies have shown that dynamic partitioning policies can alleviate the problem of adapting to workload changes, at the expense of additional reconfiguration overhead [13, 15, 7, 4, 9, 12, 10, 14]. In particular, [15, 7, 4, 9] analyze the benefits of dynamic partitioning on uniform-access, shared-memory systems and show that dynamic reconfiguration policies outperform all other space-sharing policies. In the realm of private-memory (message-passing) systems, [12] have also demonstrated that dynamic reconfiguration policies outperform the other policies. A discussion of the effects of different processor scheduling policies, and reconfiguration overhead, in dynamically reconfigurable systems can be found in [10, 8].

Our work differs from the previously mentioned research in that we have implemented a working environment for a commercial message-passing system (IBM SP2) that supports dynamic reconfiguration of processor partitions. We provide the language extensions and run-time services that allow the user to easily port their existing SPMD applications to execute on reconfigurable partitions. We also provide all the resource control and scheduling mechanism to coordinate the execution of these jobs.

As an alternative to simulation, some authors have developed analytical performance models of dynamically partitioned multiprocessors [8, 10, 14]. The available models, however, were not appropriate for evaluating the DRMS system at the level of detail we wanted.

7 Conclusions

We have implemented the Distributed Resource Management System (DRMS) for the IBM SP2 as a framework for application-assisted dynamic scheduling on multicomputer systems. The DRMS resource management and scheduling subsystem is capable of taking full advantage of reconfigurable applications by dynamically changing their processor partitions as the system conditions change. We have measured the performance of DRMS using reconfigurable parallel CFD applications, determining the reconfiguration overhead to be in the range of 5–20s.

We have used the performance parameters of DRMS and our application suite to perform a simulation study of reconfigurable *vs.* non-reconfigurable scheduling policies under typical scientific an engineering workloads. The results from our study indicate that the reconfigurable scheduling policy outperforms the other policies as long as the job mix is such that there are sufficient job reconfiguration opportunities. However, not all jobs need to be reconfigurable. Thus, the RS policy is ideally suited for multiprocessor based supercomputing centers where the job mix has large variations with significant fraction of the cycles being consumed by large and long running jobs. We have shown that jobs from all categories benefit from a reconfigurable scheduling policy, even when only a fraction of all jobs are reconfigurable. In several instances of our experiments, the average response time across all categories was reduced by a factor of two.

Acknowledgements: This work is partially supported by NASA under the HPCCPT-1 Cooperative Research Agreement No. NCC2-9000.

References

1. Agerwala, T., Martin, J. L., Mirza, J. H., Sadler, D. C., Dias, D. M., and Snir, M. SP2 system architecture. *IBM Systems Journal*, 34(2):152–184, 1995.
2. Bailey, D., Barszcz, E., Dagum, L., and Simon, H. NAS parallel benchmark results. *IEEE Parallel & Distributed Technology*, 1:43–51, 1993.
3. Ekanadham, K., Moreira, J. E., and Naik, V. K. Application oriented resource management on large scale parallel systems. In Dharma P. Agrawal, editor, *Proceedings of the 1995 ICPP Workshop on Challenges for Parallel Processing*, pages 56–63, August 14 1995.
4. Gupta, A., Tucker, A., and Urushibara, S. 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.
5. Jain, R. *The Art of Computer Systems Performance Evaluation*. John Wiley & Sons, New York, 1991.
6. Konuru, R. B., Moreira, J. E., and Naik, V. K. Application-assisted dynamic scheduling on large-scale multi-computer systems. Technical Report RC 20390, IBM Research Division, February 1996. Available at `http://www.watson.ibm.com:8080/main-cgi-bin/search_paper.pl/entry_ids=7991` .
7. Leutenneger, S. T and Vernon, M. K. 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.
8. Madhukar, M., Leuze, M., and Dowdy, L. Petri net model of a dynamically partitioned multiprocessors system. In *Proceedings of the Sixth International Workshop on Petri Nets and Performance Models*, pages 73–82, October 3-6 1995.
9. McCann, C., Vaswami, R., and Zahorjan, J. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146–178, May 1993.
10. McCann, C. and Zahorjan, J. 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.
11. Naik, V. K. A scalable implementation of NAS parallel benchmark BT on distributed memory systems. *IBM Systems Journal*, 34(2):273–291, 1995.
12. Naik, V. K., Setia, S. K., and Squillante, M. S. Processor allocation in multiprogrammed, distributed-memory parallel computer systems. Technical Report RC 20239, IBM Research Division, October 1995. Submitted to Journal of Parallel and Distributed Computing.
13. Park, K.-H. and Dowdy, L. W. Dynamic partitioning of multiprocessor systems. *International Journal of Parallel Programming*, 18(2):91–120, 1989.
14. Squillante, M. S. *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, chapter On the benefits and limitations of dynamic partitioning in parallel computer systems, pages 219–238. Springer-Verlag, 1995.
15. Tucker, A. and Gupta, A. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 159–166, December 1989.