Class-Partitioning Job Scheduling for Large-Scale Parallel Systems

Su-Hui Chiang
Computer Science Department
Portland State University
P.O. Box 751-CMPS
Portland, OR. 97207

Mary K. Vernon
Computer Science Department
University of Wisconsin-Madison
1210 W. Dayton Street
Madison, WI. 53706

Abstract

This paper addresses the problem of poor response time for large parallel jobs under nonpreemptive backfill scheduling policies. Four monthly workloads from the large-scale NCSA O2K system are used to evaluate new scheduling policies. Our key result is that partitioning the system based on job class, where class is defined by the number of requested processors, significantly improves overall performance and the performance of large jobs, while still providing similar performance for smaller jobs, compared to the best previous priority backfill policies. Another key advantage of the class-partitioning policies is that selecting high-performance policy parameters is simpler and more intuitive than determining the number of reservations and other parameters of the priority backfill policies.

1 Introduction

Job scheduling for high-performance parallel systems is a great challenge in the presence of large-resource jobs, each of which requires a large fraction of the processors and/or memory available in a given system, and may run for a very long time. Particularly in heavily utilized systems with non-preemptive scheduling, it is difficult to find enough resources for such jobs, and to balance the scheduling of such large jobs against responsive scheduling of a very large number of smaller jobs with highly variable runtimes.

In this paper, we consider job scheduling for the Origin 2000 (O2K) system at the National Center for Supercomputing Applications (NCSA). The NCSA O2K system is a large-scale system with 600 gigabytes of memory and over 1,400 processors. The processors and memory are partitioned into twelve hosts, and each job is constrained to run within the boundary of a single host, due to a high overhead of processor communication across host boundary. The largest O2K job requires the maximum resources available on any host, and runs for up to 400 hours. In order to provide a good performance for the jobs that utilize an entire host, as well as for jobs that require smaller numbers of processors, the scheduling policy adopted on the O2K system designated three hosts to run mainly the largest jobs, leaving the remaining hosts entirely for all other jobs. To take advantage of the processing time not used by the largest jobs, smaller jobs that are under five hours can also run on the partition used by the largest jobs.

One question that has not, to our knowledge, been studied is whether or not non-preemptive scheduling policies that partition the system based on job sizes effectively improve system performance, compared to policies that don't partition the system? If so, how should such partitioning policies be configured?

We address the above questions in the context of nonpreemptive backfill scheduling policies, which have been shown to have promising performance among non-preemptive policies [4, 3, 5]. In particular, backfill policies that give some priority to short jobs, such as LXF&W-backfill [2], have been shown to be competitive with preemptive policies which are significantly more difficult to implement and have higher scheduling overhead.

We evaluate the performance of partitioned and non-partitioned scheduling policies by simulation using four monthly job traces that ran on NCSA O2K during December 2002 - March 2003. We evaluate policies using more complete measures than in most of the previous papers, including average, 95th-percentile, and maximum wait time over all jobs and over each range of requested processors and runtime. The remainder of the paper is organized as follows. Section 2 provides some background information for this study. Sections 3 - 4 present our results of policy evaluation. Section 5 gives a summary.

2 Background

This section provides background information for the scheduling policy evaluation. Table 1 defines the symbols for the job characteristics that are used throughout the paper. Section 2.1 describes the O2K

Table 1. Job Characteristics

Symbol	Definition
M	Requested Memory
P	Requested Processors
R	Requested Runtime
T	Actual Runtime

Table 2. NCSA O2K: System Size

# Processors:	1440
Memory:	600 GB
11 hosts:	128 processors each
1 hosts:	32 processors

Table 3. NCSA O2K: Job Limits

Shared:	$P \leq 64$ & $M \leq 25$ GB & $R \leq 400$ hrs
Dedicated:	$P \leq 128$ & $M \leq 64$ GB & $R \leq 400$ hrs* ($*$: $R \leq 50$ hours prior to 12/19/02)

system and Section 2.2 provides the most important characteristics of the four monthly O2K workloads that are used to evaluate the policies. Section 2.3 defines the priority backfill policies to be evaluated.

2.1 NCSA O2K System

Tables 2-3 show the O2K system capacity and the bounds on job requests for processors, memory, and runtime, during December 2002 - March 2003. The O2K jobs were scheduled by the Maui Scheduler, which implements a priority backfill policy. Shared jobs can space-share the processors on the same host as long as there are enough resources, but each dedicated job runs exclusively on a single host. For the period of time studied, three 128-processor hosts on the O2K were assigned to run dedicated jobs. Those hosts also run shared jobs that request up to 5 hours of runtime when no dedicated jobs are available to run. The remaining nine hosts run only shared jobs.

The workload that consists of only the O2K shared jobs were used in previous work [2, 1] for evaluating backfill and preemptive scheduling policies. In this study, we use the full O2K workload.

2.2 O2K Workload Characteristics

Table 4 provides an overview of the monthly workloads on the NCSA O2K system during December 2002 - March 2003. For each month, the table shows the number of jobs and total processor demand (denoted by "proc. demand") for jobs submitted in the month, as well as for jobs in each given range of requested processors. The processor demand of a given set of jobs is the fraction of the total system that is utilized, on average, by that class during the month. That is, the processor demand is the sum of the number of requested processors (P) multiplied by the actual runtime (R) for each job, expressed as a fraction of the total processor-hours available on all twelve hosts during the month. The memory demand is at least 20% lower than the processor demand each month, not shown to conserve space. Thus, processors are a more scarce resource than memory on the O2K.

As shown in the table, (1) the monthly total processor demand is in the range of 73-92%, (2) both January and February 2003 have a heavy load (i.e., $\geq$ 88%), (3) March 2003 has the lowest load, and (3) in each month, the jobs that request over 64 processors utilize up to 22% of the total system resources, even though these jobs account for a very small fraction ($<$ 3%) of all jobs.

The three months in 2003 are chosen because the load due to the jobs requesting over 64 processors in these months is significantly higher than in most other months (due to a change in the system configuration). In December 2002, the largest class of jobs account for a lower load and this month is included to have a variety of workloads for study.

Figure 1 plots the distribution of requested runtime in Feb. 2003. Other months have a similar distribution. As shown in the figure, about half of the jobs request 5, 50, 200, or 400 hours of runtime, and another significant fraction of jobs request 10 minutes or 1 hour. As in previous workloads, the majority of jobs (i.e., 80-85% for the O2K) requested a power of two processors (not shown).

2.3 Definition of Priority Backfill Policies

Priority backfill policies use a weighted sum of job characteristics for prioritizing the jobs that are waiting to be scheduled. Table 5 defines the weights and job metrics that are relevant to the policies that will be evaluated. Table 6 provides the values of the weights used in each policy that will be evaluated: FCFS, LxF&W, LxF&W&P, and LPF (i.e., largest requested processors first). Previous work shows that LxF&W-backfill has significantly lower average and 95th-percentile waiting time than FCFS-backfill, while having comparable maximum wait time, for the shared O2K jobs [1]. Both LxF&W&P and LPF functions use a non-zero weight for the number of requested processors, and thus have the potential to improve the performance of the largest jobs.

Under the priority backfill policies, a small number of waiting jobs with the highest priority are given

Table 4. NCSA O2K Monthly Workload (December 2002 - March 2003)

Month	Measure	Total	Requested Processors (P)							
			1	2	3-4	5-8	9-16	17-32	33-64	65-128
Dec. 02	#jobs	7370	3380	294	1189	1138	1059	539	242	56
	proc. demand	81%	2%	2%	8%	15%	24%	7%	12%	12%
Jan 03	#jobs	6861	3167	270	1326	869	977	333	210	183
(heavy)	proc. demand	92%	2%	1%	8%	12%	25%	7%	15%	22%
Feb 03	#jobs	8121	3424	300	1957	1245	995	517	142	146
(heavy)	proc. demand	88%	2%	2%	8%	15%	16%	14%	10%	21%
Mar 03	#jobs	7552	2801	300	1484	1331	1174	722	452	94
	proc. demand	73%	2%	2%	8%	14%	17%	7%	5%	18%

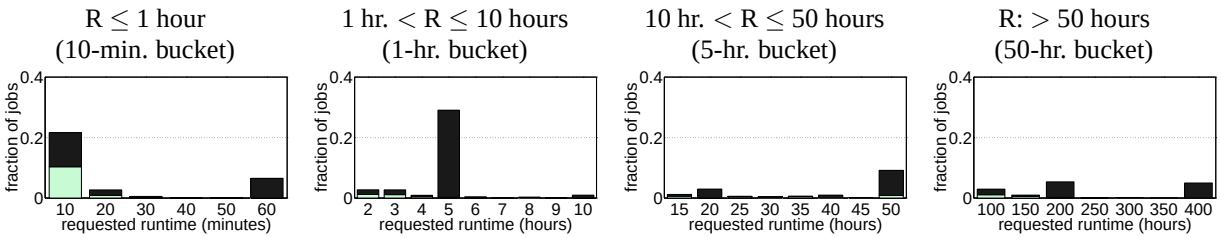


Figure 1. Distribution of Requested Runtime (Feb. 2003)

Table 5. Job Priority Function Weights

Weight	Job Metric to be Multiplied
W_w	J_w, current job wait time in *hours*
W_x	J_x, square root of current estimated expansion factor = $\frac{J_w + R\ in\ hours}{R\ in\ hours}$
W_p	P, requested processors

Table 6. Policy Priority Functions

Policy	Weight Values		
	W_x	W_w	W_p
FCFS	0	1	0
LxF&W	1	0.02	0
LxF&W&P	1	0.02	0.01
LPF	0	0	1

(Note LxF&W was called L$\sqrt{X}$F&W in [1])

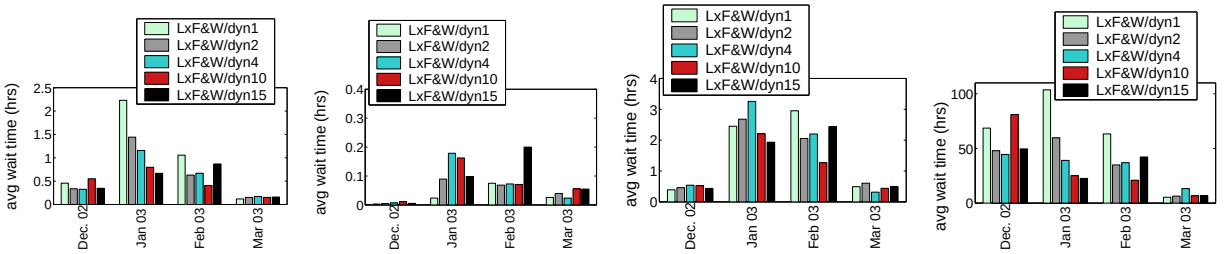
a scheduled start time (i.e., a reservation). The number of jobs that receive a reservation is a parameter of the policy. Smaller jobs that have lower priority can be scheduled in priority order on idle resources as long as they do not delay any reserved start times.

In previous work [1], we evaluated the impact of the number of reservations on various priority backfill policies for the O2K shared jobs (i.e., P $\leq$ 64). We found that using a few reservations (2-4) improves the maximum wait without hurting other performance measures studied, but using more than a few reservations makes no or minimal further improvement on the maximum wait and in fact degrades the average wait time of all jobs.

In Section 3, we again study the impact of the number of reservations on backfill policies, but for the full O2K workloads that include jobs that require over 64 processors. We assume *dynamic* reservations [1], denoted by "dynK", which means that at each point in time the reservations are given to the K jobs that *currently* have the highest priority.

3 Single-Class Backfill Policies

In this section, we evaluate alternative priority backfill policies, including the impact of the number of reservations in each policy, assuming the policy is used to schedule jobs on all twelve of the O2K hosts. That is, hosts are not designated to run particular classes of jobs. One goal is to study whether using more reservations or giving priority to jobs that request more processors can improve the response time for large jobs. Another goal is to improve the priority backfill policies and to identify the *best* backfill policy among the policies considered, for comparisons with class-partitioning policies in Section 4. We vary the number of reservations in the range of 1 to 40 for each policy: LxF&W-backfill, FCFS-backfill, LPF-backfill, and LxF&W&P-backfill. To show policy performance for jobs that have different ranges of requested processors, we show results for



(a) Average Wait All jobs (b) Average Wait Small jobs (c) Average Wait Medium jobs (d) Average Wait Large jobs

Figure 2. Performance Impact of Reservations

Table 7. Definition of Job Classes

Small	Medium	Large
$P < 32$	$32 \leq P \leq 64$	$64 < P \leq 128$

three classes of jobs: small, medium, and large, as defined in Table 7.

Figures 2(a)-(d) show the average wait time each month for all jobs, and each of the three job classes, respectively, under LxF&W-backfill with 1, 2, 4, 10, and 15 reservations. The results for the maximum wait (not shown) are qualitatively similar. With one reservation, the average wait increases from under 0.2 hour for small jobs and under 3 hours for medium jobs to over 50 hours for large jobs during the two heavy-load months (i.e., Jan. and Feb. 2003), which illustrates the potential performance problem of non-preemptive backfill policies for large jobs. Using 2 - 4 reservations for LxF&W-backfill significantly improves the average wait of all jobs as well as large jobs, while still keeping the average wait for small and medium jobs reasonably low, compared to that of using one reservation. Previous results [1] showed that using a small number of reservations (2-4) results in the best performance for LxF&W-backfill for the shared O2K jobs. However, the figure shows that when large jobs (i.e., $P > 64$) are included in the workload, a larger number of reservations may improve performance. In particular, during the two heavy load months, using 10 reservations is best, as it reduces the average wait of large jobs and also improves the mean wait for medium jobs, while providing still reasonable performance for the small jobs. The results for 20 - 40 reservations (not shown) are similar or worse than that of 10 reservations, except for the December 2002 workload. For the December workload, using 25 reservations significantly improves the average and maximum wait (by 30-50%) for large jobs and at the same time having the same or better average wait for other jobs, compared to using 10 reservations. One problem with determining the number or reservations is that the best number changes as the workload varies. For example, as shown in Figure 2, the 10-reservations policy is best has almost double the average wait time for the large jobs compared to 2 or 4 reservations, during December 2002.

Similar to that for LxF&W-backfill, we have evaluated each of the other three priority backfill policies by varying the number of reservations, not shown to conserve space. The key result is that the policies that use a weight for requested processors, namely LPF and LxF&W&P, and FCFS have similar or worse performance than LxF&W-backfill with 10 reservations for the three months in 2003 and LxF&W-backfill with 25 reservations in December 2002. Another conclusion from tuning these backfill policies is that the performance of using more than a few reservations can be unpredictable, due to the complex interaction between the reservations and the workloads.

To compare against the class-partitioned policies in the next section, we use LxF&W-backfill with 25 reservations for December 2002 and 10 reservations for the three months in 2003.

4 Class-Partitioned Policies

On a heavily utilized large-scale system, partitioning the system based on classes of jobs, where class is defined by the number of requested processors, may reduce the problem of finding enough processors to run a large job when it has the highest priority among the waiting jobs. In this section, we evaluate whether and to what extent partitioning the system based on job classes improves the performance of the O2K system.

One problem that occurs in many production job schedulers is that the interaction between different policy configuration parameters is complex and it is difficult to optimize the parameter values over the range of workloads that occur in practice. Thus, simplicity is one key goal in policy design. To that end, we extend from the *best* previous policy, whenever possible adding pa-

Table 8. Processor Demand By Job Classes on NCSA O2K

	P > 64	P ≤ 64		
		R ≤ 5h	R ≤ 20h	R ≤ 50h
Dec. 02	12%	6%	8%	19%
Jan. 03	22%	4%	5%	15%
Feb. 03	21%	5%	7%	15%
Mar. 03	18%	4%	6%	13%

rameters that have an impact that is reasonably intuitive or predictable.

We consider partitioning the hosts into two partitions: a *class-one* partition used principally for large jobs (defined to be jobs with P $>$ 64), and a *class-two* partition used principally for small and medium jobs. Note that host boundaries remain within each partition. Two aspects of the policy remain to be determined. First, how big should each partition be? Second, to what extent should each partition run other job classes.

Dedicated partitions, i.e., no sharing of resources between different job classes, can result in under-utilization of resources in each partition. Thus, similar to the policy used on the NCSA O2K (discussed in Section 2.1), the class-one partition gives priority to large jobs but will backfill other jobs that are *short*. For the class-two partition, we consider two versions of the policy: part-L/SM and part-L/A. In part-L/SM, only small and medium jobs can run on class-two partition; in Part-L/A, all jobs can run on the class-two partition. In both versions of the class-partitioned policy, at most two reservations can be made for resources in the class-two partition, but large jobs can make reservations in the class-one partition until up to two times the available resources on each host are reserved. The former rule was found to be optimal for small and medium jobs in previous work [1]. Allowing waiting large jobs to reserve all of the resources on each class-one host limits the fragmentation of the class-one partition when large jobs are waiting. The LxF&W policy prioritizes all of the waiting jobs, and then schedules the highest priority jobs that are allowed to run, backfill, and make reservations in each partition.

There are two parameters in the above class-partitioning policies: (1) the number of the 128-processor hosts, n, that form the class-one partition, and (2) the runtime limit, r, for the small and medium jobs that are eligible to run in class-one partition.

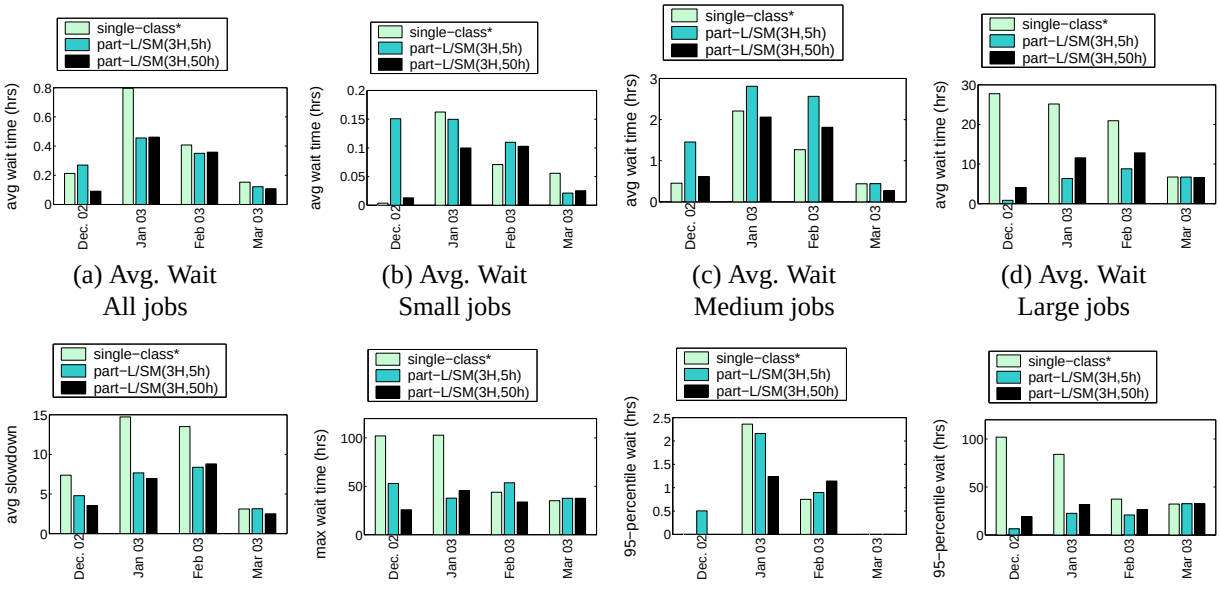
Clearly, the *best* values of the two parameters are not independent, but there are simple guidelines for choosing their values. First, the total processing time available on the hosts in the class-one partition should satisfy all or most of the processing time requirement of the large jobs. Second, the processing time required by all jobs eligible to run in class-one partition should be greater than the available time on the partition. Third, the value of r should be selected to limit the maximum wait time for the first large job in the queue.

Table 8 shows the processor demand of the largest jobs and of the small and medium jobs requesting up to each given runtime (5, 20, 50 hours). Based on the data in the table, three 128-processor hosts (i.e., about 27% of the total number of processors on O2K) will be needed in class-one partition to accommodate the processor demand of large jobs (21-22%) during the two heavy-load months. In the case of part-L/A, we also consider $n = 2$, since large jobs are also allowed to run on the class-two partition.

For the value of r, we consider 5 and 50 hours. Based on the data in Table 8, any value under 5 hours for r may result in under-utilization of the class-one partition, even in the two heavy-load months. Since only a very small fraction of jobs requesting a runtime greater than 5 hours and smaller than 50 hours in the workloads studied (Figure 1), using a value between 5 and 50 hours is similar to using $r = 5$.

In the figures below, the "single-class*" policy refers to the best priority backfill policy with no partitioning, from the previous section. The name of each class-partitioning policy consists of two components: (1) either "part-L/SM" or "part-L/A" partitioning, and (2) (nH, r), e.g., (3H,50h), specifying the number of 128-processor hosts assigned to class-one partition and the requested runtime limit for the small and medium jobs that can run in class-one partition.

To begin with, we compare three policies: single-class*, part-L/SM(3H,5h), and part-L/A(3H,50h). Figures 3(a)-(d) show the average wait time of all jobs, small, medium, and large jobs, respectively, under each of the three policies. Figures 3(e)-(g) show the average slowdown, maximum wait, and 95th-percentile wait of all jobs, and Figure 3(h) shows the 95th-percentile wait of large jobs under each policy. The results in Figures 3(a), (d), and (e)-(h) show that, compared to single-class*, part-L/SM(3H,50h) provides the same or improved performance for all jobs in each month for the performance measures studied, and greatly reduces the average and 95th-percentile wait for the large jobs in three of the four monthly workloads. Figure 3(b)-(c) show that part-L/SM(3H,50h) also provides reasonably low average wait for small and medium jobs. The results for the maximum wait for each job class (not shown) are similar to that of the average wait. Considering the overall performance and considerably improved



(a) Avg. Wait All jobs (b) Avg. Wait Small jobs (c) Avg. Wait Medium jobs (d) Avg. Wait Large jobs

(e) Avg. Slowdown All jobs (f) Max. Wait All jobs (g) 95th-percentile Wait All jobs (h) 95th-percentile Wait Large jobs

Figure 3. Policy Performance Comparison: Overall Measures

performance for the large jobs, we conclude that part-L/SM(3H,50h) outperforms the single-class policy. The key to the higher performance of the class-partitioned policy is the limited fragmentation of the resources in the class-one partition where the large jobs are run. On the other hand, part-L/SM(3H,5h) performs significantly less well than part-L/SM(3H,50h) for small and medium jobs, because the total demand of all jobs eligible to run on the class-one partition is too small for the small value of r (i.e., 5 hours).

Figure 4 provides more detailed policy performance comparisons for the January 2003 workload, which had the largest performance difference between the two policies. Figure 4(a)-(c) plot the average, 95-percentile, and maximum wait time for jobs with different ranges of actual runtime. Figure 4(d)-(g) plot the average and maximum wait for each range of requested processors and for each range of requested runtime. The results in Figure 4 show that part-L/SM(3H,50h) significantly improves the average wait and maximum wait time for most ranges of actual runtime, the large jobs, and long jobs (R $>$ 5 hours), compared to that of single-class*.

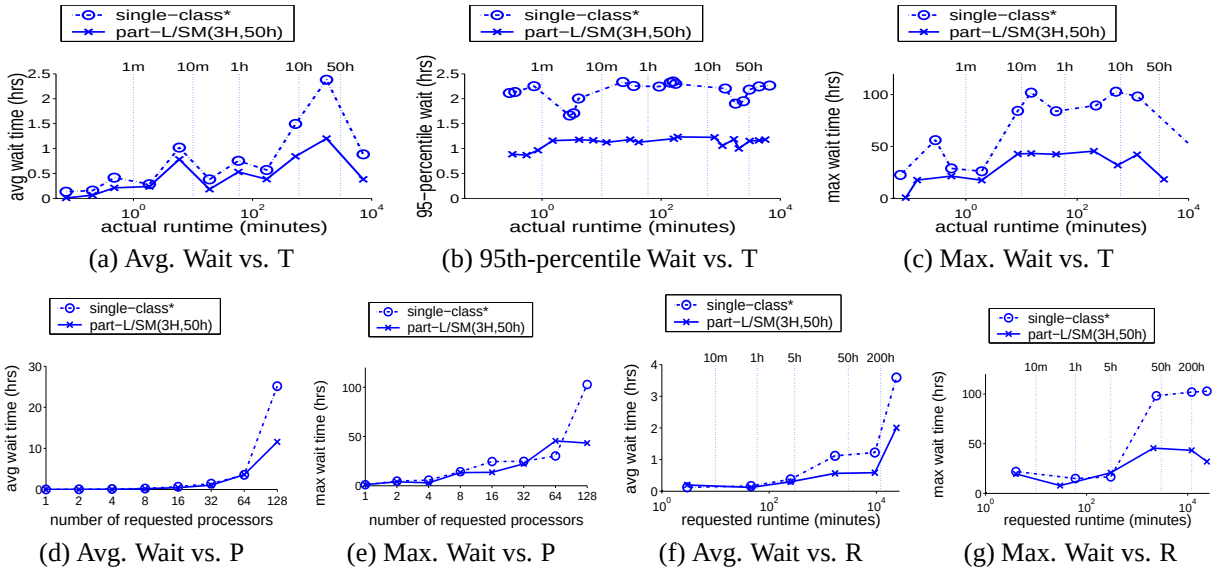
We also evaluated part-L/A partitioning by comparing part-L/A(2H,5h) and part-L/A(3H,50h) to part-L/SM(3H,50h), and found that part-L/A(3H,50h) has very similar performance to part-L/SM(3H,50h) each month. On the other hand, part-L/A(2H,5h) has poor performance during the two heavy-load months, as illustrated in Figure 5(a)-(d), which plot the maximum wait versus number of requested processors under each policy during these two months and average and maximum wait of all jobs each month. Thus, although large jobs can run on either partition in the part-L/A policy, for high performance the class-one partition must have enough resources to satisfy the demand of the large jobs.

5 Conclusions

In this paper, we examined the problem of providing good performance for large-processor jobs in a heavily utilized system with many small jobs, under priority backfill policies. We use four monthly workloads from the NCSA O2K system, which is a large-scale parallel systems partitioned into twelve hosts during the period when the workloads ran on the system.

The key result is that partitioning the system based on classes of job requested processors, such that sufficient resources are designated to give priority to the largest jobs in the system, can significantly improve the wait times of the largest jobs without significantly impacting smaller jobs, compared to using many reservations ($\geq$ 10) in backfill policies.

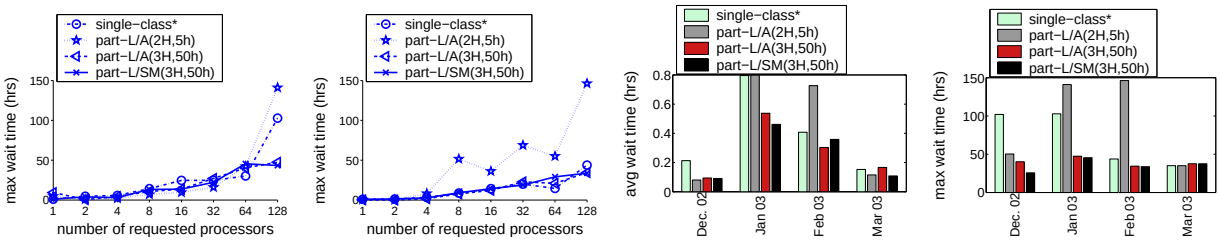
Although the configuration for the class-partitioned policies depends on the load of the large jobs, it is simpler and more intuitive to select the partitioned configuration than to tune the number of reservations for backfill policies. Furthermore, if too many resources are allocated to the large jobs, the resources are still utilized by the appropriate range of small and medium jobs, which can be dynamically adjusted in response to



(a) Avg. Wait vs. T (b) 95th-percentile Wait vs. T (c) Max. Wait vs. T

(d) Avg. Wait vs. P (e) Max. Wait vs. P (f) Avg. Wait vs. R (g) Max. Wait vs. R

Figure 4. Policy Performance Comparison: More Detailed Measures



(a) Max. Wait vs. P Jan. 2003 (b) Max. Wait vs. P Feb. 2003 (c) Avg. Wait All jobs (d) Max. Wait All jobs

Figure 5. Performance of part-L/A

changes in system load.

For the workloads studied, our results show that as long as sufficient resources are used serve the largest jobs, the remaining hosts can run only the smaller jobs or can also run the largest jobs, with minimal difference on the performance. However, allowing more resource sharing among job classes as in part-L/A may be beneficial for other workloads.

On-going and future work includes (1) evaluating class-partitioned policies for workloads on clusters or other multi-host systems (such as NCSA IBM-p690); (2) investigating alternative approaches to improve scheduling performance for workloads with mixtures of large and small jobs (e.g., [4, 3]).

References

[1] S.-H. Chiang, A. Dusseau-Arpaci, and M. K. Vernon. The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance. In Proc. 8th Workshop on Job Scheduling Strategies for Parallel Processing, Scotland, July 2002.

[2] S.-H. Chiang and M. K. Vernon. Production Job Scheduling for Parallel Shared Memory Systems. In Proc. Int'l. Parallel and Distributed Processing Symp. 2001, San Francisco, April 2001.

[3] 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 Syst., 12(6):529–543, June 2001.

[4] J. Subhlok, T. Gross, and T. Suzuoka. Impact of Job Mix on Optimizations for Space Sharing Schedulers. In Proc. 1996 ACM/IEEE Supercomputing Conf., Pittsburgh, November 1996.

[5] D. Zotkin and P. J. Keleher. Job-Length Estimation and Performance in Backfilling Schedulers. In 8th IEEE Int'l Symp. on High Performance Distributed Computing, Redondo Beach, August 1999.