Job Scheduling for Prime Time vs. Non-prime Time

Virginia Lo
University of Oregon
Eugene, OR 97403
lo@cs.uoregon.edu

Jens Mache
Lewis and Clark College
Portland, OR 97219
jmache@lclark.edu

Abstract

Current job scheduling systems for massively parallel machines and Beowulf-class compute clusters support batch scheduling involving two classes of queues: prime time vs. non-prime time. Jobs running in these queue classes must satisfy different criteria with respect to jobsize, runtime, or other resource needs. These constraints are designed to delay big jobs to non-prime time in order to provide better quality service during the prime time workday hours.

This paper surveys existing prime time/non-prime time scheduling policies and investigates the sensitivity of scheduling performance to changes in the jobsize and runtime limits allowed during prime time vs. non-prime time. Our simulation study, using real workload traces from the NASA NAS IBM SP/2 cluster, gives strong evidence for the use of specific prime time limits and sheds light on the performance trade-offs regarding response times, utilization, short term scheduling algorithm (FCFS vs. EASY backfilling), and success and overflow rates.

1. Introduction

Current job scheduling systems for massively parallel machines and Beowulf-class compute clusters, such as EASY/Load Leveller, LSF, PBS and Maui suppport batch scheduling with several distinct scheduling queues. A common arrangement involves two classes of queues: prime time vs. non-prime time. Jobs running in prime time must satisfy more stringent criteria with respect to jobsize, runtime, or other resource needs. These constraints are designed to delay big jobs to non-prime time in order to provide better quality service during the prime time workday hours. The basic premise, which is supported by both theoretical and empirical research, is that allowing big jobs to run at arbitrary times results in poor performance.

This paper investigates the sensitivity of scheduling performance to the choice of scheduling parameters for prime time versus non-prime time queues. We focus on two metrics used to distinguish between big jobs versus small jobs: jobsize and runtime.

We seek to answer the following questions: (1) Which is a more critical metric of the size of a job: jobsize or runtime? (2) What is the biggest job that should be allowed to run during prime time in order to maximize performance? (3) How does the best prime time schedule compare to that achieved by a single scheduling queue? (4) Which has greater impact on performance: the long term scheduling policy (prime time vs. non-prime time) or the short term scheduling policy (FCFS vs. EASY backfilling)?

Through simulations, we systematically study how changes in the jobsize and runtime limits associated with prime time and non-prime time, respectively, impact average slowdown, average job response time, waiting time, percentage of jobs completed at the end of each scheduling slot, percentage of jobs completed at the end of each 24 hour period, and system utilization. We look at each of these metrics for prime and non-prime time separately, as well as for entire day.

The study reported in this paper was conducted through simulations using real workload traces from the NASA NAS IBM SP/2 cluster. Additional simulations are currently being conducted using MPP traces and Linux cluster traces captured from PBS and Maui scheduling logs to investigate whether these trends hold across a wide variety of workloads.

2. Scheduling large jobs: background and related work

We focus on two resource needs of large jobs: (1) jobsize (number of processors) and (2) runtime (execution time).

Several empirical studies of workload traces for *parallel jobs* [4, 2, 9] captured from production supercomputer sites show a great deal of variety in the runtime and jobsize characteristics. Many of these traces exhibit the common

Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'02)
0-7695-1745-5/02 $17.00 © 2002 IEEE

characteristic that while the percentage of very large jobs is small, they consume a disproportionately large percentage of the total resources of the machine as measured by the time-space product.

Fair scheduling in the presence of resource hogs can be addressed through the per queue short term scheduling policy, as well as through prime/non-prime time scheduling policies. It is widely acknowledged that the FCFS scheduling policy yields poor performance due to the fact that small jobs can pile up behind a large job whose resource needs cannot be immediately satisfied. Backfilling algorithms schedule around larger jobs by allowing smaller jobs to jump ahead. The superior performance of backfilling has been clearly established [8, 3, 7]; thus, many commercial scheduling systems include a backfilling scheduler. Scheduling systems such as PBS, Maui, and LSF support multiple queues within the prime and non-prime time slots, allowing the systems administrator to tune the queue parameters to the specifics of the target machine and the site workload.

We note that it has not been established whether jobsize or runtime is the critical determinant in whether a resource hog can adversely impact overall performance. An earlier study we conducted [6] showed that certain size-related characteristics do significantly affect performance: proportion of power of two jobsizes, and degree of correlation between jobsize and runtime.

3. Survey of prime/non-prime time policies for real machines

The notion of prime time versus non-prime time scheduling is focused on the goal of providing fair and effective service to jobs with a wide range of resource needs. Basically, the large resource hogs are relegated to non-prime time to provide fast response times for smaller jobs. Prime time scheduling also meets human needs, providing better service to users during regular working hours. This division seems to work well in practice since users are willing to wait overnight for large resource-consuming jobs.

Tables 1 and 2 shows the configuration of scheduling queues for two sample parallel machines, part of our worldwide survey of existing prime/non-prime time scheduling policies. Space limitations prevent us from including more machines, some of which include memory and disk constraints as well. What is apparent is that there is quite a bit of variety in terms of both the number of queues and the constraints associated with each queue. This information motivated us to investigate the underlying principles that govern prime time scheduling.

	jobsize	runtime (hrs)
Prime	16	0.25
6am-7pm	32	4
	144	0.25
Non-prime	100	10
7pm-6am	8	0.25

Table 1. NASA NAS/IBM SP/2 (144 nodes)

	jobsize	runtime (hrs)
Weekdays		
Prime	32	0.25
7am-7:30pm	64	0.33
	128	0.33
	512	0.25
	256	0.33
6am-7pm	512	1.00
Non-prime		
8pm-6am	512	4.44
11pm-4am	64	6.0
8pm-8am	512	1.0
reservation	512	4.44
Weekends		
Non-prime		
F 8pm-M 7am	512	1.00
F 8pm-M 6am	512	4.44
F 11pm-M 6am	64	6.00
F 8pm-M 8am	512	1.00

Table 2. NASA/Cray T3E (512 nodes)

4. Simulation Experiments

In our experiments, we model a system in which user jobs are submitted to a batch scheduler which enforces both long term and short term scheduling (per queue) policies. The long term scheduing policy is a prime/non-prime time scheme with two queues, the prime time job queue and the non-prime time job queue. Newly arriving jobs are queued in the prime time queue if and only if their jobsize and runtime estimates do not exceed the static limits set for prime time. A job whose resource needs exceeds these limits is queued in the non-prime time queue which is served during non-prime time. The short term (per queue) scheduling discipline decides which of the jobs in a given queue is selected for execution on the parallel machine. In our experiments, we analyzed the performance of both FCFS and EASY backfilling for both the prime time and non-prime time job queues. All experiments were conducted using ProcSimity, a simulation tool we developed for evaluation of job scheduling and processor allocation algorithms for distributed memory parallel machines.

Workload traces. For this set of experiments we used a workload trace obtained from the NASA NAS IBM SP/2 cluster. (Simulations for additional MPP and Linux cluster traces are still being conducted.) The NAS SP/2 at Nasa

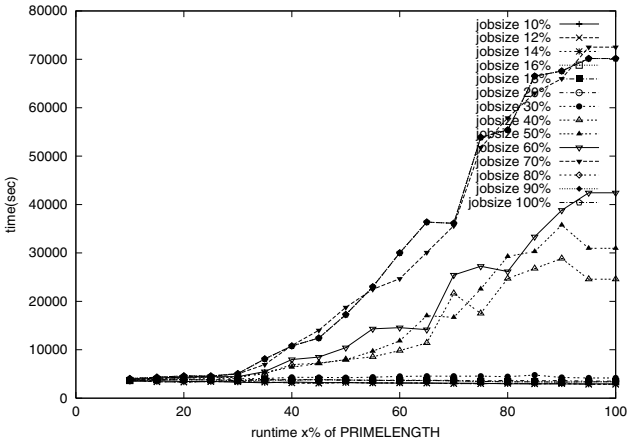

Figure 1. Average response time during PRIME TIME (EASY)

Ames has 160 nodes which are RS6000/690 workstations connected by a high performance switch Omega network. The trace of 39,518 jobs consisted primarily of CFD applications, gathered during the year 1997. Prime time was from 6 AM to 7 PM (13 hours); non-prime time from 7 PM to 6 AM (11 hours). Table 1 shows the queue structure within the prime time and non-prime time classes. Our simulations used the arrival time, jobsize, and runtime data provided by the trace. These traces were sanitized to remove user-specific information and pre-processed to correct for system downtimes, sporadic missing data, and reformatted for use in ProcSimity. Detailed information about the characteristics of this trace and others can be found in [1].

Assumptions. We assumed a 13 hour prime time slot followed by an 11 hour non-prime time slot for all experiments to match the current NAS SP/2 slots. Large jobs that did not complete by the end of a given non-prime time slot were considered *overflow jobs* and were not carried over to the next non-prime time period. In real systems, overflow jobs are either killed or check-pointed/restarted; we assumed the former for simplicity. Small jobs were allowed to run in either prime time or non-prime time. Note that the terms *small job* and *large job* are relative rather than absolute since they depend on a given setting of values for the runtime and jobsize cutoffs. *Small jobs* are those that fall below both of these limits and are allowed to run during prime time, while *large jobs* are those fall above one or both of these limits and thus must run during non-prime time. We also include in the *small jobs*, any job with runtime $\leq$ 15 minutes regardless of jobsize and any job with jobsize $\leq$ 3% nodes regardless of runtime.

Experimental parameters. (independent variables) included the following: (1) *per queue scheduling algorithm*: We used FCFS and EASY backfilling [5] for both the prime time and non-prime time queues. FCFS, as expected, always chooses the job at the head of the job queue. The EASY backfilling algorithm seeks to avoid the bottleneck created by large jobs. When the job at the head of the scheduling queue is unable to run due to lack of resources, EASY backfilling allows short jobs to skip ahead provided they do not delay the job at the head. (2) *jobsize and runtime limits*: We varied the prime time limits on jobsize and runtime as percentages of the machine size and length of the prime time slot, respectively, from 10% to 100% by increments of 10%. Note that 100% limits corresponds to no constraints on the size of job that can run in prime time, i.e. just one scheduling queue. (3) *machine size/system load*: We varied the offered load by fixing the workload but varying the machine size. A smaller machine size corresponds to a heavy load, while the largest machine size corresponds to a light load. We varied the machine size from 50% to 100% of the actual target machine size corresponding to the workload being tested, by increments of 10 nodes. Thus, for the NAS SP/2, machine size ranged from 80 nodes to 160 nodes in increments of 10. We define *load* to capture the static demand for space-time resources divided by the space-time capacity: $(avgjobsize * avgruntime * \#jobs)/(machinesize * totaltime)$.

Performance metrics. The following standard performance metrics (dependent variables) were each plotted against jobtime and runtime limits, and for both FCFS and EASY scheduling policies: average response time, queue waiting time, slowdown, and system utilization. Recall that *slowdown* is the ratio of response time divided by runtime, and is a more discerning measure of performance in the presence of very large (and very small) jobs. We also looked at metrics unique to the prime/non-prime time model: *success rate*: percentage of jobs that complete in the slot for which they were queued, and *overflow*: percentage of jobs that were killed (checkpointed) at the end of non-prime time.

5. Results and Discussion

Average response time during PRIME time. Our preliminary experiments indicated that runtime limits in the range 20% to 40% of the total prime time slot form a critical limit. Similary, the evidence indicates that jobsize limits in the range 20% to 40% of the total machine size form a critical limit. Interestingly, it is only necessary to keep one of the constraints, *either* runtime *or* jobsize, below the critical limit for optimal performance during prime time.

These phenomena are illustrated in Figure 1 which show average response time as a function of runtime and jobsize limits. While we only have space to include graphs for EASY backfilling on an 80 node machine, the performance

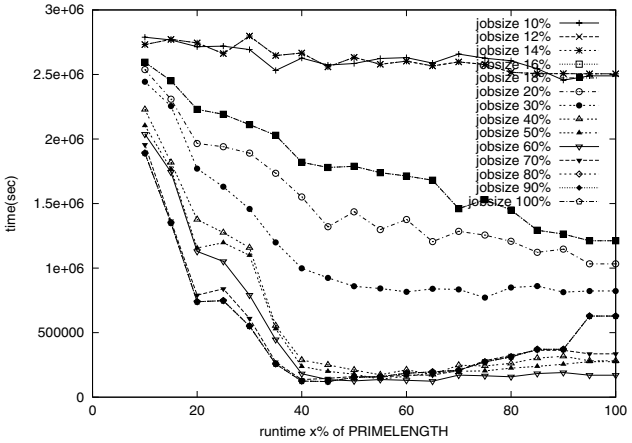

Figure 2. Average response time during NON-PRIME time (EASY)

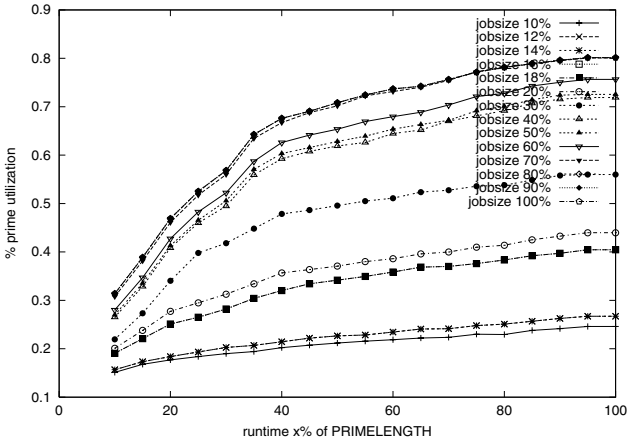

Figure 3. Utilization as a function of runtime limits: PRIME time (EASY)

pattern appears to be universal, regardless of scheduling algorithm (FCFS and EASY) or load/machine size, and for the other metrics related to response time (slowdown and queue waiting time) as well. The only situation which deviates is that of a lightly loaded machine ($\geq$ 140 nodes). In this case, the critical range is wider and thus less sensitive to the cutoff values.

In the figure, we see the following pattern: for limits on runtime below the critical point (30%), average response time for the small jobs is at a minimum and stable regardless of the jobsize limit. In other words, if the prime time runtime limit is set below 30% of the 13 hour prime time slot, performance is optimal without any constraint on jobsize!

However, above the critical point, we see two classes of behavior: in group 1 (jobsize limits $\leq$ 30%), average response time remains stable above the 30% cutoff for runtime, but in group 2 (jobsize $\geq$ 40%) average response time rises dramatically above the critical cutoff. Similarly, if the prime time jobsize limit is set below 30% of the machine size, performance is optimal without any constraint on runtime! We again see two classes of behavior above the critical point: the behavior of group 1 (runtime limit $\leq$ 30%) is stable while the average response time of group 2 (runtime limit $\geq$ 40%), grows dramatically above 30%.

Intuitively, keeping runtime and/or jobsize limits small ensures that no big jobs are scheduled during prime time, yielding good service for small jobs. In these experiments, the 30% limit excludes 5% to 10% of the entire workload from prime time. Excluding this small group of resource hogs from prime time yields two orders of magnitude improvement in average response time for small jobs under FCFS scheduling and an order of magnitude for EASY (see discussion of short term scheduling algorithms below).

On the other hand, setting the cutoffs for runtime and jobsize too low is counterproductive, since it relegates almost all jobs to non-prime time, yielding very poor utilization during prime time (see discussion of utilization below).

We conclude that the best average response time during prime time is achieved when either the runtime limit or the jobsize limit is set to be around 30%. This setting yields utilization levels of 50-65% which, is acceptable, even desirable, during prime time because it means fast service for prime time jobs.

Average response time during NON-PRIME time. The runtime and jobsize limits that benefit non-prime time metrics are, not surprisingly, greater than those that benefit small jobs. As a general rule, setting the prime time limits above 40% for *both* runtime and jobsize yields the best average response time during non-prime time. This is in contrast to the situation for small jobs in which it is sufficient to set either runtime or jobsize *below* the critical point for best performance.

Figure 2 shows average response time during non-prime time for EASY backfilling on an 80 node machine. From this graph, we can see that the critical point is at about 40%, and that if either the runtime or jobsize limits fall below 40%, average response time rises dramatically. Again, this same behavior appears to be universal in our experiments.

Thus, smaller cutoffs benefit prime time while larger cutoffs benefit non-prime time. Fortunately, the critical points are not far from each other: $\leq 30\%$ for prime time and $\geq 40\%$ for non-prime time, and the limits can be selected so as to give greater benefit to prime time (or non-prime time), or to compromise between the two.

System utilization. The impact of runtime and jobsize limits on system utilization is interesting because the trend

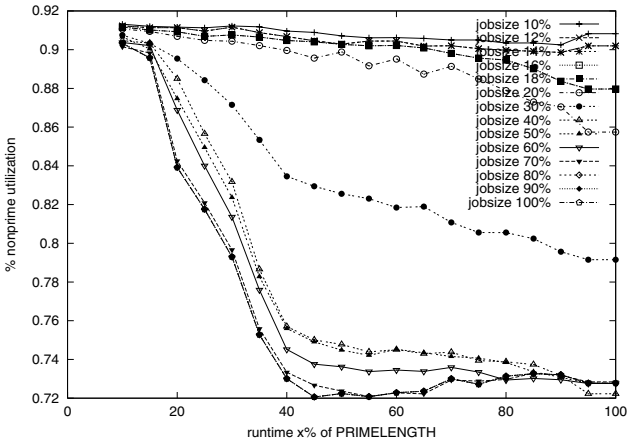

Figure 4. Utilization as a function of runtime limits: NON-PRIME time (EASY)

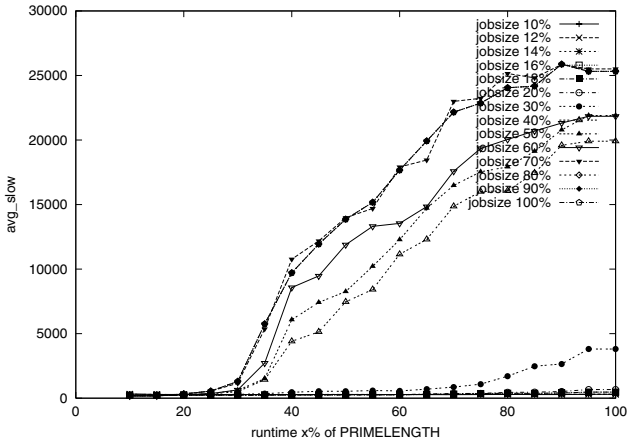

Figure 5. Average slowdown for all jobs (FCFS)

is the reverse of that for average response time. In particular, prime time is best served when both limits are $\geq 40\%$ while non-prime time is best served when both limits are $\leq 20\%$.

Also, prime time utilization is much more sensitive to cutoff values than is non-prime time utilization. For prime time, utilization levels range from 15% to 80%, and thus a poor choice of limits can have a severely adverse impact; for non-prime time, utlization levels range only from 72% to 91%, and thus are not as sensitive to the choice of prime time limits. These patterns are universal in that they hold true for both scheduling algorithms and for all load/machine size levels. Figures 3 and 4 illustrate this behavior.

We conclude that higher runtime and jobsize limits are preferable because low cutoffs yield very poor utilization during prime time, while non-prime time utilization levels are good to excellent, regardless of the chosen limits. Note, however, that high system utilization is not a goal in itself. The tuning of runtime and jobsize cutoffs should focus on the overall per job metrics with an eye to realizing reasonable levels of utilization.

Prime/non-prime time vs. short term scheduling. The issue investigated in this study is the relative benefits of prime time vs. non-prime time queues: (a) which has a greater impact – use of prime time/non-prime time scheduling or choice of short term scheduling algorithm, and (b) for which short term scheduling algorithm, FCFS or EASY, does prime/non-prime time scheduling offer the greatest benefits?

The impact of prime/non-prime time scheduling can be seen by comparing performance when runtime and jobsize limits are both set at 100% (thereby removing the non-prime time queue) to the best performance under prime/non-prime time. For FCFS, Figure 5 shows the average slowdown without prime/non-prime time scheduling to be two orders of magnitude worse than the best achievable. For EASY, Figure 6, we see that average slowdown without prime/non-prime time queues is about one order of magnitude greater than the best achievable. Thus, for both FCFS and EASY, use of two queues does yield huge payoffs, with especially large benefits to be reaped by FCFS. For comparison purposes, the average slowdown achieved soley by changing from FCFS to EASY without any prime/non-prime time queues improves by a factor of 10, also clearly a significant gain.

Success rates and overflow rates. The payoff for prime/non-prime time scheduling is evident when considering the *scheduling success rate*, i.e. the percentage of jobs that arrive and complete execution during prime time. The intuition is that the ability to defer resource hogs to non-prime time improves the service to jobs during prime time. By optimizing the prime time limits, under FCFS it is possible to achieve 80-90% success rates (runtime limits = jobsize limit= 30%). Without prime time queues, FCFS's success rate during the prime time hours is only around 3%!!! With EASY, the highest success rate at 98% (runtime limit = jobsize limit = 30%) and the success rate without prime time queues is about 69%. Again, the benefits of using prime/non-prime queues is strong for both FCFS and EASY, with much greater benefits to be reaped by FCFS.

We are also interested in the impact of prime time scheduling on the percentage of jobs that do not complete by the end of the 24-hour day, which we call the *overflow rate*. It is important that this percentage be low due to the high cost of checkpointing or worse yet, killing, the overflow jobs.

It turns out the prime time scheduling is at a relative dis-

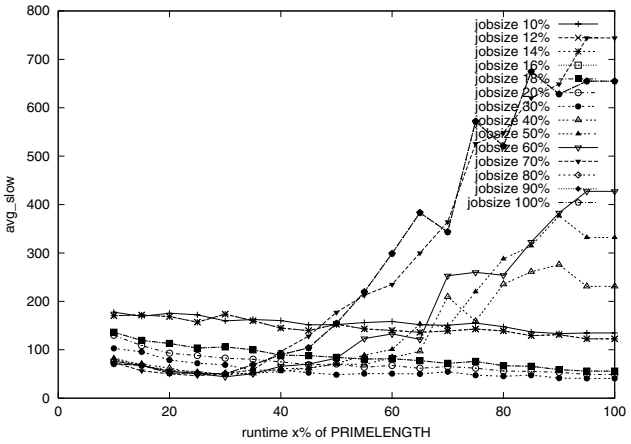

Figure 6. Average slowdown for all jobs (EASY)

advantage in meeting 24-hour deadlines. In particular, the overflow rate is minimum and very low (below one percent) *without* prime time scheduling. Most prime time limits incur a failure rate of 10-18% although rates of less than 5% can be achieved by setting by runtime and jobsize limits at greater than 70%. The tradeoff that occurs to achieve low overflow rates lies in poorer prime time performance: response time, slowdown, and prime time success rate are all penalized significantly.

6. Conclusions

This study shows clearly that prime/non-prime time scheduling is very beneficial for batch scheduling, across short term scheduling algorithms (FCFS and EASY), across workloads and varying levels of system load.

The difficulty lies in choosing the optimum prime time limits for runtime and jobsize because of the competing needs of small vs. large jobs, prime vs. non-prime time service. The rules of thumbs that have materialized out of this study include the following: (1) For best prime time service, choose cutoffs around 30% for either runtime and jobsize; (2) For best non-prime time service, choose cutoffs around 40% for both runtime and jobsize. (3) Maximum system utilization and minimum overflow rates are achieved when both runtime and jobsize are as large as possible.

The Prime Time Rule of Thumb:The best compromise, assuming static limits for runtime and jobsize, is to fix one of them at around 30% and set the other at 100%. Table 3 shows the performance for all metrics using this rule and illustrate tradeoffs for other prime/non-prime time cutoff values. The data shown is for EASY scheduling; FCFS showed the same trends.

Table 3. Performance metrics for three prime time cutoffs (EASY). Ratios are cutoff value divided by best value

Metric	30-100 ratio	80-100 ratio	100-100 ratio
avg resptime-all	1.81	5.17	6.37
avg resptime-prime	1.73	18.87	23.91
avg resptime-non	4.55	2.58	5.19
slowdown-all	1.24	12.80	16.09
slowdown-prime	1.77	18.55	23.16
slowdown-non	6.74	2.32	1.47
Metric	**30-100 percent**	**80-100 percent**	**100-100 percent**
success-prime	98	72	69
overflow	16	4	0
utilization-all	69	75	76
utiilization-prime	57	78	80
utilization-non	79	73	73

References

[1] V. Appaya, D. Guzman, J. B. Lin, V. Lo, J. Mache, and W. Zhu. An analysis of the production parallel workload on the NASA Ames IBM SP/2 and the KFA Cray T3E. Technical Report CIS-TR-98-13, University of Oregon, 1998.

[2] D. Feitelson. Packing schemes for gang scheduling. In *Proceedings of the 2nd Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '96*, 1996.

[3] D. Feitelson and A. M Weil. Utilization and predictability in scheduling the ibm sp/2 with backfilling. In *Proceedings 12th International Parallel Processing Symposium*, 1998.

[4] S. Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. In *Proceedings of the 2nd Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '96*, 1996.

[5] D. Lifka. The ANL/IBM SP scheduling system. In *Proceedings of the 3nd Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '97*, 1997.

[6] Virginia Lo, Jens Mache, and Kurt Windisch. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In *Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '98*, 1998.

[7] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE Trans. Parallel and Distributed Systems*, 12(6), 2001.

[8] J. Subhlok, T. Gross, and T. Suzuoka. Impacts of job mix on optimizations for space sharing schedulers. In *Proceedings of Supercomputing '96*, 1996.

[9] K. Windisch, V. Lo, D. Feitelson, B. Nitzberg, and R. Moore. A comparison of workload traces from two production parallel machines. In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation*, 1996.