Production Job Scheduling for Parallel Shared Memory Systems

Su-Hui Chiang suhui@cs.wisc.edu
Mary K. Vernon vernon@cs.wisc.edu
Computer Sciences Department
University of Wisconsin-Madison

Abstract

This paper addresses open job scheduling questions for the challenge workloads *that run on the large scale parallel systems at supercomputer centers. Simulation results for six recent one-month job traces from the NCSA Origin 2000 (O2K) system are used to evaluate (1) the experimentally tuned NCSA LSF* policy, (2) the FCFS-backfill policy, (3) the Priority-backfill policy with alternative priority functions and with limited preemption to provide immediate service to each arriving job, and (4) the spatial equipartitioning (EQspatial) policy with an optional modification to reduce the maximum waiting time for the largest jobs in the challenge workloads. Measurements on the O2K validate the simulation results for two of the policies. The priority-backfill policy with immediate service and a starvation-free priority measure that favors short jobs is shown to be the most promising if jobs cannot adapt to changing processor allocations at runtime, but EQspatial provides significantly better 95th percentile waiting time.*

1. Introduction

The workloads that run on the large-scale parallel systems at supercomputer centers are challenging from a scheduling perspective due to high job arrival rate (several hundred submissions per day), high utilization of system processors and memory (e.g., 80-90%), and the unpredictable arrival of jobs that run for hundreds of hours while occupying a significant fraction of system resources.

Production job schedulers employed at supercomputing centers use a variety of nonpreemptive scheduling policies. These policies include the **LSF*** scheduler employed until recently on the NCSA SGI O2K, the **Priority-backfill** scheduler on the IBM SP at the Maui High Performance Computing Center (MHPCC), and the **FCFS-backfill** scheduler on the IBM SP at Argonne National Laboratory.

Previous research has shown that preemptive scheduling policies, particularly those that provide all jobs in the system with (close to) an equal fraction of the total system processing cycles, have the potential to provide much better job turnaround times than non-preemptive policies [13, 26, 3, 19]. However, to our knowledge, these preemptive policies have not been compared against the production scheduling policies identified above, in the context of the memory and processing requirements of the challenge workloads. This paper thus investigates the following open questions:

1. What is the relative performance of the above three non-preemptive scheduling policies for the workload that runs on the NCSA O2K?
2. Are there particular priority measures that significantly improve the performance of the Priority-backfill scheduler for the NCSA workload?
3. To what extent is performance improved or degraded in the shared memory system by adding a limited form of preemption to the nonpreemptive scheduling policies, so that each job that arrives is immediately allocated a small amount of memory and a short quantum of running time on the number of processors it requests?
4. Assuming a programming and runtime environment that enables each job to dynamically adjust to the number of processors allocated to it, such as Cilk [8], or the adaptive system in [21], to what extent are job turnaround times improved by a spatial processor equipartitioning policy (EQspatial) with low scheduling overhead?

The alternative scheduling policies are evaluated using simulation with six one-month job traces that were logged on the O2K during October 1999 through March 2000. The job logs include the processors, memory, and runtime *requested* and *used* by each job. Section 3 provides a characterization of the processing and memory requirements of the jobs in each of the six workloads. This data shows that the job mix submitted to the O2K system is very similar from month to month.

Table 1. NCSA O2K LSF Job Class Definitions

Class Name	Resource Request Limits		Reserve
Time	CPU Time Per Processor	Run Time Per Job	Wait Time
vst (very short)	$\leq$ 5 hrs	$\leq$ 5.5 hrs	$\geq$ 12 hrs
st (short)	$\leq$ 50 hrs	-	$\geq$ 45 hrs
mt (medium)	$\leq$ 200 hrs	-	$\geq$ 180 hrs
lt (long)	$\leq$ 400 hrs	-	$\geq$ 360 hrs
Size	# Processors	Memory	
sj (small)	≤ 8	$\leq$ 2 GB	
mj (medium)	≤ 16	$\leq$ 4 GB	
lj (large)	≤ 64	$\leq$ 16 GB	

Results in Section 4 predict significant improvement in job turnaround time for the FCFS-backfill and Priority-backfill policies, as compared with the experimentally tuned NCSA LSF* policy. Based on projected performance improvements, NCSA recently changed the scheduler on the O2K to a priority-backfill scheduler with similar parameter settings. The measured improvement in system performance agrees with the improvement predicted by the simulations. Further simulation results for the NCSA workloads predict that (1) using a job priority measure that favors short jobs significantly improves the performance of the Priority-backfill policy, (2) adding *limited* preemption to the Priority-backfill policy to provide immediate service for each arriving job greatly improves turnaround time for very short jobs without significantly impacting jobs with longer execution time, (3) the EQspatial policy has the potential for further substantial improvement in system turnaround time, but perhaps not as substantial as previous research results would suggest, and (4) it may be important to modify the EQspatial policy to reduce the maximum waiting time of the longest running jobs.

2. Background

This section defines the scheduling policies evaluated in this paper and provides a review of the relevant previous work. The scheduling policies are described as they apply to the non-industrial batch jobs that run in a space-sharing fashion on the NCSA O2K. Extensions to serve special job classes and requests (e.g., industrial jobs, requests for dedicated time, advance reservations [24]) are beyond the scope of this paper.

2.1. The NCSA-LSF* Scheduler Policy

The NCSA-LSF* scheduler [18] on the O2K is configured with the twelve experimentally tuned static job classes defined in Table 1. A job belongs to one of the classes based on its requested number of processors, memory, cpu time, and running time, which are specified when the job is submitted. For example, a job that requests 16 processors, 4 gigabytes of memory, and $300 \times 16 = 4800$ hours of cpu time, is in the lt-mj class.

The O2K has eight "hosts", each configured, as shown in Table 2, with a set of experimentally tuned job class priorities and constraints that provide particular relative levels of service to the twelve job classes. For example, host "eir" will run up to three lj jobs and four mj jobs simultaneously. Within these constraints, vst-lj jobs have highest priority for scheduling on eir, followed by other lj jobs. If three lj jobs are currently running on eir, then vst-mj jobs have highest priority for scheduling, followed by other mj jobs, and so forth. Within each priority class, jobs are considered for scheduling in first come first serve (FCFS) order. Note that one host runs only sj jobs, one host runs only mj jobs, and the other six hosts give priority to lj jobs.

When a job is scheduled on a host, it is assumed to occupy the number of processors and amount of memory it requested until it terminates.

When a job terminates on a host A, if there is any job that (a) has been in the wait queue longer than the Reserve Wait Time for its job class, and (b) does not yet have a special reservation on any other host, whichever such waiting job has highest priority for running on host A is either scheduled on host A (if there are enough processors and memory available) or is given a special reservation on host A.

If no special reservations are pending for a host, the host's free processor and memory resources are assigned to waiting jobs. To assign the free resources, the scheduler can skip over jobs that are too big in the time class priority ordering. The scheduler cannot skip to a lower priority job size class unless the upper bound on the number of executing higher priority size jobs is reached.

Note that if a special reservation is pending, the free processors and memory are kept idle until the job with the reservation can be scheduled. Special reservations are made infrequently on the O2K due to the high waiting time thresholds. The idle time due to holding host resources for special reservations is measured to be under 4% for the workloads studied in this paper.

2.2. The Priority-backfill Policy

The Priority-backfill scheduling policy evaluated in this paper is derived from the Maui Scheduler (MS) on the IBM SP at MHPCC. The MS policy is augmented to accommodate the host boundaries and job memory requirements on the NCSA O2K.

A submitted job specifies a requested number of processors, amount of memory, and running time. Each job has an individual priority that is dynamically computed

Table 2. **Host Resources and Scheduling Priority on NCSA Origin2000**

Host Name	**Number Processors**	**Memory** Total (GB)	Per Processor (MB)	**Scheduler Configuration** Job limits	Job priority
eir	128	64	512	lj≤3; mj≤4	lj > mj > sj; vst > st,mt,lt
nerthus	128	64	512	lj≤3; mj≤4	lj > mj > sj; vst > st,mt,lt
jord1	128	32	256	lj≤3; mj≤4	lj > mj > sj; vst > st,mt,lt
saga1	128	32	256	lj≤3; mj≤4	lj > mj > sj; vst > st,mt,lt
huldra	128	32	256	lj≤3; mj≤4	lj > mj > sj; vst > st,mt,lt
hod1	128	64	512	lj≤6; mj≤2	lj > mj > sj; vst > st,mt,lt
mimir	128	32	256	-	sj only; vst > st,mt,lt
modi2	64	16	256	-	mj only; vst > st,mt,lt
System-wide job limits: mt ≤ 48; lt ≤ 20					
Per-user job limit = 3 (the job limit is infinity in our simulations)					

Table 3. **Job Metrics and Policy Weights**

Weight Symbol	**Weight Value** Priority-bf	LXF&W-bf	**Job Measure**
W_w	1	0.0167	job wait time, J_w, in hours
W_x	5	1	estimated expansion factor, $J_x = \frac{J_w + requested\ runtime\ in\ hours}{requested\ runtime\ in\ hours}$
W_p	0.2	0	requested number of processors, J_p
W_m	0	0	requested memory in MB/358.4*, J_m
W_{pe}	0	0	max$\{J_p, J_m\}$
W_{bypass}	0	0	the number of lower priority jobs scheduled ahead of the job

* The average memory per processor on the eight hosts is 358.4 MB.

as the weighted sum of several job metrics defined in Table 3[1]. Jobs are scheduled in priority order, using backfill to schedule lower priority jobs on processors and memory that would otherwise be idle in the strict priority schedule. At each point in time, a specified number of the highest priority jobs that are waiting in the queue are each given a scheduled start time on the hosts that have the earliest guaranteed start times. The policy comparisons in this paper assume this number is one. (Simulations with larger numbers did not improve policy performance.) If an arriving job can be scheduled on more than one host that has enough unallocated processors and memory, the job is placed on the host that has the fewest unallocated processors.

The dynamic priority calculation enables a job's priority to increase as the job waits to be scheduled. Note that if $W_x > 0$, the priority of shorter jobs increases faster than that of longer jobs as a function of the job waiting time. Except as otherwise noted, the waiting job that has the scheduled start time is dynamically the highest priority waiting job.

[1] We've found through simulation that the weights in the table give better performance for the NCSA O2K than the weights used on the SP at MHPCC.

2.3. The FCSF-, SJF-, and LXF-backfill Policies

The following four policies that are similar to Priority-backfill but use different definitions of job priority are also evaluated in this paper:

In the FCFS-backfill policy each job's priority is statically defined by its arrival time.

LXF&W-backfill uses the weights defined in Table 3 to compute the dynamic job priorities.

In the SJF-backfill policy, each job's priority is statically equal to the inverse of its requested runtime. To reduce starvation, once a job has a scheduled start time, it does not release its start time to a shorter job that arrives before it runs.

2.4. Backfill Policies with Immediate Service

To provide fast turnaround for short jobs, the backfill policies can be extended to preemptively give each arriving job an immediate small memory allocation and small quantum of running time on the number of processors it has requested. Note that a job requesting a long running time might terminate prematurely due to execution behavior that can't be anticipated ahead of time. For example, in the six one-month workloads on the NCSA

O2K studied in this paper, 12-33% of the jobs that request over an hour of running time terminate in under one minute. For jobs that request over ten hours of running time, 11-42% terminate in under one minute.

The initial memory allocation and quantum of run time should be selected so as to improve turnaround time for a significant fraction of small jobs without negatively impacting the other jobs in the system. For the evaluations in this paper, the size of the quantum and the amount of memory allocated during the quantum are derived in section 3.

When choosing the job(s) to be preempted, only the hosts that have enough unallocated memory and the jobs that have been executing without interruption long enough (i.e., 10 minutes) are considered. On each such host, the eligible job(s) with the smallest value of current *slowdown* (i.e., total time in system so far divided by total runtime so far) are selected to be preempted, until enough host processors are available to start the new job. A preempted job vacates its processors but occupies its memory during the quantum. The host that will have the smallest number of idle processors after preempting the selected jobs and starting the new job, is selected to run the new job for its quantum.

The backfill policies with and without immediate service as described above are compared to see what impact the immediate service has on the waiting time of the other jobs in the system.

2.5. The Spatial Equipartitioning Policy

The spatial equipartitioning policy (EQspatial) dynamically adjusts the number of processors assigned to each job, attempting to give each executing job an equal number of the system processors [26, 15]. We evaluate this policy, modified for the partitioned-host O2K and to reduce scheduling overhead, as described below.

Each submitted job specifies a requested amount of memory and (maximum) number of processors. To avoid memory overcommitment, each executing job is allocated its requested memory, and an arriving job is only scheduled on a host that has enough unallocated memory to satisfy its memory request. When a job arrives, if more than one host has the requested amount of unallocated memory and the requested number of unallocated processors, the job is placed on the one of those hosts with the fewest unallocated processors. If no hosts have the requested number of unallocated processors, the job is placed on the host with enough unallocated memory that has the maximum ratio of number of processors on the host to the total number of processors requested by the jobs currently running on the host (including the new job that might be placed there). The host processors are then partitioned among the jobs executing on the host as equally as allowed by the jobs' processor requests. (Several other host placement heuristics were evaluated, and the heuristic defined here provides the least deviation from EQspatial applied to an unpartitioned 960-processor O2K system.)

If no host has enough unallocated memory to satisfy a new job's memory request, the job is given a small quantum and small initial memory allocation (as in the backfill policies with immediate service). If the job exceeds the memory allocation or doesn't finish within the quantum, it waits until a host has enough free memory to satisfy its memory request.

While a job waits for memory, the amount of processing time it would receive if memory were available is computed. When the job is scheduled, it is given its requested processors until its processing time is approximately the same as it would have been had it been scheduled when it arrived. During the time that the job is given extra processors, the extra processors are deducted from the number of processors on the host when determining which further jobs to place on the host.

A MinInterval parameter defines the minimum amount of time that must elapse between processor repartitionings. When a job terminates or a job is scheduled on a host according to the above rules, if the time since the last repartitioning of the host is shorter than MinInterval, a newly scheduled job can execute on unallocated processors (which may have been vacated by a job that terminated since the last repartitioning), but the system waits until the MinInterval is satisfied before repartitioning the processors.

To use the EQspatial policy, considerable effort would be required to modify a significant fraction of the applications to use a a runtime system (e.g., Cilk [2, 8]) that supports adaptive job parallelism. This paper provides quantitative data to evaluate whether the effort to retarget the applications would be justified.

2.6. Related Work

Backfill has been implemented in several system schedulers [10, 14, 23] and has been shown to be effective in increasing the system utilization [11].

Feitelson and Weil [7] show that the performance of FCFS-backfill is relatively insensitive to the number of jobs that are given scheduled start times. Talby and Feitelson [25] propose a FCFS-backfill policy with a more flexible backfill deadline, and show that this flexible backfill deadline improves average wait by 10-20% for twelve monthly IBM SP/2 workloads. (Flexible backfill deadlines are not evaluated in this paper.) Leinberger et al. [12] propose a modified FCFS-backfill that balances the utilization of different resources (i.e., processors and memory), which yields 10-20% lower average

Table 4. Total Monthly Processor and Memory Demand By LSF Job Class

Month	Overall	LSF Job Class											
		vst_sj	st_sj	mt_sj	lt_sj	vst_mj	st_mj	mt_mj	lt_mj	vst_lj	st_lj	mt_lj	lt_lj
Oct. 1999													
#jobs	9974	3234	2188	2224	116	707	748	145	27	218	243	73	51
proc demand	75%	1%	9%	13%	3%	1%	8%	8%	1%	1%	13%	11%	4%
mem demand	55%	1%	4%	8%	2%	1%	4%	7%	5%	0%	4%	11%	8%
Nov. 1999													
#jobs	7348	2155	2216	502	139	629	711	208	40	304	333	90	21
proc demand	80%	1%	9%	7%	2%	1%	14%	10%	3%	2%	14%	14%	3%
mem demand	53%	1%	6%	6%	2%	1%	5%	10%	5%	0%	7%	7%	5%
Dec. 1999*													
#jobs	5863	1605	2075	552	114	190	571	189	28	101	275	134	29
proc demand	89%	1%	8%	9%	4%	0%	12%	17%	4%	0%	10%	15%	9%
mem demand	72%	0%	5%	7%	3%	0%	5%	11%	4%	0%	10%	14%	11%
Jan. 2000													
#jobs	7046	2033	2601	550	71	493	585	161	61	130	250	89	22
proc demand	83%	1%	9%	11%	3%	1%	9%	13%	6%	1%	8%	11%	10%
mem demand	74%	0%	6%	7%	3%	1%	5%	10%	6%	0%	11%	10%	16%
Feb. 2000*													
#jobs	8122	3252	2252	465	70	549	697	212	32	173	312	88	20
proc demand	88%	1%	9%	11%	3%	1%	10%	13%	3%	1%	18%	12%	5%
mem demand	74%	1%	7%	7%	3%	1%	5%	10%	5%	0%	10%	12%	15%
March 2000													
#jobs	7034	1668	2666	469	57	450	624	213	70	173	497	122	25
proc demand	85%	1%	11%	9%	3%	1%	11%	15%	4%	1%	14%	13%	3%
mem demand	78%	0%	7%	6%	3%	1%	6%	9%	8%	0%	15%	19%	4%

* High-load months: Dec 1999 and Feb 2000.

turnaround time for two NAS SGI O2K job traces.

Zotkin and Keleher [29] evaluate a priority-backfill with highest priority for the shortest requested runtime, and show that it improves mean slowdown for two job traces by over 50% compared to FCFS-backfill. However, Zhang et al. [28] did not find such improvement. Aida [1] showed that FCFS-firstfit has better mean response time than FCFS-backfill.

Chiang et al. [3] evaluate a non-preemptive policy (ASP) with a preemptive small initial quantum, and show that immediate service improves the performance of ASP. Perkovic and Keleher [20] propose augmenting a variety of backfill policies with a short 'test run' that doesn't preempt executing jobs. The 'test run' along with assuming actual runtime is 10% of the requested run time during backfill, improves average slowdown by 50% for a job trace from a 430-node cluster.

Previous papers that compare spatial equipartitioning to nonpreemptive policies either do not consider memory requirements [15, 17], or assume that each job has a uniform distribution for the minimum number of processors it can execute on to represent the job's memory requirement ([16, 19]).

3. NCSA Workload Characterization

This section provides an overview of the O2K batch workloads used to evaluate the scheduling policies in Section 4, with particular attention to the job characteristics that differ from system workloads reported previously in [6, 9, 4, 5, 22].

Table 4 summarizes the one-month NCSA O2K workloads obtained from system logs during October 1999 through March 2000. The "processor demand" is the product of a job's requested number of processors and actual runtime, summed over all jobs (in a given class) and expressed as a fraction of the total processing time the eight hosts can provide during the month. The memory demand is the equivalent measure for the job memory requests.

Note that the aggregate execution time of the submitted jobs is typically 80-90% of the total available time on the O2K hosts. Furthermore, the large jobs (lj) have high demand (i.e., over 30% of the total available processing time during four of the months). This makes job scheduling challenging, as it is difficult to find free resources for the large jobs on such a highly-loaded sys-

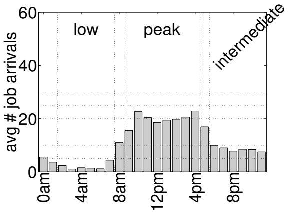


Figure 1. Job Arrivals Each Hour (Mondays-Fridays, Mar 2000)

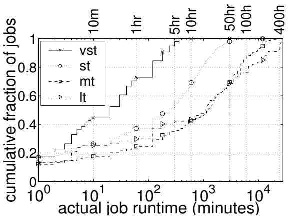


Figure 2. Distribution of Actual Job Runtime (Typical Weekday Peak)

tem, and scheduling several large jobs simultaneously can make it difficult to schedule future arrivals.

There are greater similarities than differences in the O2K workload from month to month. For example, the mt-lj jobs demand 10-15% of the processing time available on the hosts each month, while the lt-mj jobs demand only 1-5% of the available processing cycles.

The number of jobs arriving per hour for each day of the week, illustrated for the weekdays in Figure 1, reveals three periods (peak, intermediate, and low) of relatively stable job arrival rate. (Similar periods are observed for weekend days). These periods are not sharply defined but are fairly similar in each of the six workloads, and are more appropriate delineations of arrival rate homogeneity for the NCSA O2K than the "daytime" and "nighttime" delineations in previous work (e.g., [6]). The scheduling policy evaluations in this paper are performed for all jobs that arrive during the month; thus, the arrival rate behavior is provided simply to characterize the workload that is used to evaluate the policies.

Similar to the workloads previously reported in [6, 9, 4, 5, 22], a large fraction of the jobs are serial and many jobs request power-of-2 processors. In some cases, a higher fraction of the jobs submitted during weekday-low periods, or on weekends, request the maximum possible (i.e., 64) processors, as in [6].

During a typical weekday peak period, a significant fraction of the jobs in each class have short execution time, as shown in Figure 2. For example, 15-20% of the jobs in each class have actual execution time less than one minute. Over 50% of the lt jobs could have requested an st runtime if runtime could be estimated accurately ahead of time. (Similarly, 25-35% of the jobs in each class have actual (maximum) memory usage that is less than 10% of the memory requested.) Note that for each order of magnitude on the x-axis in the figure, a significant fraction of jobs have actual job execution time in that range. Hence, we will use the log scale for showing mean wait vs actual job execution time in Section 4.

Nearly all jobs that complete in under one minute use less than one gigabyte of memory. These (conservative) values will be used for the initial quantum and memory allocation, respectively, for the backfill policies that provide immediate service to arriving jobs. Fine tuning the initial allocations is left for future work.

For most periods, there is a positive correlation between the actual job runtime and the number of processors requested, as in [6, 27]. Jobs that have shorter actual runtime also tend to have lower ratio of memory used to processors requested, as was found in previous workloads [5, 22]. However, during the weekday peak periods (as well as many other periods), serial jobs tend to have higher ratio of peak memory usage divided by requested number of processors than parallel jobs. This differs from previous workloads [5].

During weekday-peak periods there is a *negative* correlation between *requested* memory per-processor and job parallelism. Furthermore, 50% of the serial jobs request more than 512 MB, which is more than the memory per processor on any of the eight O2K batch hosts.

4. Policy Performance Comparisons

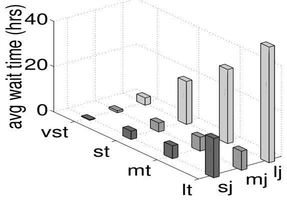
This section provides policy comparisons for the batch workloads that run on the eight O2K hosts. For a given month, the policy is simulated on a trace of the jobs, starting with job arrivals seven days prior to the month of interest, and ending when every job submitted during the month has completed execution. Average, maximum, and 95 percentile waiting times, are reported for all jobs that arrive during the indicated month.

4.1. Experimentally-Tuned NCSA LSF*

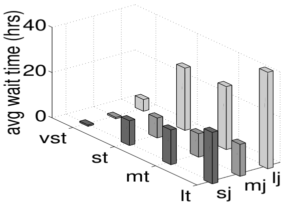
Figure 3 shows the average wait for each job class in four workloads studied. Average wait for most classes is measured in hours or tens of hours, while maximum wait (not shown) is measured in hundreds of hours.

The average and maximum wait measures apply to the jobs that have short actual runtime, which occur in every queue. To illustrate this point, Figure 4 plots the job wait time versus the actual job runtime, for the 5% of the jobs with the largest waiting times during January 2000. Note that the largest waits are scattered over the range of runtime; for example, two jobs with actual runtime less than 1 minute have waiting time over 50 hours.

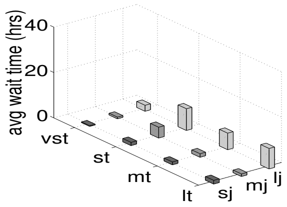
Another key point is that there are differences in NCSA-LSF* performance from month to month even though the overall workload characteristics are similar for each month, as was shown in Table 4. This makes tuning the NCSA-LSF* policy difficult. For example, increasing the resources available to lt jobs might be appropriate for March 2000, but would be inappropriate for November 1999. There isn't an obvious characteristic difference in the two workloads that could have



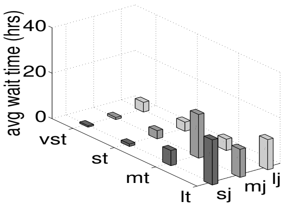
(a) High load (Dec 1999)



(b) High load (Feb 2000)



(c) Typical load (Nov 1999)



(d) Typical load (Mar 2000)

Figure 3. Performance of NCSA-LSF*

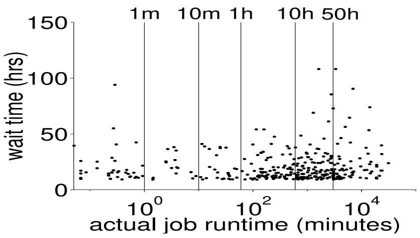


Figure 4. NCSA-LSF* Highest 5% Wait Times vs. Actual Job Run Time (Jan 2000)

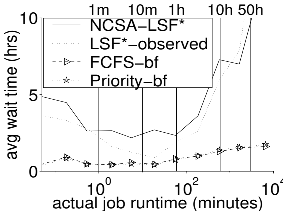
been used to dynamically tune the policy. Extensive experimental tuning on the O2K system, aided by simulations under changes in the host priorities and constraints, has not identified any significant improvements over the NCSA-LSF* configuration evaluated here.

Other policies that are evaluated below have more similar performance from month to month. Since only 4% of the total processing cycles are idle while a special reservation is pending under NCSA-LSF*, the performance sensitivity of NCSA-LSF* appears to be due to (1) having *static* job priority classes and (2) the complex interactions among the set of host priorities and constraints that define the policy.

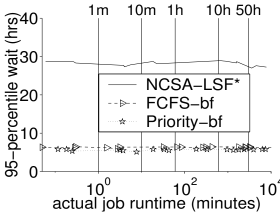
4.2. Backfill Policies

Figure 5 shows the performance of Priority-backfill (as configured in Table 3), FCFS-backfill, and NCSA-LSF*, as a function of actual job execution time.

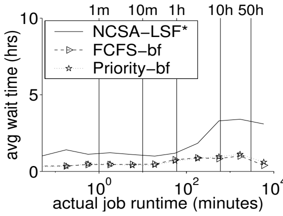
Figure 5(a) shows the measured (i.e., observed) average wait versus actual job runtime for the NCSA-LSF*



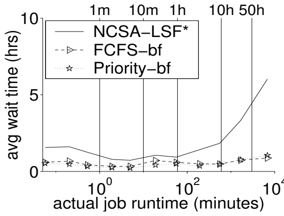
(a) Average Wait (Dec 1999)



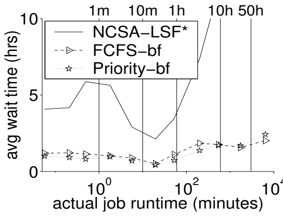
(b) 95-percentile Wait (Dec 1999)



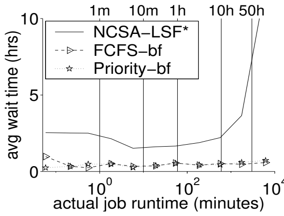
(c) Average Wait (Nov 1999)



(d) Average Wait (Jan 2000)

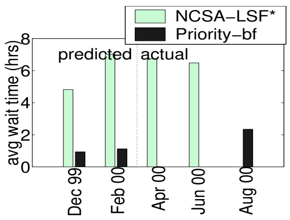


(e) Average Wait (Feb 2000)

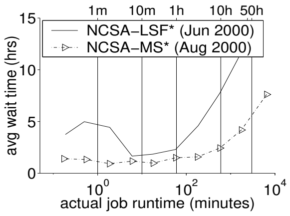


(f) Average Wait (Mar 2000)

Figure 5. Performance of Priority-backfill, FCFS-backfill and NCSA-LSF*



(a) Predicted and Measured Avg wait

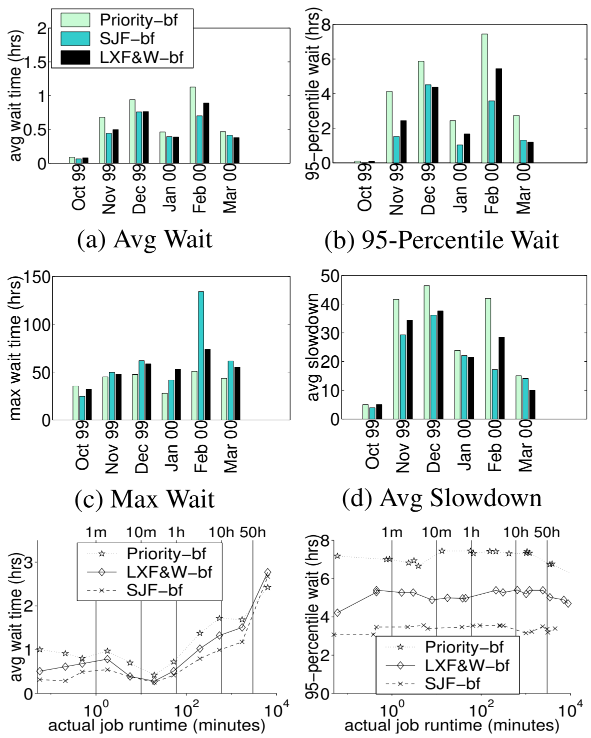


(b) Measured Avg Wait vs. Runtime

Figure 6. Measured Improvement

policy, which shows that the simulation estimates are slightly conservative and qualitatively correct for this policy.[2] Figure 5(b) plots the 95 percentile waiting time for each equal-sized logarithmic range of the execution time. Figures 5(c) through (f) show the variation in

[2]For unknown reasons, the NCSA-LSF* implementation sometimes violates the specified host constraints and scheduling priorities, which causes discrepancies between the simulated and observed waiting times. However, this discrepancy is small when compared with the quantitative differences in performance between NCSA-LSF* and the other scheduling policies evaluated in this paper.



(a) Avg Wait (b) 95-Percentile Wait

(c) Max Wait (d) Avg Slowdown

(e) Avg Wait (Feb 2000) (f) 95-percentile Wait (Feb 2000)

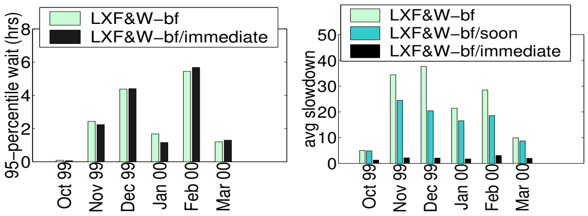
Figure 7. Alternative Priority Measures

scheduling policy performance from month to month.

A key conclusion is that FCFS-backfill and Priority-backfill have similar performance, with each having significantly lower average, maximum (see Figure 7(c)), and 95 percentile waiting time than NCSA-LSF*.

Based on projected performance improvements, the priority-backfill Maui Scheduler has been ported to the NCSA O2K, configured with parameters similar to the parameters used in this study (NCSA-MS*), and put into operation in July 2000. Figure 6 shows that the measured improvement in average wait time (e.g., for August as compared with April and June 2000) is qualitatively similar to the performance improvement estimated by the simulations, which were performed prior to the change in the system scheduler.

We next consider policies that provide higher relative priority to shorter jobs. Figure 7 shows that LXF&W-backfill and SJF-backfill (as defined in Section 2.3) outperform Priority-backfill on every measure provided except maximum wait. The improvement in average slowdown for SJF-backfill compared with FCFS-backfill is only as substantial as in [29] for the February 2000 workload. SJF-backfill outperforms LXF&W-backfill in most cases; however, starvation is potentially a problem in SJF-backfill (as reflected in the maximum wait for the Feb. 2000 workload). Thus LXF&W-backfill, which has significantly better average and 95 percentile wait than Priority-backfill, may be the preferred policy.



(a) 95-Percentile Wait (b) Avg Slowdown

Figure 8. Performance of LXF&W-backfill with 1 Minute of Immediate Service

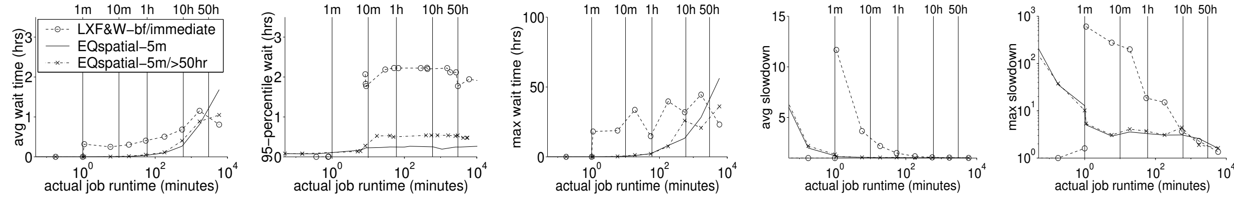
4.3. LXF&W-backfill with Immediate Service

As the maximum wait for the backfill policies is measured in tens of hours even for short jobs, we next consider the LXF&W-backfill policy with limited preemption for immediate service, defined in section 2.4. Figure 8 shows that this policy greatly improves the overall average slowdown, while not adversely affecting the 95 percentile wait time (or the average or maximum wait time, not shown). In fact, immediate service slightly reduces the 95 percentile wait in two workloads, since jobs that terminate prematurely can have among the top 5% of the waiting times when immediate service is not provided. Also included in Figure 8(b) is an LXF&W-backfill/soon policy that gives an arriving job a short quantum as soon as possible without preempting other jobs, as proposed in [20]. The results show that the immediate quantum is significantly more effective.

For each month (not shown) LXF&W-backfill with preemptive immediate service has similar slowdown and wait time as a function of actual job runtime greater than one minute, as LXF&W-backfill without immediate service. The key conclusion is that the immediate service policy considerably improves the turnaround time for the jobs that complete within the quantum without significantly impacting the longer jobs.

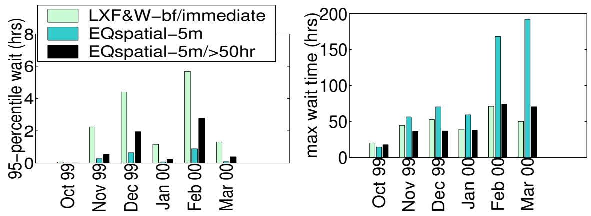
4.4. Performance of EQspatial

Figures 9 and 10 compare the performance of EQspatial with MinInterval equal to 5 minutes (EQspatial-5m) against the LXF&W-backfill with immediate service policy. Also included is EQspatial-5m/>50hr, in which the jobs that have been running for longer than 50 hours are allocated the number of processors they requested for the remainder of their execution time, even if this is larger than the equipartition value, as long as each other job assigned to the host has at least one processor



(a) Avg Wait (b) 95-percentile Wait (c) Max Wait (d) Avg Slowdown (e) Max Slowdown

Figure 9. Performance of EQspatial and LXF&W-backfill with Immediate Service (Nov 1999)



(a) 95-Percentile Wait (b) Max Wait

Figure 10. Summary Performance of EQspatial vs. LXF&W-backfill

to run on. The EQspatial-5m policy has nearly the same performance as EQspatial-0m (not shown).

The policy performance vs job runtime results are similar for the other five O2K workloads (not shown), with slightly larger performance differences between policies during the high-load months (12/99 and 2/00). More than 99% of the arriving jobs in each workload are placed on a host for full execution immediately upon arrival, which indicates that the processor and memory usage on each host is reasonably well-balanced by the heuristic host placement policy defined in Section 2.5.

A job's waiting time under EQspatial is defined as its total time in the system minus its execution time on the number of processors it requested. Linear slowdown is assumed whenever the job executes on fewer processors under EQspatial, although in an adaptive system the job might execute more efficiently on fewer processors; thus, the predicted performance improvement for EQspatial is conservative. The maximum wait time under EQspatial-5m is extremely high for the largest jobs in February and March. Thus, EQspatial-5m/>50hr may be preferred, depending on the desired trade-off between maximum and 95 percentile waiting times.

The immediate service quantum might be tuned to improve the slowdown of LXF&W-backfill for jobs with execution times of one to ten minutes. Thus, the key conclusions are that (1) EQspatial significantly improves the 95 percentile wait, and (2) LXF&W-backfill/immediate is perhaps surprisingly competitive with EQspatial with respect to the other performance measures.

5. Conclusions

This paper has used trace-driven simulation to compare alternative scheduling policies for the challenge workloads that run on the NCSA Origin 2000. The main results are that (1) the current estimated job expansion factor provides a good starvation-free measure for the priority-backfill policies, (2) with an appropriate choice of quantum size and preemption victim, adding immediate service to backfill policies improves performance for very short jobs without significantly degrading performance for other jobs, (3) the EQspatial policy considerably improves the 95 percentile waiting time for jobs that don't complete in the short initial quantum of LXF&W-backfill with immediate service; however, the latter policy is quite competitive with EQspatial with respect to average wait, overall maximum wait, and average slowdown, and (4) it may be important to modify the EQspatial policy to reduce the maximum waiting time of the longest running jobs in the challenge workloads.

Possible directions for future work include a more complete characterization of the NCSA O2K workloads, experimentation with more flexible backfill deadlines, further tuning of the priority measure and host placement policy for the LXF&W-backfill policy, and evaluation of two-level foreground-background scheduling policies for challenge workloads.

Acknowledgments

The authors thank Wayne Hoyenga, Dave Jackson, Michael Pflugmacher, Michael Shapiro, and John Towns for many useful discussions about the NCSA LSF* scheduler, the Maui Scheduler, and the goals of production job scheduling for supercomputing workloads.

References

[1] K. Aida. Effect of job size characteristics on job scheduling performance. *Proc. 6th Workshop on Job Scheduling Strategies for Parallel Processing*, Cancun, May 2000, pp. 1-17. Lecture Notes in Comp. Sci. Vol. 1911, Springer-Verlag.

[2] R. Blumofe, C. Joerg, B. Kuszmaul, C. Leiserson, K. Randall, and Y. Zhou. Cilk: An efficient multithreaded runtime system. *Proc. 5th ACM SIGPLAN Symp. on Prin-*

ciples and Practice of Parallel Programming, Santa Barbara, June 1995, pp. 207-216.

[3] S.-H. Chiang, R. Mansharamani, and M. Vernon. Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. *Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Nashville, May 1994, pp. 33-44.

[4] D. G. Feitelson. Packing schemes for gang scheduling. *Proc. 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, Honolulu, Apr. 1996, pp. 89-110. Lecture Notes in Comp. Sci. Vol. 1162, Springer-Verlag.

[5] D. G. Feitelson. Memory usage in the LANL CM-5 workload. *Proc. 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, Geneva, Apr. 1997, pp. 78-94. Lecture Notes in Comp. Sci. Vol. 1291, Springer-Verlag.

[6] D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. *Proc. 1st Workshop on Job Scheduling Strategies for Parallel Processing*, Santa Barbara, Apr. 1995, pp. 337-360. Lecture Notes in Comp. Sci. Vol. 949, Springer-Verlag.

[7] D. G. Feitelson and A. M. Weil. Utilization and predictability in scheduling the IBM SP2 with backfilling. *Proc. 12th Int'l. Parallel Processing Symp.*, Orlando, Mar. 1998, pp. 542-546.

[8] M. Frigo, C. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded language. *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, Montreal, June 1998, pp. 212-223.

[9] S. Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. *Proc. 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, Honolulu, Apr. 1996, pp. 27-40. Lecture Notes in Comp. Sci. Vol. 1162, Springer-Verlag.

[10] *Intel Corp., iPSC/860 Multi-User Accounting, Control and Scheduling Utilities Manual. Order number 312261-002*, May 1992.

[11] J. P. Jones and B. Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. *Proc. 5th Workshop on Job Scheduling Strategies for Parallel Processing*, San Juan, Apr. 1999, pp. 1-16. Lecture Notes in Comp. Sci. Vol. 1659, Springer-Verlag.

[12] W. Leinberger, G. Karypis, and V. Kumar. Job scheduling in the presence of multiple resource requirements. *Proc. 1999 ACM/IEEE Supercomputing Conf.*, Portland, Nov. 1999.

[13] S. T. Leutenneger and M. K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. *Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Boulder, May 1990, pp. 226-236.

[14] D. Lifka. The ANL/IBM SP scheduling system. *Proc. 1st Workshop on Job Scheduling Strategies for Parallel Processing*, Santa Barbara, Mar. 1995, pp. 295-303. Lecture Notes in Comp. Sci. Vol. 949, Springer-Verlag.

[15] C. McCann, R. Vaswani, and J. Zahorjan. Dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Trans. on Computer Systems*, 11(2):146–178, May 1993.

[16] C. McCann and J. Zahorjan. Scheduling memory constrained jobs on distributed memory parallel computers. *Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Ottawa, May 1995, pp. 208-219.

[17] V. K. Naik, S. K. Setia, and M. S. Squillante. Performance analysis of job scheduling policies in parallel supercomputing environments. *Proc. 1993 ACM/IEEE Supercomputing Conf.*, Portland, Nov. 1993, pp. 824-833.

[18] *Running Jobs on NCSA Origin2000*. http://www.ncsa.uiuc.edu/SCD/Hardware/Origin2000/Doc/Jobs.html.

[19] E. W. Parsons and K. C. Sevcik. Implementing multiprocessor scheduling disciplines. *Proc. 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, Geneva, Apr. 1997, pp. 166-192.

[20] D. Perkovic and P. J. Keleher. Randomization, speculation, and adaptation in batch schedulers. *Proc. 2000 ACM/IEEE Supercomputing Conf.*, Dallas, Nov. 2000.

[21] A. Scherer, H. Lu, T. Gross, and W. Zwaenepoel. Transparent adaptive parallelism on NOWs using OpenMP. *Proc. 7th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, Atlanta, May 1999, pp. 96-106.

[22] S. K. Setia, M. S. Squillante, and V. K. Naik. The impact of job memory requirements on gang-scheduling performance. *Performance Evaluation Review*, 26(4):30–39, Mar. 1999.

[23] J. Skovira, W. Chan, H. Zhou, and K. Lifka. The EASY-Loadleveler API Project. *Proc. 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, Honolulu, Apr. 1996, pp. 41-47. Lecture Notes in Comp. Sci. Vol. 1162, Springer-Verlag.

[24] Q. Snell, M. Clement, D. Jackson, and C. Gregory. The performance impact of advance reservation metascheduling. *Proc. 6th Workshop on Job Scheduling Strategies for Parallel Processing*, Cancun, May 2000, pp. 137-153. Lecture Notes in Comp. Sci. Vol. 1911, Springer-Verlag.

[25] D. Talby and D. G. Feitelson. Supporting priorities and improving utilization of the IBM SP2 scheduler using slack-based backfilling. *Proc. 13th Int'l. Parallel Processing Symp.*, San Juan, Apr. 1999, pp. 513-517.

[26] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. *Proc. 12th ACM Symp. on Operating Systems Principles*, Litchfield Pk., Dec. 1989, pp. 159-166.

[27] K. Windisch, V. Lo, D. Feitelson, B. Nitzberg, and R. Moore. A comparison of workload traces from two production parallel machines. *Proc. 6th Symp. on the Frontiers of Massively Parallel Computation*, Oct. 1996.

[28] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam. Improving parallel job scheduling by combining gang scheduling and backfilling techniques. *Proc. Int'l. Parallel and Distributed Processing Symp.*, Cancun, May 2000, pp. 133-142.

[29] D. Zotkin and P. Keleher. Job-length estimation and performance in backfilling schedulers. *8th IEEE Int'l Symp. on High Performance Distributed Computing*, Redondo Beach, Aug. 1999, pp. 236-243.