The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance

Su-Hui Chiang Andrea Arpaci-Dusseau Mary K. Vernon

Computer Sciences Department
University of Wisconsin-Madison
{suhui, dusseau, vernon}@cs.wisc.edu

Abstract

This paper addresses the question of whether more accurate requested runtimes can significantly improve high-performance priority backfill policies, for production workloads running on leading edge systems such as the 1500 Origin 2000 system at NCSA or the new TeraGRID. This question has been studied previously for FCFS-backfill using a limited set of performance metrics. The new results for higher performance backfill policies, heavier system load, and for a broader range of performance metrics show that more accurate requested runtimes have much greater potential to improve system performance than suggested in previous results. Furthermore, the results show that (a) using user test runs to improve requested run time estimates can achieve most of the benefit, and (b) users who provide more accurate requested runtimes can expect improved performance, even if other users do not provide the more accurate requests.

1 Introduction

Many state-of-the-art production parallel job schedulers are non-preemptive and use a requested runtime for each job to make scheduling decisions. For example, the EASY Scheduler for the SP2 [Lif95, SCZL96] implements the First-Come First-Served (FCFS)-backfill policy, in which the requested runtime is used to determine whether a job is short enough to be backfilled on a subset of the nodes during a period when those nodes would otherwise be idle. The more recent Maui Scheduler ported to the NCSA Origin 2000 (O2K) [NCSa] and the large NCSA Linux Cluster [NCSb] implements a parameterized priority-backfill scheduler that uses the requested runtime to determine job priority as well as whether it can be backfilled. Recent work [CV01a] has shown that the priority-backfill policy on the O2K has similar performance to FCFS-backfill, but that modifying the policy priority parameters to favor short jobs, in a manner analogous to some of the high performance scheduling policies originally developed for uniprocessor systems, provides superior average wait, 95th-percentile wait, and average slowdown, as well as similar maximum wait time as for FCFS-backfill, for the large production workloads that run on the O2K. Thus, the requested runtimes are needed not only for backfill decisions, but also to enable favoring short jobs in a way that improves service for nearly all jobs.

The advantages of nonpreemptive scheduling policies include low scheduling overhead and relatively easy implementation. Furthermore, simulation results for the O2K job traces show that non-preemptive backfill policies that give priority to jobs with short requested runtime can have performance that is reasonably competitive with high performance (but more difficult to implement) preemptive policies such as gang scheduling or spatial equi-partitioning [CV01a]. This relatively high performance is achieved in spite of the fact user estimated runtimes are often highly inaccurate [FW98, MF01, CB01, CV01b]. For example, analysis of the NCSA O2K logs shows that 30% of the jobs that request 200 or more hours of runtime terminate in under ten hours [CV01b].

The key open question addressed in this paper is whether the high performance backfill policies could be further improved with more accurate requested runtimes. Several previous simulation studies of FCFS-backfill show that more accurate requested runtime has only minimal impact on the average wait time or average slowdown [FW98, STF99, ZFMS00, ZFMS01, MF01]. We briefly revisit the question for FCFS-backfill, using workloads from recent months on the O2K that have significantly heavier system load (e.g., up to 100% cpu demand), and using a more complete set of performance measures. More importantly, we investigate the question of whether more accurate requested runtimes can sig-

nificantly improve the higher-performance backfill policies that use requested runtimes to favor short jobs. We evaluate these questions using complete workload traces from the NCSA O2K and consider not only average wait time and average slowdown as in previous studies, but also the maximum and 95th-percentile wait time. Each of these measures is obtained as a function of actual job runtime and as a function of the number of processors, to determine how performance varies for jobs with different actual runtime or jobs that use large or small numbers of processors.

To study the above key question, two design issues that relate to preventing starvation in backfill policies that favor short jobs require further investigation. As discussed in Sections 2.3 and 3.1, preventing starvation was not fully addressed in previous work. In particular, the problem is more significant for the heavier system load in recent months on the O2K. The first design issue is the reservation policy; that is, how many jobs are given reservations and whether the reservations are *fixed* or they *dynamically* change for dynamic priority functions. The second design issue is the relative weight in the priority function for requested job runtime and current job wait time. A more complete analysis of these issues is needed in order to set these parameters properly for studying the potential improvement of more accurate requested runtimes.

The key results in the paper are as follows:

- For a set of high performance backfill policies that favor short jobs (i.e., LXF&W-, SJF&W-, $LX^{1/2}$F&W-, and $ST^{1/2}$F&W-backfill), more accurate requested runtimes dramatically improve the average slowdown, greatly improve the average and maximum wait for short jobs without increasing the average wait for long jobs, and greatly improves the 95th-percentile wait for all jobs.

- Nearly all of this improvement is realized even if requested runtime is up to a factor of two times the actual runtime. Furthermore, (a) much of the improvement can be achieved even if only 50% - 80% of the jobs provide the approximately accurate runtime requests, and (b) test runs to more accurately estimate requested runtime do not reduce the performance gains.

Additional contributions of the paper include:

- A summary of the very recent workloads (October 2000 - July 2001) on the O2K, including several months with heavier processor and memory demand than workloads used previously to design scheduling policies. Note that heavier system load can have a significant impact on the magnitude of the performance differences among alternative scheduling policies.

- For the NCSA O2K architecture and workload, using a small number of reservations (2 to 4) outperforms a single reservation, but a larger number of reservations results in poor performance during months with exceptionally heavy load.

- Compared to the highest performance previous backfill policy, namely LXF&W-backfill with single reservation, LXF&W-backfill with two to four reservations or two proposed new priority backfill policies ($LX^{1/2}F$&W and $ST^{1/2}$F&W-backfill), with two reservations, significantly improve the maximum wait time.

- In systems where only 60% of the jobs provide approximately accurate requested runtimes, the jobs with improved runtime requests have nearly the same wait time statistics as if 100% of the jobs provided approximately accurate requested runtimes, thus providing a significant incentive for individual users to improve the accuracy of their runtime requests.

The remainder of this paper is organized as follows. Section 2 provides background on the system and workloads used in this study, and on related previous work. Section 3 provides results concerning the impact of reservation policies and the relative priority weight between job requested runtime and current job wait time for backfill policies. Section 4 studies the potential benefit of using more accurate requested runtimes in priority backfill policies. Section 5 shows whether the benefit of more accurate requested runtimes can be realized by using test runs to estimate the more accurate requested runtimes. Section 6 provides the conclusions of this work.

2 Background

2.1 The NCSA Origin 2000 System

The NCSA O2K is a large production parallel system that provides 960 processors and 336 GB of memory for processing batch jobs that do not request a dedicated host. The processors are partitioned into eight hosts, each of which has 64 or 128 processors and 32 or 64 GB of memory. The jobs are scheduled using a ported version of the Maui Scheduler that implements a backfill policy with a parameterized priority function, and evicts a job if it has run one hour longer than its requested runtime. More detail about the system and scheduler configuration can be found in [NCSa, CV01a].

Table 1. Summary of Monthly NCSA O2K Workloads

Month	Overall	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
Oct00													
#jobs	6552	1342	2491	576	276	248	624	240	50	57	362	208	78
proc demand	82%	1%	11%	9%	7%	0%	10%	11%	2%	0%	14%	13%	4%
mem demand	81%	0%	6%	7%	9%	0%	6%	6%	2%	0%	6%	18%	20%
Nov00													
#jobs	6257	1719	2279	417	60	287	499	186	16	146	513	110	25
proc demand	85%	1%	10%	8%	3%	1%	9%	12%	3%	1%	21%	13%	3%
mem demand	61%	1%	5%	5%	2%	0%	5%	6%	1%	0%	11%	11%	14%
Dec00													
#jobs	4782	1114	2056	563	164	100	203	215	59	45	135	113	15
proc demand	89%	0%	10%	10%	9%	0%	4%	18%	4%	0%	8%	13%	12%
mem demand	63%	0%	6%	8%	5%	0%	2%	10%	6%	0%	3%	13%	9%
Jan01													
#jobs	4837	945	2000	649	164	185	267	158	151	37	170	97	14
proc demand	*102%	1%	9%	13%	7%	0%	4%	18%	10%	0%	9%	15%	14%
mem demand	76%	0%	6%	8%	5%	0%	3%	10%	14%	0%	6%	9%	14%
Feb01													
#jobs	6784	2328	2264	479	180	357	333	119	63	281	219	91	70
proc demand	*97%	1%	9%	9%	8%	0%	6%	13%	7%	0%	11%	12%	22%
mem demand	*87%	1%	6%	5%	5%	0%	4%	8%	8%	0%	8%	14%	28%
Mar01													
#jobs	5929	1915	1869	644	221	372	290	140	50	78	224	87	39
proc demand	*100%	1%	12%	11%	10%	1%	4%	10%	5%	0%	11%	18%	17%
mem demand	*92%	1%	7%	8%	9%	0%	3%	6%	8%	0%	9%	11%	30%
Apr01													
#jobs	6206	2106	2304	643	202	235	238	70	78	47	159	90	34
proc demand	78%	1%	13%	12%	9%	0%	5%	8%	5%	0%	8%	8%	8%
mem demand	77%	1%	6%	7%	7%	0%	3%	3%	9%	0%	9%	8%	25%
May01													
#jobs	6573	2220	2012	611	191	364	355	115	96	214	246	104	45
proc demand	*99%	2%	12%	10%	10%	1%	6%	10%	12%	1%	8%	19%	9%
mem demand	*92%	1%	5%	9%	6%	0%	3%	4%	14%	1%	10%	18%	20%
Jun01													
#jobs	6364	2076	2317	690	82	271	346	113	86	91	189	84	19
proc demand	86%	2%	12%	15%	6%	1%	8%	10%	9%	1%	9%	10%	4%
mem demand	75%	1%	7%	11%	4%	0%	4%	4%	12%	0%	9%	17%	6%
Jul01													
#jobs	5705	1363	2070	664	136	243	415	177	111	102	263	131	30
proc demand	89%	1%	12%	15%	5%	1%	7%	14%	6%	1%	12%	10%	5%
mem demand	81%	1%	8%	9%	5%	0%	4%	7%	13%	0%	9%	18%	8%

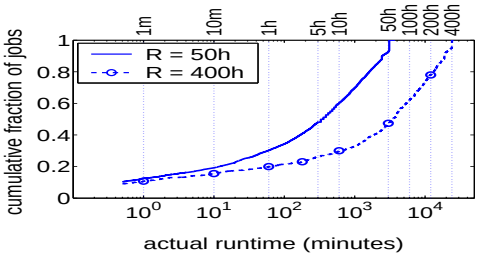
'*' indicates exceptionally high load.

Job Class	vst	st	mt	lt	sj	mj	lj
	$\leq$5hrs	[5, 50) hrs	[50, 200) hrs	[200, 400) hrs	P $\leq$ 8 M $\leq$ 2GB	P $\leq$ 16 M $\leq$ 4GB	P $\leq$ 64 M $\leq$ 25GB

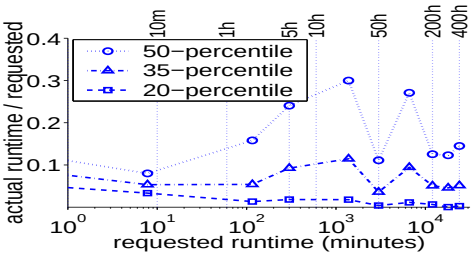
Time class: requested runtime
P = requested processors
M = requested memory

2.2 Workloads

In this study, we evaluate scheduling policy performance using simulations with ten different one-month job traces (obtained during October 2000 - July 2001) from the O2K. Three of these months (October - December 2000) were fully characterized in [CV01b]. The load during each month is summarized in Table 1. The overall processing demand ("proc demand") per month is based on the actual runtimes of the jobs submitted that month, and is expressed as a percentage of the total available processor-minutes for the month. Similarly the overall memory demand per month is the sum of the memory used by jobs submitted during the month, expressed as



(a) Distribution of Actual Runtime



(b) Distributions of Actual/Requested Runtime

Figure 1. Inaccuracy in the Requested Runtime (R) for O2K Workloads
(January 2001 - July 2001)

a fraction of the memory available that month. Percentages of processor and memory demand are also given for each job class, where job class is defined by the requested runtime and requested processor and memory resources, as defined below the table.

There are two key differences in the traces summarized in the table compared to those considered previously [CV01a, CV01b]. First, the actual job runtime in these traces includes the initial data *setup time*, during which the job occupies its requested resources (i.e., processors and memory) but it has not yet started its computation. The data setup time adds negligible ($\leq$ 1%) total cpu and memory load each month, but it is significant (e.g., 10 hours) for some jobs. Second, the traces include four months (January - March and May 2001) that have exceptionally high demand for processor resources (i.e., very close to 100%), and three of those months (February, March, and May 2001) also have exceptionally high memory demand ($>$ 90%). The other three months in 2001 (April, June, and July) have cpu demand (80 - 90%) and memory demand (70 - 80%) that is typical in earlier O2K workloads [CV01a]. Results in the paper will be shown for three of the heavy load months (January - March 2001), one of the months that follows a heavy load month (April 2001) and one typical month (July 2001).

Other characteristics of the workloads during 2001 are similar to the previous months in 2000. In particular, there is an approximately similar mix of job classes (i.e., sizes) from month to month (as shown in Table 1), and there is a large discrepancy between requested and actual runtimes, as illustrated in Figure 1. For jobs submitted during January - July 2001, Figure 1(a) plots the distribution of actual runtime for jobs that requested 50 hours, and jobs that requested 400 hours. Figure 1(b) plots points in the distribution (i.e., the 20th-, 35th-, and 50th-percentile) of the ratio of the actual runtime divided by the requested runtime as a function of requested runtime. These results show that the requested runtimes can be very inaccurate. For example, Figure 1(a) shows that almost 30% of the jobs that request 400 hours of runtime actually terminate in under 10 hours, and another 10% have actual runtime between 10 and 50 hours. Furthermore, for requested runtime of 50 hours or 400 hours, approximately 10% of the jobs terminate in under 1 minute. Figure 1(b) shows that for any range of requested runtimes greater than one minute, 35% of the jobs use less than 10% of their requested runtime (i.e., requested runtime is a factor of 10 or more times the actual runtime), and another 15% of the jobs have actual runtime between 10% and 30% of the requested runtime. Similarly large discrepancies between requested and actual runtimes have also recently been reported for many SP2 traces [MF01, CB01]. In particular, the results by Cirne and Berman [CB01] show that for four SP2 traces, 50-60% of the jobs use under 20% of the requested runtime, which is very similar to the results for the NCSA O2K workloads reviewed above.

2.3 Previous Work

In this section, we review previous work on three topics: alternative priority functions for backfill policies, the impact of reservation policies, and the impact of using more accurate requested runtimes on backfill policies.

The most comprehensive previous comparison of alternative priority backfill policies [CV01a] shows that, among the priority functions defined in Table 2, the LXF&W-backfill policy that gives priority to short jobs while taking current job waiting time into account outperforms FCFS-backfill, whereas SJF-backfill has the problem of starvation (i.e., large maximum wait) under high load. This previous paper also provides a review of earlier papers [ZK99, ZFMS00, PK00] that compare the SJF-backfill and FCFS-backfill policies.

Reservation policies concern (a) the number of jobs waiting in the queue that are given (earliest possible)

Table 2. Priority Functions of Previous Backfill Policies

Priority Weight				Job Measure
FCFS	SJF	LXF	LXF&W	
1	0	0	$w(0.02)$	current wait time, J_w, in hours
0	1	0	0	inverse of requested runtime ($\frac{1}{R}$)
0	0	1	1	current job expansion factor ($\frac{J_w + R\ in\ hours}{R\ in\ hours}$)

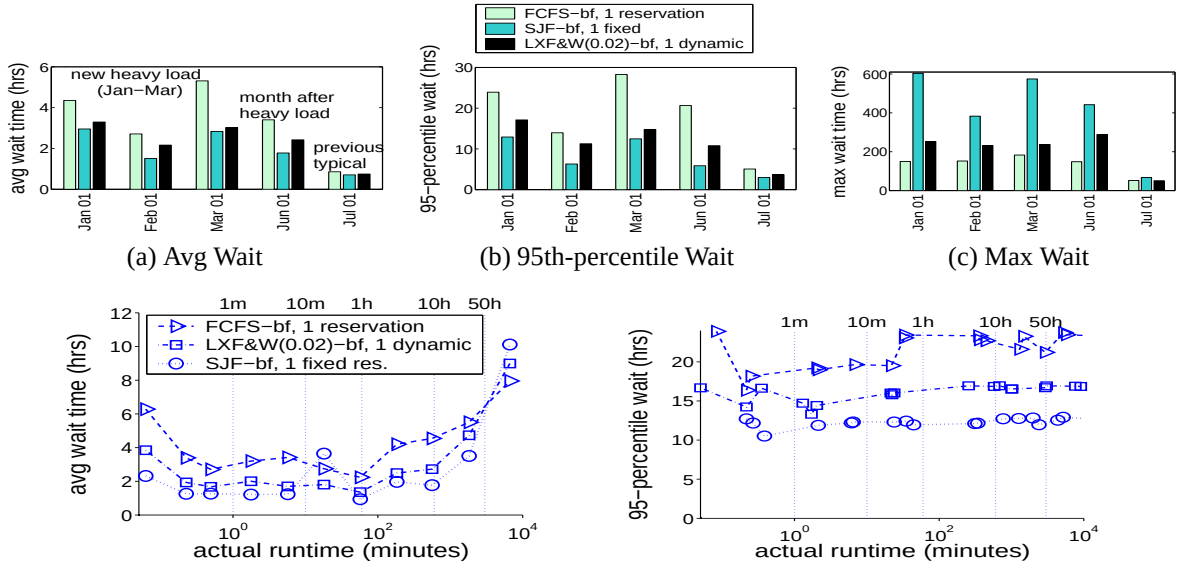
reservations for processor and memory resources, and (b) whether the reservations are dynamic or fixed. Previous results by Feitelson and Weil [FW98] show that, for FCFS-backfill and a set of SP workloads, average slowdown is similar when only one (i.e., the oldest) waiting job has a reservation or when all jobs have a reservation. In more recent work [MF01] they find similar results for further SP2 workloads, for workloads from other systems, and for many synthetic workloads, but they find that for many other SP2 monthly workloads, a single reservation significantly improves the average slowdown (by $>$ 40%) and average response time (by $>$ 30%). Several papers study backfill policies that have reservations for all waiting jobs [STF99, PK00, ZFMS00], while still other papers evaluate backfill policies that give reservations to only one waiting job [Lif95, SCZL96, Gib97, ZK99, CV01a].

With dynamic reservations, job reservations and the ordering of job reservations can change when a new job arrives, or if the relative priorities of the waiting jobs change with time. For example, in SJF-backfill with a single dynamic reservation, an arriving job will preempt the reservation held by a longer job. With *fixed* reservations, in contrast, once a job is given a reservation, it may be given an earlier reservation when another job terminates earlier than its requested runtime, but recomputed job reservations will have the same order as the existing reservations, even if a job that has no reservation or a later reservation attains a higher priority. A single fixed reservation is used to reduce starvation in SJF-backfill in [CV01a]. In [PK00], each job is given a reservation when it arrives. They compare a form of dynamic ("no guarantee") reservations, in which reservations are only recomputed if and when a job finishes early but the recomputed reservations are done in priority (i.e., FCFS or SJF) order, against "guaranteed reservations", in which job reservations are recomputed only in the same order as the existing reservations. They find that the dynamic reservations have lower average slowdown and average wait than guaranteed reservations for the priority backfill policies studied, including SJF-backfill. Results in this paper include the maximum wait measure, which shows that fixed reservations improve SJF-backfill; otherwise the results in this paper are consistent with their results.

Two previous papers show that perfectly accurate requested runtimes for FCFS-backfill improve the average slowdown by no more than 30% [MF01] and the average wait time by only 10 - 20% [STF99], compared to using the highly inaccurate requested runtimes given in SP traces. Several papers [FW98, ZK99, ZFMS00, ZFMS01, MF01] compare the performance of various models of requested runtimes against perfectly accurate runtime requests. For a given actual runtime, they model the requested runtime overestimation (i.e., requested runtime - actual runtime) as a factor times the actual runtime, where the factor is drawn from a uniform distribution between 0 and a fixed parameter C. The paper [ZK99] also includes a model where the factor is deterministic. The results in those papers show that even for C as large as 300 [FW98, MF01] (or 50 [ZK99] or 10 [ZFMS00, ZFMS01]), the average slowdown or average wait is similar to, or even slightly better than that of C = 0. Additional results in [MF01] show that multiplying the user requested runtimes by two slightly improves on average slowdown and response time for SP workloads and FCFS-backfill. These papers conclude that there is no benefit of using accurate requested runtimes for FCFS-backfill and SJF-backfill. We note that for large C (or when multiply requested runtime by two), jobs with long runtimes can have very large runtime overestimation, which leaves larger holes for backfilling shorter jobs. As a result, average slowdown and average wait may be lower, as reported in these previous papers. On the other hand, these systems may have poorer maximum wait, which was not studied in any of these previous papers.

3 Reducing Starvation in Systems that Favor Short Jobs

Backfill policies that favor short jobs have the potential problem of poor maximum wait for long jobs. Mechanisms for reducing the maximum wait include using a larger number of reservations, and increasing the priority weight on the current job wait time. On the other hand, either of these mechanisms may increase the aver-



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

(d) Avg Wait vs. Actual Runtime (January 2001)

(e) 95th-percentile Wait vs. Actual Runtime (January 2001)

Figure 2. Performance Comparisons of Previous Backfill Policies

age and 95th-percentile wait for all jobs. The goal of this section is to provide a more comprehensive evaluation of the trade-offs in the wait time measures for different reservation policies and for alternative priority functions that give different relative weight to the current job waiting time. In evaluating the tradeoffs for each policy, we seek to achieve a maximum wait that is no greater than the maximum wait in FCFS-backfill, while reducing the average and 95th-percentile wait time as much as possible.

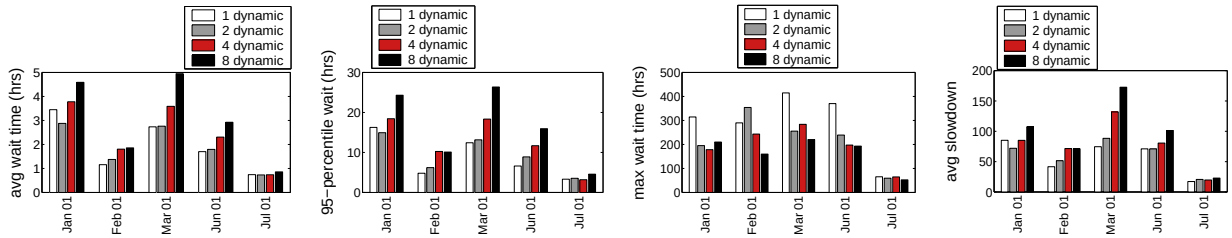
In this section, and in the remainder of the paper, policy comparisons will be shown for five representative workloads. These workloads include (a) three of the four new *exceptionally heavy load* months (i.e., January - March 2001), which are the most important months for policy optimization, (b) June 2001, which has similar policy performance at April 2001 since both of these months follow an exceptionally heavy load month, and (c) July 2001 which has a typical load and policy performance similar to October - December 2000 and other previously studied workloads. The other new exceptionally heavy load month (May 2001) has somewhat lower wait time statistics for each policy than the other three exceptionally heavy months, due to a larger number of short jobs submitted that month.

Section 3.1 evaluates previous backfill policies to show that starvation is a more significant problem for the new exceptionally heavy load workloads on the NCSA O2K. Section 3.2 evaluates several alternative reservation policies. Section 3.3 evaluates several new priority functions with different relative weights on the current job waiting time and compares the best new priority backfill policies against FCFS-backfill.

3.1 Re-evaluation of Previous Policies

In this section, we use the recent O2K workloads to re-evaluate the FCFS-backfill, SJF-backfill, and LXF&W-backfill policies (defined in Table 2). Note that both SJF-backfill and LXF&W-backfill favor short jobs, but LXF&W-backfill also has a priority weight for current job wait time. The reservation policies in these previously defined schedulers are: FCFS-backfill uses one reservation, LXF&W-backfill uses one dynamic reservation, and SJF-backfill uses one fixed reservation (to reduce the maximum wait).

Figure 2 compares the three policies, showing (a) overall average wait, (b) 95th-percentile wait, (c) maximum wait, and (d)-(e) average and 95th-percentile wait, respectively, as a function of actual runtime, during a representative heavy load month. Comparisons in previous work [CV01a] are similar to the comparisons for the July 2001 workload in figures (a) - (c). Conclusions for the new heavy load months that are similar to previous work are that (1) both SJF-backfill and LXF&W-backfill have significantly lower 95th-percentile wait (for all ranges of actual runtime) than that of FCFS-backfill, and (2) SJF-backfill has the problem of poor maximum wait for many of the workloads, as shown in figure (c). Conclu-



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

Figure 3. Impact of Number of Reservations on LXF-backfill
(Dynamic Reservations)

sions for the new heavy load months that *differ* from the results in previous work (and also differ from the results for July 2001), are that (1) both LXF&W-backfill and SJF-backfill have considerably lower *average wait* than FCFS-backfill (for most ranges of actual runtimes), and (2) LXF&W-backfill also has significantly *higher maximum wait* than FCFS-backfill.

The starvation problems of LXF&W-backfill and SJF-backfill are addressed in the next two sections. The questions are (1) whether multiple reservations can improve the performance, particularly the maximum wait, of SJF-backfill and LXF&W-backfill, (2) whether fixed reservations can improve the maximum wait for LXF&W-backfill, and (3) whether new priority functions, such as adding a priority weight for current waiting time to the SJF-backfill priority function, or more generally whether new relative priority weights between requested job runtime and current job wait time, can improve on the previous policy priority functions. Section 3.2 addresses the first two questions. Section 3.3 studies the third question.

3.2 New Reservation Policy Comparisons

This section studies the impact of reservation policies, i.e., the number of reservations and dynamic versus fixed reservations, on backfill policies. We use three simple priority backfill policies to evaluate the reservation policies, namely: FCFS-backfill, SJF-backfill, and LXF-backfill (all with weight on current waiting time equal to zero). Adding weights for current waiting time will be studied in the next section.

For each of the three policies, we evaluated the performance for the following numbers of reservations: 1, 2, 4, 6, 8, 12, and 16. For the LXF-backfill and SJF-backfill policies that have dynamic priority functions, we evaluate the performance of both dynamic and fixed reservations, each over the entire range of number of reservations.

Figure 3 shows the performance of LXF-backfill with up to eight dynamic reservations. Twelve and sixteen reservations have similar or worse performance as that of eight reservations. The impact of the number of reservations is similar for FCFS-backfill and SJF-backfill (not shown to conserve space), except that four reservations performs slightly better for SJF-backfill. The key conclusion is that using a few (i.e., 2-4) reservations significantly reduces the maximum wait time compared to using a single reservation, by about 30% for most of the new heavy load months, as shown in Figure 3(c). Furthermore, using more than a couple of reservations usually makes minimal further improvement for the maximum wait, yet significantly increases the average and 95th-percentile wait, as shown in Figure 3(a) and (b), for the new heavy load workloads or immediately following the heavy load months. For months with a typical O2K load (e.g., July 2001), the impact of reservation policies on backfill policies is minimal, which agrees with previous results for the average slowdown of FCFS-backfill in [FW98].

Other results omitted to conserve space show that fixed and dynamic reservations (with 2-4 reservations) have similar performance for LXF-backfill and the policies developed in the next section. However, for SJF-backfill, dynamic reservations has higher maximum wait than fixed reservations because (particularly under heavy load) dynamic reservations for jobs with long requested runtimes are often usurped by newly arriving jobs that have short requested runtimes.

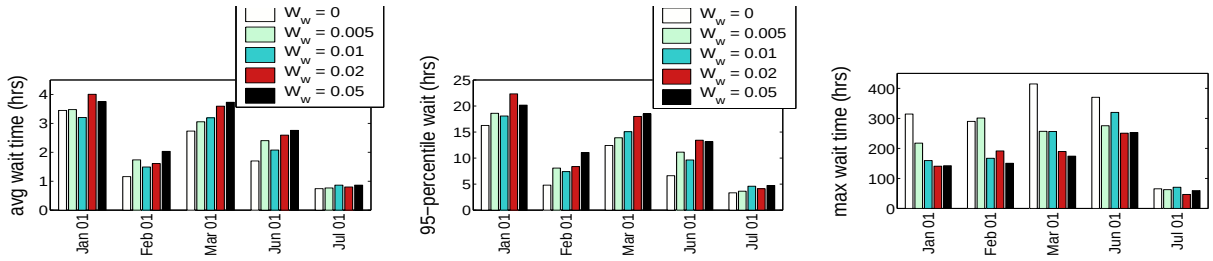
3.3 New Priority Functions

In this section, we propose three alternative new priority functions and study the impact of alternative priority weights for current job wait time together with the impact of reservation policies. The next section will compare the best new priority functions against the previous backfill policies.

Table 3. Weights for New Priority Backfill Policies

Priority Weight			Job Measure
SJF&W	**S$T^{1/2}$F&W**	**L$X^{1/2}$F&W**	
w(0.05-0.2)	w(0.01-0.05)	w(0.01-0.02)	current wait time, J_w, in hours
1	0	0	$J_r = \frac{400^*}{R \text{ in hours}}$
0	1	0	$\sqrt{J_r}$
0	0	1	$\sqrt{J_x}$, where $J_x = \frac{J_w + R \text{ in hours}}{R \text{ in hours}}$

(* The maximum O2K requested runtime, R, is 400 hours.)



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

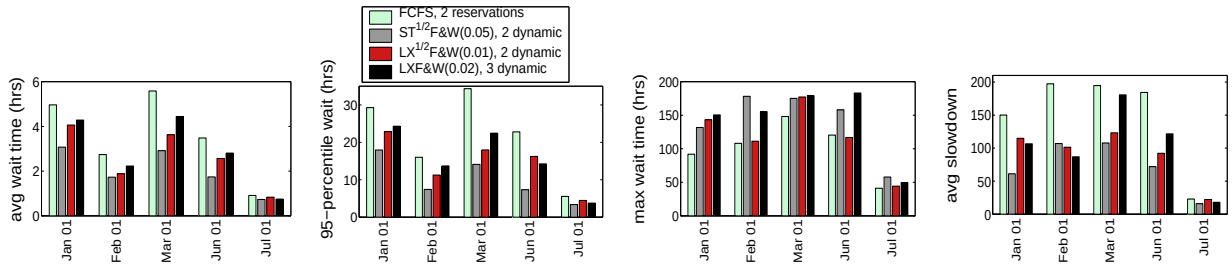
Figure 4. Performance Comparisons of Job Wait Priority Values (W_w) for L$X^{1/2}$F&W-backfill
(One Dynamic Reservation)

The new (dynamic) priority functions are defined in Table 3. The SJF&W priority extends the previous SJF function with a weight for the current job wait time. Note that in the J_r metric for the SJF&W and S$T^{1/2}$F&W priority functions, the inverse of requested runtime ($1/R$) is normalized to the maximum allowed requested runtime (i.e., 400 hours). The S$T^{1/2}$F&W the L$X^{1/2}$F&W priority functions are designed to reduce discrimination against longer requested runtimes by applying a square root to the job metric that includes requested runtime. We find that S$T^{1/2}$F&W-backfill and L$X^{1/2}$F&W-backfill (with an appropriate priority weight W_w) only very slightly outperform SJF&W-backfill and LXF&W-backfill, as will be shown below. Thus, further alternatives for discriminating against longer requested runtimes are not likely to lead to any significant improvement.

Figure 4 shows the impact of alternative priority weights for current wait time (W_w) on L$X^{1/2}$F&W-backfill with one dynamic reservation. Results are similar for 2-4 reservations, and for the other two new priority functions, as well as for LXF&W (not shown). The Figure shows that average and 95th-percentile wait are not highly sensitive to W_w in the range of 0.005 - 0.05, and that during heavy load months, this range of W_w values significantly reduces the maximum wait (by 30-50%) compared to W_w = 0. Larger values of W_w (e.g., W_w = 1) significantly increase the average and 95th-percentile wait time, with only small improvements in the maximum wait (not shown).

Similar to the previous section, we find that using a small number of reservations (i.e., two or three) outperforms a single reservation for each of the alternative new priority functions.

Figure 5 compares the performance of FCFS-backfill, LXF&W(0.02)-backfill, and the two best alternative new priority backfill policies (i.e., L$X^{1/2}$F&W(0.01) and S$T^{1/2}$F&W(0.05)-backfill, which slightly outperform SJF&W-backfill), each with 2 - 3 reservations. One key result is that using 2-4 reservations instead of one reservation has improved the overall performance of all four policies. For example, compared to Figure 2, the maximum wait for FCFS-backfill and LXF&W(0.02)-backfill is reduced by up to 30% while the average or 95th-percentile wait is increased by on the order of 10% or less. Another key result is that LXF&W(0.02)-backfill with 2-4 reservations has maximum wait that is reasonably competitive with FCFS-backfill, yet significantly outperforms FCFS-backfill for the other wait time statistics. L$X^{1/2}$F&W-backfill has very slightly better overall performance than LXF&W-backfill. Finally, S$T^{1/2}$F&W-backfill has better average and 95th-percentile wait than L$X^{1/2}$F&W-backfill, but more often has significantly poorer maximum wait than FCFS-backfill (e.g., in February and June 2001).



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

Figure 5. Comparisons of New Priority Backfill Policies that Favor Short Jobs

The overall conclusion is that, similar to results in [CV01a], giving priority to short jobs but also using an appropriate weight for current job wait can significantly outperform FCFS-backfill. In the remainder of this paper, we study the impact of more accurate requested runtimes on these high performance backfill policies that favor short jobs.

4 More Accurate Requested Runtimes

There is reason to believe that runtimes can be more accurately estimated for the jobs that run on the O2K. In particular, a majority of the jobs use one of the default requested runtimes, which are "very small" (5 hours), "small" (50 hours), "medium" (200 hours), and "large" (400 hours). This indicates that users have the habit of specifying hugely approximate requested runtimes due to the course-grain defaults that are available. Furthermore, since the current priority-backfill policy provides similar 95th-percentile waiting time for the entire range of job runtimes (see Figure 2(e) and results in [CV01a]), there isn't currently any incentive for an individual user to provide a more accurate requested runtime. This explains why, for example, many of the jobs that have actual runtime of 10 hours have requested runtime of 50, 200, or 400 hours.

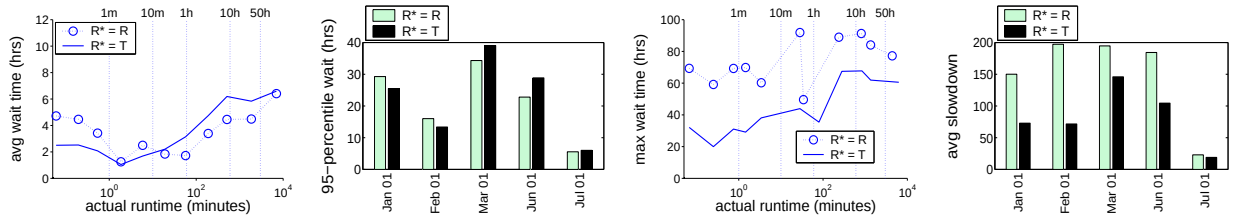
This section uses notation defined in Table 4 and evaluates the performance improvement, as a function of actual job runtime, for various scenarios of more accurate requested runtimes. Letting T denote the actual job runtime from the trace, we consider three cases. In the first (implausible) case, each requested runtime is perfect (i.e., $R* = T$). In the second case, the requested runtimes are imperfect but are *approximately* accurate (i.e., $R* = \min\{R, kT\}$, $1 < k \leq 2$). In the third case, only a fraction (e.g., 80% or 60%) of the jobs have these approximately accurate requested runtimes, while the rest of the jobs, selected randomly, have requested runtimes as given in the job log, which are generally highly inaccurate. The first case is used to provide a bound on the maximum benefit of more accurate runtimes, while the second and third cases are used to assess performance gains that are more likely to be achievable. Section 5 will explore the performance impact of using short test runs to achieve the more accurate runtime requests.

We present results for $k = 2$. We also considered smaller values of k, in particular $k = 1.2$, which results in slightly better performance, but we omit those results below to conserve space. As noted in Section 2.3, several previous papers [FW98, ZK99, ZFMS00, ZFMS01] have used a random (i.e., uniform) distribution of requested runtime overestimations, with a large upper bound (e.g., 10, 50, or 300). In contrast, our scenarios assume that requested runtime is never larger than the actual requested runtime in the workload trace. The fraction of jobs that use inaccurate requested runtimes from the trace represent runtimes that can't (easily) be accurately estimated and/or carelessly specified runtime requests. This fraction is varied in the experiments.

Previous results suggest that using more accurate requested runtimes has only minimal impact on the average slowdown and average wait time for FCFS-backfill [FW98, STF99, ZK99, ZFMS00, ZFMS01]. This section investigates whether the benefit of more accurate requested runtimes is more significant for the above scenarios, higher system loads, and more complete performance measures, as well as for priority backfill policies that use requested runtimes to favor short jobs. Section 4.1 reassesses the impact of more accurate requested runtimes on FCFS-backfill, whereas Section 4.2 evaluates the impact of more accurate requested runtimes on the policies that favor short jobs.

Table 4. Notation

Symbol	Definition
T	Actual job runtime
R	User requested runtime from the O2K logs
R*	Simulated requested runtime
P	Number of requested processors



(a) Avg Wait vs. T (June 2001) (b) 95th-Percentile Wait (c) Max Wait vs. T (Jan. 2001) (d) Avg Slowdown

Figure 6. Impact of Perfect Requested Runtimes for FCFS-backfill

4.1 FCFS-backfill Results

Figure 6 compares the performance of perfectly accurate requested runtimes (i.e., R* = T) against user requested runtimes from the trace (i.e., R* = R) for FCFS-backfill with two reservations. The results for previous typical O2K workloads (e.g., July 2001) agree with previous results in [FW98]; that is, using more accurate runtimes has only very slight impact on system performance. Moreover, perfect requested runtimes have minimal impact on the overall average waiting time for each month (not shown), and on the 95th-percentile wait each month, shown in Figure 6(b). On the other hand, as shown in Figure 6(a) for June 2001, accurate runtime requests improve the average wait of very short jobs ($T < 30$ minutes) during and immediately following the new exceptionally heavy load months. More significantly, Figure 6(c) shows that accurate requested runtimes significantly improve maximum wait time for most actual job runtimes, for many of the exceptionally heavy load months and immediately following new heavy load months. Figure 6(d) shows that actual runtimes significantly reduce average slowdown under and immediately following new heavy load months (by up to 60% in Feb. 2001).

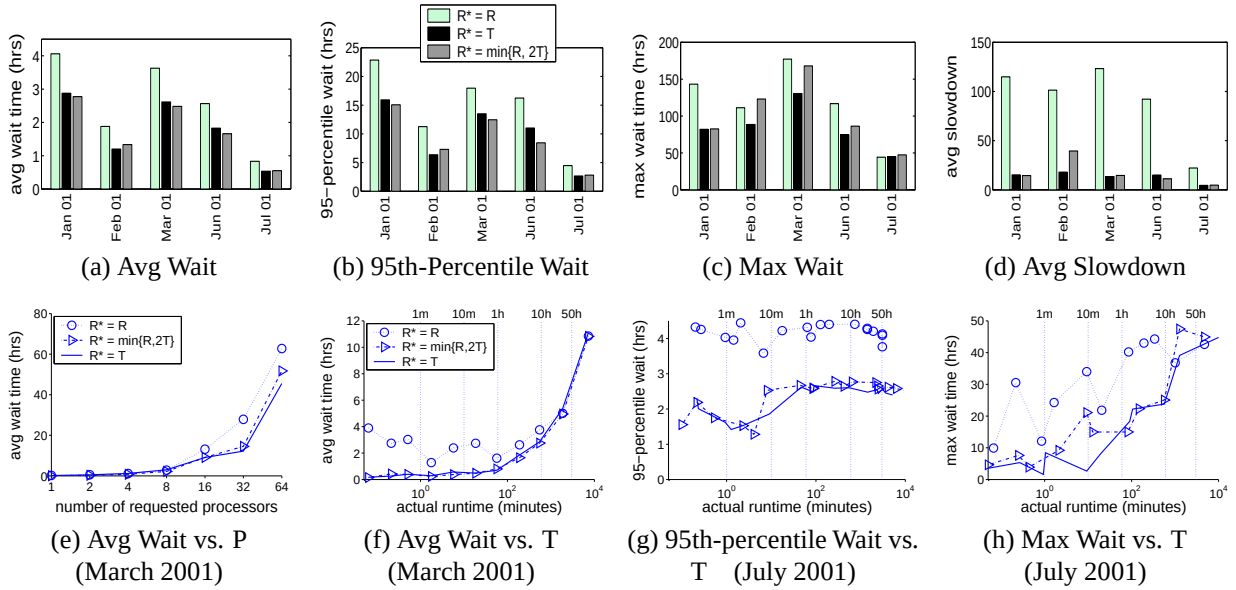
We note that perfect requested runtimes generally improves the wait time for short jobs because these jobs can be backfilled more easily. Accurate requested runtimes also improve the maximum wait for long jobs due to shorter backfill windows. Using approximately accurate requested runtimes (i.e., $R* = kT$) has a somewhat lower impact on system performance than using perfect runtime requests (not shown to conserve space).

4.2 Results for Policies that Favor Short Jobs

This section studies the potential impact of using more accurate requested runtimes for policies that favor short jobs. We present the results for L$X^{1/2}$F&W-backfill. Results are similar for the other priority backfill policies that favor short jobs (such as S$T^{1/2}$F&W-backfill). First, we consider the case where *all* jobs have requested runtimes within a small factor of their actual runtimes. Then, we consider the impact of the case where only 80% or 60% of the jobs have approximately accurate requested runtimes.

Figure 7 compares the performance of perfectly accurate runtime requests (i.e., R* = T) and approximately accurate runtime requests (i.e., R* = Min{R, 2T}) against requested runtimes from the trace (i.e., R* = R). Graphs (a)-(d) contain the overall performance measures each month, whereas graphs (e)-(h) show performance versus requested number of processors or actual runtime for the March 2001 workload. Results for other months (not shown) are similar.

In contrast to the FCFS-backfill results in the previous section (shown in Figure 6), there is an even larger potential benefit of using more accurate requested runtimes for L$X^{1/2}$F&W-backfill, because accurate runtime requests enable L$X^{1/2}$F&W-backfill to give priority to jobs that are actually shorter. In particular, Figures 7(a) - (d) show that perfectly accurate runtime requests improve not only the maximum wait and average slowdown, but also the average and 95th-percentile wait time over all jobs. Furthermore, the average slowdown is dramatically improved in *every* month, including the typical



(a) Avg Wait (b) 95th-Percentile Wait (c) Max Wait (d) Avg Slowdown

(e) Avg Wait vs. P (March 2001) (f) Avg Wait vs. T (March 2001) (g) 95th-percentile Wait vs. T (July 2001) (h) Max Wait vs. T (July 2001)

Figure 7. Performance Improvement for Accurate Runtime Requests under L$X^{1/2}$F&W-backfill

O2K load months (e.g., July 2001). These four graphs also show that even if the the requested runtimes are only approximately accurate (i.e., R* = Min$\{$R, 2T$\}$), similar improvement in average and 95th-percentile wait time, as well as similar dramatic improvement in average slowdown, can be achieved.

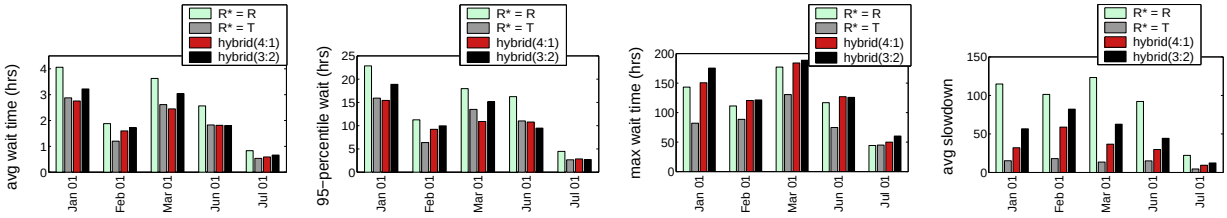
Figure 7(e) shows that accurate or approximately accurate requested runtimes improve the average wait time for jobs with a large number of requested processors ($\geq$ 32).

Figures 7(f)-(h) show that more accurate requested runtimes improve the average wait for short jobs (up to 10 hours), 95th-percentile wait for all jobs, and the maximum wait for all but the very largest jobs. Note also that the improvement in the average wait for short jobs is significantly larger than the improvement for accurate runtime requests in FCFS-backfill, *and* the improvement is achieved without increasing the average wait time for longer jobs. Furthermore, when requested runtimes are accurate or approximately accurate, the average wait under L$X^{1/2}$F&W-backfill decreases (monotonically) as the actual runtime decreases; this is a desirable property that, to our knowledge, has not been demonstrated for any previous system with a backfill policy.

Next, we consider scenarios in which not all jobs have approximately accurate requested runtimes. Two systems are evaluated: hybrid(4:1) and hybrid(3:2), where in the hybrid(4:1) system, 4 out of 5 jobs (i.e., 80% of jobs), selected randomly, have approximately accurate requested runtime (i.e., R* = minR, 2T). Again, results will be shown for L$X^{1/2}$F&W-backfill, and the results are similar for the other priority backfill policies that favor short jobs.

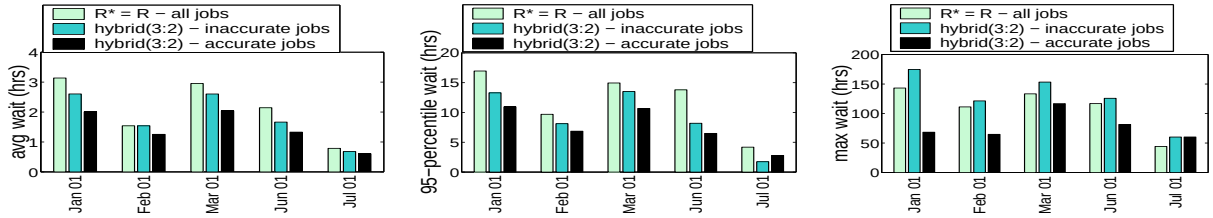
Figure 8 compares hybrid(4:1) and hybrid(3:2) against the case where all jobs have perfectly accurate runtime requests (i.e, R* = T), and the case where all jobs use requested runtimes from the trace (i.e, R* = R). The key conclusion is that much of the benefit of accurate requested runtimes can be achieved even if only 60% or 80% of the jobs have approximately accurate requested runtimes. Specifically, Figures 8(a) and (b) show that hybrid(4:1) has similar average and 95th-percentile wait time as the system with perfect runtime requests. Figure 8(c) shows that hybrid(4:1) has somewhat higher maximum wait than when requested runtimes are perfectly accurate, but has lower maximum wait than for user requested runtimes in the trace. Figure 8(d) shows that hybrid(4:1) has much lower average slowdown than the system with user requested runtimes from the trace. If only 60% of the jobs have improved requested runtimes, i.e., hybrid(3:2), the performance improvement is smaller than that in hybrid(4:1), but hybrid(3:2) still has lower average and 95th-percentile wait time and significantly lower average slowdown than that of using very inaccurate requested runtimes from the trace. Further reducing the fraction of the jobs to have improved requested runtimes results in a system more similar to that of using requested runtimes from the trace.

We now provide results that show that the jobs that have more accurate requested runtimes see most of the performance benefit. In particular, Figure 9 compares the wait time statistics for *'accurate jobs'* (i.e., R* $\leq$ 2T) in the

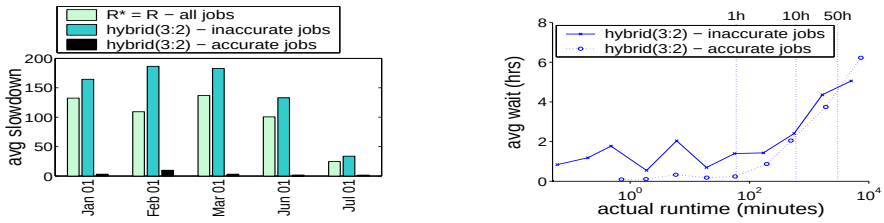


(a) Avg Wait (b) 95th-Percentile Wait (c) Max Wait (d) Avg Slowdown

Figure 8. Performance of Hybrid(x:y) Approximately Accurate:Inaccurate Requested Runtimes
(L$X^{1/2}$F&W-backfill, Approximately Accurate R* = Min{R, 2T})



(a) Avg Wait ($T \leq 50$ hours) (b) 95th-percentile Wait ($T \leq 50$ hours) (c) Max Wait ($T \leq 50$ hours)



(d) Avg Slowdown ($T \leq 50$ hours) (e) Avg Wait vs. T (June 2001)

Figure 9. Performance Benefit for Jobs with More Accurate Requested Runtimes
(L$X^{1/2}$F&W-backfill; Accurate R* = Min{R, 2T})

hybrid system against the wait time statistics for *'inaccurate jobs'* (i.e., R* = R $>$ 2T) in the hybrid system. The figure also includes the performance when all jobs that have requested runtimes as in the workload trace (i.e., R* = R - all jobs). The results are shown for hybrid(3:2), in which only 60% of the jobs have improved requested runtimes. Note that only the jobs with under 50 hours of actual runtime are considered in the first four graphs because requested runtime accuracy has more impact on these jobs than on jobs with longer actual runtime (as can be seen in Figure 9(e)). Figures 9(a) - (c) show that during and immediately following the extremely heavy load months, for actual runtime up to 50 hours, jobs with accurate runtime requests have 20% lower average and 95th-percentile wait time and up to 50% lower maximum wait time than the jobs with inaccurate runtime requests. Furthermore, the jobs with accurate runtime requests improve the average and 95th-percentile wait time of inaccurate jobs, compared to when all jobs have the requested runtimes from the trace. Figure 9(d) shows that for any month, the average slowdown of jobs with accurate runtime requests is dramatically lower than that of jobs with inaccurate requests, and also lower than the overall average slowdown if all jobs use the requested runtime from the trace (i.e., R* = R). Figure 9(e) further shows that for actual runtime of up to 10 hours, jobs with accurate requests achieve significantly lower average wait time than that of inaccurate jobs, and average wait decreases monotonically as actual runtime decreases for the jobs with accurate requests.

5 Test Runs for Improving Requested Runtimes

Results in Section 4 show that if a large fraction of the jobs (e.g., at least 60%) have estimated runtimes within a

factor of two of their actual runtime, shorter jobs would have lower expected and maximum wait than large jobs, and large jobs will have better 95th-percentile and maximum wait than with the very inaccurate requested runtimes in the current O2K system. Thus, if users are provided with incentives and tools to provide more accurate requested runtimes, the users will reap significant performance benefit.

Some jobs have inherently inaccurate requested runtimes because the runtime is highly unpredictable for any given (new) set of input parameters. For example, some applications, including stochastic optimization, have a number of iterations that is dependent on how quickly the solution converges, which generally can't be predicted ahead of time. However, approximately accurate requested runtimes could be provided for these latter codes if the computation is broken into several runs, each except the last run having requested runtime likely to be somewhat smaller than needed to reach final convergence, and with the solution from one run being input to the next run. For many other applications, codes are often run with similar input parameters to previous runs, or with changes in the input parameters that will affect run time in an approximately predictable way (e.g., runtime can be estimated within a factor of two), or the runtime request can be more accurate if a short test run is made before the full run. Example applications that can estimate requested runtime after a test run include those that involve iterative computation in which the number of iterations and/or the time per iteration are dependent on the input data, but can be estimated reasonably well have running the first few iterations.

This remainder of this section investigates whether most of the benefit of more accurate requested runtimes shown in the previous section can still be realized if (some or all) users perform a test run to better estimate the requested runtime for their jobs.

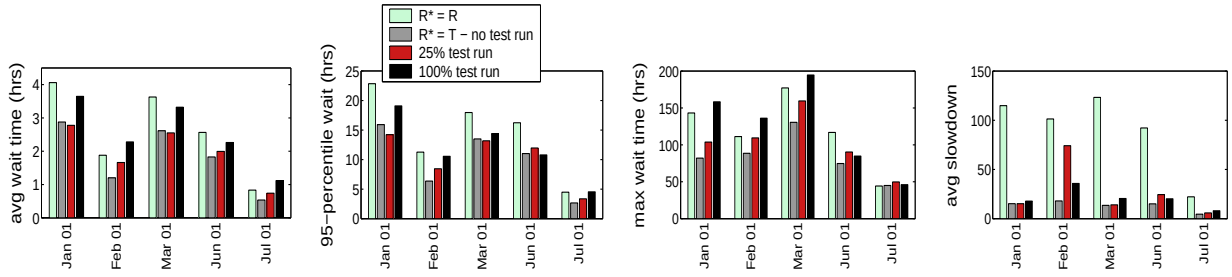
The assumptions regarding the test runs are as follows. If the user requested runtime is already within a factor of two of the actual runtime, we assume that it is likely that the user is aware that a test run is not needed, and the job is simply submitted as it was in the actual system. Otherwise, the requested runtime for a test run is equal to: (a) 10% of the user requested runtime (if the user requested runtime is under 10 hours) or (b) one hour (if the user requested runtime is greater than 10 hours). The requested runtime for the test run represents the runtime needed to estimate the full job runtime within a small factor. Note that because the user requested runtimes are highly inaccurate, the job might complete during the test run. In the actual system, jobs might complete during the test run either due to the user's lack of experience in how long the test run should be, or due to an unexpected error in the execution. If the job does not complete during this test run, the job is resubmitted with an improved requested runtime (i.e., an estimated requested runtime that is a small factor times the actual job runtime). As in Section 4.2, we use R* = Min$\{$R, 2T$\}$ for approximately accurate requested runtimes. We also consider the case where some fraction of the jobs do not make test runs, and some fraction of those just just use the requested runtime from the workload logs, representing jobs for which the user is either not able or not interested in estimating runtime more accurately.

Section 5.1 considers the scenario in which all full runs have requested runtime within a factor of two of the actual runtime, but two different fractions of jobs (i.e., 100% or 25%) (randomly selected) make test runs before submitting with the approximately accurate requested runtime. Section 5.2 considers the scenario in which 20% or 50% of the jobs provide the same requested runtimes as in the job trace, while the other jobs provide approximately accurate requested runtimes. Of the jobs that provide approximately accurate runtime requests, 25% make the test run before submitting with the approximately accurate request.

5.1 Improved Requested Runtimes for All Jobs

This section studies the impact of test runs for the optimistic ("best case performance") scenario in which all jobs provide approximately accurate requested runtimes. In one case ("25% testrun"), 25% of jobs that do not have approximately accurate user requested runtime from the trace are randomly selected to have a test run; the remaining 75% of such jobs will have improved runtime estimate without a test run, due to previous user experience with the code. In the other case ("100% testrun"), every job with improved runtime request requires a test run. Note that "100% testrun" is a pessimistic assumption that is not likely to occur in practice, whereas "25% testrun" is more likely to be representative of a realistic setting, particularly since many applications are run a large number of times, and in many cases previous executions can be used to improve runtime request accuracy. Thus, we consider the "25% testrun" experiments to be representative of the practical impact of using test runs to improve runtime estimate accuracy.

During each month, 35-45% of the jobs have inaccurate requested runtimes (i.e., R $>$ 2T) and actual runtime T $>$ Min$\{$1 hour, 10% R$\}$. For such jobs, if a test run is used



(a) Avg Total Wait (b) 95-Percentile Total Wait (c) Max Total Wait (d) Avg Slowdown

Figure 10. Impact of Test-Runs to Determine Requested Runtimes
(L$X^{1/2}$F&W-backfill; wait includes testrun wait and overhead; R* = Min{R, 2T})

to improve requested runtime, the job is resubmitted after the test run. The total extra load due to the test runs is very small (only 1-3% increase in processor and memory demand each month), even for 100% testrun. However, the additional waiting time for the test run, and the test run, must be included in the average and other measures of total job waiting time.

Figure 10 compares 100% testrun and 25% testrun against the optimal case where all jobs use actual runtimes (i.e., R* = T) without test runs and the case where all jobs use the requested runtimes from the trace (i.e., R* = R). The average total wait, 95th-percentile total wait, maximum total wait, and average slowdown, are shown for representative recent O2K workloads. For each of these measures except average slowdown during February 2001, the performance of the 25% testrun case is very similar to the case where R*=T. There are small increases in maximum total wait during the heavy load months and in the other measures for February 2001, but overall the results show that a significant fraction of test runs can be made to improve requested runtimes, and if the improved requested runtimes are within a factor of two of the actual runtime, then nearly the maximum possible benefit of accurate requested runtimes can be achieved.

The test run overhead becomes prominent if all jobs with R $>$ 2T require a test run (i.e., 100% testrun). Figure 10(a) shows that the overall average wait time in the case of ”100% testrun” is 0.5 - 1 hour longer than that of using actual runtimes (without test runs), due to test run wait and test run time incurred by over 35% of the jobs each month. Similarly, Figure 10(b) and (c) show that ”100% testrun” has worse maximum wait and 95th-percentile wait time than that using actual runtimes in new heavy load months. Even so, ”100% testrun” still has lower average and 95th-percentile wait and especially lower average slowdown than that of using requested runtimes from the trace, under and immediately following new heavy load months. In practice, we expect an even higher benefit of test run than that of ”100% testrun” compared to using requested runtimes from the trace.

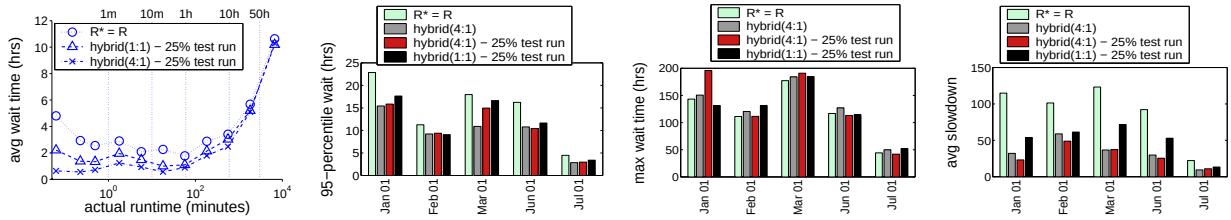
5.2 Improved Requested Runtimes for a Majority of the Jobs

This section considers scenarios where only 50% or 80% of the jobs have improved requested runtime accuracy, and test runs are needed in 25% of the cases to estimate the improved requested runtimes. Again, we use R* = Min{R, 2T} for approximately accurate requested runtimes. The two scenarios to be evaluated are thus named hybrid(4:1) and hybrid(1:1) - 25% testrun. Note that hybrid(1:1) with 25% testrun represents a reasonably pessimistic, but possibly realistic scenario, in which only 50% of the jobs have approximately accurate requested runtimes and one out of four jobs requires a test run to improve requested runtime accuracy.

Figure 11 compares the above two hybrid scenarios with 25% test run against that of using requested runtimes from the trace (i.e., ”R* = R”). The performance for hybrid(4:1) without test run is also included in Figure 11(b) - (d) for comparison with hybrid(4:1) with 25% testrun. The results show that for both hybrid systems with testruns, the average wait for short jobs, the 95th-percentile wait, and the average slowdown is significantly better than for the requested runtimes in the O2K traces. The results also show that test runs do not introduce significant overhead in the hybrid (4:1) system.

6 Conclusions

In this paper, we used ten one-month more recent traces from the NCSA O2K to evaluate whether high-performance backfill policies can be further significantly



(a) Avg Wait (Representative Jan. 2001) (b) 95th-Percentile Wait (c) Max Wait (d) Avg Slowdown

Figure 11. Performance of Hybrid(x:y) with Test Runs
(L$X^{1/2}$F&W-backfill, R* = Min{R, 2T})

improved if the requested runtimes can be more accurate. Many of these months have exceptionally heavy load, which tends to make the policy performance differentials larger than for lower load used in previous work.

To improve backfill policies for studying our key question, we more fully evaluated the trade-offs between favoring short jobs and preventing starvation for backfill policies. Our results show that a few reservations (2 - 4) can significantly reduce the maximum wait time but a larger number of reservations result in poor performance; fixed reservation or dynamic reservation makes minimal difference in most cases, except for SJF-backfill which requires fixed reservation to reduce starvation.

Our results of using higher-performance backfill policies (such as L$X^{1/2}$F&W-backfill and S$T^{1/2}$F&W-backfill with two reservations), heavier system load, and a more complete set of performance measures show that the potential benefit of more accurate requested runtimes is significantly larger than suggested in previous FCFS-backfill results, even if each requested runtime is up to twice the actual runtime. Furthermore, we show that most of the benefit of more accurate requested runtimes can be achieved by using test runs to improve requested runtime accuracy even though there is test run overhead. Our results also show that users who provide more accurate requested runtimes can expect improved performance, even if other jobs do not provide more accurate requested runtimes.

References

[CB01] W. Cirne and F. Berman. A Comprehensive Model of the Supercomputer Workload. In *Proc. IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX., December 2001.

[CV01a] S.-H. Chiang and M. K. Vernon. Production job scheduling for parallel shared memory systems. In *Proc. Int'l. Parallel and Distributed Processing Symp. (IPDPS) 2001*, San Francisco, April 2001.

[CV01b] S.-H. Chiang and M. K. Vernon. Characteristics of a large shared memory production workload. In *Proc. 7th Workshop on Job Scheduling Strategies for Parallel Processing*, Cambridge, MA., June 2001.

[FW98] Dror G. Feitelson and Ahuva Mu'alem Weil. Utilization and predictability in scheduling the IBM SP2 with backfilling. In *Proc. 12th Int'l. Parallel Processing Symp.*, pages 542–546, Orlando, March 1998.

[Gib97] R. Gibbons. A historical application profiler for use by parallel schedulers. In *Proc. 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, Geneva, April 1997. Lecture Notes in Comp. Sci. Vol. 1291, Springer-Verlag.

[Lif95] D. Lifka. The ANL/IBM SP scheduling system. In *Proc. 1st Workshop on Job Scheduling Strategies for Parallel Processing*, pages 295–303, Santa Barbara, March 1995. Lecture Notes in Comp. Sci. Vol. 949, Springer-Verlag.

[MF01] 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 & Distributed Syst.*, 12(6):529–543, June 2001.

[NCSa] *NCSA Scientific Computing SGI Origin2000*. http://www.ncsa.uiuc.edu/SCD/Hardware/Origin2000.

[NCSb] *NCSA Scientific Computing: IA-32 Linux Cluster*. http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IA32LinuxCluster.

[PK00] D. Perkovic and P. J. Keleher. Randomization, speculation, and adaptation in batch schedulers. In *Proc. 2000 ACM/IEEE Supercomputing Conf.*, Dallas, November 2000.

[SCZL96] J. Skovira, W. Chan, H. Zhou, and K. Lifka. The EASY-Loadleveler API Project. In *Proc. 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, pages 41–47, Honolulu, April 1996. Lecture Notes in Comp. Sci. Vol. 1162, Springer-Verlag.

[STF99] W. Smith, V. Taylor, and I. Foster. Using runtime predictions to estimate queue wait times and improve scheduler performance. In *Proc. 5th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 202–219, San Juan, April 1999. Lecture Notes in Comp. Sci. Vol. 1659, Springer-Verlag.

[ZFMS00] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam. Improving parallel job scheduling by combining gang scheduling and backfilling techniques. In *Proc. Int'l. Parallel and Distributed Processing Symp. (IPDPS) 2000*, Cancun, May 2000.

[ZFMS01] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam. An analysis of space- and time-sharing techniques for parallel job scheduling. In *Proc. 7th Workshop on Job Scheduling Strategies for Parallel Processing*, Cambridge, MA., June 2001.

[ZK99] 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*, pages 236–243, Redondo Beach, August 1999.