Effective Selection of Partition Sizes for Moldable Scheduling of Parallel Jobs*

Srividya Srinivasan, Vijay Subramani, Rajkumar Kettimuthu, Praveen Holenarsipur, and P. Sadayappan

Ohio State University, Columbus, OH, USA
{srinivas,subraman,kettimut,holenars,saday}@cis.ohio-state.edu

Abstract. Although the current practice in parallel job scheduling requires jobs to specify a particular number of requested processors, most parallel jobs are moldable, i.e. the required number of processors is flexible. This paper addresses the issue of effective selection of processor partition size for moldable jobs. The proposed scheduling strategy is shown to provide significant benefits over a rigid scheduling model and is also considerably better than a previously proposed approach to moldable job scheduling.

1 Introduction

The issue of effective scheduling of parallel jobs on space-shared parallel systems has been the subject of several recent research studies [4], [12], [14], [21]. Most of the research to date on this topic has focused on the scheduling of rigid jobs, i.e. jobs for which the number of required processors is fixed. This matches the practice at all supercomputer centers to our knowledge: users specify a specific single value for the number of processors required by a job. However, most parallel applications are moldable [10], i.e. they can be executed on different numbers of processors. If the machine were empty, the fastest turnaround time for a particular job would be obtained by specifying as large a number of processors as possible. But on a machine with heavy load, specifying fewer processors may actually provide a faster turnaround time than specifying the maximum number of processors. Although a run with more processors is likely to take less time than on fewer processors, the waiting time in the queue may be much longer. It would be desirable to have an intelligent scheduler determine the number of processors to allocate to different jobs, without forcing the user to specify a single specific value.

Several studies have considered job scheduling for the case of malleable jobs, where the number of processors for a job can be varied dynamically [1], [2], [3], [15], [16], [17]. In this paper, we consider the scheduling under a moldable job model. Recently Walfredo Cirne [5], [6], [7] evaluated the effectiveness of moldable job scheduling. Using synthetic job traces, he showed improvement in turnaround times of jobs under a moldable scheduling model, when compared

* Supported in part by a grant from Sandia National Laboratories.

to a standard conservative backfilling scheme [14], [22]. A greedy strategy was employed at job submission time for selecting the processor partition size for each job - among a set of possible choices identified for each job (using random variables characterizing the job's scalability), the one that gave the earliest estimated completion time was chosen.

Using a subset of a one-year job trace from the Cornell Theory Center [9], we evaluated the effectiveness of a greedy partition selection strategy for moldable job scheduling. Instead of restricting the range of processor choices for each job by use of statistical distributions (as done in Cirne's experiments), we allowed each job a range of choices from one processor to the total number of processors in the system. We found the performance improvement provided by moldable scheduling over standard non-moldable job scheduling to be considerably lower than that reported by Cirne. The results of the experiments highlighted the importance of careful selection of the processor partition size for each job. A greedy selection strategy over a wide range of choices for partition size was problematic - most jobs tended to choose very wide partitions, resulting in deterioration of performance for small jobs.

Using the insights from the initial experiments, we develop a more effective strategy for selection of partition size for moldable job scheduling. We show that the proposed scheme provides considerable overall improvement in job turnaround time of large jobs, without significantly impeding small jobs.

This paper is organized as follows. In Section 2, we provide some background information pertinent to this paper. Section 3 evaluates the previously proposed greedy submit-time moldable scheduling approach. A new approach to selection of processor partition size is presented and evaluated in Section 4. An enhancement to this approach is proposed and evaluated in Section 5 and we provide conclusions in Section 6.

2 Background and Workload Characterization

Scheduling of parallel jobs is usually viewed in terms of a 2D chart with time along one axis and the number of processors along the other axis. Each job can be thought of as a rectangle whose width is the user estimated run time and height is the number of processors requested. The simplest way to schedule jobs is to use the First-Come-First-Served (FCFS) policy. This approach suffers from low system utilization. Backfilling [13], [14], [22] was proposed to improve system utilization and has been implemented in several production schedulers [11]. Backfilling works by identifying "holes" in the 2D chart and moving forward smaller jobs that fit those holes. There are two common variants to backfilling - conservative and aggressive (EASY)[14], [18]. In conservative backfill, every job is given a reservation when it enters the system. A smaller job is moved forward in the queue as long as it does not delay any previously queued job. In aggressive backfilling, only the job at the head of the queue has a reservation. A small job is allowed to leap forward as long as it does not delay the job at the head of the queue.

A job is said to be moldable if it can run on multiple processor request sizes [10]. If the user requests a large number of processors, the execution time of a job may be lower, but it may have to wait for a long time in the queue before all needed processors are available. If the user requests a smaller number of processors, the wait time may be lower but the run time will be higher. There is a need to balance these two factors since the job turnaround time, which is of primary interest to the user is the sum of the job wait time and the job run time. Since the scheduler has the snapshot of the current processor allocation, if the task of deciding the job request size is left to the scheduler, the performance could potentially be better than if the decision is made by the user at submit time.

2.1 Workload Characterization

We use trace based simulation to evaluate the various schemes using the CTC workload log from Feitelson's archive [9]. Any analysis that is based on the aggregate turnaround time of the system as a whole does not provide insights in to the variability within different job categories. Therefore in our discussion we classify the jobs in to various categories based on their weight (i.e processor seconds needed) and analyze the average turnaround time for each category. Since job logs from supercomputer centers only include a single specific number for each job's processor requirement, an important issue in evaluating moldable scheduling approaches is that of job scalability. It is necessary to estimate each job's run-time for different possible processor partition sizes. The user estimated execution time and the actual execution time for the original processor request can be determined from the trace file. We use the Downey model [8] to generate the execution times for new processor request sizes. We assume that the ratio of the user estimated run time with the actual execution time remains the same as the processor request size changes. Cirne [5], [6], [7] used Downey's model of job scalability, with statistical distributions for model parameters.

3 Submit-time Greedy Selection of Partition Size

We first evaluate the approach to moldable scheduling that has been previously proposed [5]. In this scheme, every job is allowed a range of processor choices, from one processor to the total number of processors in the system. Among these partition sizes, the one that results in the best turnaround time for the job is chosen. The decision of which choice of processor count to allocate is made at the time of job submission - and hence once a choice is made, the job is made rigid and is no longer moldable(Submit-time moldability). Under this scheme, many of the jobs can be expected to choose very wide partition sizes, because of the local greedy nature of the strategy. Consider the arrival of jobs one after the other. For the first arriving job, the widest partition size will give the least turnaround time and hence it chooses a partition size that is equal to the total number of processors in the system. As a result, the next arriving job also chooses a partition size equal to the total number of processors in the system (as it results in the least turnaround time) and so on.

We first evaluated the submit-time greedy strategy under an assumption of perfect scalability of all jobs (i.e. $\sigma = 0$ for the Downey model). Table 1 shows the resulting distribution of partition sizes for the jobs in the system. We observe that a majority of the jobs tend to choose very wide partition sizes. Fig. 1 shows the percentage change in the average turnaround time for the greedy scheme with respect to the standard conservative backfilling scheme.The overall average turnaround time improves in comparison to conservative backfilling. This is because, under the greedy scheme, jobs choose wide partition sizes and hence execute one after the other as opposed to choosing narrow partition sizes and executing in parallel. But the average turnaround time for the small jobs deteriorates to a large extent because these jobs find very few "holes" in the schedule to backfill under the greedy scheme.

As σ is increased (i.e. jobs are less scalable), the performance of the scheme can be expected to deteriorate. This is because there is a resource usage penalty for using wide partition sizes - the total number of processor-seconds needed for a job increases as more processors are used. When job scalability is not perfect, a load-sensitive partition size selection strategy is called for. If there is a single job in the system, it might be appropriate for it to utilize all available processors. However, when there are several queued jobs, wide partition choices for each of the jobs is wasteful of resources; instead narrower choices would be more efficient. Fig. 2 shows performance data for $\sigma = 1$, and performance has indeed deteriorated. Performance deteriorates further as σ is increased, but we omit the data for space reasons.

Table 1. Distribution of jobs based on partition sizes

Partition Size	Non-Moldable	Greedy Moldable
1	2223	1169
2	394	174
3-4	655	212
5-8	494	55
9-16	617	0
17-32	327	0
33-64	172	51
65-128	72	36
129-256	34	70
>256	12	3233

4 Load-sensitive Selection of Partition Size

The greedy scheme is not preferable because most of the jobs choose wide partition sizes and as a result, the turnaround times of the small jobs deteriorates to

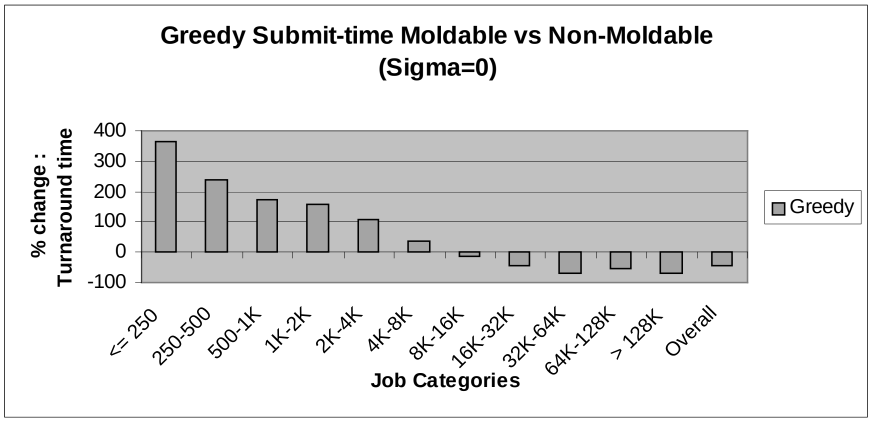

Fig. 1. Category-wise comparison of the performance of the greedy submit-time moldable scheme vs non-moldable scheduling with conservative backfilling. Although the greedy scheme improves the overall turnaround time, it degrades the average turnaround time of the small jobs to a large extent

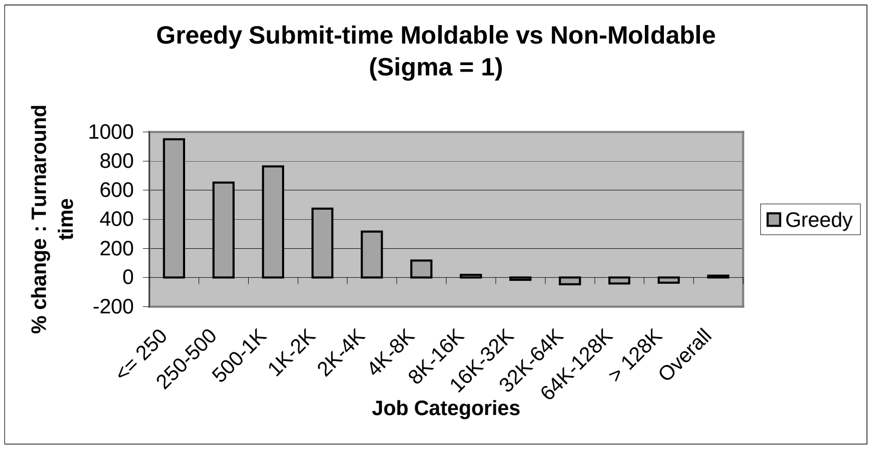

Fig. 2. Performance of the greedy scheme vs non-moldable conservative backfilling. As σ is increased the performance of the greedy scheme further deteriorates

a large extent. Thus no job should be allowed to use up all the processors in the system. Hence we impose a limit of 90% on the maximum number of processors that can be allocated to a single job. This means that no job will be allowed to expand to more than 90% of the total number of processors in the system. Using such a limit alone is not sufficient - although it would improve backfilling opportunities for the small jobs, it would be indiscriminate in allocating the available set of processors to jobs and hence could result in most of the jobs expanding to occupy 90% of the processors in the system. Instead, it is desirable that the available set of processors be allocated to the waiting jobs based on load considerations. For example, consider the case where there are four jobs with relative resource requirements of one, four, eight, and twelve. Clearly it would be beneficial to allocate more processors to the heaviest job (i.e. the one with the largest resource requirement) and fewer processors to the lightest job. However, it would not be desirable to allocate most or all processors to the heaviest job and have the others wait. On the other hand, when there is only a single heavy job in the system, we should allow it a wider partition limit. A suitable strategy might be to determine a "fair-share" processor-count for each job, based on the fractional resource requirement of this job compared to the total requirements of all pending jobs. Thus the maximum allowable partition size (called the fair share limit) should be different for different jobs and it should depend upon what fraction of the total weight of the jobs currently in the system (both idle and running) a job constitutes. The fair share limit for a job is defined as follows :

Fair Share Limit of a Job = (Total Processors in the System) * Weight of the Job/Sum of the Weights of all Jobs Currently in the System.

Since the above limit may be overly restrictive, a stretch factor of 2 was used. i.e the maximum allowable processor limit for a job was set to twice its fair share limit. Thus for each job, a range of choices for partition sizes, from one to twice its fair share limit is tested and the one which gives the best expected turnaround time is chosen.

Fig. 3 shows the percentage change in the average turnaround time for the greedy and the fair share scheme with respect to non-moldable conservative backfill scheduling. We observe that the fair share scheme results in a 50% decrease in the overall average turnaround time compared to conservative backfilling. Most of the job categories improve significantly, with only a slight deterioration for the small jobs. Comparing the greedy scheme with the fair share scheme, the fair share scheme improves the overall average turnaround time and the turnaround time for all the jobs except the very large jobs (whose weight is greater than 128,000 seconds). Fig. 4 shows performance data for σ=1. The relative improvements achieved by the proposed scheme compared to the greedy scheme increases with increasing σ.

5 Schedule-time Moldability and Aggressive Backfilling

In this section, we attempt to enhance performance by modifying the scheduling strategy in two ways:

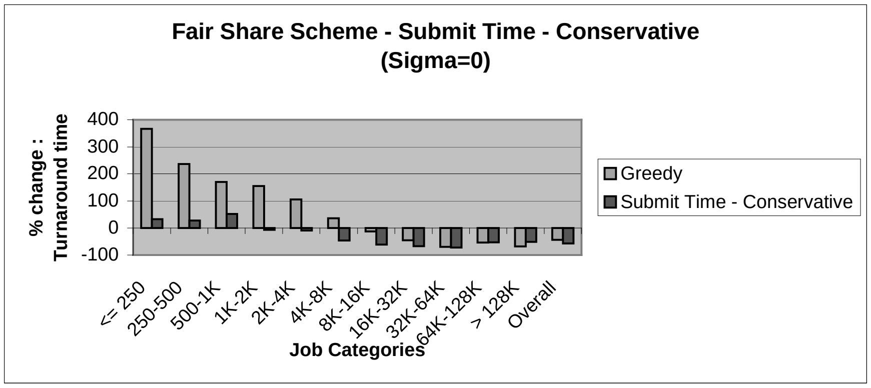

Fig. 3. Performance of the load-sensitive submit-time moldable conservative backfilling scheme. There is a 50% decrease in the overall average turnaround time compared to conservative backfilling. Compared to the greedy scheme the average turnaround time of all categories except the very large jobs improves

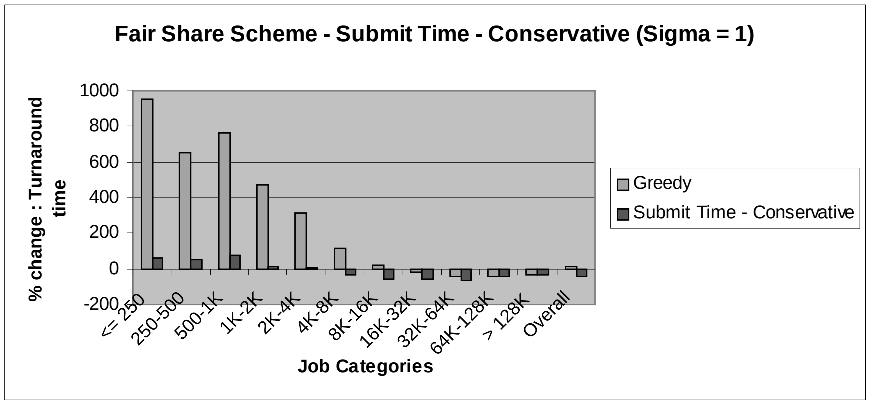

Fig. 4. Performance of the load-sensitive submit-time moldable conservative backfilling scheme. The relative improvement achieved by this scheme compared to the greedy scheme increases with increasing σ

– Instead of freezing the partition size choice for each job at job submission time, defer it till actual job start time (Schedule-time moldability).
– Instead of using conservative backfilling, use aggressive backfilling

There is a fundamental trade-off between conservative and aggressive backfilling. Conservative backfilling provides reservations to all jobs at submission time, while aggressive backfilling has only one reservation at any time. Thus the aggressive scheme has much more backfilling. However, job categories that have difficulty backfilling (such as very wide jobs) suffer from the lack of reservations. Overall, whether conservative or aggressive backfill is better, depends on the mix of the jobs since some some jobs (Short Wide) consistently do better with conservative backfill while others (Long Narrow) do better with aggressive backfill [19], [20].

However, when we consider a moldable job scheduling model, the disadvantage of aggressive backfilling disappears! This is because, there are no longer jobs that are forced to be "short and wide" - they can mold themselves to be "long and narrow" instead if that gets them quicker completion. This prompts the development of a schedule-time-moldable aggressive backfilling strategy, that we evaluate next.

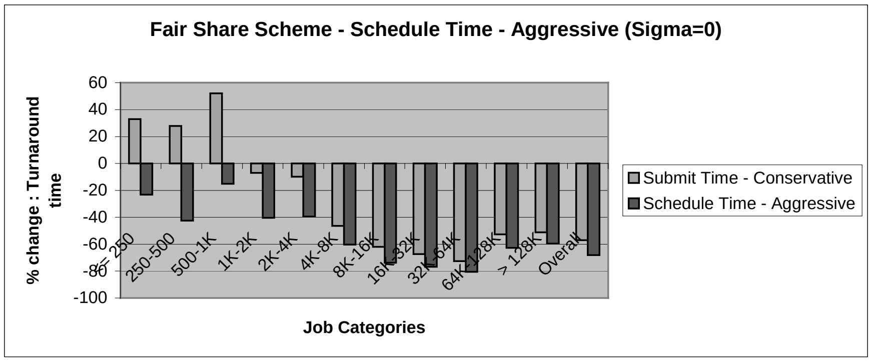

Fig. 5. Performance of the load-sensitive schedule-time-moldable aggressive backfilling scheme. This scheme clearly outperforms standard conservative backfilling and submit-time-moldable conservative backfilling

Fig. 5 shows the performance of the fair share scheme using aggressive backfill and schedule time moldability. We observe that this scheme clearly outperforms both the non-moldable conservative backfilling scheme and the fair-share scheme using conservative backfill and submit time moldability. The overall average turnaround time improves by almost 70% compared to conservative backfilling. Also, the turnaround times of all job categories improve under the fair share based schedule time moldable aggressive backfilling strategy compared to the other two schemes. Fig. 6 shows performance for σ=1. The performance of the moldable schemes deteriorates compared to σ=0 because the runtime for a job

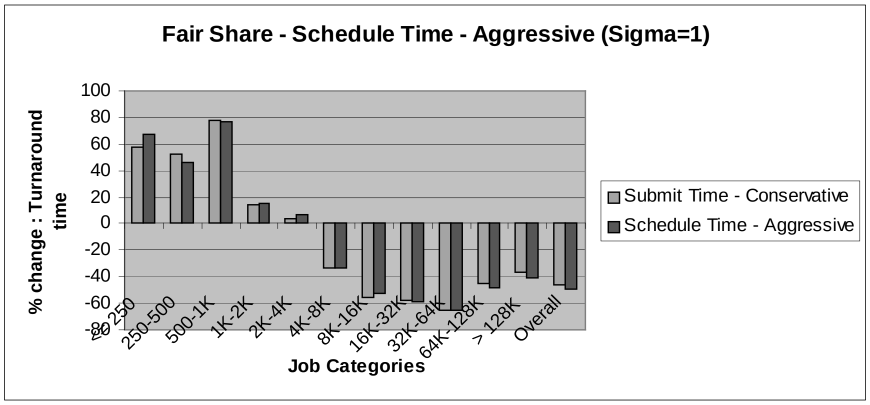

Fig. 6. Performance of the load-sensitive schedule-time-moldable aggressive backfilling scheme when jobs are less scalable

on a wider partition is higher with σ=1 than with σ=0. However the performance is still considerably better than the non-moldable case and the greedy scheme for moldable scheduling.

6 Conclusion

In this paper we addressed the issue of effective selection of processor partition size for moldable jobs. The proposed scheduling strategies were shown to provide significant benefits over a rigid scheduling model and were also considerably better than a previously proposed approach to moldable job scheduling.

References

1. S. V. Anastasiadis and K. C. Sevcik. Parallel Application Scheduling on Networks of Workstations. *Journal of Parallel and Distributed Computing*, 43(2):109–124, 1997.
2. O. Arndt, B. Freisleben, T. Kielmann, and F. Thilo. A Comparative Study of Online Scheduling Algorithms for Networks of Workstations. *Cluster Computing*, 3(2):95–112, 2000.
3. 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 *SIGMETRICS*, pages 33–44, 1994.
4. S. H. Chiang and M. K. Vernon. Production Job Scheduling for Parallel Shared Memory Systems. In *Proceedings of the International Parallel and Distributed Processing Symp*, 2001.
5. W. Cirne. Using Moldability to Improve the Performance of Supercomputer Jobs. Ph.D. Thesis. Computer Science and Engineering, University of California San Diego, 2001.
6. W. Cirne. When the Herd is Smart: The Emergent Behavior of SA. In *IEEE Trans. Par. Distr. Systems*, 2002.

7. W. Cirne and F. Berman. Adaptive Selection of Partition Size for Supercomputer Requests. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 187–208, 2000.
8. A. B. Downey. A Model For Speedup of Parallel Programs. Technical Report CSD-97-933. University of California at Berkeley, 1997.
9. D. G. Feitelson. Logs of real parallel workloads from production systems. http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.
10. D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and Practice in Parallel Job Scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing* , pages 1–34.
11. D. Jackson, Q. Snell, and M. J. Clement. Core Algorithms of the Maui Scheduler. In *Wkshp. on Job Sched. Strategies for Parallel Processing*, pages 87–102, 2001.
12. P. J. Keleher, D. Zotkin, and D. Perkovic. Attacking the Bottlenecks of Backfilling Schedulers. *Cluster Computing*, 3(4):245–254, 2000.
13. D. Lifka. The ANL/IBM SP Scheduling System. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 295–303, 1995.
14. A. W. Mu'alem and D. G. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. In *IEEE Trans. Par. Distr. Systems*, volume 12, pages 529–543, 2001.
15. E. Rosti, E. Smirni, L. W. Dowdy, G. Serazzi, and B. M. Carlson. Robust Partitioning Policies of Multiprocessor Systems. *Performance Evaluation*, 19(2-3):141–165, 1994.
16. S. Setia and S. Tripathi. A Comparative Analysis of Static Processor Partitioning Policies for Parallel Computers. In *Proc. of the Intl. Wkshp. on Modeling and Simulation of Computer and Telecomm. Syst. (MASCOTS)*, pages 283–286, 1993.
17. K. C. Sevcik. Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems. *Performance Evaluation*, 19(2-3):107–140, 1994.
18. J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY - LoadLeveler API Project. In *Wkshp. on Job Sched. Strategies for Parallel Processing*, pages 41–47, 1996.
19. S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Characterization of Backfilling Strategies for Parallel Job Scheduling. In *Proceedings of the ICPP-2002 Workshops*, pages 514–519, 2002.
20. S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Selective Reservation Strategies for Backfill Job Scheduling. In *Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, 2002.
21. A. Streit. On Job Scheduling for HPC-Clusters and the dynP Scheduler. In *Proc. Intl. Conf. High Perf. Comp.*, pages 58–67, 2001.
22. D. Talby and D. Feitelson. Supporting Priorities and Improving Utilization of the IBM SP Scheduler Using Slack-Based Backfilling. In *Proceedings of the 13th International Parallel Processing Symposium*, 1999.