Evaluation of Parallel Job Scheduling Policies for Production Scientific Workload

Su-Hui Chiang

Technical Report #1434

January 2002

Evaluation of Parallel Job Scheduling Policies for Production Scientific Workloads

By

Su-Hui Chiang

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN – MADISON

2001

Abstract

This thesis addresses open questions regarding high-performance job scheduling policies for *challenging workloads* that run on large-scale production parallel systems. As part of this study, the production workload that runs on the National Computational Science Alliance (NCSA) Origin 2000 (O2K) system is completely characterized with respect to the policies evaluated.

Trace-driven simulation with six one-month O2K job traces are used to evaluate the following policies: (1) the experimentally tuned NCSA-LSF* multi-class priority scheduler, (2) two previously proposed First-Come First-Served (FCFS)-backfill and Shortest-Job-First (SJF)-backfill; (3) two new priority backfill policies: Priority-backfill and LXF&W-backfill, designed to give short jobs better service without starving long jobs; (4) limited preemption for backfill policies to give a limited immediate quantum to each new job that cannot be started immediately due to insufficient resources; (5) the spatial equipartitioning (EQspatial) policy modified to provide better service for the largest and longest jobs.

Measurements on the O2K validate simulation results for two policies used on the system. Simulation results show that the backfill policy using current job expansion plus a small weight for waiting time (LXF&W) to prioritize jobs, with limited preemptive immediate service, is the most promising if jobs cannot adapt to changing processor allocations during execution or preemption is not fully supported. On the other hand, EQspatial provides significantly better 95th-percentile waiting time for the O2K workload.

The full workload characterization includes the distributions of job interarrival time, requested number of processors, requested memory, requested runtime, ratio of actual to requested runtime, ratio of peak memory usage to memory request, as well as the conditional distributions needed to generate synthetic workloads that have the observed correlation among the key job characteristics in the O2K workload. Many distributions and conditional distributions have not been provided in previous studies. Jobs submitted to the O2K are very similar across different periods of the day (i.e., peak, intermediate, and low arrival rate periods), days of the week, and from month to month over an entire year. The workload characteristics are largely similar to previously reported production parallel scientific workloads, with only a few exceptions noted in the thesis.

Acknowledgements

My greatest thanks go to my advisor, Professor Mary Vernon, without whose advice this thesis would not have been possible. I thank her for teaching me performance modeling. It has been my good fortune to learn from an expert in this area. I am truly grateful for her constant support, criticism, and encouragement.

I thank Professor Andrea Arpaci-Dusseau, Miron Livny, Remzi Arpaci-Dusseau, and Vicki Bier for serving in my final defense committee and constructive criticism that helped improve the thesis. I also thank Professor Jim Goodman for serving in my prelim committee.

My appreciation also goes to Michael Shapiro, Wayne Hoyenga, Dave Jackson, Michael Pflugmacher, 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.

Special thanks go to a then fellow student Rajesh, who has given me valuable lessons on sample path analysis and helped me getting started with simulation of job scheduling policies. I would also like to thank my fellow students, Jussara and Ian for their friendship and technical discussions. I am also indebted to my friends Huey-Chi, Min, Yuh-Jye, and Liang-Yin for their companionship outside computer science, and Joyce, Huei-Fung, Herman, Annie, Su-Ling, Erik, and more for offering great friendship and support over the course of my Ph.D. Finally, it is a great pleasure to thank my parents for having confidence in me, and my sister and brothers for their encouragement and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many different parallel job scheduling policies have been proposed and evaluated in previous research. Several policies have been identified that outperform other policies that have been evaluated. Some of these promising policies are used on production parallel systems. For example, a FCFS-backfill policy has been proposed and shown to improve the strict FCFS policy on an SP/1 and is currently used on many SP/2 systems; the Gang Scheduling policy has been shown to improve FCFS-backfill and has been employed on some Intel Paragon, CM-5, and SP/2 systems. Other of the high-performance policies are being considered for use on production parallel systems. For example, the dynamic equal spatial partitioning policy (i.e., EQS) is implemented in the Cilk prototype runtime system and is under consideration for use on the National Computational Science Alliance (NCSA) Origin 2000 (O2K). There are also a few configurable commercial schedulers that are used on production systems. For example, LSF, a configurable non-preemptive multi-class priority scheduler, has been used on O2K.

In spite of a large body of prior work, the performance of the scheduling policies is still not well understood, for at least four reasons. First, some promising policies have not been compared against each other. For example, the EQS policy has not been compared with FCFS-backfill; and the non-preemptive multi-class priority policies have also not been compared with FCFS-backfill. As a result, it is difficult for a high-performance production computing facility, such as that at NCSA, to decide which of the existing policies to use for their production parallel systems. Second, most previous policy comparisons use a single performance measure, such as average waiting time or average slowdown (defined to be the ratio of job response time to actual runtime). This yields an incomplete understanding of the relative policy performance. For example, a policy may have a very low average waiting time and/or average slowdown but a very high maximum waiting time, and thus may not be the preferred policy by a high-performance production computing facility. As another example, the overall average wait may be low but a particular class of jobs may experience high wait. Third, starvation-free priority measures for backfill policies that favor short jobs may improve the overall performance of FCFS-backfill but have not been proposed or studied. The fourth problem is that of the workloads used to evaluate policies in previous work. Many previous studies have evaluated policies experimentally using a set of benchmark applications, or by simulation using job traces from particular production systems, but have not provided a detailed enough workload characterization data to know whether the results are applicable for a given other workload. Other previous simulation studies use synthetical workloads, based on hypothetical distributions that have not been shown to match observed distributions and do not apply for production workloads in general. Furthermore, key workload features are missing in the previous synthetic workloads. For example, several previous studies have used the actual runtime as the requested runtime or a hyperexponential distribution is proposed for the observed actual runtime without providing data to show the fit of the distribution. Conditional distributions of requested runtime capturing observed correlation with the key job characteristics, such as requested number of processors and the requested memory, are missing in previous workload models.

The goal of this thesis is to provide a better understanding of the relative performance of parallel scheduling policies by more completely comparing promising existing policies and by proposing and comparing important new policies, with a key goal of providing sufficient information for organizations such as NCSA to choose a policy that is appropriate for their production parallel systems. We evaluate policies by simulation, using six one-month O2K job traces from late 1999 and early 2000. Complete characterization of the workloads is provided so that further workloads can be compared and the policy evaluation results can be applied as appropriate. Furthermore, the workload characterization helps in designing the proposed new policies, and provides a model that can be used to generate synthetic workloads that have the distributions and observed correlations among key job measures in the O2K workload.

Section 1.1 further provides background for the scheduling policies. Section 1.2 lists the contributions of this thesis. Section 1.3 lists the key results of this thesis. The organization of the remainder of this thesis is given in Section 1.4.

1.1 Background: Job Scheduling

Parallel job scheduling policies can be classified into non-preemptive and preemptive policies. Under non-preemptive policies, each job runs to completion without interruption. The advantage of non-preemptive policies is that they are simpler to implement and incur minimal scheduling overhead. Under preemptive policies, a job may be preempted during execution. With this flexibility, preemptive schedulers can give prompt service to each new arrival job even at high system load, by reassigning processors from executing jobs to the new job and/or by time sharing processors among current executing jobs and the new job. Thus, preemptive policies provide better turnaround for short executing jobs, if scheduling overhead is appropriately kept small. The costs of preemption are more complex implementation of schedulers and the requirement of special system support for processor repartitionings and/or for time sharing.

A FCFS policy that allocates each job the number of processors that it requests has a head-of-line queue blocking problem. That is, the first job in the queue that requests more processors or memory than currently available blocks those jobs that are small enough to be scheduled on the free resources but are queued behind the first job. Skipping the jobs that are too large to be scheduled (i.e., FCFS with first fit) avoids this problem, but creates a different problem; namely, starving the large jobs.

FCFS-backfill [Lif95] was proposed to improve FCFS without starving large jobs. Under FCFS-backfill, the first waiting job is given a scheduled start time, while smaller jobs that are short enough can be scheduled out of order on the resources that would be otherwise idle. The requested runtime for each job is required to determine the start time of the first job and which jobs can be backfilled without delaying the job at the front of the queue.

Three other priority functions for backfill policies have been evaluated [ZK99, PK00, ZFMS00]. SJF-backfill (i.e., shortest job first with backfill) improves the turnaround for short jobs and may have little effect on most jobs, but may starve jobs that need long running times. The largest-processors-first and smallest-processors-first policies with backfill have also been evaluated [ZFMS00]. They have been shown to have worse average waiting times at high load, and a similar average waiting time at low to moderate loads, compared to FCFS-backfill. Alternative priority measures that favor short jobs without the problem of starving long jobs may improve the overall performance of FCFS-backfill, and need to be further explored.

Previous analyses of job scheduling policies for uniprocessor systems have identified several fundamental results [Kle76]. Shortest-remaining-time-first minimizes mean response time if the execution time is known. Processor sharing achieves same expected *expansion factor* (i.e., response time divided by execution time) for any execution time. Furthermore, the mean response time of processor sharing is insensitive to the coefficient of variation of the process execution time, C_D, and thus has lower mean response time than FCFS if $C_D > 1$ (which is expected for both uniprocessor and multiprocessor workloads). If $C_D > 1$, multilevel feedback queueing which gives priority to the process that has so far received the least service can be used to give higher priority to short jobs and further improve the mean response time of processor sharing.

These job scheduling policies for uniprocessor systems cannot be directly applied to the parallel workloads for several reasons. First, frequent preemptions required in these policies may result in unacceptable high memory overhead for parallel computers. Second, parallel jobs have widely varying processor requirements. Third, the mean response time is not the only performance measure for optimization on general-purpose parallel computers, because providing good service to large jobs is often an important performance goal on the systems at high-performance computing facilities. However, the idea in these fundamental uniprocessor scheduling policies can be used to guide the design of the job scheduling policies for the parallel workloads.

The *estimated current job expansion factor*, which is similar to the expansion factor defined above, except that it is computed according to requested runtime and the current wait time of the job is an example of a job measure that can be used for prioritizing jobs for backfill policies to give *relatively* higher priority to short jobs, rather than absolutely highest priority to the shortest jobs (as in SJF-backfill).

Gang and EQS policies are preemptive policies that give an approximately equal processing power to each job. They have been shown to have similar performance to each other and better mean response time than non-preemptive policies [ST91, CMV94, SST93, NSS93b, PD96, PS97].

1.2 Contributions of This Thesis

The contributions of this thesis are summarized below:

- We simulate with six one-month O2K job traces to evaluate the policies. The O2K traces contain more information than in previous job traces. For example, the job requested runtime, the requested memory, and the actual processor and memory usage over each 30-second period are available in the O2K traces; some or all of these data are missing in previous job traces. In addition, the O2K jobs collectively request a higher fraction (i.e., 90-100%) of the available processing time than the jobs in previous workloads.

- We provide a characterization of the large production parallel workload that runs on the O2K over six one-month periods in early and late 2000. This characterization is more complete than previous parallel workload characterizations, including: (1) a comparison of the workload mix from month to month; (2) an analysis of the number of job arrivals each day and for each hour, to examine the variation in the number of job arrivals across different days of the week and different hours of the day; (3) new distributions of requested memory, requested runtime, and the utilization on the requested processors; (4) a more extensive use of conditional distributions to capture the observed correlations among the requested number of processors, requested memory, requested runtime, peak memory usage, and actual runtime in the workload; (5) sufficient data and a systematic procedure for creating synthetic workloads that have the characteristics observed in the O2K workload; (6) differences and similarities between the O2K workload and previous workloads (e.g., the Cornell Theory Center SP/2 and Intel Paragon workloads); (7) an analysis of the characteristics of the fifteen jobs that have the largest processor demand in each month; and (8) characteristics of jobs that are submitted during periods of approximately stationary job arrival rate.

- New performance measures are used for evaluating the job scheduling policies, in addition to the average waiting time and average slowdown over all jobs. These new performance measures are: the 95th percentile and maximum waiting time over all jobs; the average, 95th percentile, and maximum waiting time versus actual job runtime; the average and maximum waiting time versus the number of requested processors and requested memory; and the average and maximum slowdown versus actual runtime. These measures provide information that is not revealed by a single measure, such as the overall average waiting time or average slowdown. For example, our performance measures allow us to examine: whether short jobs have good turnaround; whether the performance of long running jobs is compromised in order to favor short jobs; and whether large-processor jobs are starved.

- We compare the performance of NCSA-LSF* against FCFS-backfill. These two policies have not been compared in previous work.

- We propose and evaluate two new backfill policies: Priority-backfill; and LXF&W-backfill.

Each gives priority to short jobs, with LXF&W-backfill having a higher priority for short jobs.

- We propose a preemptive quantum of immediate service that provides good turnaround time for short jobs. Our policy is different from previous proposals. In particular, in our policy, only processors are preempted but not memory, to avoid memory overhead. Furthermore, we choose the quantum of processing time based on the workload characteristics; the quantum is large enough to benefit a significant number of jobs, but small enough to have minimal impact on long running jobs.
- The quantitative performance difference between EQspatial and the best of the priority backfill policies evaluated is presented. Comparisons between EQspatial and the backfill policies have not been provided in previous work.
- We evaluate the potential performance improvement for job traces with requested runtime replaced by actual runtime, and for two scenarios of more accurate estimates for FCFS-backfill and several priority backfill policies. Our results for FCFS-backfill are compared against previous results. No previous work has examined the impact of perfect or more accurate runtime estimates on priority backfill policies that favor short jobs.

1.3 Principal Results

The key results in this thesis are summarized below:

- In most cases, the characteristics of the O2K workload are similar to the characteristics that are reported for previous production scientific workloads on systems such as the Cornell Theory Center SP/2, the Intel Paragon, the Los Alamos National Lab CM/5, and NAS iPSC/860 workloads Exceptions include the following: (1) the coefficient of variation of the job interarrival time is in the range of 1-2 during each period of approximately stationary hourly arrival rate. (2) jobs require an *average* of 50-100 processor hours, depending on the month. (3) 10-15% of the jobs run for over 20 hours, (4) 15-20% of the jobs each month request a higher fraction of memory than processors, (5) jobs that request more processors tend to request and use less memory per processor. and (6) there is not an appreciable correlation between the number of processors requested and the job runtime.

- Interesting characteristics of the O2K workload not previously studied, include the following: (1) the fifteen largest jobs in a typical month have an average running time of over 200 hours, and use an average of 4000-8000 processor hours, (2) most of the fifteen largest jobs in a typical month arrive during peak arrival rate period (i.e., 8am - 5pm), similar to other jobs, (3) overall utilization on the allocated processors is fairly high (approximately 80%) whereas overall utilization on the allocated memory is closer to 50%, (4) most requested runtimes are default values (i.e., 5, 50, 200, or 400 hours), (5) whether or not a default runtime is requested, more than 50% of the jobs have actual runtimes less than 20% of their requested runtimes, and (6) over 20% of the jobs that request 400 hours terminate in more than 5 but fewer than 50 hours.

- We show that the Priority-backfill and FCFS-backfill policies significantly improve the overall performance of NCSA-LSF*. Based on this improvement, NCSA changed the scheduler on the O2K to a priority backfill scheduler configured with a similar priority function to that in Priority-backfill. The improvement measured on the O2K is consistent with the predicted improvement from our simulations.

- We show that the estimated current job expansion factor with a very small weight for the current job waiting time is a starvation-free job measure that favors short jobs and significantly improves Priority-backfill and FCFS-backfill.

- Adding a limited preemptive immediate service for the backfill policies, with an appropriate quantum of processing time and selection of the victim jobs for preemption, significantly improves the turnaround for jobs that complete in the quantum, without compromising the performance of other jobs.

- EQspatial policy with a low processor repartitioning overhead has the potential to further significantly improve the performance (especially the 95th percentile waiting time) of jobs that don't complete in the short quantum, even in the best of the priority backfill policies with limited preemption.

- It may be important to modify EQspatial to reduce the maximum waiting time of the jobs that have long running time and a large requested number of processors.

- The simulation results of using more accurate requested runtime suggest that the performance of several priority policies that favor short jobs can be significantly improved if all jobs, including short jobs, provide better runtime estimates.

1.4 Organization of this Thesis

The organization of this thesis is as follows. Chapter 2 provides background, including an overview of the NCSA O2K system, a review of the related previous work on workload characterization, and a review of previous results for parallel job scheduling policies. Chapter 3 discusses the methodology used in this thesis to evaluate the scheduling policies, including the definition of our baseline policy (i.e., NCSA-LSF*), the definitions of the other policies to be evaluated, our approaches for characterizing the O2K workload, the performance measures used to evaluate the policies, the evaluation results for NCSA-LSF*, and validation of our simulations of the schedulers on the O2K (i.e., NCSA-LSF* and the current priority backfill policy, called NCSA-MS*) against the measured performance on the O2K.

Chapter 4 provides characterization results for the NCSA O2K workload, including: the system load each month; the characteristics of several large job class and the fifteen largest jobs for each month; and the distributions of and the relationships among the interarrival time, requested number of processors; requested memory, requested runtime, actual runtime, and peak memory usage. Chapter 4 also gives a procedure for using the provided distributions and conditional distributions to create synthetic workloads that have the characteristics observed in the O2K workload.

Chapter 5 presents the results for the backfill policies, including: the comparisons between NCSA-LSF* versus FCFS-backfill and Priority-backfill; the performance of alternative priority backfill policies, including LXF&W-backfill and SJF-backfill; the impact of alternative reservation rules for the backfill policies; and the benefit of adding limited preemption for the best of the backfill policies. Chapter 6 presents further policy comparison results, including the performance of EQspatial versus backfill policies, and an investigation of the potential benefit of using more accurate runtime estimates for backfill policies. Finally, Chapter 7 provides conclusions and possible directions for future work.

Chapter 2

Background

This section provides an overview of the NCSA O2K system used for policy evaluation in this thesis, and background on workload characterizations and job scheduling policies.

Parallel job scheduling policies can be classified into non-preemptive and preemptive (i.e., dynamic) policies. The advantage of non-preemptive policies is that they are simple to implement and have minimal scheduling overhead, while preemptive policies incur higher scheduling overhead and are more complex to implement. The advantage of preemptive policies is the flexibility of reassigning processors by space-slicing or time-slicing in response to changes in system load. As a result, preemptive policies in general provide higher performance than non-preemptive policies.

Section 2.1 gives an overview of the NCSA O2K system and workload. Section 2.2 reviews related previous workload study results. Non-preemptive parallel job scheduling policies are reviewed in Section 2.3, and preemptive policies are reviewed in Section 2.4. A summary of the previous results on scheduling policies is provided in Section 2.5.

2.1 Overview of NCSA Origin 2000 System

This section gives a brief overview of the O2K system, including the host configuration, the job schedulers, and the job logs available on the system. The NCSA O2K is a large shared-memory multicomputer based on the MIPS R10000 processor with non-uniform memory access, The system studied consists of 1520 processors and 616 GB of memory, which is partitioned into twelve unequal-size hosts. Table 1 summarizes the number of processors, available memory,

Table 1. NCSA O2K Batch Hosts Configuration

Host Name	Number Proces-sors	Memory		Jobs Served
		Total (GB)	Per Processor (MB)	
eir	128	64	512	space-shared
nerthus	128	64	512	
hod1	128	64	512	
jord1	128	32	256	
saga1	128	32	256	
huldra	128	32	256	
mimir	128	32	256	
modi2	64	16	256	
aegir	128	64	512	dedicated*
forseti1	128	76	608	
balder	256	128	512	

* can also run very short space-shared jobs (under 5 hours).

and the types of jobs served on each of the eleven hosts that run batch jobs.

Eight hosts, with 960 processors and 336 GB of memory in total, are used for running batch jobs in a space-shared mode. Another three hosts, with 512 processors and 268 GB of memory in total, give priority to batch jobs that request a dedicated host. These 3 hosts also run jobs that request less than 5 hours of runtime in a space-shared mode when no dedicated jobs are waiting. The remaining host (not shown) with 48 processors and 12 GB of memory, is used for interactive jobs only.

Each space-shared job, when submitted to the O2K, requests a number of processors (up to 64), an amount of memory (up to 16 GB), and a runtime (up to 400 hours). Each job that requests a dedicated host specifies either 128 or 256 processors, and a runtime (up to 50 hours).

Prior to July 2000, the LSF Scheduler from Platform Computing [Pla] was used to dispatch the O2K jobs. In July 2000, a higher-performance priority backfill job scheduler (derived from the Maui scheduler [Mau] on an SP/2 at MHPCC [MHP]) replaced LSF for dispatching purposes, although LSF still records the information about the jobs.

Information about each job is recorded by the LSF Scheduler and by a JMD 'daemon' (implemented by NCSA). LSF records the submission time, start time, and completion time, and the requested number of processors, memory, and runtime. JMD monitors the execution of each job and records the actual number of processors and memory in use by each job once every 30 seconds. Appendix A shows the format of the job logs, and explains how we extracted the required information from the job logs for our analysis.

2.2 Previous Production Workload Characterizations

Previous production workload studies provide characterization results for the workloads on: the 128-processor Intel iPSC/860 at the NAS Facility of NASA's Ames Research Center; the 96-processor Intel Paragon at ETH Zuerich; the 128-processor SP/1 at Argonne National Lab (ANL); the 128-processor Butterfly at Lawrence Livermore National Lab (LLNL); the 512-processor SP/2 at Cornell Theory Center (CTC); the 400-processor Intel Paragon at San Diego Supercomputing Center (SDSC); the Cray T3E at SDSC; the 512-processor Cray T3D at Pittsburgh Supercomputing Center (PSC); the 1024-processor CM-5 at Los Alamos National Lab (LANL); the 100-processor SP/2 at the Royal Institute of Technology (KTH) in Stockholm, Sweden; the 64-processor SGI Origin2000 at NAS; the 320-processor ASCI Blue-Pacific at Lawrence Livermore National Lab (LLNL); and the Lewis network of workstations at NASA. Section 2.2.1 summarizes two partial workload models. Section 2.2.2 reviews the results of further workload characterization efforts, each focusing on one or two job characteristics.

2.2.1 Two Partial Workload Models

Table 2 summarizes two partial workload models provided in previous work; i.e., Feitelson96 model and the CTC SP/2 model, each of these models is further discussed below.

The Feitelson96 model [Fei96] is derived from the workloads that run on the following six

Table 2. Two Partial Workload Models

Job Characteristic	Workload Model	
	Feitelson'96 [Fei96]	CTC SP/2
Interarrival Time	Poisson job arrivals + Zipf for # repeated submissions of each application	hyperErlang for each P [JPF$^+$97]
# Requested Processors (P)	harmonic (hand-tuned to emphasize small #, interesting #)	log-uniform [Dow97b]
Actual Runtime (T)	2-stage hyperexponential (smaller P: higher probability choosing the smaller mean)	(1) observed avg runtime for 15 equal-number sets of jobs partitioned according to #processors [Hot96] (no clear relation between T & P) (2) observed distributions for four equal-number sets of jobs partitioned according to #processors [DF99] (smaller P: weight on shorter time)
P $\times$ T	-	(1) hyperErlang for each P [JPF$^+$97] (2) log-uniform [Dow97b, DF99]
Requested Memory Per Processor	-	five-ranged distribution [Hot96] (85% of the jobs: the smallest size)

production systems: iPSC/860; SP1; CTC SP/2; ETH Paragon; SDSC Paragon; and Butterfly. This model provides the job arrival process, the distribution of the requested number of processors, and the distribution of the actual runtime conditioned on the requested number of processors, as given in Table 2. Results are based on the following observations:

- Users have the tendency to submit sequences of the same job, one after another (observed in the iPSC/860 and SP1 traces, the only two traces that have job names). The maximum observed is 402 runs of the same job in the SP1 workload.

- There is a positive correlation between the actual runtime and the requested number of processors (observed in the iPSC/960 and SDSC Paragon workloads).

- Small numbers of requested processors are more common than large numbers of requested processors.

- The number of requested processors is most commonly a power of two; perfect squares;

multiples of ten, the maximum number of processors, etc., also stand out in many of the workloads studied.

The Feitelson96 model captures the correlation between actual runtime and the number of requested processors as follows: the actual runtime is generated by the two-stage hyperexponential distribution with the probability for choosing the stage that has the higher mean monotonically increasing with the number of requested processors. This correlation model has been used in at least two papers ([LKK99, Aid00]) to create synthetic workloads for evaluating the parallel job scheduling policies.

Note that some characterization data for several of the above systems are also provided in other papers; they are the iPSC/860 in [FN95], the SDSC Paragon in [WLF$^+$96, WMKS96], and the ETH Paragon in [SGS96].

The CTC SP/2 model is a collection of characterization results for the CTC SP/2 system studied in [Hot96, JPF$^+$97, Dow97b, DF99]. This model includes hyper-Erlang distributions conditioned on the number of processors for both the interarrival time and the total processing time (i.e., the product of the requested number of processors and the actual runtime), proposed in [JPF$^+$97]. The model parameters are chosen so that the hyper-Erlang distribution matches the first three moments of the observed distribution for each number of processors. At least two papers [FJM$^+$99, ZFMS00] use this model to generate the interarrival time and the total processing time or actual runtime for synthetic workloads. Most jobs in the SP/2 workload request power-of-two numbers of processors, and a significant fraction of the jobs are serial, as in the Feitelson96 model; however, [Dow97b] uses the log-uniform distribution to model the requested number of processors because they believe that requesting powers of two is due to the user habits and the interface to the queueing system. Note that [Dow97b] uses the log-uniform distribution to model the requested number of processors in the SDSC Paragon workload, for the same reasons. For the distribution of actual runtime in the SP/2 workload, the study in [Hot96] does not find a clear correlation between the mean actual runtime and the requested number of processors; on the other hand, [DF99] finds that the cumulative distribution of the actual runtime for jobs with fewer nodes has more weight at the short runtimes, and suggests that the model for the correlation between actual runtime and the number of requested processors defined in the Feitelson96 model could be used for the SP/2 workload. The difference in these two results could be due to different granularity in partitioning the jobs according to number of requested processors; [DF99] has a coarse partitioning only (4 categories of jobs, compared to 15 in [Hot96]). Finally, the observed fraction of jobs requesting each of five different ranges (i.e., $\leq$ 128, 129-256, 257-512, 512-1024, and 1024-2049 MB) of requested memory per processor is provided for the SP/2 [Hot96] workload; about 85% of the jobs request under 128 MB, and these jobs account for 96% of the total processor demand. Note that the scheduler places these jobs on the 128 MB nodes if there are free 128 MB nodes; otherwise, they can be placed on free nodes with more memory per node. Thus, the 128 MB nodes are over-requested.

2.2.2 Further Previous Workload Characterizations

This section reviews further previous workload characterization results, which provide one or two specific job characteristics that are not included in the two workload models discussed in the previous section. Four papers [FN95, WLF$^+$96, HSO96, Gib97a] report the number of job arrivals for each hour on the iPSC/860, the SDSC Paragon, the CTC SP/2, and the Lewis network of workstations at NASA. Except for the Paragon workload [WLF$^+$96], the other systems all have similar job arrival patterns; that is, a peak arrival rate period between 8am and 6pm, a low arrival rate period before 8am, and an intermediate arrival rate period after 6pm on weekdays. The number of arrivals for each hour between 8am and 6pm on weekends is significantly lower than that during weekdays. They notice the peak and low arrival periods on weekdays, but not the intermediate arrival period. The arrival rate for each hour in the Paragon workload differs in that there is only a two-hour peak arrival rate period (2-4pm) on weekdays, and no particular pattern for the number of jobs during other hours. Furthermore, the number of jobs for each hour on weekends is similar to that on weekdays, except during the two-hour peak period.

Two previous papers focus on the peak memory usage for CM-5 [Fei97a] and T3E [SSN99]. Both papers report the observed distribution of peak memory usage per processor for each job and find that a large fraction (20-50%) of the jobs use a small fraction ($<$ 15%) of the total memory available on each processor (i.e., 32 MB for the CM-5, and 128 MB for the T3E). They also attempt to characterize the relationship between the peak memory usage and the actual runtime. [Fei97a] reports that the average peak memory usage for jobs, partitioned according to their actual runtimes, increases with the actual runtimes. [SSN99] reports the average peak memory usage per processor for sets of jobs that have particular ranges of actual runtime, and find that average peak memory usage per processor is larger for jobs that have longer actual runtimes. [Fei97a] also report the observed cumulative distributions of memory usage (both per processor and in total) for each number of requested processors. Their results show that the weight of the distribution moves to higher total memory usage values for larger numbers of processors. In addition, they report the observed distribution of the memory usage as a fraction of the requested memory. A significant fraction (about 15%) of the jobs use more memory than requested; the distribution for the remaining jobs is approximately uniformly distributed between 0 and 100%.

[LKK99] provides the only information on the job memory usage for shared-memory workload. They report the distribution of the memory usage as a fraction of the total memory available on the system; a large fraction ($>$ 40%) of the jobs use less than 1% of the total memory available on the two NAS O2K systems.

[FW98] provide the only information on job requested runtime. They plot the distribution of the ratio of the actual runtime to requested runtime of all jobs. They find that about 16% of the jobs use 99-100% of the requested runtime, but nearly all of them are killed due to exceeding the requested runtime; for the remaining jobs, the actual runtime as a fraction of the requested runtime is approximately uniformly distributed between 0 and 99%. They provide a hypothetical model in which the value of the requested runtime divided by the actual runtime is generated by the uniform distribution (with the lower bound equal to 1 and a given upper bound). This model is also used in another paper [ZFMS00] for generating the requested runtime from the actual runtime. The relationship of the ratio of the actual runtime to the requested runtime with other job measures for creating synthetic workloads has not been examined.

2.2.3 Summary of Previous Workload Studies

There are many previous workload characterization results for production systems. However, these results do not provide sufficient data for generating synthetic workloads. In particular, the distribution of requested runtime conditioned on other job measures is missing; the distribution of requested memory for shared-memory workloads is also missing; and there is inconsistency in the results provided by two papers (i.e., [Hot96, DF99]) that examine the relationship between the actual runtime and the number of requested processors in the CTC SP/2 workload.

Furthermore, all previous workload studies aggregate the jobs from multiple days and months, without investigating (or providing data to show) whether such aggregation is statistically sound. For example, jobs submitted on Mondays may have different characteristics from jobs submitted on Fridays. Two papers [FN95, WLF+96] delineate the workload into system-defined day and night time periods, but do not examine whether aggregation within each period is statistically appropriate.

2.3 Evaluation of Non-Preemptive Scheduling Policies

Parallel job scheduling policies can be classified into non-preemptive and preemptive policies. This section reviews non-preemptive policies. Preemptive policies are reviewed in Section 2.4.

The simplest scheduling policy is perhaps the non-preemptive strict FCFS (i.e., first-come-first-serve), which allocates each job as many processors as it requests, and stops scheduling if the first waiting job can't be started due to insufficient free processors. The strict FCFS suffers from a serious problem; i.e., the first large waiting job could potentially block jobs that arrive later for a long time. On the other hand, FCFS with firstfit [Aid00] (i.e., in which jobs can be started out of order), has the problem of starving large jobs.

One solution to the problem with strict FCFS is to use adaptive initial processor allocations, Under such policies, the number of processors allocated to each job is determined when the job is started rather than at the compilation time. This added flexibility allows a job to start as long as there are free processors and sufficient memory, thus avoiding the head-of-line queue blocking problem. Another solution is to use *backfilling*, which has been proposed and implemented to improve strict FCFS on an SP/1 at ANL [Lif95]. FCFS-backfill is widely used on many SP/2 systems (e.g., [Lif95, SCZL96, AAN+98]).

Non-preemptive policies with adaptive initial processor allocations are further reviewed in Section 2.3.1. The results for backfill policies are reviewed in Section 2.3.2. Finally, Section 2.3.3 reviews non-preemptive policies that use limited preemption and restart to provide better turnaround time for short jobs.

2.3.1 Space Partitioning with Adaptive Initial Processor Allocations

The requirement for adaptive initial processor allocations is that each job be configured so that it can run on different numbers of processors initially. Such jobs are called *moldable* in [Fei97b]. The key question under such policies is how many processors to allocate to each job.

Several papers have suggested or assumed that it is beneficial to allocate each job a number of processors based on job execution characteristics, such as the average parallelism [Sev89], the knee of the execution time and efficiency profile [EZL89, MEB91], or equivalently the processor working set (independently derived in [GST91]). The *efficiency* of a job running on a particular number of processors is defined to equal the speedup achieved on the processors (relative to the time for the job to complete if allocated only one processor), divided by the number of processors. The speedup is assumed to be 1 on one processor, and the efficiency on more than one processor is generally ≤ 1. The measures of both average parallelism and processor working set for each job provide the information of cost-efficient operating points for the job. In particular, [EZL89] prove that allocating each job a number of processors equal to the average parallelism or processor working set of each job achieves at least 50% of the maximum possible speedup and at least 50% of efficiency A key conclusion from these studies is that policies that allocate each job no more than its processor working set or average parallelism outperform policies that do not have such constraints at moderate to high loads. Another key conclusion is that the initial number of processors allocated to each job should decrease as the load increases.

Setia and Tripathi [ST91, ST93] propose an adaptive static partitioning policy (ASP), which does not use job execution characteristics, but adapts the initial processor allocations to the load by partitioning the free processors as equally as possible among the waiting jobs (under the constraint that no job is allocated more than it requests). Several papers [SRDS93, RSD$^+$94, DY98] propose variants of ASP, including adding a maximum number of processor allocations for each job (i.e., ASP-max). In our previous work [CMV94], we show that ASP-max has lower mean response time than particular processor-working-set and average-parallelism based policies for moderate to high loads under particular workloads where the coefficient of variation of the total processing time is greater than 1. The key conclusion of these studies is that when there is a high variation in the job execution time, it is more important to adapt the initial processor allocations to the load than to allocate the number of processors based on job execution characteristics. On the other hand, [Dow97a] shows that policies with adaptive initial processor allocations in which the processor allocations have an upper bound equal to the processor working set (or average parallelism) outperform ASP for a particular workload model. Their explanation is that ASP is sensitive to short-term fluctuation in the system load.

2.3.2 Space Partitioning with Backfill

As mentioned earlier, FCFS-backfill has been proposed as an improvement to strict FCFS [Lif95]. Specifically, FCFS-backfill schedules jobs in order of the job arrival times, and allocates each job as many processors as requested. The backfill comes into play when there are not enough free processors to start the first waiting job. The scheduler then reserves the free processors for the waiting job, while backfilling the smaller jobs (with lower priority) on the idle processors as long as they will not interfere with the expected start time of the first job. Backfill scheduling will be illustrated and further explained in Section 3.2.1.

The backfill scheduler requires each job to submit an estimated runtime, in order to compute the expected start time for the highest-priority waiting job and determine whether a job can be backfilled on the idle processors. However, most papers that study backfill policies assume that the actual runtime is known to the schedulers. FCFS-backfill is shown to improve the performance of FCFS in terms of the average wait time [SGS96, Gib97b], average slowdown [FW98, ZK99], and processor utilization [JN99].

A different style of backfill, called 'conservative' backfill, is proposed for FCFS in [FW98]. This policy gives a scheduled start time to each waiting job, rather than only to the first job. The motivation is to ensure that the scheduling of a job will not delay the expected start time of any jobs that have an earlier arrival time, and to provide a predicted waiting time for each job. They show that conservative backfill does not affect the overall average slowdown for several job traces where the requested runtime is not available and thus the actual runtime was used, and also a job trace where the requested runtime is available and was used in the simulation. However, we note that in general, users make poor estimates for the job runtimes, as also noted in [FW98]. In fact, [PK00] provides data showing that under FCFS with conservative backfill, the predicted waiting time (given by the initial scheduled start time) is more than 16 times longer than the actual waiting time on average. Furthermore, only 15% of the jobs have predicted waiting times within 167% of the actual waiting times. They conclude that scheduled start times are inaccurate as a predictor of when jobs will run.

Many papers propose different approaches to improve FCFS-backfill, including using non-FCFS priority functions, allowing more flexible deadlines for the scheduled start times, and using more accurate requested runtimes. These approaches are reviewed below.

Only three non-FCFS priority functions for the backfill policies have been compared with FCFS-backfill in the previous work. They are shortest-job-first (SJF), largest-processors-first, and smallest-processors-first.

SJF-backfill has been shown to improve FCFS-backfill, with 50% lower average slowdowns [ZK99, PK00] for several production workloads. On the other hand, [ZFMS00] did not find such improvement for a particular synthetic workload model. Specifically, they show that the system under either SJF-backfill or smallest-processors-first (called "worst-fit" in their paper) with backfill saturates at a cpu load of about 85%, while that under FCFS-backfill does not. Furthermore, at high load (> 90%), FCFS-backfill has significantly (more than 50%) lower average waiting time than largest-processors-first (called best-fit in their paper) with backfill. Except for [PK00], the other two studies assume that schedulers have information of the actual runtime.

One study [LKK99] takes the job memory requirement into account, and proposes to extend FCFS-backfill by choosing the next job for backfilling so as to achieve the most balanced allocation of processors and memory. For a real trace and several synthetic workloads, they show that a particular resource balancing priority algorithm called backfill-balanced improves on FCFS-backfill not only for the mean response time (by at least 15%) but also for the mean response time weighted by the product of the job execution time and the number of processors, They also assume that information on the actual runtime and actual job memory usage is known to the scheduler.

The paper in [TF99] proposes a version of FCFS-backfill that allows a non-zero upper-bound on the delay in the initially assigned scheduled start time for each job. The idea is to allow a larger number of small jobs to be backfilled. They show that using a flexible deadline improves the average waiting time for FCFS-backfill by 10-20% for most of the twelve one-month traces studied. The user-supplied requested runtime is used in their simulation. Their paper also studies priority schedulers based on political considerations (e.g., favoring users who pay more for the use of the computers) for backfill policies.

Finally, several papers study the impact of more accurate runtime estimates for backfill policies. [FW98] shows that FCFS-backfill using the actual runtime results in about 25% lower average slowdown than FCFS-backfill using the user-supplied requested runtime, for a particular one-year trace. On the other hand, using a hypothetical model for the overestimation of runtimes results in a similar average slowdown to that obtained using the actual runtime under FCFS-backfill, for most of the workloads they study. In particular, they use the uniform distribution to model the overestimation in the runtime (i.e., the ratio of the requested runtime in excess of the actual runtime, divided by the actual runtime); overestimation by as much as 300 times the actual runtime is simulated. [ZFMS00] shows that this hypothetical model of requested runtime for FCFS-backfill results in similar average waiting time, compared to that obtained using actual runtime; nevertheless, jobs that have more accurate runtime information incur lower waiting times than jobs with larger overestimation. [JJS01] shows that the utilization for a particular priority backfill policy is improved by 10% when the accuracy in the requested runtime is increased from 10% to 40%, but no further improvement is achieved beyond 40% accuracy. Note that, however, the accuracy is not defined in that study.

2.3.3 Limited Preemption and Restart

One serious drawback to the non-preemptive space sharing policies discussed in the previous two sections is that if there are not enough free resources to satisfy the job resource requests, a new job may have to wait for a long time in the queue, even if the job runs for only a short time (e.g., under 1 minute). Backfill policies may not be able to backfill these short jobs soon, because of generally overestimated requested runtimes. This motivates the use of limited preemption for ASP in [CMV94] and a non-preemptive 15-minute test run and restart for FCFS-backfill and SJF-backfill in [PK00]. Related work in [HB00] proposes multi-level queueing with restart, in the context of more specialized distributed-server systems. This work is motivated by the heavy-tail property in the distribution of job execution times observed on Unix sequential systems [HBD96]. No prior work has proposed the use of limited preemption for backfill policies.

2.4 Dynamic Scheduling Policies for Parallel Systems

The uniprocessor processor sharing policy, as discussed in Section 1.1, can be applied for parallel systems to achieve high performance. In particular, previous work shows that equi-allocation policies that allocate an equal processing power per job outperforms the policies that allocate equal processing power per process when the coefficient of varion of the execution time $>$ 1 [LV90, GL99].

Parallel equal allocation can be implemented through either time-sharing (i.e., EQT) or space-sharing (i.e., EQS). For efficient interprocess communication, EQT requires support for scheduling and preempting cooperating processes running on different processors at the same time. EQS with runtime support for jobs to adapt to changes in processor allocations ensures that different processes of the same job are always scheduled simultaneously.

Here we use the term EQT to refer to any preemptive policy that gives an approximately equal quantum of time to each job by time slicing the computer among the jobs, and has support for co-scheduling processes that must be scheduled at the same time for interprocess communication. Similarly, EQS refers to any preemptive policy that partitions the processors among the jobs as equally as possible, and has runtime support for processor repartitioning. Thus, the above definitions exclude those preemptive policies that time-share or space-share the machine among jobs, but do not have the required runtime support for efficient execution of parallel jobs.

Section 2.4.1 below reviews EQT policies. Section 2.4.2 then reviews EQS policies.

2.4.1 Dynamic Time Sharing Policies

Coscheduling is proposed in [Ous82], and subsequently refined and called Gang Scheduling policy in [FR90]. These policies give each job as many processors as requested, and time-slice each processor among the processes from different jobs. To facilitate interprocess communication, these policies require coordinated context switching of processes across different processors, so that processes of the same job are scheduled and preempted at the same time.

Many papers propose specific implementations or extensions of these policies. Specific implementations for efficiently coordinating the scheduling of processes on different processors are proposed in [MCF$^+$98, SS99]. Implicit (or dynamic) coscheduling policies, achieved by the detection of message exchanges between the cooperating processes, are proposed and studied in [SW95, DAC96, SPWC98, YJ01]. These policies differ from Gang Scheduling in that only those processes that communicate with each other must be scheduled at the same time. Several papers extend Gang Scheduling to take job memory requirements into account [Set97, SSN99, BF00, PF00].

Many papers study the impact of different factors on the performance of Gang Scheduling, in an attempt to further improve the performance of Gang Scheduling. A major problem in Gang Scheduling is that of processor fragmentation; i.e., some processors are left idle in some time slice(s), because there are not enough processors to schedule any other job in another time slice. Note that each process also makes slower progress as the number of time slices increases. To reduce fragmentation, previous papers study alternative job packing schemes [Fei97a, ZJWB98], the benefit of using process migration [Fei97a, Set97, WPS97, SSN99, ZFMS01], and the benefit of employing backfill for Gang Scheduling [ZFMS00, ZFMS01]. [GL99] studies the effect of the quantum allocation for each time slice, and shows that making quantum length inversely proportion to the number of processes that are running in a given time slice results in lower mean response time than allocating an equal quantum to each time slice under particular workloads.

Several papers [FJ97, WPS97, FJM+99, ZFMS00] show that Gang Scheduling policies improve on FCFS-backfill. However, these studies do not take job memory requirements into account.

2.4.2 Dynamic Space Sharing Policies

Tucker and Gupta [TG89] proposed the first dynamic equal space sharing policy (i.e., EQS), which partitions the processors among jobs as equally as possible upon each job departure and arrival. To allow coscheduling of the processes of each job, they propose an implementation called "Process Control", in which each job is informed when processor allocations change, so that the job can adapt the number of processes to match the number of allocated processors. Thus, EQS requires that each job adapt to changes in processor allocations, and therefore incurs processor repartitioning overhead. One advantage of EQS over time-sharing policies is that EQS does not have the problem of processor fragmentation. Another advantage of EQS is that the system may have a higher processor efficiency because the jobs may be given fewer processors under EQS than under EQT. Most applications have higher execution efficiency (i.e., the speedup achieved on the processors divided by the number of processors) when running on a smaller number of processors, due to a smaller interprocess communication overhead.

In addition to Process Control, particular implementations of EQS have been proposed to improve performance, or to accommodate special requirements in the workloads. [ZM90] proposes more dynamic EQS policy, which not only responds to the load but also responds to the change in job parallelism by reassigning processors among jobs as needed (i.e., taking away extra processors from a job that has reduced parallelism, and reassigning to another job that can use more processors). In the case where each job can run only on power of two processors, [MZ94] propose Folding, which always allocates each job numbers of processors equal to power of two and, achieves equal sharing by rotating small and large partitions among jobs. In the context of distributed-memory systems, they propose an EQS that implements minimum processor allocations imposed by job memory requirements to avoid memory overhead [MZ95].

Many run-time support systems for the processor repartitioning required under EQS have been implemented. These include Cilk [BJK+95, BL97], DRMS [MNK96, JEM97], Octopus [IPS+97, IPSG97], automatic self-allocating threads [SE97], WoDi [PL95], and data redistribution [CNSW97].

When the coefficient of variation of the total processing time is greater than 1, many papers show that EQS has better performance than non-preemptive policies, such as ASP [ST91, CMV94, SST93, NSS93b, NSS93a, PD96, PS97], and policies that allocate a number of processors based on processor working set [ST91, CMV94, SST93] or average parallelism [ZM90].

EQS policies have been shown to have higher performance than EQT policies for workloads where jobs have poor execution efficiency when running on a large number of processors [GTU91, MVZ93, YL95]. This is because under EQS, each job may be allocated a number of processors smaller than requested at moderate to high loads (as discussed above), resulting in better execution efficiency both for the jobs and for the system under such workloads. However, we note that for workloads where most jobs can efficiently use the number of processors requested, such as in the O2K workload (to be characterized in Chapter 4), the performance of EQT is expected to be comparable to that of EQS. Other high-performance parallel computer systems are likely to have similar characteristics, as users have an incentive to request numbers of processors that their jobs can efficiently use, in order to get better turnaround time and to spend their quota for using the computers wisely.

2.5 Summary of Previous Policy Studies

The FCFS-backfill policy is widely used on SP/2 multicomputers. Only three priority functions have been compared with FCFS-backfill. None of these priority backfill policies has been shown to outperform FCFS-backfill overall. Specifically, SJF-backfill has been shown to provide significantly lower average slowdown of FCFS-backfill [ZK99, PK00]. However, SJF-backfill has the potential of starving jobs with long requested runtime, and attempts to avoid starvation by placing a limit on the waiting time of each job compromise the performance [ZK99]. The smallest-processors-first and the largest-processors-first policies with backfill have been shown to have significantly worse average waiting time than FCFS-backfill. A few previous papers [FW98, ZFMS00, ZFMS01] have studied the impact of overestimated runtime on backfill policies, and conclude that the impact is minimal. However, no previous work has studied the impact of more accurate runtime on priority backfill policies that favor short jobs. Note that more accurate requested runtime has the potential to be beneficial for priority backfill policies that give priority to short jobs.

A starvation-free priority function for backfill policies that favor short jobs has not yet been proposed or studied. Furthermore, most previous studies assume that the actual job runtime is known to the schedulers and most studies (except [LKK99]) do not take the job memory into account.

Preemptive EQT and EQS have both been shown to have high performance. In particular, EQT has been shown to outperform FCFS-backfill; however, job memory is not taken into account. Furthermore, EQT has not been compared against priority backfill policies, and no prior papers have compared EQS against backfill policies.

Chapter 3

Evaluation Methodology

Our methodology for evaluating job scheduling policies consists of characterizing the O2K workload that will be used to evaluate the policies, evaluating and tuning the performance of the schedulers that dispatch the jobs on the NCSA O2K, and designing improved alternative schedulers that can be used for the O2K (as well as for the other production parallel computers). The baseline policy that will be evaluated is NCSA-LSF*, which was the production job scheduler on the O2K prior to July 2000. We also evaluate priority backfill policies, and using trace-driven simulation with the O2K workload. show that our initial backfill policy, called Priority-backfill, significantly improves NCSA-LSF*. Based on these results, a priority backfill policy, called NCSA-MS*, configured with a similar priority function as that in Priority-backfill, replaced NCSA-LSF* in July 2000. We further evaluate alternative priority measures for backfill policies to determine whether they improve performance. In addition, several other approaches to improving the scheduler are evaluated; namely (1) adding limited preemption for backfill policies, (2) using the preemptive spatial equal allocation (i.e., EQspatial) policy, and (3) using improved requested runtime estimates for backfill policies.

Section 3.1 defines NCSA-LSF*. Other scheduling policies evaluated in this thesis are defined in Section 3.2. Section 3.3 discusses our approach for characterizing the O2K workload. Section 3.4 describes our trace-simulation method and the simulator we develop. The performance metrics used to evaluate the scheduling policies are discussed in Section 3.5. Section 3.6 provides the performance evaluation and some of the tuning results for NCSA-LSF*. Section 3.7 provides the results that validate the performance of our simulated NCSA-LSF* and

Table 3. NCSA O2K LSF Job Class Definitions

Class Name	Resource Request Limits		Drain Wait Time
Time	CPU Time Per Processor	Run Time Per Job	
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)	$\leq$ 8	$\leq$ 2 GB	
mj (medium)	$\leq$ 16	$\leq$ 4 GB	
lj (large)	$\leq$ 64	$\leq$ 16 GB*	

(* $\leq$ 25 GB in October-December 2000)

NCSA-MS* against the measured performance on the O2K.

3.1 Baseline Policy: NCSA-LSF*

This section describes the scheduling parameters and algorithm of NCSA-LSF*. Table 3 provides the definition of LSF job classes. Table 4 summarizes the LSF scheduling parameters; the processors and memory available on each host are repeated from Table 1 for convenience.

When submitted to NCSA-LSF*, each job requests a number of processors, an amount of memory, and the cpu time in average per processor. According to each job's requests, NCSA-LSF* classifies that job into one of the four time classes (i.e., vst, st, mt, and lt) and one of the three size classes (i.e., sj, mj, and lj). For example, a job that requests 16 processors, up to 4 GB of memory, and 50 hours of cpu time on average per processor (or equivalently, 50×16 hours of total cpu time) is in the st-mj class. When a job is started, it is allocated as many processors and as much memory as requested, and occupies the allocated processors and memory during its entire execution.

Each of the eight LSF hosts is configured with job priority and class limits. There are also system-wide job limits on the mt and lt classes to avoid long jobs dominating the resources;

Table 4. LSF Host Scheduling Priority on NCSA Origin2000

Host Name	Number Processors	Memory		Scheduler Configuration	
		Total (GB)	Per Processor (MB)	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
hod1	128	64	512	lj≤6; mj≤2	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
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)					

specifically, at most 48 mt jobs and 20 lt jobs can run at the same time in the system, as shown in Table 4. For example, the host "eir" can run up to 3 lj jobs and 4 mj jobs at the same time. The jobs in the vst-lj class have the highest priority on the host, followed by the other lj jobs, subject to the class limits. If three jobs in the lj class are already running on the host, or if no jobs in the lj class are available for scheduling, the jobs in the vst-mj class have the next highest scheduling priority on the host, followed by the other mj jobs. Similarly, if 3 lj jobs and 4 mj jobs are running on the host or no mj and lj jobs are available for scheduling, the jobs in the vst-sj class have the highest scheduling priority on the host, followed by the other sj jobs. Within each priority class, the jobs are scheduled in order of arrival time. Note that one host (mimir) schedules sj jobs only, and one host (modi2) schedules mj jobs only; all of the other six hosts give priority to lj jobs.

When a job departs from a host, LSF schedules the waiting jobs according to the priority and class limits described above. A job is skipped if not enough free processors and memory are available to satisfy the job's requests. If this job has the highest size class priority (e.g., lj priority), then other jobs in the same size class (i.e., lj) can be started on the host, but not jobs in the other size classes. However, vst priority jobs do not block the scheduling of jobs from other time classes. When a job arrives, if more than one host has enough free processors and memory (and the required priority and quota) to start the job, the job is started on the host with the largest number of free processors.

If a job has been waiting longer than the per-class "Drain Wait Time" (defined in the last column in Table 3), the job is given a draining priority on a randomly chosen host that does not already have a job with such priority. Thus, at most one host can be drained for each such job, and each host can be drained for at most one job. A job with draining priority on a host is started when enough cumulative free resources on that host have become available (or when another host can start the job). As long as a job has draining priority on a host, no other jobs can be scheduled at that host. However, a job looses draining priority if the appropriate limit for the time class of that job has been reached while the job is waiting. Also, if the size class of a job no longer has scheduling priority on a host while that job is still waiting, a different host that is not drained for any job is used to give the job another draining priority.

Note that host draining does not occur frequently, because of the high wait time threshold for draining priority. The measured total idle processor time due to draining on the eight hosts for each month is under 4% of the total processor time available in that month for the O2K workloads studied in this thesis.

3.2 Other Policies Evaluated

This section continues to define other job scheduling policies evaluated in this thesis. Section 3.2.1 defines the backfill policies. The preemptive immediate service option for the backfill policies is defined in Section 3.2.2. The EQspatial policy is defined in Section 3.2.3.

3.2.1 Backfill Policies

Backfill policies compute the priority of each job as a weighted sum of several job metrics, such as the current job waiting time, the estimated current job expansion factor (i.e., the sum of the

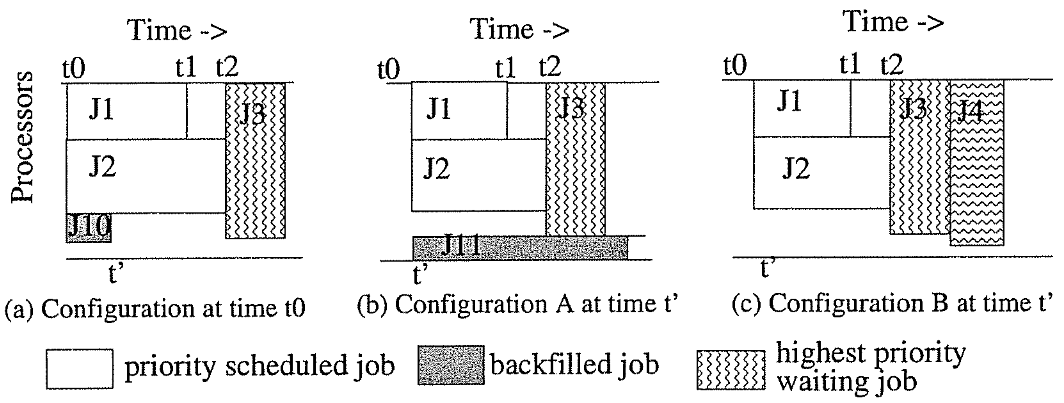

Time ->
Processors
t0
t1
t2
J1
J2
J3
J4
J10
J11
t'
(a) Configuration at time t0
(b) Configuration A at time t'
(c) Configuration B at time t'
priority scheduled job
backfilled job
highest priority waiting job

Figure 1. Example Backfill Schedules

elapsed time since the job was submitted and the job requested runtime, divided by the job requested runtime), etc. This section describes the general algorithm for backfill policies. The priority functions used in the priority backfill policies evaluated in this thesis will be defined in Section 5.1.

Under the non-preemptive backfill policies evaluated in this thesis, when a job is started, it is allocated as many processors and as much memory as requested, and occupies the allocated processors and memory during its entire execution, as in NCSA-LSF*. Instead of draining as in NCSA-LSF*, backfill policies allow small jobs that have lower priority to use the processors and memory that would be otherwise idle, as illustrated in Figure 1.

For the purpose of illustration, let's consider the processor allocations only and assume the FCFS priority, i.e., the jobs are prioritized in the order of their arrival time and thus the job priority never changes. In the figure, the jobs are numbered in the order of their arrival time. Suppose Job J3 can't be started at t0 and is given a scheduled start time to occur at time t2, i.e., the processors required to run J3 at t2 are reserved for as long as the estimated runtime of the job. At time t0, Job J10 is backfilled because it is the highest-priority job that can run on the idle processors without delaying the scheduled start time of J3. At time t', if J3 is the only job with a scheduled start time, Job J11 is backfilled, as shown in Figure 1(b). However, if both J3 and J4 are given a scheduled start time as shown in Figure 1(c), Job J11 can't be backfilled at

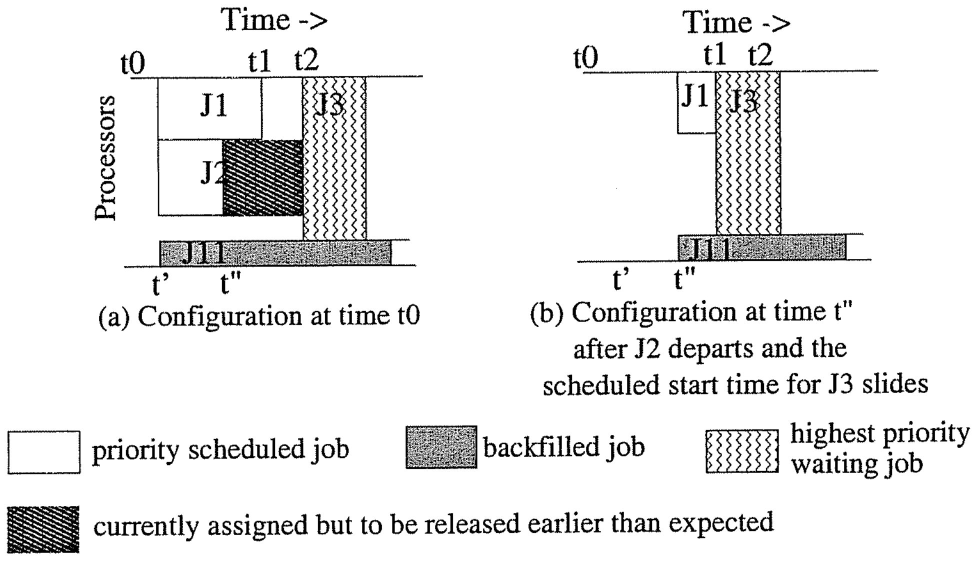

Time ->
t0
t1
t2
Processors
J1
J2
J3
J11
t'
t"
(a) Configuration at time t0
Time ->
t0
t1
t2
J1
J3
J11
t'
t"
(b) Configuration at time t" after J2 departs and the scheduled start time for J3 slides
priority scheduled job
backfilled job
highest priority waiting job
currently assigned but to be released earlier than expected

Figure 2. Example Sliding Reservation Upon a Job Departure

time t' because it would delay J4's start time. The number of jobs that can be given a scheduled start time is a parameter, N, of the backfill policies. Previous proposed backfill policies either use N = 1 (i.e., the scheduled start time is given to the first job only) [Lif95, LKK99] or N = ∞ (i.e., each waiting job is given a scheduled start time) [Gib97b, FW98, TF99, ZFMS00]. Both values of N are studied in [PK00]. Note that the motivation of using N = ∞ is these previous papers is to ensure that no job delays the start time of any jobs that have a higher priority.

In the case the job priority can dynamically change, the reservation can be *dynamic-job* or *fixed-job* (called "CURRHIGHEST" and "HIGHEST", respectively, in the Maui Scheduler). With the dynamic-job reservation, the scheduled start time is always given to dynamically the current N highest-priority jobs. On the other hand, under fixed-job reservation, once a job is given a scheduled start time, the job does not release the scheduled start time until it is started, even if the job is no longer one of the N highest-priority jobs. Note that under FCFS-backfill, fixed-job reservation is equivalent to dynamic-job reservation, as the job priority never changes.

When a job departs earlier than expected, the existing scheduled start times are rescheduled to occur sooner (which can be the current time, i.e., the job can start now). As illustrated in

Figure 2, Job J2 departs at t" (earlier than the expected complete time, t2), and the scheduled start time for Job J3 slides to an earlier time, t1, as shown in Figure 2(b).

Using fixed-job reservation can potentially reduce the worst waiting time of that using dynamic-job reservation for priority backfill policies, especially SJF-backfill, that gives the highest priority to the job with the shortest requested runtime. Using $N > 1$ can also potentially reduce the worst waiting time of that using $N = 1$. Note that with both $N = \infty$ and fixed-job reservation, any non-FCFS backfill policy will use the FCFS order to give the scheduled start time to the jobs, and only use the priority to backfill the jobs.

The default reservation used for the backfill policies evaluated in Chapter 5 is dynamic-job with $N = 1$ reservation (i.e., only the job with dynamically the highest priority is given the reservation). We also evaluate $N > 1$ and fixed-job reservation in an attempt to improve the performance of SJF-backfill.

The backfill policies need to accommodate the job memory requirement and the host boundaries on the O2K. We augment backfill policies as follows:

- A job can be started on a host only if there are sufficient free memory as well as free processors to satisfy the job's requests.
- If more than one host can start a job, the host with the smallest free processors is chosen to start the job. The motivation is to minimize the processor fragmentation so as to increase the chance the jobs that request a large number of processors can be started.
- When a job is given a scheduled start time, the reservation is made on the host that is expected to start the job at the earliest time, computed according to the requested runtimes of all currently executing jobs.

3.2.2 Backfill Policies with Limited Preemption for Immediate Service

Similar to the other production workloads, a significant fraction of the O2K jobs terminate in a very short time (e.g., under 1 minute), including the jobs that request a very long runtime (e.g., over 50 hours), to be shown in Section 4.5.1. To provide a good response time for these very short jobs, we propose to extend the backfill policies with preemptive immediate service of a short quantum.

With the preemptive immediate service, if a new job can't be started as a full job, the job is given a preemptive priority to start on as many processors as requested and an initial memory for a very short quantum. The initial cpu quantum and initial memory are the parameters of the policy. The initial quantum should be short enough to not have adverse impact on longer jobs but long enough to help a significant number of jobs. The values we choose are motivated by the characteristics observed in the O2K workload (to be presented in Chapter 4, and are discussed in Chapter 5.5 as we evaluate the performance of immediate service.

Below further explains the algorithm of giving an immediate service to a new job and the parameter t.

If a host has enough free (i.e., unoccupied) processors and the initial memory to give an immediate service to the new job, the job receives an immediate service on the host without preemption. If more than one such hosts are found, the host that has the smallest number of free processors is chosen. Otherwise, the currently executing jobs are chosen to be preempted as follows. To avoid memory overhead, only the processors (but not the memory) of the currently executing jobs can be preempted, i.e., a job vacates all the processors but occupies the memory allocated when it is preempted. An executing job is eligible to be preempted if (1) it has been running without interruption for longer than t, and (2) its expected remaining runtime is longer than t. For each host that has at least an amount of the initial memory, each job eligible for preemption is considered in the order of increasing estimated current job *slowdown*, which is the ratio of the elapsed time since the job arrived divided by the time it has actually run so far (i.e., the time it has been preempted is excluded). As many such jobs on the host as required are selected, until the sum of the total number of processors occupied by these jobs plus the currently free processors on the host is at least as large as requested by the new job. Note that if the number of these processors is greater than requested, some of the selected jobs may be de-selected to avoid unnecessary preemption. The host that will have the fewest number of idle processors due to preemption is chosen to give a preemptive immediate service to the new job. A preempted job is assumed to completely stop execution.

If the new job completes in the initial quantum and the initial memory, it leaves the system. Otherwise, it is killed and waits in the queue to be scheduled as a full job.

In our evaluation in Section 5.5, t equals 10 times the initial quantum (e.g., $t = 10$ minutes if the initial quantum is 1 minute). Thus, the total delay incurred by any job due to preemption is under 10% of its actual runtime. We expect that the preemptive immediate service option will not adversely impact the long running jobs. Our comparisons of backfill policies in Section 5.5 will confirm that this is the case.

3.2.3 The EQspatial-MinInterval Policy

The previously proposed equal space partitioning policy (i.e., EQspatial) [TG89] dynamically repartitions the processors as equally as possible among the jobs, upon each job departure and arrival. We extend this policy to accommodate the job memory requirement and host boundaries on the O2K and to have reduced processor repartitioning overhead, described below.

As in all other policies evaluated in this thesis, EQspatial allocates each job as much memory as it requests, to avoid memory overcommitment.

The key algorithms of EQspatial are the host placement policy and the immediate service.

The host placement policy determines which host to schedule a new job as follows. Only the hosts with enough free memory to satisfy the job requested memory are considered. If more than one such hosts are found, the job is scheduled on the host that has the largest ratio of the total number of processors available on the host divided by the total number of requested processors of all executing jobs on the host and the new job. We've examined several other more complex host placement heuristics and found that this simple heuristic results in least deviation from the 'true' equi-allocation (computed as if there is no host boundaries).

If no host has enough free memory to satisfy the request, the job is given an immediate service and an initial memory for a short quantum on the host that has at least the required initial memory, as in the immediate service option for the backfill policies. If the job does not finish in the initial quantum and memory, it is killed and waits in the queue for enough free memory.

During the time while a job is waiting in the queue, the cumulative processor time the job could have received under true equi-allocation is computed. Once the job is started, it is given as many processors as requested until it has received what it should have received while it was waiting in the queue. Note that if there are such jobs running on a host, the processors occupied by these jobs are deducted from the total number of processors available on the host and the requested processors of these jobs are excluded in the total number of requested processors of the jobs on the host, when applying the host placement heuristic defined above.

To reduce the processor repartitioning overhead, a parameter, MinInterval, is used to define the minimum time between any two consecutive processor repartionings on each host. If a new job is scheduled on a host on which the processors cannot be repartioned due to the MinInterval constraint, the job can run on the currently free processors (if any) until the processors can be repartitioned again. Similarly, the processors released by a job departure on a host are left idle until the MinInterval for the host is satisfied. We use 5 minutes for MinInterval (i.e., EQspatial-5m), which should have negligible processor repartitioning overhead for current parallel computers.

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 above assumption of the job execution efficiency although results in a more pessimistic performance for EQspatial, it is consistent with the characteristics of the O2K workload, in which the overall efficiency on the allocated processors is about 80% each month and that a large fraction (60%) of the jobs have high processor efficiency (> 80%) on the number of the processors requested (and allocated), to be shown in Chapter 4.

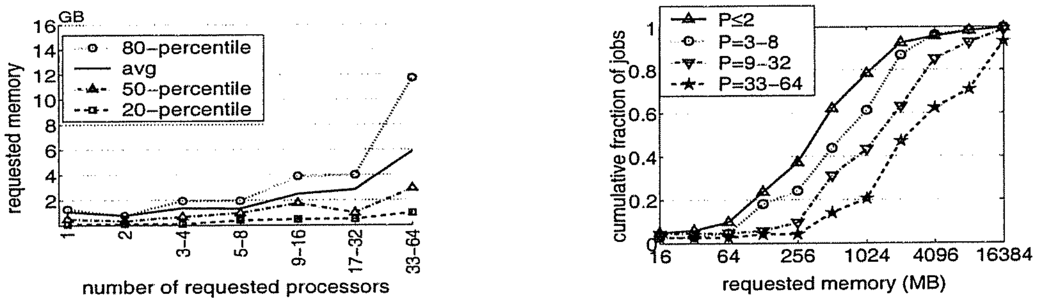
In the case if each job can only run on a power of two number of processors, EQspatial can be designed to accommodate the workload feature as proposed in Folding [MZ95] (Section 2.4.2).

To use EQspatial, the existing applications require recompiling or even more costly conversion, to use the runtime support for processor repartioning. Our goal of evaluating the performance of EQspatial is to study quantitatively how much it can improve the backfill policies, which helps us to determine whether or not it's worthwhile to implement the required support.

3.3 Workload Characterization

Realistic workload is an important part for evaluating the scheduling policy performance. While it is now very common to use the job traces from the production workloads for scheduling policy evaluation, the advantage of the synthetic workloads is to allow the flexibility for varying the key workload parameters for the expected workloads in a systematic way. However, as discussed in Section 2.2, no sufficient data is provided in previous studies for generating realistic synthetic workloads. One of the goals of this thesis is to provide such data.

Six one-month O2K workloads, during January - March and October - December 2000, are analyzed. We identify the periods of relatively stable arrival rate and characterize the distribution of job interarrival time and the characteristics of the jobs that arrive in each period. For each period of relatively stable job arrival rate, we analyze the key job characteristics, which are three job resource request measures (i.e., the requested number of processors, requested memory, and requested runtime), and two job resource usage measures (i.e., the actual runtime and memory usage). The characteristics of several other new and important job measures, such as per-job processor utilization, are also provided.



(a) Requested Memory vs. Requested Processors

(b) Conditional Distributions of Requested Memory (P: requested number of processors)

Figure 3. Example Conditional Distribution Analysis

To generate the synthetic workloads that have the observed characteristics of the O2K workload, the relations between the key job characteristics are required. We characterize the distribution of the requested number of processors, and use that as the first-order job measure. The distributions of each of the other job measures is characterized conditioned on the job measures for which the distribution or conditional distributions are already characterized. As an example, Figure 3 illustrates our approach to deriving the distributions of the requested memory conditioned on the number of requested processors. First, the jobs are partitioned according to the number of requested processors of the jobs. Since most of the jobs request a power of two number of processors (to be shown in Section 4.4.1), it's natural to partition the jobs using the power of two processors, i.e., 1 processor, 2 processors, 3-4 processors, 5-8 processors, etc. For each set of the jobs, we examine the distribution of the requested memory (measured by the 20, 50, 80-th percentiles, and the average of the requested memory), as shown in Figure 3(a). Different sets of the jobs that have a similar distribution of requested memory are identified. For example, the jobs that request 1-2 processors have an *approximately* similar distribution of requested memory, the jobs that request 3-8 processors have a similar distribution of requested memory, etc. Based on the results, the observed distributions of the requested memory are provided for four different sets of the jobs, as shown in Figure 3(b). The workload generator can use the provided conditional distributions in Figure 3(b) to determine the requested memory after the number of requested processors of the job has been determined according to the observed distribution of the number of requested processors.

The conditional distributions of the other job measures are analyzed using the same approach, except that the distributions are conditioned on more than one job measure. For example, the distribution of the requested runtime is conditioned on both the requested number of processors and requested memory.

In order to keep the workload model manageable and the characterization possible, approximation is used to identify the sets of the jobs that have a similar distribution of a particular job measure. In addition, unless there is data to suggest otherwise, a set of only a few number of jobs is often analyzed with an adjacent set of the jobs, because the observed distribution based on a few number of jobs can be very unreliable due to the statistical fluctuation in a small number of jobs.

3.4 Trace-Driven Simulation and the Simulator

We evaluate the job scheduling policies by trace-driven simulation using six one-month workloads derived from the O2K job logs, during October-December 1999 and January-March 2000.

Note that three more recent one-month workloads (i.e., October-December 2000) characterized in Chapter 4 are not used for policy evaluation, because our simulation of the policies was performed prior to October 2000. Nevertheless, since we find that the workload does not change from January-March 2000 to October-December 2000, the policy performance evaluation results would be the same for the more recent three months.

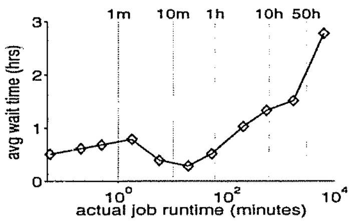
Only the space-shared hosts and the jobs that run on the space-shared hosts are simulated. Thus, the jobs in the vst class that run on the dedicated hosts (when there are no dedicated jobs available for running) are not included in our simulation.

For each month, the simulation of each policy starts a week in advance of the month to remove the 'cold-start' effect. Only the jobs submitted during that month are analyzed to compute the performance measures for the policy. Under non-preemptive policies, a simulation is complete when all the jobs submitted during the month have been started; under policies that allow the jobs to be preempted (i.e., backfill policies with limited preemption and the EQspatial policies), the simulation is complete when all the jobs submitted during the month have completed.

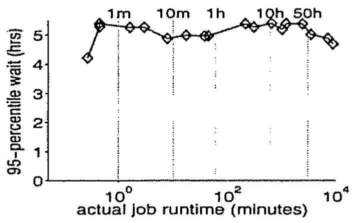
We develop the simulator using the Java programming language. The inputs to the simulator are: the month to be simulated, the policy to be simulated with the scheduling parameters. The NCSA-LSF* policy has additional inputs that specify the host priority and job limits. The simulator reads the appropriate job traces with a known format to the simulator, according to the month specified. The simulation is event-driven; the events include the next job arrival and the next job departure for any policy. The limited preemption has an additional event that wakes up the scheduler when a job has become eligible to be preempted by a new job that is still waiting for an immediate service. The EQspatial with non-zero MinInterval has an additional event that wakes up the scheduler to perform a delayed processor repartitioning. Each simulation run produces a data file that records the *response time* of each job analyzed (i.e., the total time the job has been in the system since it is submitted), together with the job actual runtime and requested number of processors and memory, etc. The results of the output files are processed by the programs written in the Matlab programming language to produce the figures for each performance metric evaluated.

3.5 Policy Performance Metrics

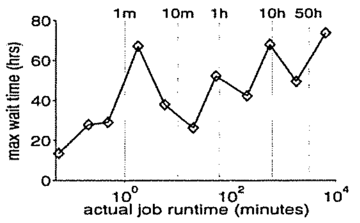
Similar to previous work, we evaluate the overall average waiting time and overall average *slowdown* (which is defined to be the response time divided by the actual runtime) over all jobs for each policy. However, these overall average measures are incomplete for understanding the policy performance. Thus, we also provide overall 95th percentile and maximum waiting time. Additionally, we show more detail performance by plotting the average, 95th percentile, and



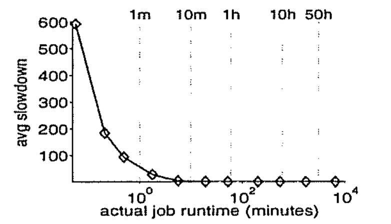
(a) Average Wait Time



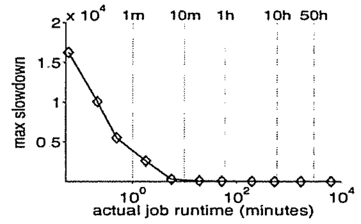
(b) 95th Percentile Wait Time



(c) Maximum Wait Time



(d) Average Slowdown



(e) Maximum Slowdown

Figure 4. Example Plots for Waiting Time Versus Actual Runtime

maximum waiting time, and the average and maximum slowdown, versus the actual runtime. In some cases, the average and maximum waiting time versus different ranges of the requested number of processors and the requested memory are provided, as an auxiliary data.

To illustrate, Figure 4 shows the performance measures versus the actual runtime. Figure 4(a) plots the average of the waiting times in each equal-size logarithmic range of actual runtime against the average of the runtime in that range. Specifically, the jobs are partitioned into eleven disjoint groups according to their actual runtimes; each group is upper-bounded by 10^i minutes, where the value of i is from -1 to 4 with an increment of 0.5. A very small fraction ($< 2\%$) of the jobs each month have actual runtime greater than 10^4 minutes, i.e., 167 hours, and they are placed in the last group. The actual runtime is plotted on a logarithmic scale to reflect the distribution of the actual runtime, i.e., an overwhelming fraction of the jobs have very short run time. For each group of jobs, the average wait time is plotted against the average of the actual runtimes in log scale, which is computed as follows. The average of the logarithmic actual runtimes is computed, and then applied by the inverse logarithm. The maximum wait time versus actual runtime is plotted in a similar way, except that it is plotted against the actual runtime for which the maximum wait time occurs, as shown in Figure 4(c). Figures 4(d) and (e) plot the average slowdown and maximum slowdown versus the actual runtime, the same way as that for the average wait time and maximum wait time, respectively.

Since the maximum wait time is a more sensitive measure, we also provide 95th percentile wait time. Figure 4(b) plots the 95th percentile waiting time versus actual runtime. For this plot, the top 5% of the wait times are first eliminated. To show more about the variation in the 95th percentile wait times in each range of actual runtime while without making the graph too complex to read, we plot more data points (namely three) in each range of actual runtime but for a fewer ranges (namely six) of actual runtime, compared to that for the average and maximum wait time versus actual runtime. Specifically, the actual runtimes are partitioned into six ranges: each of the first four ranges consists of equal-size logarithmic range of actual runtime upper-bounded by 10^i minutes, where i = 0, 1, 2, 3; the last two ranges are the same as that in the other plots (i.e., 10^3 to $10^{3.5}$ and $10^{3.5}$ to 10^4 minutes. The top three wait times (from the remaining wait times) for each of the six ranges of actual runtime are plotted against the actual runtimes for which the waiting times occur.

The plots of performance measures versus actual runtime as shown in Figure 4 reveal important detail of the policy performance that can't be illustrated by the simple overall measures. For example, the plot of the average wait time versus actual runtime shows how the average wait time varies with the increase in the actual runtime; the maximum wait time versus actual runtime shows how poor the wait time is for small as well as large actual runtime, etc.

Note the wait time for each job under non-preemptive policies is the elapsed time since the job arrives until the job is started. For the backfill policies with limited preemption and the EQspatial policies, the waiting time for each job is computed to be the response time of the job minus the actual run time of the job on the requested number of processors.

3.6 Performance of NCSA-LSF*

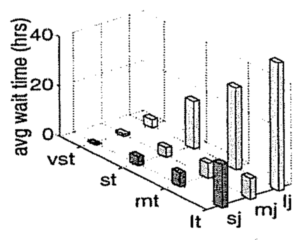
This section provides the evaluation results of our baseline policy, i.e., NCSA-LSF*.

Since under NCSA-LSF*, the scheduling parameters are defined based on the time and size classes, Figure 5 plots the average waiting time of each job class for each of four months. Figure 6 provides the results for the maximum waiting time. One key observation is that the average waiting time is measured by hours and tens of hours for lj classes, and the maximum wait is over 50 hours and even hundreds of hours. The poor waiting time applies to any job, including very short running jobs. To illustrate, Figure 6, plots the largest 5% of the job waiting times against the actual runtime for a typical-load month (January 2000). The figure shows that the poor maximum wait (> 50 hours) applies to any runtime, including two jobs that have very short runtime (< 1 minute).

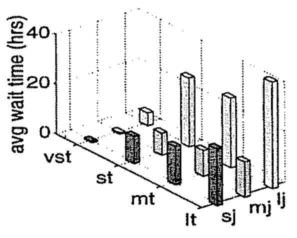
Another observation is that the relative performance of LSF job classes varies from month to month even though the job mix per month is similar (to be shown in Chapter 4). This has made tuning for NCSA-LSF* difficult. For example, it might be appropriate to increase the resource available for 'lj' class for February 2000 to improve their average wait time, but this would not be appropriate for March 2000. Section 5.2 will further show how the performance of NCSA-LSF* varies from month to month, while the performance of Priority-backfill and FCFS-backfill is more similar from month to month.

The scheduling parameters for NCSA-LSF* have been experimentally tuned by NCSA based on our simulation results. It's difficult to further improve its performance. Appendix E provides some of our simulation and analysis results for tuning the performance of NCSA-LSF*.

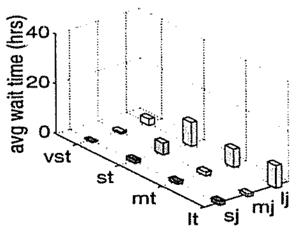
Since draining in NCSA-LSF* occurs only very infrequently (as mentioned in Section 3.1), the poor performance in NCSA-LSF* is mainly due to the use of static job classes with higher priority to larger size classes (lj > mj > sj on six out of eight hosts) and thus requires limits on large jobs (lj and mj) in order to prevent starving small jobs. The job limits causes significant waiting times for large jobs when there are more of them in the workload to be scheduled. Furthermore, the performance of NCSA-LSF* is sensitive to the specific order of job arrivals; and it is difficult to tune the scheduling parameters due to the complex interaction between the



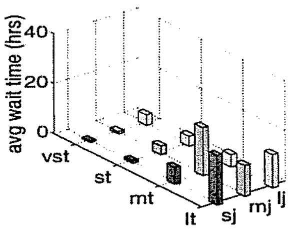
(a) High load (Dec 1999)



(b) High load (Feb 2000)

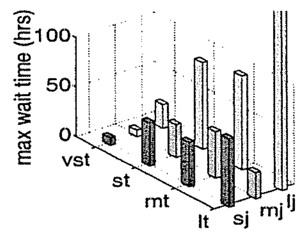


(c) Typical load (Nov 1999)

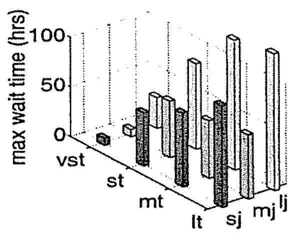


(d) Typical load (Mar 2000)

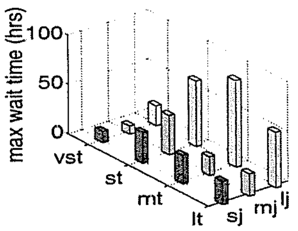
Figure 5. Performance of NCSA-LSF*: Average Wait Time



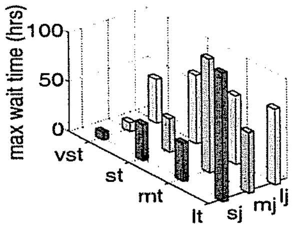
(a) High load (Dec 1999)



(b) High load (Feb 2000)



(c) Typical load (Nov 1999)



(d) Typical load (Mar 2000)

Figure 6. Performance of NCSA-LSF*: Maximum Wait Time

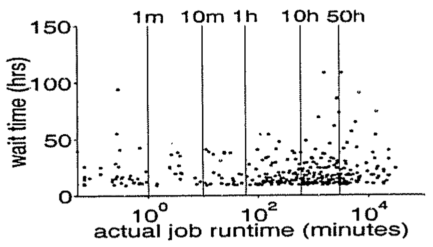


Figure 7. NCSA-LSF* Highest 5% Wait Times vs. Actual Job Run Time (Jan 2000)

class priority and limits.

To improve the scheduler, we evaluated priority backfill policies and show that our initial priority backfill policy, i.e., Priority-backfill, significantly outperforms NCSA-LSF*. Based on our predicated improvement, a higher performance priority backfill policy, i.e., NCSA-MS*, configured with a similar priority function as in Priority-backfill, replaced NCSA-LSF* in July 2000, as mentioned at the beginning of this chapter.

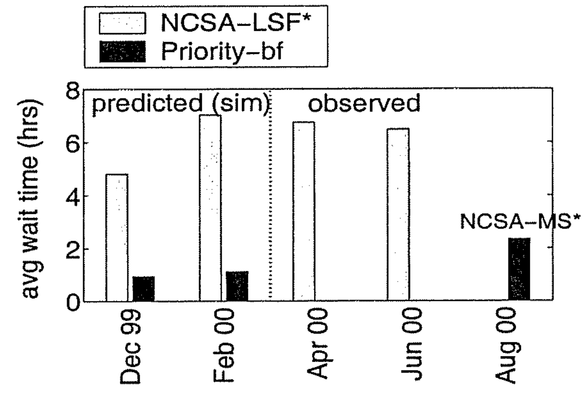


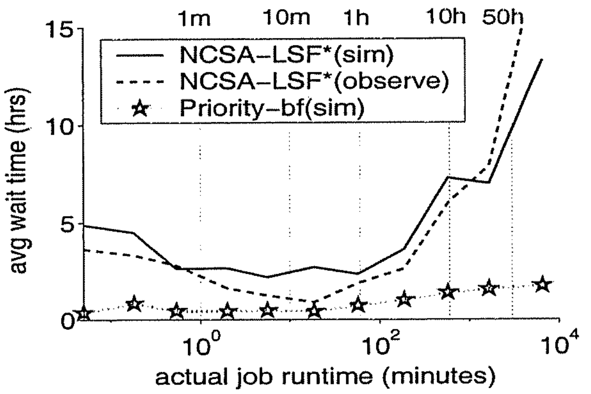
Figure 8. Predicted and Observed Improvement of NCSA-MS* over NCSA-LSF*

3.7 Validation of Simulation

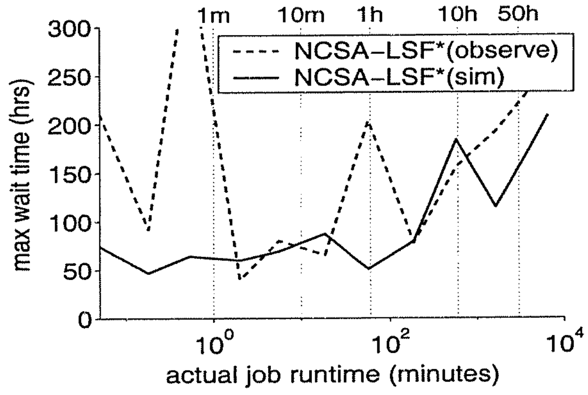
This section provides results that validate our simulation of the scheduling policies used on the NCSA O2K (i.e., NCSA-LSF* prior to July 2000 and NCSA-MS* since July 2000) and our predicted improvement of Priority-backfill over NCSA-LSF*.

Figure 8 shows that the improvement in the monthly average wait time of the simulated Priority-backfill over NCSA-LSF* (shown for two high-load months, December 1999 and February 2000) is qualitatively similar to the measured improvement of NCSA-MS* (for a high-load month, August 2000) over NCSA-LSF* (for a high-load month, June 2000) on the O2K.

Figure 9 further compares Priority-backfill and NCSA-LSF* and validate our simulation of NCSA-LSF* for a high-load month (December 1999), in which month NCSA-LSF* still dispatched the jobs on the O2K. Figure 9(a) plots the average wait time versus the actual runtime for the simulated Priority-backfill and NCSA-LSF* for a high-load month, December 1999. Also included in the graph is the measured NCSA-LSF* on the O2K (i.e., NCSA-LSF*(observe)). The figure shows that Priority-backfill has significantly lower average wait time than that of NCSA-LSF* over the entire range of actual runtime. Furthermore, it shows that the simulated NCSA-LSF* has similar average wait time versus actual runtime than that of the observed NCSA-LSF*, except that simulated performance is somewhat worse for runtime under

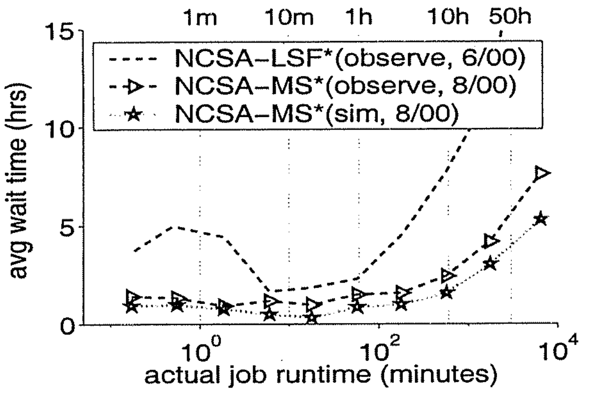


(a) Avg Wait vs. Actual Runtime

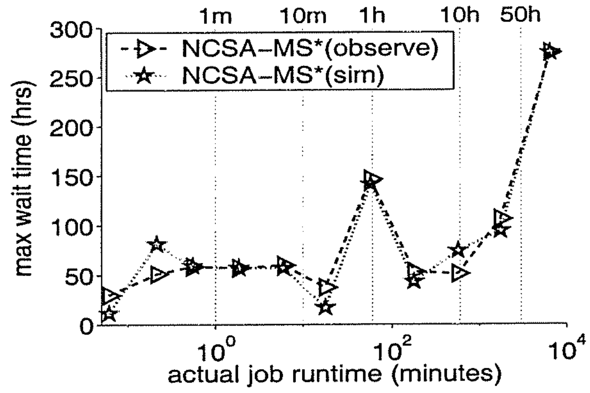


(b) Max Wait vs. Actual Runtime

Figure 9. Predicted Performance Improvement of Priority-backfill over NCSA-LSF* and Validation of NCSA-LSF* Simulation (December 1999)



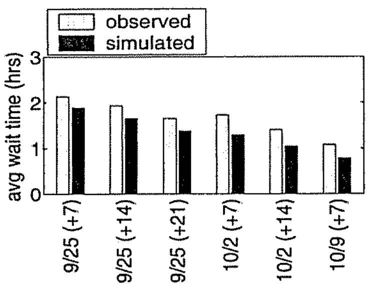
(a) Avg Wait vs. Actual Runtime



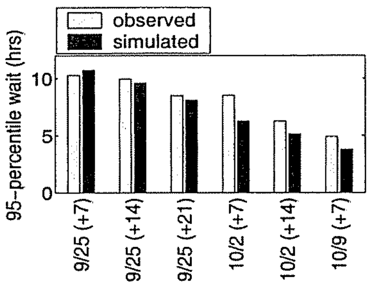
(b) Max Wait vs. Actual Runtime (August 2000)

Figure 10. Observed Performance Improvement of NCSA-MS* over NCSA-LSF* and Validation of NCSA-MS* Simulation

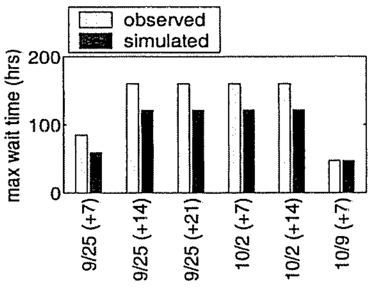
50 hours and better for runtime > 50 hours. The discrepancy in the simulated and observed performance of NCSA-LSF* is mainly due to the anomaly in the NCSA-LSF* scheduler on the O2K. We found that NCSA-LSF* sometimes scheduled the jobs in violation of the scheduler parameters configured on the O2K; in particular, many sj jobs were scheduled on the hosts when the mj or lj jobs have the priority and there are mj or lj jobs waiting. This sometimes causes a lower overall average waiting time but higher overall maximum waiting time. As shown in Figure 9(b), the maximum waiting time versus actual runtime for the observed NCSA-LSF* is



(a) Avg Wait



(b) 95th percentile Wait



(c) Max Wait

Figure 11. Performance Discrepancy Between Actual and Simulated NCSA-MS*

(Six Workloads During Sepetember 25-October, 2000)
The label "mm/dd (+n)" means the n-day period starting at mm/dd.
For example, "9/25 (+14)" is the 2-week period starting at Sep. 25.

much worse than that of the simulated NCSA-LSF*. Appendix D explains the types of anomaly found in the NCSA-LSF* scheduler in more detail.

Figure 10 compares the performance of the priority backfill policy (for August 2000) and NCSA-LSF* (June 2000) observed on the O2K. Note August 2000 is the first full month in which NCSA-MS* is used on the O2K. The figure also included simulated NCSA-MS* with the scheduling parameters configured for the O2K. First, Figure 10(a) shows that the improvement in the average wait time by simulated NCSA-MS* over NCSA-LSF* on the O2K qualitatively resembles that in Figure 9(a) performed prior to the installation of NCSA-MS* on the O2K. Second, Figures 10(a)-(b) show that the simulated NCSA-MS* has somewhat lower average wait time (especially for runtime > 50 hours), and very similar maximum wait time versus the actual runtime. We note that there was a bug in the NCSA-MS* in that month. The bug caused the scheduler not able to start the jobs sometimes even though enough free resources are available for scheduling more jobs.

Figure 11 further shows the difference in the average, 95th percentile, and maximum waiting time between the simulated and measured NCSA-MS*. We chose the periods in which the above mentioned bug in NCSA-MS* has been fixed and there is no or minimal system downtime (to the best of our knowledge) for this study. As shown in the figure, the difference in each performance measure for each period is under 20% for almost periods simulated. There could be numerous reasons for the discrepancy, e.g., events that are not available in the traces, such as a scheduled or unscheduled downtown event on particular hosts, a scheduled special reservation for particular users on particular hosts, etc. In addition, there is a 5 mintue interval between two scheduling cycles for NCSA-MS* on the O2K, but not in the simulation. This could have also caused some performance difference, since after a job departure but before the end of that 5 minutes interval, new jobs may arrive and may have higher priority and get to be started before the previously waiting jobs can be scheduled. However, since the difference in the measured and simulated performance is not significant, we did not pursue further study of the discrepancy.

Chapter 4

Characteristics of A Large Shared Memory Production Workload

A good understanding of the workload is useful for designing high-performance policies for the workload and for creating realistic synthetic workloads. In this chapter, we analyze NCSA O2K workload, used for evaluating scheduling policies in Chapters 5-6.

This chapter provides the characteristics of the batch jobs that do not request a dedicated host. These jobs have more widely varying processor and memory requests than the jobs that request a dedicated host. The jobs analyzed were submitted in the six one-month periods (i.e., January - March and October - December 2000).

Usage on the O2K is dominated by scientific applications from chemistry, physics, astronomical sciences, molecular and cellular biosciences and materials research. Most programs are written in Fortran programming language; others use C/C++ languages; only a very small fraction of programs are in Java. Most programs use MPI (message passing interface) for parallelization; others use openMP (for shared memory programming models), native Fortran or C compiler-support parallelization, and Posix.

The O2K workload is interesting for several reasons. First, the system, consisting of 1520 processors and 616 GB of memory, is larger than any previously studied workload. Second, the largest O2K job runs for as long as 400 hours (i.e., over 16 days) on 64 processors, compared to only a few tens of hours in previously studied workloads. Third, the jobs submitted each month require a higher fraction (i.e., 90-100%) of the total processing time available than

Table 5. Notation

Symbol	Definition
M	Requested Memory
P	Requested Processors
T	Actual Runtime
M×T	Memory Demand
P×T	Processor Demand

that in previously studied workloads, to be shown later in Section 4.1. Fourth, the O2K is a shared-memory system, wheres the previous workload studies only focus on distributed-memory systems.

Our characterization is more thorough than that in previous workload studies. The results in this chapter make four main contributions. (1) we identify the different features in the O2K workload from that in previously studied workloads. (2) we provide job characteristics for each period of approximately stationary job arrival rate, rather than over a period of several months as in the previous workloads without establishing whether the workload is approximately stationary over those months. (3) we provide new measures that have not been studied, including the distribution of utilization on the allocated processors for each job, the average memory utilization for each job, the distributions of requested runtime and requested memory, the characteristics of the fifteen largest jobs in each month. (4) we provide new conditional distributions that are required for creating synthetic workloads but are missing in the previous workload models. For example the distribution of requested runtime conditioned on the number of requested processors and requested memory; the ratio of the actual runtime to the requested runtime conditioned on requested runtime, requested number of processors and requested memory.

Some of the notations that will be used in this chapter are given in Table 5. The organization of this chapter is as follows. Section 4.1 provides the analysis results of the job mix and load on the O2K hosts. Section 4.2 show the system load due to several large job classes, including the fifteen largest jobs of each month. Section 4.3 present the analysis of the number of job arrivals for each day and for each hour during the six months studied. Section 4.4 provides the characterization results for the distributions and conditional distributions among the job request measures, which are the requested number of processors, requested memory, and requested runtime for each period of approximately stationary job arrival rate per hour. Section 4.5 provides the results for the job usage measures, which are the actual job runtime and job memory usage. A summary is given in Section 4.6.

4.1 Overview of NCSA O2K Workload

Table 6 summarizes the O2K workload for each of the nine months during October - December 1999, January - March 2000, and October - December 2000. The early three months in October - December 1999 are included, because they are used for policy evaluation that are performed prior to October 2000. Our workload characterization results in the rest of this chapter are provided for the other six months in the year 2000. The LSF scheduler is used for the O2K during the three months in January - March 2000 and a priority backfill scheduler similar to the Maui Scheduler [Mau] is used during the late three months in October - December 2000.

For each month, the table shows the following information for all jobs (i.e., "All") and for each of the twelve LSF job classes (classified by the time and resource requests, as defined in Section 3.1): (1) the total number of submitted jobs; (2) the processor demand (i.e., P×T), which is the sum of the product of the number of requested processors and the actual runtime of each job, expressed as a fraction of the total processing time available on the eight O2K hosts during the month (i.e., the product of the total number of the processors and the total time during the month). Note that the total processor demand can be greater than 100% because the jobs that are submitted to the vst classes but run on the three dedicated hosts that give priority to dedicated jobs are also included. These vst jobs account for a total of 4-9% of the

Table 6. Total Monthly Processor and Memory Demand By LSF Job Class

Month	All	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	13562	5186	2190	2226	118	1483	750	145	27	1067	244	74	52
P×T	86%	2%	9%	14%	3%	3%	8%	8%	1%	9%	13%	11%	4%
M×T	58%	1%	4%	8%	3%	2%	4%	7%	5%	2%	4%	11%	8%
Nov 1999													
#jobs	10916	3884	2231	510	141	1312	722	212	40	1404	345	94	21
P×T	91%	2%	9%	7%	2%	3%	14%	10%	3%	7%	16%	15%	3%
M×T	57%	1%	6%	6%	2%	1%	5%	10%	5%	2%	8%	7%	5%
Dec 1999													
#jobs	8778	3223	2081	557	114	732	575	189	28	840	276	134	29
P×T	95%	1%	8%	10%	4%	1%	13%	17%	4%	4%	10%	15%	9%
M×T	74%	1%	5%	7%	3%	0%	5%	11%	4%	1%	10%	14%	11%
Jan 2000													
#jobs	9652	3622	2606	553	71	950	589	163	61	671	252	91	23
P×T	88%	2%	9%	11%	3%	2%	9%	13%	6%	4%	8%	12%	10%
M×T	76%	1%	6%	7%	3%	1%	5%	10%	6%	1%	11%	10%	17%
Feb 2000													
#jobs	11290	5296	2269	466	71	1128	698	219	33	686	314	90	20
P×T	96%	2%	9%	11%	3%	3%	10%	13%	3%	6%	18%	12%	5%
M×T	78%	1%	7%	7%	3%	2%	5%	11%	5%	1%	10%	12%	15%
Mar 2000													
#jobs	12101	4671	2678	472	57	1808	631	216	70	850	500	123	25
P×T	94%	2%	11%	9%	3%	4%	11%	15%	4%	4%	14%	14%	3%
M×T	83%	1%	7%	6%	3%	2%	7%	9%	8%	2%	16%	18%	4%
Oct 2000													
#jobs	9768	3012	2488	580	278	881	627	241	50	957	367	209	78
P×T	90%	1%	11%	9%	7%	2%	10%	11%	2%	5%	14%	13%	4%
M×T	84%	1%	6%	7%	9%	1%	6%	6%	2%	2%	6%	18%	20%
Nov 2000													
#jobs	8708	2982	2279	416	60	711	497	187	16	912	513	110	25
P×T	91%	2%	10%	8%	3%	2%	9%	12%	3%	6%	20%	13%	3%
M×T	63%	1%	5%	5%	2%	1%	5%	6%	1%	2%	10%	11%	14%
Dec 2000													
#jobs	7798	2581	2190	565	164	801	252	215	59	667	176	113	15
P×T	102%	2%	11%	10%	9%	2%	5%	18%	4%	6%	11%	13%	12%
M×T	68%	1%	6%	8%	5%	1%	2%	10%	6%	2%	4%	13%	9%

processor demand each month. (3) the memory demand (i.e., M×T), which is the same as processor demand, but for the requested memory.

Key observations from Table 6:

- The processor demand for each month is very high, typically 90-100%;
- The memory demand for each month is lower, but is still in the range of 60-80%;

- The job mix each month is very similar from month to month, except that there is a smaller number of jobs but slightly higher processor demand in December 2000 than in other months;
- The large jobs ('lj') account for over 40% of the processor and memory demand each month.

4.2 Monthly Load Due To Large Jobs

This section studies the system load due to large jobs in the O2K workload. Section 4.2.1 provides the results for several large job classes. Section 4.2.2 provides the results for the fifteen largest jobs in each month. The characteristics for these largest jobs each month are also provided.

4.2.1 Large Job Classes

Previous workload studies show that a relatively small fraction of the jobs account for a large fraction of the total processor demand. For example, Feitelson and Nitzberg [FN95] reported that 90% of the jobs on an iPSC/860 have runtime under 15 minutes, but the remaining 10% of the jobs account for 90% of the processor demand of all jobs. As another example, Hotovy [Hot96] reported that under 50% of the jobs on the CTC SP/2 are use more than 1 processor (i.e., parallel jobs), but they account for over 90% of the processor demand of all jobs.

The O2K jobs have a longer runtime in average than that in previously studied workloads, e.g., greater than 10% of the O2K jobs each month have over 20 hours of actual runtime (shown in Table 7). Nevertheless, the O2K workload has a similar characteristic in that a small fraction of the jobs account for a large fraction of the processor demand of all jobs. For example, Table 6 in the last section shows that 50-60% of the jobs request under 5 hours of runtime but the remaining 40-50% of the jobs account for over 90% of the processor demand of all jobs.

Table 7. Summary of the Load of Various Large Job Classes

Month	Job Class	Jobs (%)	Demand/All		Job Size			
			P×T	M×T	Avg P	Avg M (GB)	Avg T (hours)	Avg P×T
Jan 2000	P×T > avg (65h)	1216 (13%)	89.8%	83.7%	15.0	2.3	49.0	465
	T > 20hr	916 (9%)	77.9%	82.0%	7.3	1.9	67.3	536
	P > 16*	493 (5%)	27.2%	9.4%	36.7	3.0	9.5	347
	M > 4GB*	302 (3%)	16.2%	46.8%	14.9	10.5	26.9	337
	P > 16, M > 4GB*	72 (1%)	5.8%	5.4%	39.6	10.5	11.9	505
Feb 2000	P×T > avg (57h)	1595 (14%)	89.3%	82.3%	18.4	2.2	35.5	359
	T > 20hr	1062 (9%)	71.0%	82.0%	6.9	1.8	59.7	428
	P > 16*	617 (5%)	35.6%	14.5%	38.6	3.0	8.6	369
	M > 4GB*	287 (3%)	25.2%	45.2%	23.3	10.4	25.9	562
	P > 16, M > 4GB*	112 (1%)	18.1%	11.7%	49.3	10.6	18.9	1035
Mar 2000	P×T > avg (55h)	1670 (14%)	88.1%	82.4%	15.2	2.8	41.0	353
	T > 20hr	1189 (10%)	71.6%	75.9%	6.3	1.9	62.4	403
	P > 16*	542 (4%)	30.9%	20.5%	39.7	4.8	10.0	381
	M > 4GB*	600 (5%)	27.1%	45.7%	16.3	10.6	14.4	302
	P > 16, M > 4GB*	167 (1%)	20.3%	18.1%	43.9	12.0	18.1	813
Oct 2000	P×T > avg (66h)	1793 (18%)	89.3%	79.0%	15.2	2.3	42.5	319
	T > 20hr	1312 (13%)	71.3%	80.9%	6.7	1.9	59.6	348
	P > 16*	695 (7%)	35.7%	9.7%	42.2	4.7	8.5	329
	M > 4GB*	653 (7%)	15.8%	51.5%	22.0	10.1	15.0	154
	P > 16, M >4GB*	265 (3%)	10.7%	7.0%	47.0	9.8	6.1	258
Nov 2000	P×T > avg (72h)	1495 (17%)	89.4%	83.3%	19.4	2.6	35.5	376
	T > 20hr	1065 (12%)	70.7%	79.6%	8.0	1.6	53.1	418
	P > 16*	767 (9%)	40.7%	21.7%	42.1	4.9	9.2	333
	M > 4GB*	502 (6%)	20.9%	53.7%	27.5	9.4	16.8	261
	P > 16, M > 4GB*	246 (3%)	14.4%	17.1%	46.8	12.0	8.6	367
Dec 2000	P×T > avg (93h)	1188 (15%)	87.8%	76.0%	14.6	1.9	63.7	537
	T > 20hr	1293 (17%)	82.2%	87.7%	6.5	1.5	69.5	462
	P > 16*	455 (6%)	36.1%	21.2%	39.2	4.2	16.8	577
	M > 4GB*	310 (4%)	15.9%	36.1%	23.2	9.5	19.4	372
	P > 16, M > 4GB*	97 (1%)	11.8%	15.9%	53.0	14.4	16.6	882

* T > 10 minutes

To examine to what extent a small fraction of the jobs account for a large fraction of resource demands in the O2K workload, Table 7 summarizes the processor and memory demand (as a fraction of the demand of all jobs) for five classes of large jobs of each month. The class "P$\times$T > avg" is the set of the jobs that have per-job processing time (i.e., the product of the requested number of processors and the actual runtime of the job) greater than the average processing time over all jobs in each month (the average is shown in parentheses, e.g., 93 processor hours for December 2000). The class "T > 20hr" is the set of the jobs that have actual runtime greater than 20 hours. The other three classes of jobs are defined by P > 16, M > 4 GB, and T > 10 minutes.

Key observations of Table 7:

- The average processing time of each job is in the range of 50-100 processor hours, which is larger than in previous workloads (e.g., about 30 processor hours on SDSC Paragon [WLF$^+$96] and under 10 processor hours on CTC SP/2 [Hot96]).
- The processor and memory demands for each large job class is very similar across different months.
- The processor and memory demands of each month are dominated by 10-20% of the jobs (having run for over 20 hours or having a processing time greater than the average processing time of all jobs each month).
- Very few jobs ($\leq$ 3% each month) request > 16 processors, > 4 GB of memory, and run for at least 10 minutes.

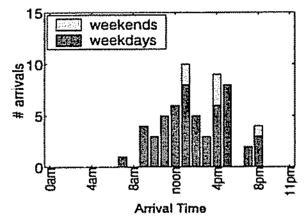
4.2.2 Characteristics of Fifteen Largest Jobs Each Month

This section provides the characterization results for the fifteen jobs that have the largest per-job processing time for each month.

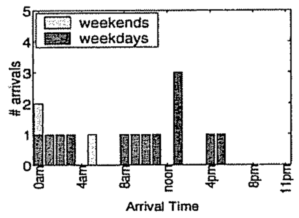
Table 8 summarizes the load due to the largest 15 jobs of each month. These 15 jobs account for impressively 10-15% of the processor and memory demand of each month. For most months, the average run time of these jobs is over 200 hours and the average processor demand is 4000 -

Table 8. Fifteen Largest Space-Shared Jobs Each Month

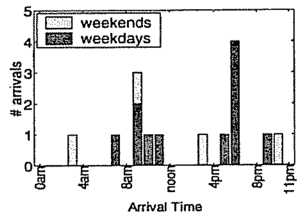
Month (Year 2000)	Demand/Demand All		Job Size			
	Processor Demand	Memory Demand	Avg Processors Requested	Avg Memory Requested (GB)	Avg Actual Runtime (hours)	Avg Processor Demand (P×T hrs)
Jan	17.1%	15.7%	27.7	6.1	311.0	7173
Feb	16.5%	17.2%	36.1	7.7	254.5	7031
Mar	15.6%	11.0%	36.1	6.6	209.5	6944
Oct	9.1%	3.4%	41.7	3.8	102.9	3888
Nov	11.1%	9.2%	39.8	5.1	168.4	4657
Dec	16.6%	13.3%	40.5	8.0	222.0	8044



(a) Jan-March & November 2000



(b) October 2000



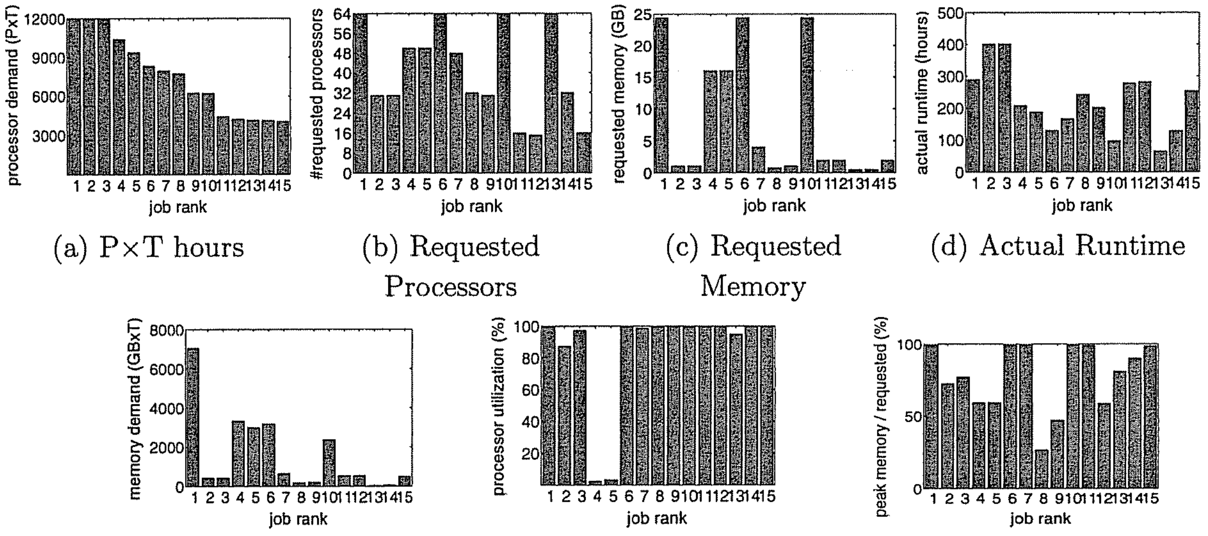
(c) December 2000

Figure 12. Job Arrival Times For the Fifteen Largest Jobs

8000 processor hours, which is at least an order of magnitude larger than the average processor demand of all jobs each month.

Figure 12 shows the arrival time of these largest jobs. Figure 12(a) provides the results for the four months (i.e., Jan-March and November 2000) that have the same results. The results for October 2000 are shown in Figure 12(b), and that for October 2000 are presented in Figure 12(c).

Figure 13 shows the characteristics of each of the 15 largest jobs in December 2000. For each job, Figures 13(a)-(g) show respectively the processing time (P×T), memory time (M×T), requested number of processors, requested memory, actual runtime, the processor utilization (i.e., total processing time kept busy by the job divided by the total processing time allocated to the job), the peak memory efficiency (i.e., the peak memory usage divided by the requested



(a) P×T hours (b) Requested Processors (c) Requested Memory (d) Actual Runtime

(e) M×T hours (f) Processor Utilization (g) Peak Memory Efficiency

Figure 13. Characteristics of the Fifteen Largest Jobs

(December 2000)

memory of the job). A set of these graphs and the graph for the arrival time of the 15 largest jobs for each of the other months are provided in Figures 83-87 in Appendix B.

The characteristics of the 15 largest jobs each month are summarized below.

- 2-6 of the 15 largest jobs each month request 64 processors (i.e., the largest number of requested processors), and nearly all of these jobs run for under 200 hours.
- Most of the 15 largest jobs request either the maximum memory allowed (i.e., 16 GB for January - March 1999 or 25 GB for October - December 2000) or under 4 GB of memory.
- 1-3 jobs each month request the maximum memory allowed, and nearly all of them also request the largest number of processors.
- At least two-thirds of the 15 largest jobs each month achieve 80-100% processor utilization; many of them achieve nearly 100% processor utilization; but 2-5 jobs each month achieve under 50%, of which a total of three jobs have utilization under 10%.
- At least one third of the 15 jobs each month achieve 80% of peak memory efficiency, but

also 1-3 jobs each month have under 50% of peak memory efficiency (except it's 9 jobs in November 1999).

- As shown in Figure 12(a), most of the 15 largest jobs each month in the four months in January - March and November 2000, arrive during 9am-6pm on weekdays, which is similar to the characteristics of all job arrivals to shown in Section 4.3. Figure 12(b) shows that a significant number of the largest jobs in October 2000 arrive before 9am; graph (c) shows that many of the largest jobs in December 2000 arrive at the beginning or the end of the period of 9am-6pm.

4.3 Characteristics of Job Arrivals

Figure 14 shows the number of job arrivals for each day during each month, with each particular day of the week (e.g., Monday) grouped together. Each week starts on Monday and ends on Sunday. In the figure, the days in the first week of each month are shown by the bars in the white color. Typically, 350-400 jobs arrive for each weekday day, and 150-200 jobs for each weekend day. Occasionally, in some weekday days, there are over 500 job arrivals per day. Other months have similar range of job arrivals per day, except that (1) in Feburary 2000, the number of jobs per weekend day is about 300, more closer to that on a weekday. (2) on the major holidays (i.e., the New Year day, Thanksgiving, and Christmas) and during the last 2-3 weeks in December 2000, the number of jobs per weekday day is more similar to that of a typical weekend day.

Next, we further examine the variation in the number of job arrivals for each hour of each day during the six months of workload.

Plots of the number of job arrivals for each hour of each day during the six months of workload do not reveal any distinct characteristics for any particular day. Figure 15 shows that the pattern of the number of jobs for each hour varies somewhat from one weekday to another (Figure 15(a)-(d)), but the average number of arrivals each hour is approximately the

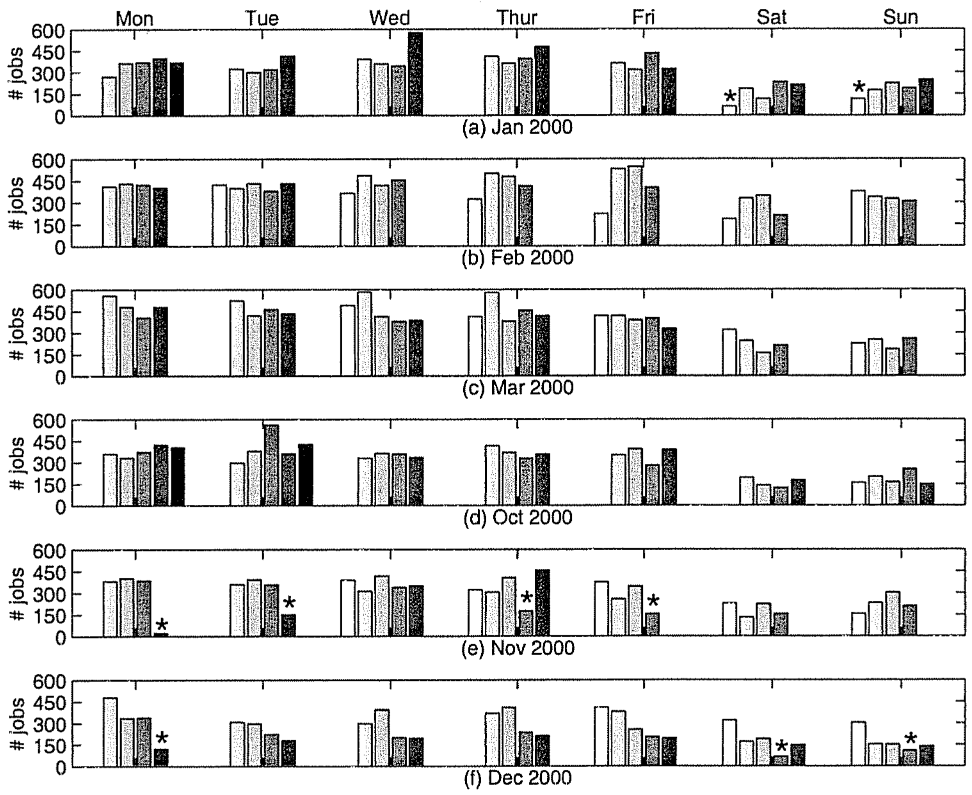


Figure 14. Number of Jobs Submitted Per Day

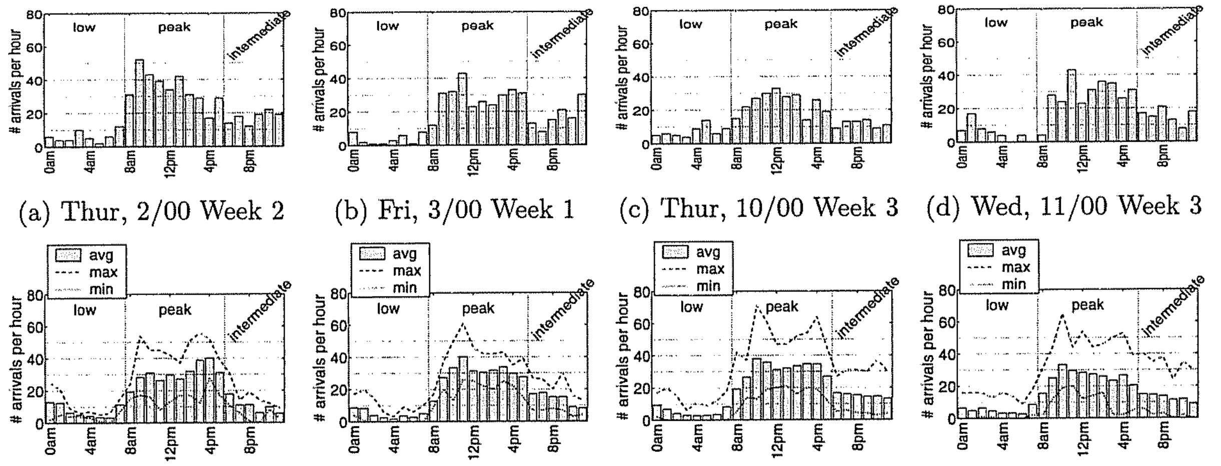
('*' marks holidays)

(Typical weekdays: 350-400/day, weekends: 150-200/day)

same whether computed over all Fridays (or other day of the week) in a given month (e.g., Figure 15(e)), or over Monday through Friday in a given week (e.g., Figures 15(f)), or over all weekdays in any given month (e.g., Figures 15(g)-(h) for March 2000 and October 2000, respectively).

Furthermore, the figure shows that when the average number of arrivals is computed over a large number of days (such as all Fridays, or a given month), three relatively stable periods of arrival rate during weekdays are readily identified, i.e., peak (8am - 6pm), intermediate (6pm - midnight), and low (midnight - 8am), as identified in the figures. The number of job arrivals for each hour is about 30 during a typical peak arrival rate period, 15-20 during a typical intermediate period, and 5-10 during a typical low period [1].

[1]Note that during the two-hour period in 0 - 2 am on some weekday days, the average number of jobs per



(a) Thur, 2/00 Week 2 (b) Fri, 3/00 Week 1 (c) Thur, 10/00 Week 3 (d) Wed, 11/00 Week 3

(e) All Fridays (2/00) (f) M-F, 3/00 Week 3 (g) All Weekdays (3/00) (h) All Weekdays (10/00)

Figure 15. Number of Job Arrivals Each Hour on Weekdays

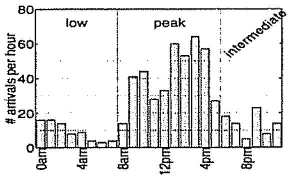
(Holidays are excluded)

(In average, peak: 30/hour; intermediate: 15-20/hour; low: 5-10/hour)

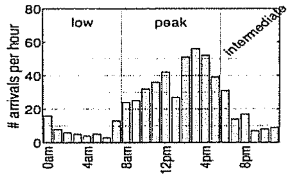
Figures 16(a)-(h) plot the results for each of eight heavy-load weekdays (> 500 jobs/day) that have a significantly higher number of jobs per day than that in other weekdays. It shows that the 'extra' jobs arrive during the peak period of arrival rate for all except for Tuesday of the 2nd week in March 2000 (Figure 16(e)), in which a significant number of extra jobs arrive during the intermediate period of arrival rate. Thus, there is a larger difference in the number of job arrivals of the peak and intermediate periods for most months, and a similar number of job arrivals per hour for the period between 8am - midnight. However, three periods of arrival rate are still appropriate for delineating the job arrivalrate for each hour.

Figures 17(a)-(d) shows the results for each of four light-load weekdays (< 300 jobs/day) that is not a holiday but has a fewer number of jobs than in the typical weekdays. It shows the number of job arrivals for each hour during the peak arrival rate period is significantly lower than that for a typical weekday day, but three periods of arrival rate are still still appropriate for these days.

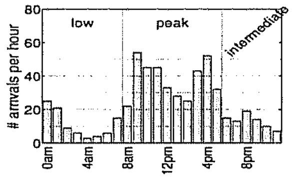
hour is more similar to that in the intermediate period (i.e., 6pm - midnight). For these days, it may be more appropriate to include these two hours in the intermediate period. However, we did not find difference in the job characteristics between from period to period and it is not likely that this two-hour period would have made the differences.



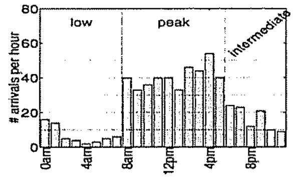
(a) Wed, 1/00 Week 5



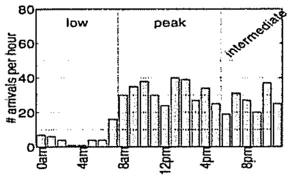
(b) Fri, 2/00 Week 2



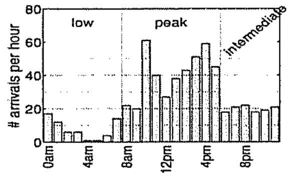
(c) Fri, 2/00 Week 3



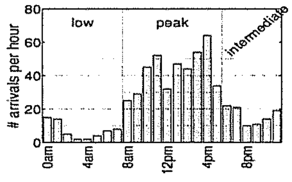
(d) Mon, 3/00 Week 2



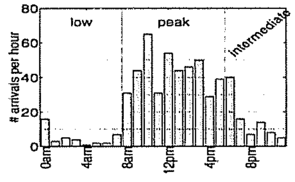
(e) Tue, 3/00 Week 2



(f) Wed, 3/00 Week 2

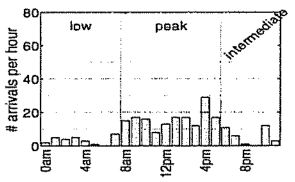


(g) Thur, 3/00 Week 2

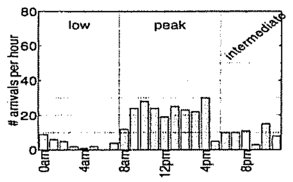


(h) Tue, 10/00 Week 4

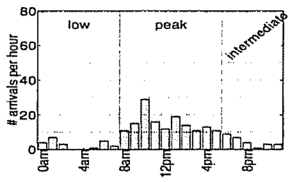
Figure 16. Number of Job Arrivals Each Hour On Weekdays with Highest Number of Jobs



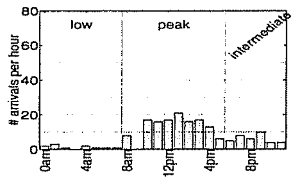
(a) Fri, 2/00 Week 1



(b) Tue, 10/00 Week 2



(c) Wed, 12/00 Week 4



(d) Tue, 12/00 Week 5

Figure 17. Number of Job Arrivals Each Hour On Weekdays with Fewer Arrivals

Figures 18(a)-(d) plot the results for each Saturday of a typical month (i.e., November 2000), and (f)-(i) for each Sunday of the given month. Figures 18(e) plots the average computed over all Saturdays in the month, and Figures 18(j) plots the average computed over all Sundays in the month. The key observation is that the number of jobs for each hour does not vary over the hours as much as that during weekdays. Similar to that for the weekdays, when the average number of jobs for each hour is computed over the Saturdays of the month (Figures 18(e) or over Sundays of the month (Figures 18(j), approximately two periods of relatively stable job arrival rate per hour are identified, i.e., intermediate and low.

Similarly, Figure 19 plots the results for the weekend days, but for the exceptional month (i.e., February 2000), in which the number of job arrivals per day in most of the weekend days is significantly higher than that in a typical weekend day. It shows that there is more variation

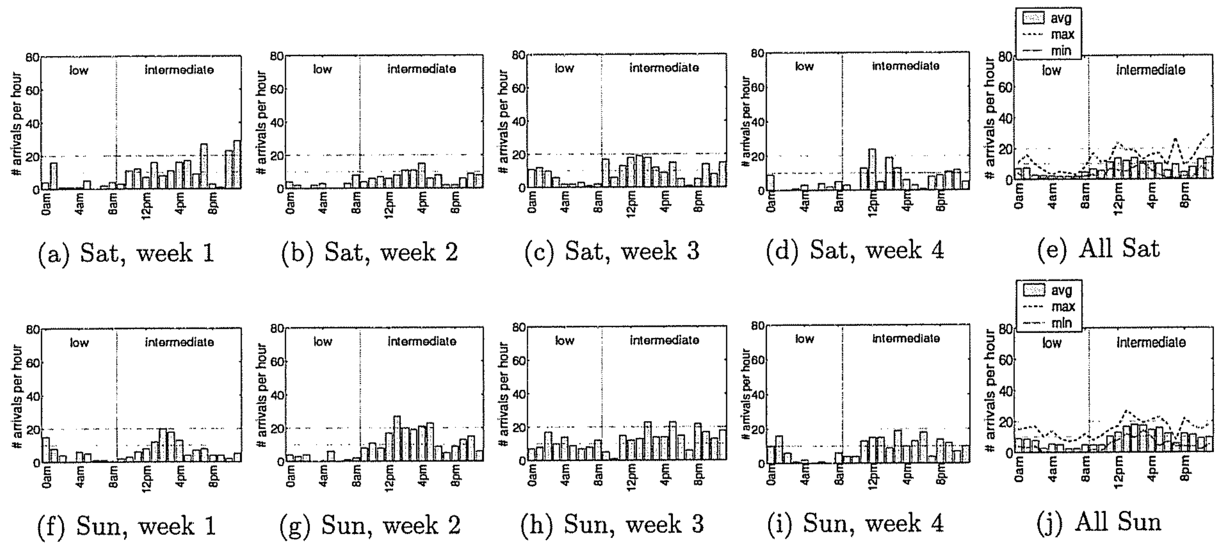


Figure 18. Number of Job Arrivals Each Hour During Normal-Load Weekends (November 1999)

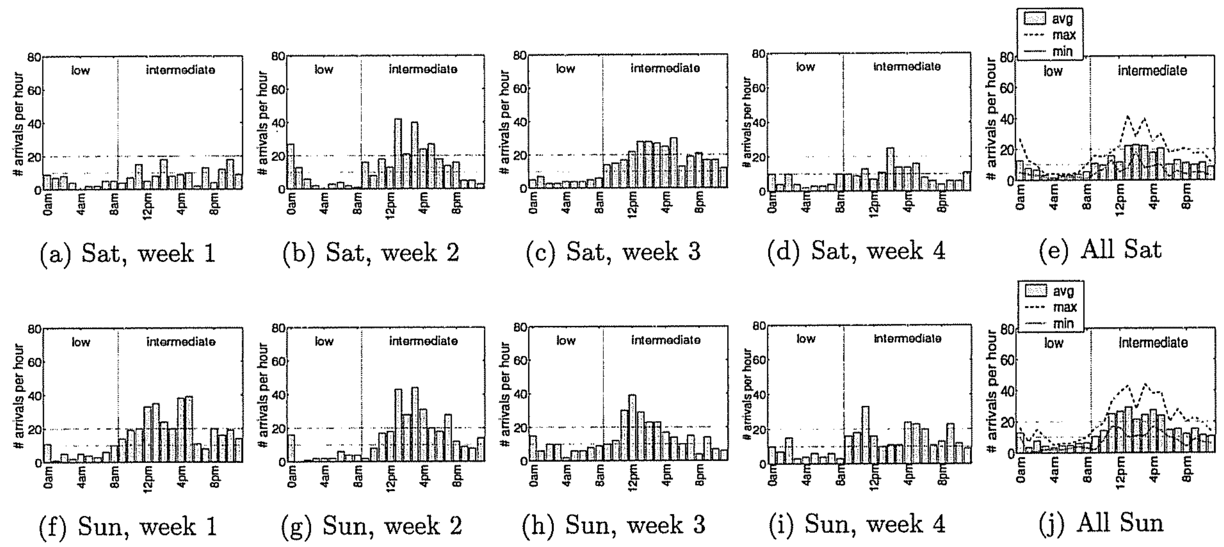


Figure 19. Number of Job Arrivals Each Hour During Heavy-Load Weekends (Feburary 2000)

in the job arrival rate per hour between 8am-midnight for many of the weekend days in this month than that in a typical weekend days. Nevertheless, since the difference in the number of jobs during these hours is not as significant as that on each weekday day, two weekend periods of arrival rate are also used for this month for the purpose of characterizing the other job characteristics.

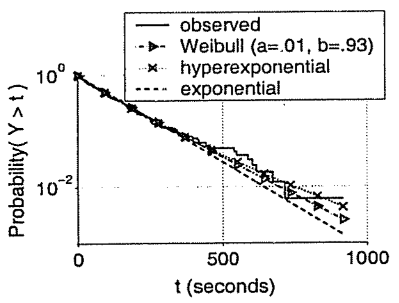
Several previous papers have reported the average number of job arrivals for each hour for the workloads on an iPSC/860 [FN95], Paragon [WLF$^+$96], SP/2 [HSO96], and a collection of workstations [Gib97a]. The per-hour job arrival pattern on these systems, except for the Paragon, is surprisingly similar to that observed on the O2K, i.e., a peak period arrival rate period during 8am and 6pm, a lower arrival rate period before 8am, and an intermediate arrival rate period after 6pm. The previous papers have noted the peak and low arrival rate periods, but not the intermediate arrival rate period. Furthermore, they do not analyze whether the per-hour job arrival pattern is similar or different across different days.

For the purpose of determining interarrival time distributions, we consider weekday peak periods from 9am-5pm because the two one-hour periods in 8-9am and 5-6pm have average arrival rate per hour that is slightly lower than that in the other peak hours; and the weekday low periods from 2am to 8am because the two-hour periods in 0-2am have an average arrival rate per hour slightly higher than that in the other low periods. For analyzing all other job characteristics, the weekday peak period is defined to be 8am-6pm and the low period is defined to be 0 am - 8am.

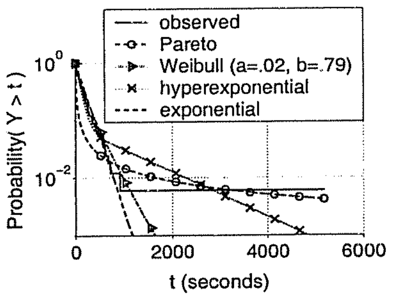
Excluding the downtime, the coefficient of variation (CV) of the interarrival time for each period of stable arrival rate is typically in 1-1.5, with occasionally slightly higher than 1.5 (but < 2). Higher CVs of interarrival time (i.e., 2.5-6) have been reported on other systems [FN95, WLF$^+$96, JPF$^+$97, SYZ99] during 'day' time, 'night' time, or during the entire period for which the traces are analyzed, rather than during stable periods of arrival rate. The paper in [Hot96] report a CV of 1.76 (more similar, but slightly higher than in most periods of arrival rate in the O2K workload) for all the interarrival times during 'day' time on an CTC SP/2.

We investigate several models for the distribution of the interarrival time for each period of relatively stable arrival rate. The distributions are the exponential distribution with the observed mean interarrival time, the two-stage hyperexponential distribution with the observed mean interarrival time and CV, and the Weibull and Gamma distributions [2]

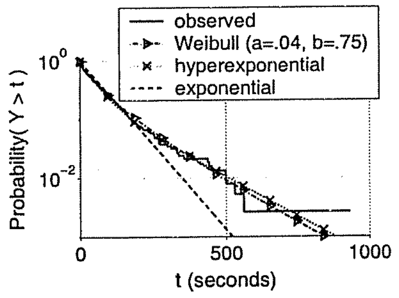
[2]For each period of arrival rate, the parameters of the two-stage hyperexponential distribution are computed



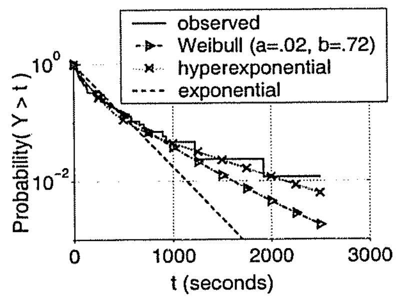
(a) Normal Load
Excluding Downtime
9am-5pm Tue, Jan'00 Week 3
(mean: 139 sec, cv: 1.12)



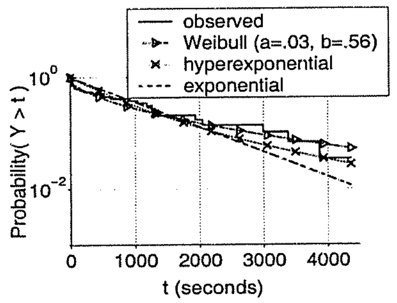
(b) Normal Load
Including Downtime
9am-5pm Tue, Jan'00 Week3
(mean: 170 sec, cv: 2.48)



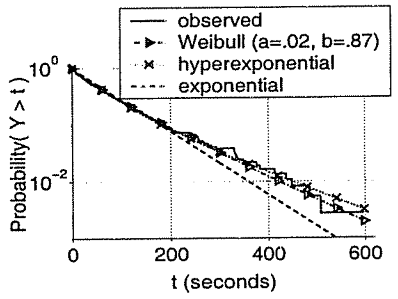
(c) High Load
No Downtime
9am-5pm Thur, Jan'00 Week 5
(mean: 76 sec, cv: 1.33)



(d) Intermediate Load
No Downtime
6pm-12am Fri, Feb'00 Week 2
(mean: 247 sec, cv: 1.64)



(e) Low Load
No Downtime
12am-8am Fri, Mar'00 Week 1
(mean: 977 sec, cv: 1.29)



(f) High Load
No Downtime
9am-5pm Thur, Mar'00 Week 2
(mean: 78 sec, cv: 1.18)

Figure 20. Distributions of Job Interarrival Time (Y)

Figure 20 illustrates the fit of these distributions for several different periods, with the interarrival time ranging from 76 seconds in the peak arrival rate period on a high-load weekday (Figure 20(c)) to 977 seconds in a typical low period of arrival rate (Figure 20(e)). The complement of the cumulative distribution is plotted on a log scale of the interarrival time to more clearly show the fit of the tail of the distribution. As shown in the graphs (a) and (c)-(f), the hyperexponential and Weibull distributions provide a better fit than that of the exponential distribution for each period, even if the CV is very close to 1 (e.g., Figure 20(a) and (f)). In

using the standard algorithm such that the products of the probability and the mean service time for each stage are equal [All90]. The maximum likelihood estimates of the Weibull and gamma distribution parameters with 95% confidence intervals [JKK92], are computed using Matlab [Mat].

some cases, the hyperexponential distribution provides a better fit than that of the Weibull distribution (e.g., Figure 20(d)). The fit of the Gamma distribution is similar to that of the Weibull distribution. For more readable figures, this distribution is not shown.

We also use the hyperErlang distribution to model the interarrival time using the algorithm proposed in [JPF$^+$97], the resulting distribution for each period is the 2-stage hyperexponential distribution that matches the first three moments of the observed interarrival time. The fit of this particular 2-stage hyperexponential distribution (not shown for more readable graphs) is similar to that of the hyperexponential distribution shown in the graphs. The two-stage hyperexponential with CV close to 3 was found to have a good fit for the distribution of all interarrival times, not separately measured over periods of approximately stationary arrival rate, for the CTC SP/2 in [JPF$^+$97].

Figure 20(b) plots the results for a period which includes a downtime (10:12-11:38am on January 11, i.e., Tuesday of the 3rd week in January) [3]. It shows that if the system downtime is included, one might erroneously conclude that the Pareto distribution provides a better fit for the distribution of the interarrival time.

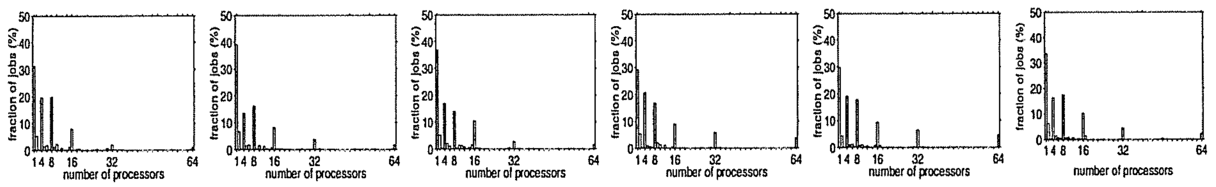
4.4 Characterizations of Job Resource Requests

This section provides characterization of the requested number of processors, requested memory, and requested runtime, for each period of relatively stable arrival rate. The characterization results for the actual job resource usage are provided in Section 4.5.

4.4.1 Requested Processors

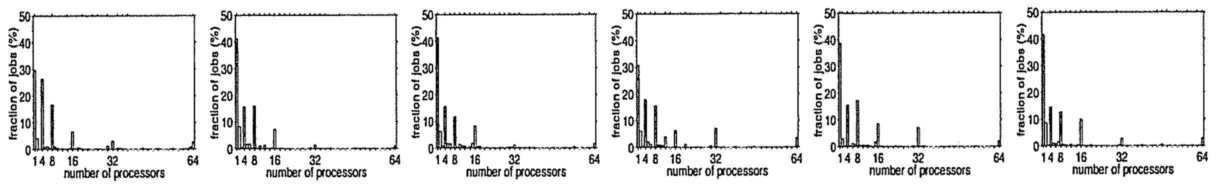
Figure 21(a)-(f) plot the probability distribution of the number of requested processors for the weekday peak arrival rate period for each of the six months studied. The results for the weekday intermediate and low periods are shown in Figures 22 and 23, respectively

[3]Downtime is often not recorded in the job logs. We implemented a special daemon process that tracks system downtime on the O2K since the O2K job logs do not contain the down time information.



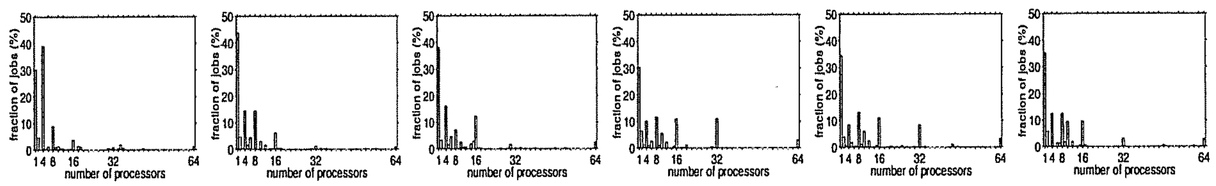
(a) Jan 2000 (mean: 7.2 cv: 1.30) (b) Feb 2000 (mean: 7.5 cv: 1.48) (c) Mar 2000 (mean: 7.4 cv: 1.40) (d) Oct 2000 (mean: 9.4 cv: 1.44) (e) Nov 2000 (mean: 10.3 cv: 1.42) (f) Dec 2000 (mean: 8.3 cv: 1.42)

Figure 21. Distribution of Requested Processors During Weekday Peak Arrival Rate Periods



(a) Jan 2000 (mean: 9.0 cv: 1.50) (b) Feb 2000 (mean: 5.9 cv: 1.51) (c) Mar 2000 (mean: 6.9 cv: 1.54) (d) Oct 2000 (mean = 9.5, cv = 1.43) (e) Nov 2000 (mean = 8.4, cv = 1.41) (f) Dec 2000 (mean = 7.4, cv = 1.64)

Figure 22. Distribution of Requested Processors During Weekday Intermediate Arrival Rate Periods

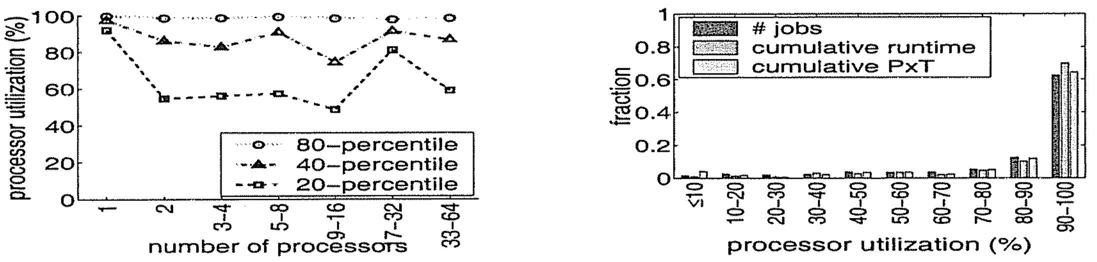


(a) Jan 2000 (mean = 6.7, cv = 1.49) (b) Feb 2000 (mean = 6.1, cv = 1.52) (c) Mar 2000 (mean = 8.3, cv = 1.43) (d) Oct 2000 (mean = 10.7, cv = 1.26) (e) Nov 2000 (mean = 10.7, cv = 1.30) (f) Dec 2000 (mean = 8.5, cv = 1.44)

Figure 23. Distribution of Requested Processors During Weekday Low Arrival Rate Periods

Similar to that in other workloads previously studied [FN95, Hot96, Fei96, WLF$^+$96, WMKS96, SGS96, Fei97a, SSN99], for each period of arrival rate of each month, a large fraction (30-40%) of the O2K jobs are serial (i.e., requesting one processor) and most jobs request power-of-two processors (except for 2 processors). The results for different months and different periods is approximately similar, except that in the late three months (especially October November 2000), there is a somewhat higher fraction of the jobs requesting 32 processors and 64 processors during each arrival rate period than that in the other months. In particular, 5-10% of the jobs for each arrival rate period of October and November 2000 request 32 processors, compared to under 2% in the first three months. This is also A higher fraction of jobs requesting 32 processors in the late three months is also observed for the periods of the weekends. Another exception is that in the weekday low period during January 2000 (Figure 23(a)), 40A similar distribution is observed in a few weekend periods.

The average number of requested processors is typically in the range of 7-10, except it is lower for several non-peak periods and weekends periods. The log-uniform cumulative distribution proposed in [Dow97b] does not fit the observed cumulative distribution of the requested processors in the O2K workload, because the number of jobs requesting 32 or 64 is much smaller than that for 4 and 8 processors, and also smaller than that for 16 processors, and there are only very few jobs that request a number of processors that is not a power of two. Thus, the harmonic distribution hand-tuned to emphasize small numbers and power of two processors proposed in [Fei96] is a better model for the O2K.



(a) vs. Requested Processors

(b) Distribution of Processor Utilization (actual runtime > 10 min)

Figure 24. Distribution of Processor Utilization

(October - December 2000)

Each job is allocated as many processors as requested during its entire execution. Figure 24 examines the utilization on the allocated processors per job (i.e., the total busy processing time divided by the total allocated processing time). Figure 24(a) plots the distribution (measured by the 80, 40, and 20th percentiles) of processor utilization for each range of number of processors (i.e., 1, 2, 3-4, 5-8, 9-16, 17-32, 33-64 processors). Note each range includes mostly the power of two processors (i.e., the upper bound of the range, since there are very few fraction of the jobs request a number of processors that is not a power of two. It shows that a high fraction of jobs utilize over 80% on the processors they requested, including jobs that request 32-64 processors. Figure 24(b) plots the fraction of the jobs (i.e., '# jobs") for each 10% range of processor utilization. Also included are the fractions weighted by the actual runtime (i.e., "cumulative runtime") and by the processor demand (i.e., "cumulative PxT"). The figure further shows that the jobs that have high utilization ($> 80\%$) include the jobs that have dominate actual runtime, and jobs that have dominant processor demand (PxT).

This leads to high total utilization of allocated processors on the O2K, to be shown in Figure 45(c).

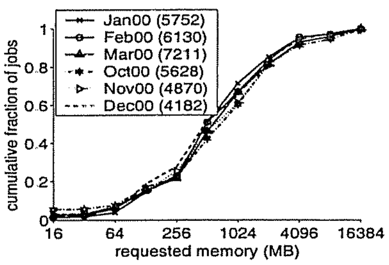
4.4.2 Requested Memory

Figure 25(a) plots the distribution of requested memory during the weekday peak period of each month. It shows that the distribution of requested memory is fairly similar for different months during the peak arrival rate period.

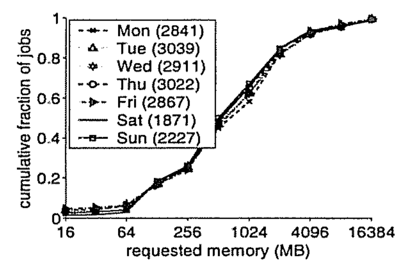
Figure 25(b) shows that the distribution of requested memory is similar for the peak arrival rate period during different days of the week (Saturday and Sunday are included [4]), in the three-month period in October - December 2000.

Figure 25(c) shows that the distribution of requested memory is very similar during different weekday period of arrival rate, except that jobs submitted during weekday intermediate periods request 256 MB to 1 GB of memory slightly more often, and 2-4 GB slightly less often but the weekday peak distribution is still a fairly good approximation for the intermediate period. Note that only a very small fraction (i.e., <1%) of jobs each month request the maximum of 25 GB (or previously 16 GB) of memory.

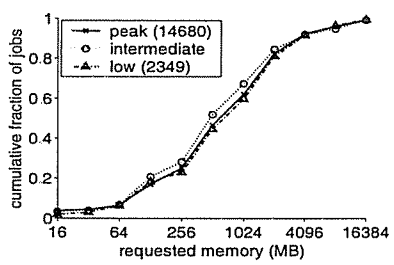
[4]Recall that there are only intermediate and low periods on each weekend day, defined in Section 4.3. The intermediate periods of Saturday and Sunday are shown in Figure 25(b) to compare with the peak period of each weekday day of the week.



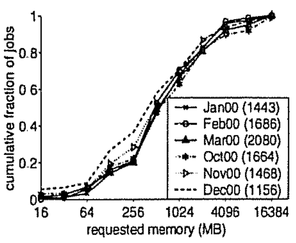
(a) Each Month Peak Period



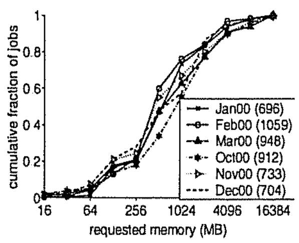
(b) Each Day of the Week Peak Period Oct-Dec 2000



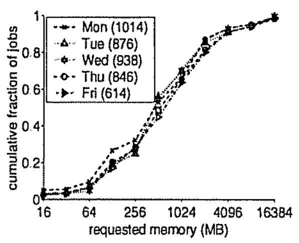
(c) Each Arrival Period Weekdays Oct-Dec 2000



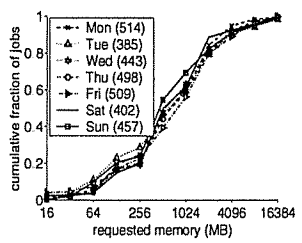
(d) Intermdiate Each Month



(e) Low Each Month



(f) Intermediate Period Each Day of Week (Oct-Dec 2000)



(g) Low Period Each Day of Week (Oct-Dec 2000)

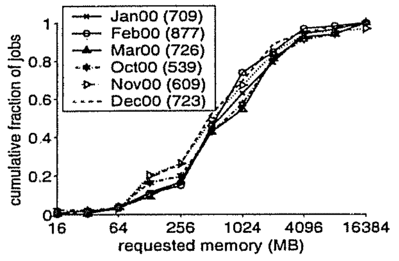
Figure 25. Variation in Distribution of Requested Memory During Weekdays

Figures 25(d)-(e) plot the results for the intermediate period and low period of different months, respectively. Figures 25(f)-(g) plot the results for the intermediate period and low period of different days of the week in the late three months. These figures show that there is a larger difference during different months for non-peak periods, due to a smaller sample of the jobs in these periods.

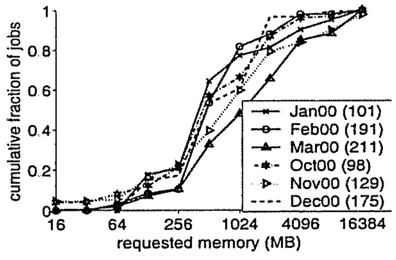
The results for the weekends are shown in Figures 26(a)-(c). As shown in these figures, due to a very small sample of jobs during these periods, there is more difference during different months and during different periods of arrival rate for the weekends. Recall that as shown in Figure 25(b), the distributions of total requested memory for Saturday and Sunday when computed over October - December 2000 is very similar to that of each weekday day.

The *normalized requested memory* for a job is defined as the amount of requested memory divided by the number of processors requested.

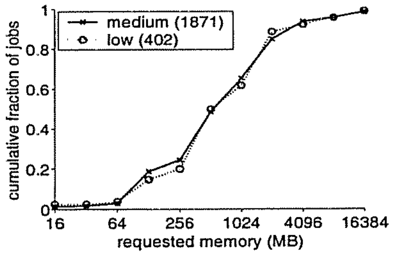
Figures 27(a) - (c) show that the distribution of normalized requested memory is very similar



(a) Intermediate Period Weekends Each Month

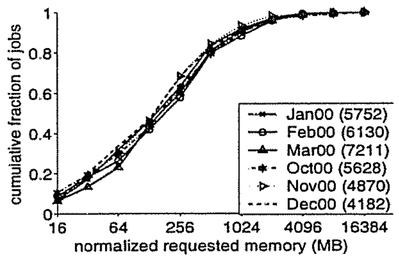


(b) Low Period Weekends Each Month

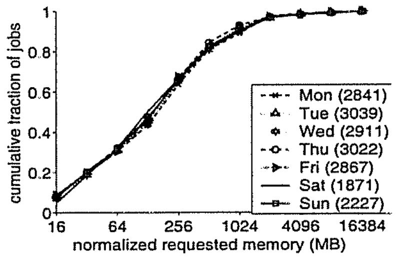


(c) Each Weekend Period Oct-Dec 2000

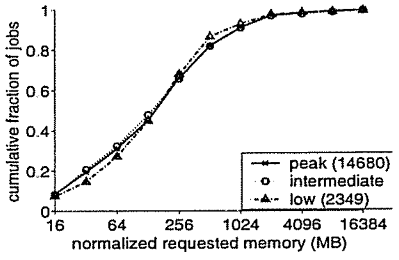
Figure 26. Variation in Distribution of Requested Memory During Weekends



(a) Each Month Peak Period



(b) Each Day of the Week Peak Period Oct-Dec 2000



(c) Each Arrival Period Weekdays Oct-Dec 2000

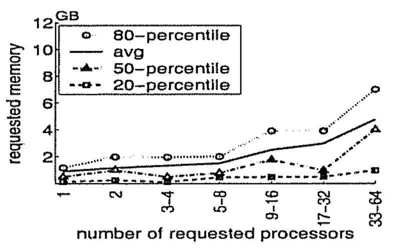
Figure 27. Variation in Distribution of Normalized Requested Memory During Weekdays

during different months, and during different periods of arrival rate, and different days of the week.

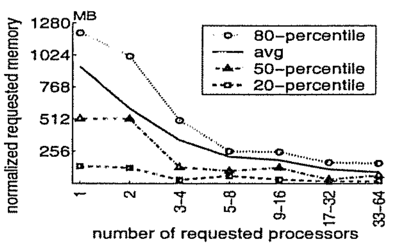
Similar to the distribution of requested memory for non-peak periods and weekend periods, the results for the distribution of normalized requested memory during these periods have a larger variation due to a small sample of jobs. The graphs for the normalized requested memory during non-peak periods and weekend periods are shown in Figures 88-89 of Appendix C.

To generate a synthetic workload, conditional distributions of requested memory are required. Next, we characterize the distribution of requested memory conditioned on the requested number of processors.

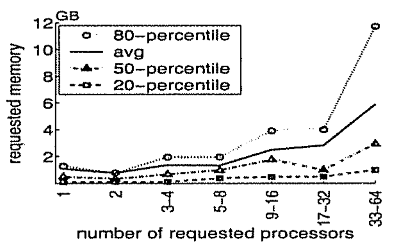
Figures 28 show that the requested memory (measured by the average, or the 20th, 50th or 80th percentile) has a strong correlation with the requested number of processors. Specifically,



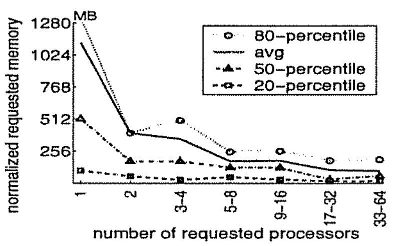
(a) Requested Memory (January-March 2000)



(b) Normalized Requested Memory (January-March 2000)



(c) Requested Memory (October-December 2000)



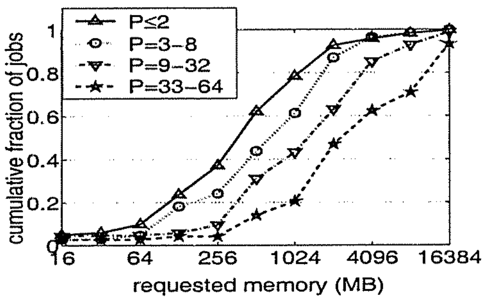
(d) Normalized Requested Memory (October-December 2000)

Figure 28. Requested Memory vs. Requested Processors

total requested memory is positively correlated with the requested number of processors, while normalized requested memory is negatively correlated with the requested number of processors. The results for Oct-Dec 2000 are slightly different from that for Jan-March 2000, in that jobs that request 33-64 processors in the late three months have higher 80th and lower 50th percentile of total requested memory, and jobs that request 2 processors have lower 80th and 50th percentile of normalized requested memory. To our knowledge, the correlations between these parameters of previous workloads have not been investigated.

Based on the results for October - December 2000 in Figure 28(c), Figure 29(a) provides the distributions of requested memory for four different ranges of the requested processors, with the mean and CV of the requested memory shown in the table in Figure 29(b). These conditional distributions can be used to generate the distribution of requested memory, after the number of requested processors is determined.

Recall from Table 1 that the total memory available per processor on the O2K is either 256 MB or 512 MB, depending on the host. As shown in Figures 25(a) and 28(a), 35-40% of



(a) Conditional Distributions

P	M		#
	Mean (GB)	CV	Jobs
≤ 2	1.05	1.95	5293
3 - 8	1.35	1.36	5713
9 - 32	2.61	1.31	3831
33 - 64	5.89	1.04	643

(b) Mean and CV

Figure 29. Conditional Distributions of Requested Memory Given Requested Processors

(October-December 2000)

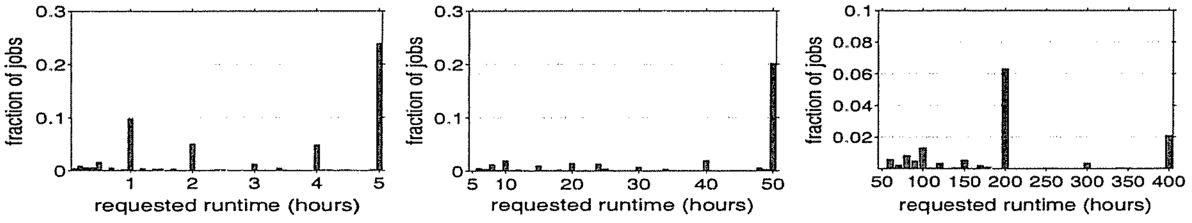
all jobs have normalized requested memory greater than 256 MB; furthermore 15-20% of all jobs and 50% of the sequential jobs have normalized requested memory greater than 0.5 GB. In contrast, in an SP/2 system where about 80% of the nodes had 128 MB of memory, and a similar fraction of the jobs requested this smallest possible normalized memory [Hot96].

4.4.3 Requested Runtime

To our knowledge, previous papers have not reported the distribution of requested job runtime, although this job measure is used by many job scheduling policies.

Figure 30 shows the distribution of requested runtime for jobs submitted during weekday peak periods, and Figure 31 compares the distributions of requested runtime for jobs submitted during different periods of approximately stationary arrival rate. General observations about these distributions include the following.

- As shown in Figures 30(a)-(c), a large fraction of jobs request the default runtime for the job class (i.e., 5, 50, 200, or 400 hours). These requested runtimes have no special meaning for the current scheduler on the O2K, except that jobs that request up to 5 hours of runtime are eligible to run on the three hosts that have higher priority for jobs that request a dedicated host.



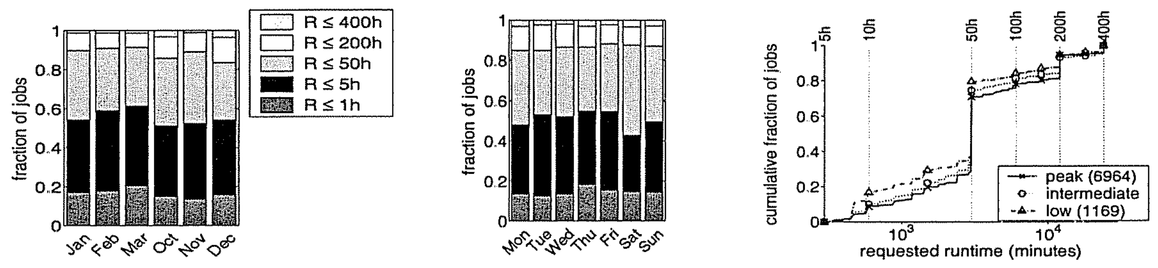
(a) 0-5 hours (vst)
bucket = 6 minutes

(b) 5-50 hours (st)
bucket = 1 hour

(c) 50-400 hours (mt, lt)
bucket = 10 hours

Figure 30. Distribution of Requested Runtime During Weekday Peak Periods

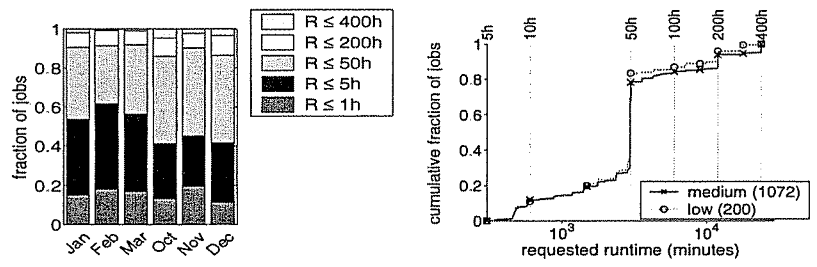
(October - December 2000)



(a) Weekday Peak Periods Each Month

(b) Weekday Peak Periods Each Day of Week (October-December 2000)

(c) Each Weekday Arrival Period (5 hours < R ≤ 400 hours) (October-December 2000)



(d) Weekend Intermediate Periods Each Month

(e) Each Weekend Arrival Period (5 hours < R ≤ 400 hours) (Oct-Dec 2000)

Figure 31. Variation in Distribution of Requested Runtime (R)

- Nearly all other jobs request some 'round' number such as 1, 2, 10, 20, or 300 hours, with approximately *uniform frequency* for each such requested runtime between a pair of default values (Figures 30(a)-(c)).

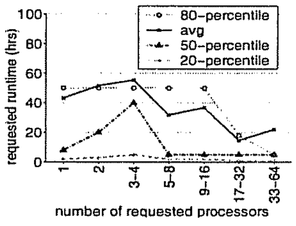
- The distribution of requested runtime is similar for jobs submitted in different months, although a somewhat larger fraction of jobs request 50 - 400 hours during October through December 2000 (Figure 31(a)). Similar results are observed for the weekends of different months, as shown in Figure 31(d).

- The distribution of requested runtime is similar for different days of the week. Over each three month period, but not during each month within the period, there are slightly fewer vst jobs submitted on Mondays and Saturdays (Figure 31(b)).
- Allowing for statistical variations in finite sample sizes, the distribution of requested runtime is very similar during different weekday arrival rate periods (Figure 31(c)). The distributions during different weekend periods of arrival rate (i.e., intermediate and low as shown in Figure 31(e)) are also similar.

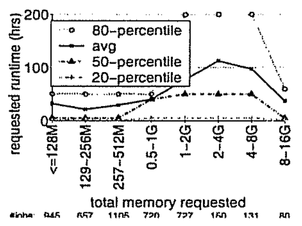
To generate a synthetic workload, the conditional distributions of requested runtime are required. We analyze the distribution of requested runtime for each range of requested processors paired with each range of requested memory.

Figure 32(a) plots the distribution of requested runtime (measured by the mean, and the 80th, 50th, and 20th percentiles) for each range of the number of processors. The figure shows that there is no appreciable correlation between the requested runtime and the number of processors and that the distribution for jobs that request over 32 processors are significantly different from that of the other jobs.

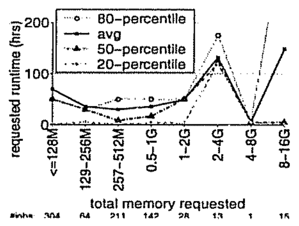
For each range of the number of processors, Figures 32(b)-(h) plots the distribution of requested runtime versus requested memory. As shown in these graphs, the requested runtime has a complex relation with the requested processors and requested memory, but the following observations can be made. For the serial jobs (Figures 32(b)), those that request larger than 1 GB of memory have higher 80th percentile and mean requested memory than the other jobs. For the 2-processor jobs (Figures 32(c)), the 50th percentile requested runtime jobs request 257 MB - 512 MB is lower than that for other jobs. Note that the requested runtime for 4-8



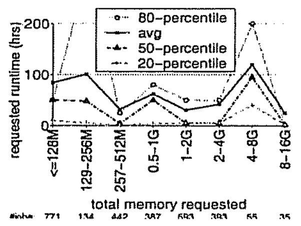
(a) All Jobs
Conditioned on P



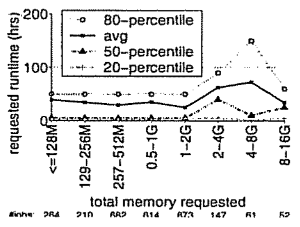
(b) P = 1
Conditioned on M



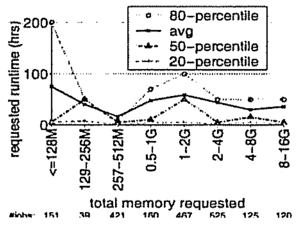
(c) P = 2
Conditioned on M



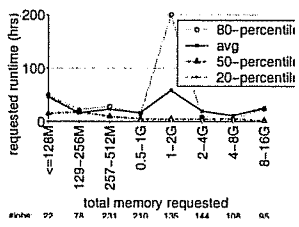
(d) P = 3 - 4
Conditioned on M



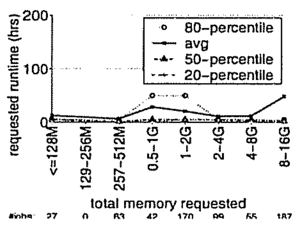
(e) P = 5 - 8
Conditioned on M



(f) P = 9 - 16
Conditioned on M



(g) P = 17 - 32
Conditioned on M

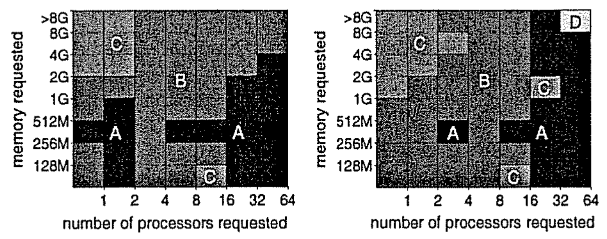


(h) P = 33 - 64
Conditioned on M

Figure 32. Distribution of Requested Runtime Conditioned on Requested Processors and Memory

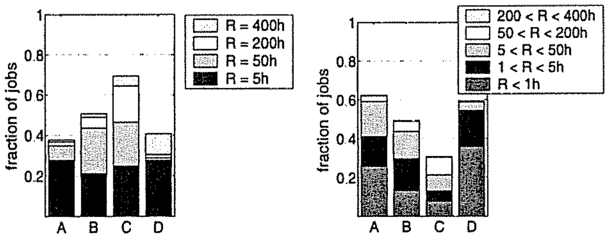
(October - December 2000)

GB is extremely low, but there is only one job in that range, and thus are not statistically meaningful. For jobs that request 3 or 4 processors (Figures 32(d)), the requested runtime for jobs that request 257 - 512 MB of memory is significantly lower than that of the other jobs. For jobs that request 5-8 processors (Figures 32(e)), those that request over 2 GB of memory have higher 80th percentile and mean requested runtime. For jobs that request 9-16 processors (Figures 32(f)), the jobs that request $\leq$ 128 MB of memory have significantly higher 80th percentile of the requested runtime than that for the other jobs, and the jobs that request 257 - 512 MB of memory have significantly lower mean and 80th percentile of requested runtime than that of the other jobs. For jobs that request 17-32 processors (Figures 32(g)), those that request 1-2 GB have significantly higher 80th and mean requested runtime than that of the other jobs. Finally, for jobs that request over 32 processors (Figures 32(h)), those that request over 8 GB of memory have significantly higher mean requested runtime than that of other jobs. Based on these analyses, for each range of requested processors, we partition the jobs according



(a) Jan-March 2000 (b) Oct-December 2000

Figure 33. Conditional Distributions (A-D) of Requested Runtime Given Requested Processors and Requested Memory



(a) Default Values of R (b) Ranges Between Defaults

Figure 34. The Distribution Functions of Requested Runtime

(October - December 2000)

Table 9. Mean and CV of Conditional Distributions of Requsted Runtime (R)

Conditional Distribution	# Jobs	All R		Default R		NonDefault R	
		Mean (hrs)	CV	Mean (hrs)	CV	Mean (hrs)	CV
A	2207	20.9	2.62	33.0	2.23	13.6	2.78
B	10828	41.0	1.74	58.8	1.45	22.8	2.02
C	1458	81.5	1.27	97.8	1.17	44.6	1.37
D	187	48.4	2.52	112.6	1.52	4.4	4.08

(October - December 2000)

to the requested memory, with the jobs that have (very approximately) similar distribution of requested runtime in the same set. For each set of jobs, the distribution of the requested runtime can be plotted as that in Figure 31 for further examination. Whenever appropriate, different sets of the jobs are combined to simplify the distributions.

The analysis reveals four distributions (i.e., A - D) of requested runtime conditioned on the requested processors and requested memory, as defined in Figure 33. The distributions are provided in Figure 34. Although the ranges of processors and memory requests over which each conditional distribution applies are complex, there are significant similarities between Figure 34(a) and (b), suggesting that the identified distributions are meaningful.

For the results in Oct - December 2000, Figure 34(a) plots the probability mass on the default requested runtime (5, 50, 200, and 400 hours), and Figure 34(b) plots the probability mass on the non-default requested runtime, for each of the four conditional distributions of requested runtime. The mean and coefficient of variation (CV) of the requested runtime for each distribution are provided in Table 9. The requested runtime of each job can be generated using these conditional distributions, after the requested processors and requested memory of the job have been determined.

The features of the conditional distributions of requested runtime are: (1) distribution A has a high fraction (about 70%) of the jobs requesting ≤ 5 hours; (2) distribution B has a 20% higher fraction of jobs requesting 50 hours and a lower fraction of jobs requesting ≤ 5 hours, compared to distribution A; (3) distribution C has a 20% higher fraction of jobs requesting > 50 hours especially 200 hours, and a lower fraction of jobs requesting < 5 hours, compared to distribution B; (4) distribution D has a significantly higher fraction of jobs requesting 400 hours and a significantly higher fraction of jobs requesting < 1 hour, compared to any other distribution. Note distribution D is for the largest jobs (i.e., $P > 32$ and $M > 8$ GB) in October - December 2000 only.

Table 9 also shows the difference in these distributions in terms of mean and CV. For both default requested runtimes and non-default requested runtimes, the distribution C has a mean requested runtime (close to 100 hours for default requested runtimes) that is about 1.5 times that of distribution B and about 3 times that of distribution A. The distribution D has an even higher mean requested runtime for defaults than that of distribution C, but distribution D has a very small mean requested runtime (i.e., under 5 hours) for the nondefaults, compared to over 40 hours for distribution C.

4.4.4 Summary of Requested Resources

The procedure for creating a synthetic workload that approximates the resource requests of the O2K weekday peak workloads can be summarized as follows:

- Job interarrival times have a two-stage hyperexponential distribution, with mean approximately in 2 minutes for a typical peak period (Figure 20(a)) about 4 minutes for a typical intermediate period (Figure 20(d)), and about 15 minutes for a typical low period (Figure 20(e)), and CV in the range of 1-2.

- Requested number of processors has a specialized harmonic distribution that emphasizes powers of two and small numbers, as shown in Figure 21-23.

- The distributions of requested memory, conditioned on the requested number of processors, are given in Figure 29.

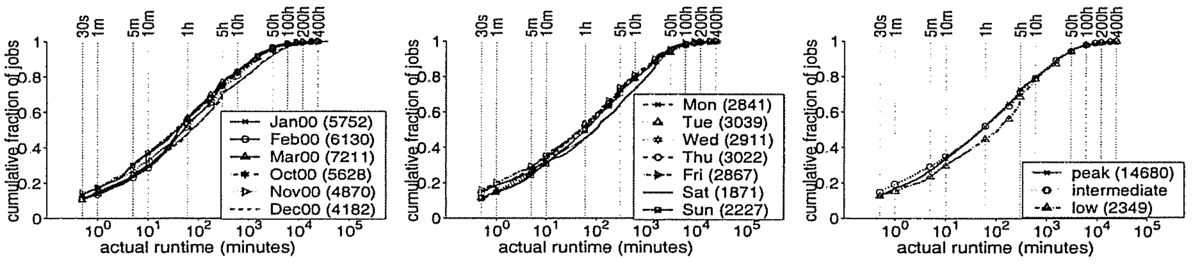
- The requested runtime distributions, conditioned on both the requested number of processors and requested memory, are given in Figures 33 - 34 and Table 9. These specialized distributions have significant probability mass at 5, 50, 200, and/or 400 hours, and relatively uniform probability for round numbers between these values, as illustrated in Figure 30.

4.5 Characterizations of Job Resource Usage

Sections 4.5.1 and 4.5.2 provide the distributions of actual job runtime and memory usage, respectively. Relationships among these quantities and the job resource requests for processors, memory, and runtime are also provided. Overall average utilization of allocated memory per month is compared against overall average utilization of allocated processors per month. Distribution of processor utilization was analyzed in Section 4.4.1. Section 4.5.3 summarizes the execution characteristics.

4.5.1 Actual Runtime

Many previous papers report the distribution of actual runtime of the jobs that arrive over a period of several months, without studying whether the workload is approximately stationary.



(a) Each Month (Weekday Peak Period) (b) Each Day of the Week (Peak Arrival Period) (c) Each Arrival Rate Period (Weekdays)

Figure 35. Variation in Distribution of Actual Runtime

Several papers have provided the distribution of actual runtime for different ranges of requested processors.

Figure 35(a) shows that the distribution of actual runtime is very similar for jobs submitted during the weekday peak arrival rate period of different months, except that in December 2000, a slightly smaller fraction of the jobs have runtime for under 1 hour, and a slightly higher fraction of the jobs have runtime for 50-100 hours than that in other months. Figure 35(b) shows that the distribution of actual runtime is very similar for jobs submitted during different days of the week in the peak arrival rate period, except that in Saturdays, a slightly higher fraction of the jobs have runtime for under 1 hour and a slightly higher fraction of the jobs have runtime for 5-10 hours than in other days of the week. Figure 35(c) shows that the distribution of actual runtime is very similar for jobs submitted during peak and intermediate periods on weekdays, a slightly smaller fraction of the jobs submitted during the low period have runtime under hours and a slightly higher fraction of the jobs submitted during the low period have runtime in 1-5 hours, that that of the jobs in peak and intermediate periods.

The CV of the overall runtime distribution for the O2K workloads during October - November 2000 is approximately 3, and is higher (i.e., 3 - 3.5) in January - March 2000. A runtime CV in the range of 2-5 has been reported for several previous parallel workloads [FN95, WLF$^+$96, Gib97b].

Similar to other workloads [FN95, WLF$^+$96, Hot96, SGS96], a significant fraction of the

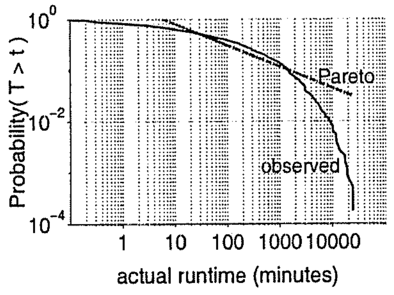


Figure 36. Distribution of Actual Runtime on Log-Log Scale

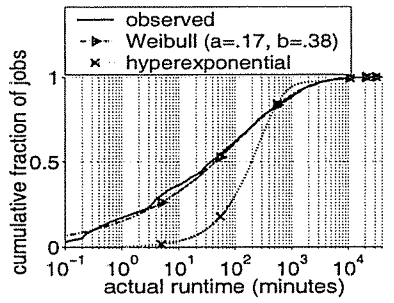
(Oct 2000 (Weekday Peak)

O2K jobs are very short running. For example, over 30% of the jobs are under 10 minutes, which is very short relative to the longest runtime of 400 hours for the O2K workload.

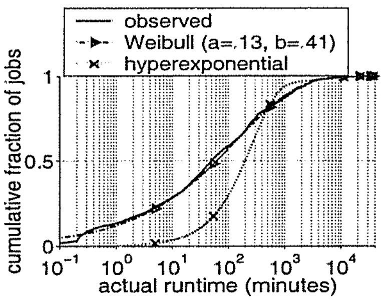
Previously proposed models for the actual runtime based on the production parallel workloads include the two-stage hyperexponential distribution [Fei96, DF99], the hyperErlang distribution [JPF+97], and the piecewise log-uniform distribution [Dow97b, DF99]. The Pareto distribution was proposed for parallel jobs on the specialized distributed-server systems, based on the observed distribution of actual runtime on sequential Unix computers [HB00].

Figure 36 plots the distribution of actual runtime and a Pareto distribution on a log-log scale for the given month (October 2000). It shows that it is not a heavy-tail distribution, i.e., the Pareto distribution. Note that the Pareto distribution on a log-log scale is always a straight line.

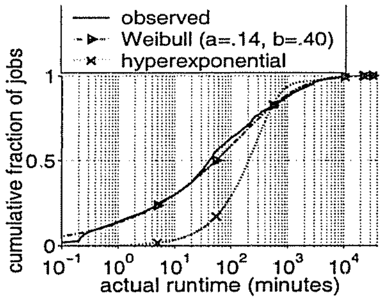
Figures 37(a)-(f) plot the fit of the models for the distribution of actual runtime for each weekday period of each month. These graphs illustrate that the Weibull distribution provides a much better fit than that of the two-stage hyperexponential distribution for the observed distribution of actual runtime. Using the algorithm proposed in [JPF+97] to fit the hyper-Erlang distribution for the observed distribution results in another two-stage hyperexponential distribution that matches the first three moments of the observed distribution (not shown for more readable graphs). The fit of this hyperexponential distributions is similarly poor as the hyperexponential distribution shown in each graph. Note that the paper [JPF+97] does not



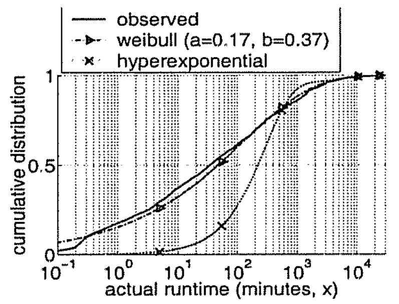
(a) Jan 2000 (Weekday Peak) (mean: 514 min, cv: 3.43)



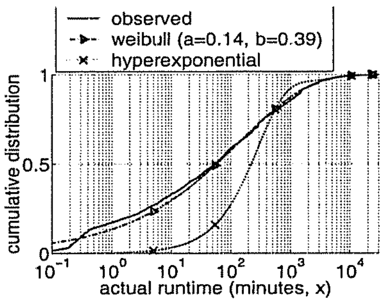
(b) Feb 2000 (Weekday Peak) (mean: 521 min, cv: 3.30)



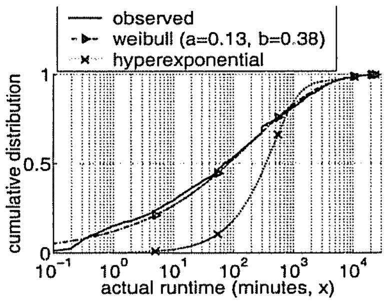
(c) March 2000 (Weekday Peak) (mean: 530 min, cv: 3.22)



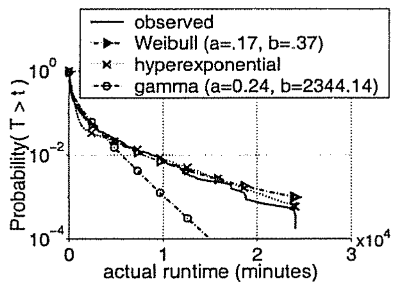
(d) Oct 2000 (Weekday Peak) (mean: 566 minutes, cv: 3)



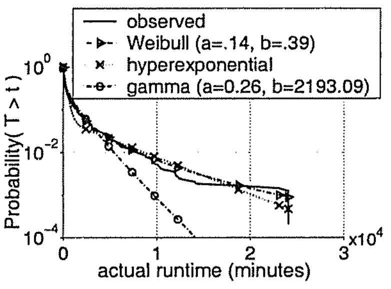
(e) Nov 2000 (Weekday Peak) (mean: 562 min, cv = 2.91)



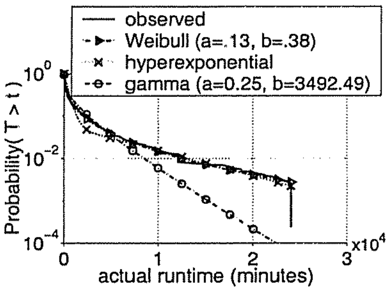
(f) Dec 2000 (Weekday Peak) (mean: 873 min, cv: 2.82)



(g) Oct 2000 (Weekday Peak) (mean: 566 minutes, cv: 3)



(h) Nov 2000 (Weekday Peak) (mean: 562 min, cv = 2.91)

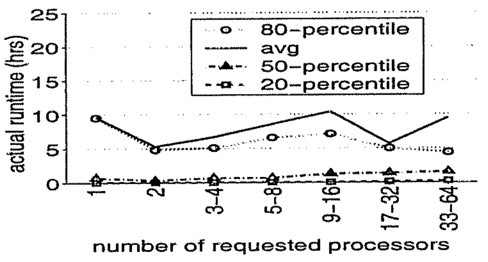


(i) Dec 2000 (Weekday Peak) (mean: 873 min, cv: 2.82)

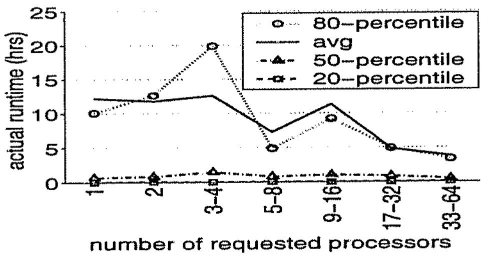
Figure 37. The Models for the Distribution of Actual Runtime (T)

provide data to show how well the hyperErlang distribution model proposed in their paper match the observed distribution of processing time in the SP/2 workload.

To show the fit of the tail of the distribution, Figures 37(g)-(i) plot the results of the cumulative distributions on a log scale of the actual runtime. The gamma distribution is included in these three graphs. The figures show that the gamma distribution does not fit the



(a) Jan-March 2000



(b) Oct-Dec 2000

Figure 38. Actual Runtime vs. Requested Processors

tail of the observed distribution.

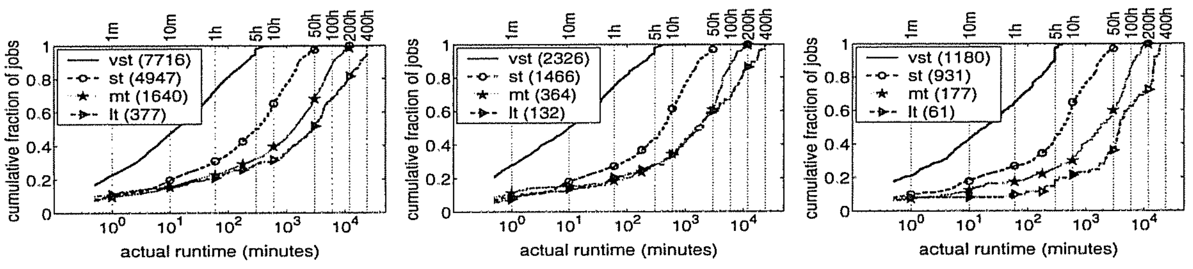
Finally, since the observed distribution is piecewise linear in the logarithm of the actual runtime, the piecewise log-uniform distribution also fits the observed distribution, but it is more complex than Weibull distribution.

Similar results are derived using the above models to fit the observed distribution of the processing time (i.e., the product of the number of requested processors and the actual runtime). That is, the Weibull and log-uniform distributions fit the observed distribution well, wheres the hyperexponential, gamma, and Pareto distributions do not.

Figure 38(a) plots the distribution (measured by average and 80th, 50th, and 20th percentiles) of the actual runtime for each range of requested processors, for the three-month workload in January - March 2000. Figure 38(b) plots the results for the late three months in October - December 2000. The graphs show that there is no appreciable correlation between the actual runtime and requested number of processors, in contrast to previous workloads on an iPSC/860 [FN95], a Paragon [WLF$^+$96]. Our results are more similar to that reported in [Hot96] for an SP/2 workload in that there is no clear correlation between these the actual runtime and the number of requested processors.

Actual Runtime vs. Requested Runtime

Previous workload studies report on the correlation between actual runtime and number of requested processors [FN95, WLF$^+$96, Hot96] as well as the distribution of the ratio of actual



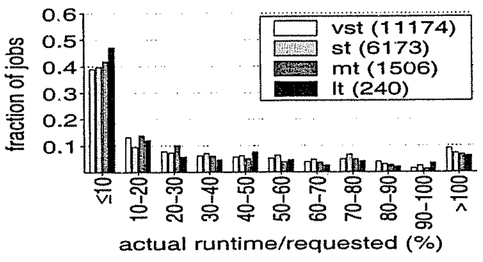
(a) Weekday Peak Periods (b) Weekday Intermediate Periods (c) Weekday Low Periods

Figure 39. Distributions of Actual Runtime Conditioned on Requested Runtime Class (October - December 2000)

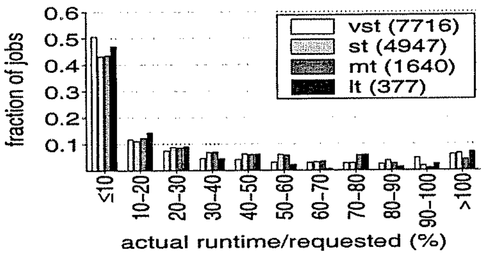
to requested runtime [FW98], but not (to our knowledge) on the correlation between the ratio of actual runtime to requested runtime with other job measures (requested runtime, requested processors, and requested memory).

Figures 39(a)-(c) plot the distribution of actual runtime for each of the default requested runtime class (i.e., vst, st, mt, and lt) for weekday peak, intermediate, and low, respectively, of the late three-month period during October - December 2000. It shows that there is a large discrepancy between the actual and requested runtime for most jobs. For example, for the jobs that request over 5 hours of runtime (i.e., st, mt, and lt classes), approximately 10% of the jobs terminate in under 1 minute. These premature job completions may be due to unanticipated errors in the codes, but it's surprising that 10% of all jobs that request over 5 hours would terminate due to an unanticipated error in the code. Furthermore, more than 20% of the jobs that request runtime greater than 50 hours actually run for 10-50 hours, and 30% of the jobs that request runtime greater than 200 hours actually use 10-100 hours. Thus, a significant fraction of the inaccurate runtime requests appear to be due to simple misestimation of the actual runtime, perhaps partially encouraged by the default requested runtime values defined in the system.

Figure 40 further plots the distribution of the ratio of the actual runtime to requested runtime for each requested runtime class for the two three-month workloads. For *any* requested



(a) January - March 2000



(b) October - December 2000

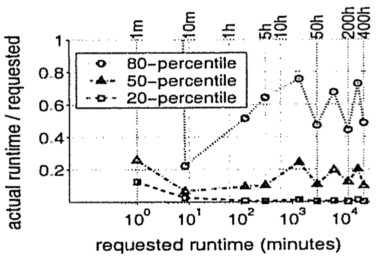
Figure 40. Distributions of Actual Runtime / Requested Runtime vs. Requested Runtime Class

runtime class, a large fraction ($> 40\%$) of the jobs use only a small fraction ($\leq 10\%$) of the runtimes requested. On the other hand, a significant fraction (about 5-10%) of the jobs use more than the runtimes requested. Note that current NCSA-MS* allows each job to run one hour longer than requested before killing the job.

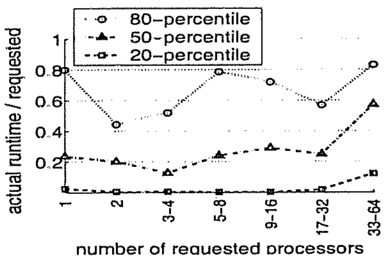
Previous paper [FW98] reports the distribution of actual runtime to requested runtime of all jobs and shows that there is a large discrepancy between requested and actual runtimes for an SP/2 workload. The main differences between the SP/2 and the O2K workloads are that possibly due to larger default requested runtimes defined on the O2K (i.e., 5, 50, 200, and 400 hours), a much higher fraction ($> 40\%$) of the O2K jobs than that (about 10%) of the SP/2 jobs use under 10% of requested runtimes, and a somewhat lower ($< 10\%$) fraction of the O2K jobs than that (about 16%) of the SP/2 jobs are killed due to exceeding the requested runtimes.

A key question is how to generate actual runtimes in a synthetic workload if requested runtimes are generated as discussed in Section 4.4.3. The percentiles in Figure 41(a) were computed for the default requested runtimes (i.e., 5,50,200, and 400 hours), each range between the defaults, and the following ranges of requested runtime (R): R=1 minute, 1 min. $< R \leq 10$ min., 10 min $< R <$ 5 hrs.[5]. This figure shows that the ratio of actual to requested runtime is statistically somewhat higher for jobs that request greater than 5 hours of runtime but not one of the default values than for jobs that request one of the default values.

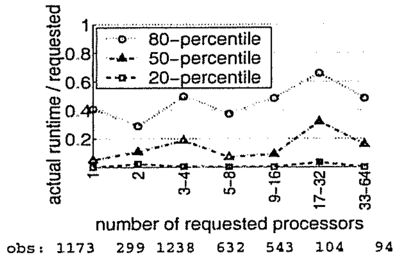
[5]The value of each percentile for each range of runtime is plotted against the average requested runtime in the range.



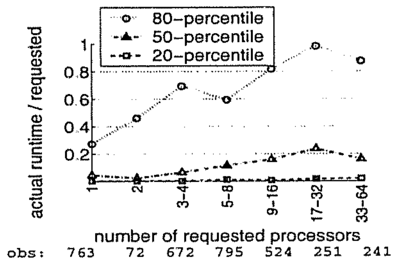
(a) Ranges of Requested Runtime (R)



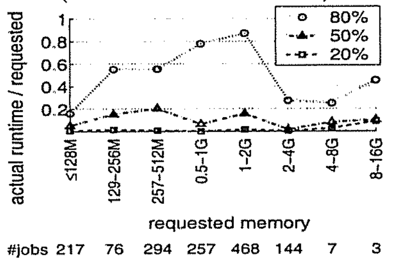
(b) Ranges of Requested Processors (R Between Defaults)



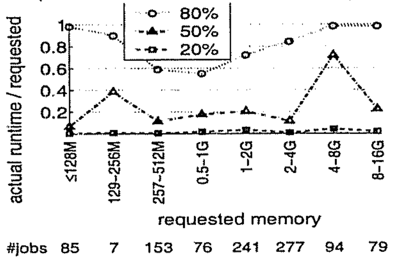
(c) Ranges of Requested Processors (R = 50, 200, or 400 hrs)



(d) Ranges of Requested Processors (R = 5 hours)



(e) Ranges of Requested Memory (R = 5 hrs, P = 3 - 8)



(f) Ranges of Requested Memory (R = 5 hrs, P > 8)

Figure 41. Actual Runtime/Requested Runtime vs. Various Request Sizes

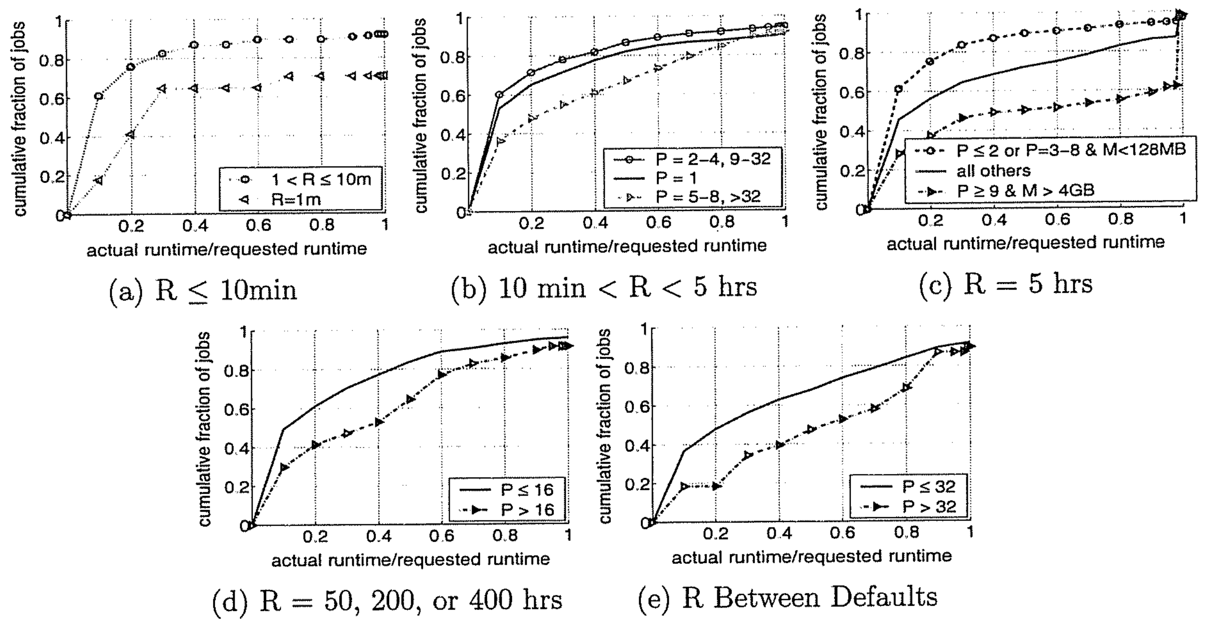
(October-December 2000)

Figure 41(b) shows that the distribution for the requested runtime between defaults is similar for each number of requested processors up to 32, but has a much higher fiftieth percentile if the requested number of processors is greater than 32.

Figure 41(c) plots the results for the default runtime requests greater than 5 hours. It shows that the distribution for the jobs has a higher 50th percentile if the jobs also request over 16 processors.

Figure 41(d) plots the results for the requested runtime equal to 5 hours. It shows that the distribution has higher 50thand 80th percentile if the jobs also request over 16 processors, and that the distribution for 3 - 8 processors has higher 50th percentile than that for 1 and 2 processors.

For categories of requested runtime and requested number of processors that have a similar distribution of the ratio of actual runtime to the requested runtime, we plot the distribution of



(a) R ≤ 10min (b) 10 min < R < 5 hrs (c) R = 5 hrs

(d) R = 50, 200, or 400 hrs (e) R Between Defaults

Figure 42. Conditional Distributions of Actual Runtime/Requested Runtime Given Requested Runtime, Requested Processors, and Requested Memory
(R: Requested Runtime)

the ratio versus different ranges of requested memory. Our intent is to identify the distributions that are *significantly* different from the other distributions. Approximations are used to keep the model reasonably simple.

For example, for requested runtime equal to 5 hours, Figure 41(d) shows that the requested number of processors equal to 3 - 8 have similar distribution of the ratio; $P \geq 9$ have statistically higher 80th and 50th percentiles of the ratio than for $P < 9$. Thus, for requested runtime equal to 5 hours, the jobs are further partitioned into three categories according to the number of requested processors: 1-2; 3-8; and > 8. The distributions of the ratio of actual to requested runtime versus requested memory for each category are further examined. Figure 41(e) plots the results for $P = 3$-8 and it shows that the distribution has a significantly lower 80th percentile for the requested memory under 128 MB than that of the other jobs. Figure 41(f) plots the results for $P > 8$. It shows that the distribution is significantly different for requested memory greater than 4 GB; in all other cases, the distribution is not significantly sensitive to the requested memory. Similar analysis are performed for the requested runtime in 10 minutes - 5 hours.

Table 10. Mean and CV of Conditional Distributions of Actual Runtime (T) / Requested Runtime (R)

Given Conditions		Actual Runtime/Requested Runtime					
Requested Runtime	Requested Processors and Memory	≤ 1			> 1		
		Mean	CV	#Jobs	Mean	CV	#Jobs
$1 < R \leq 10$ min	all	0.12	1.41	149	5.01	0.68	13
R = 1m	all	0.23	0.67	12	46.06	0.47	5
$10m < R \leq 5$ hrs	P = 2-4, 9-32	0.15	1.44	1601	1.70	0.68	90
	P = 1	0.16	1.39	1160	1.76	0.36	124
	P = 5-8, > 32	0.29	1.00	1144	1.48	0.42	100
R = 5 hours	$P \leq 2$, P = 3-8 & M < 128MB	0.14	1.58	1021	1.27	0.38	31
	all others	0.28	1.20	1998	1.13	0.08	91
	P > 8 & M > 4GB	0.51	0.83	174	1.21	0.00	3
R = 5, 50, 200, 400 hrs	$P \leq 16$	0.19	1.20	3722	1.02	0.06	163
	P > 16	0.32	0.85	95	1.02	0.01	9
Ranges Between Defaults	$P \leq 32$	0.29	1.03	2604	1.05	0.04	239
	P > 32	0.46	0.69	34	1.04	0.02	4

(October - December 2000)

Based on these analysis results, Figure 42 provides the distributions of the ratio of actual to requested runtime, conditioned on the requested runtime, requested processors, and requested memory. Table 10 provides the mean, CV, and number of jobs for each conditional distribution in Figure 42, with the ratio (T/R) ≤ 1 and the ratio > 1 shown separately.

Recall that actual runtime may exceed requested runtime by one hour before the job is killed. The results in Figure 42(a) show that a significant fraction of the jobs that request one minute of runtime exceed their request. The results in Figure 42(c) show that a significant fraction of the jobs that request 5 hours of runtime, > 8 processors, and > 4 GB of memory, are tuned to run for exactly 5 hours.

A synthetic workload generator can use the distributions provided in Figure 42 to generate the actual runtime as a fraction of requested runtime after requested runtime (Section 4.4.3), number of processors (Section 4.4.1), and requested memory (Section 4.4.2) have been generated.

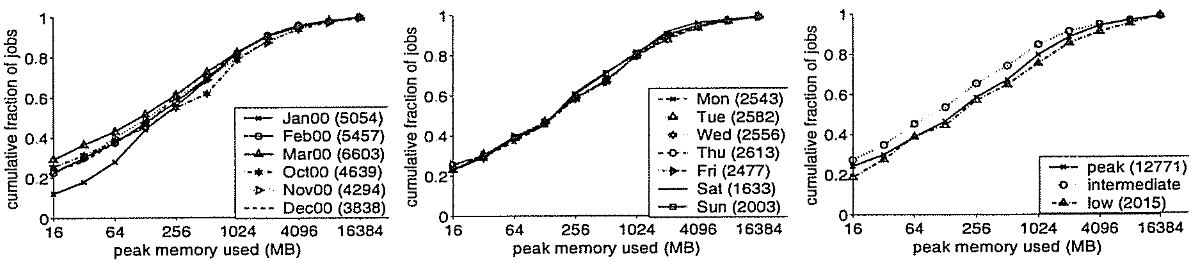
4.5.2 Actual Memory Usage

For each job, the JMD process records the memory usage of the job every 30 seconds. The format of the jmd log is described in Appendix A. A small fraction of the jobs that run over 1 minute but do not have the information of the memory usage. These jobs are not included in our analysis of memory usage in this section.

The peak memory used by each job is the maximum memory over 30-second periods reported for the job. The normalized memory is the ratio of the peak memory divided by the requested number of the processors of the job. The average memory is computed as the total memory time used by the job divided by the actual runtime of the job. The peak memory efficiency is defined to be the peak memory divided by the requested memory of the job. Similarly, the average memory efficiency is defined to be the average memory usage divided by the requested memory of the job.

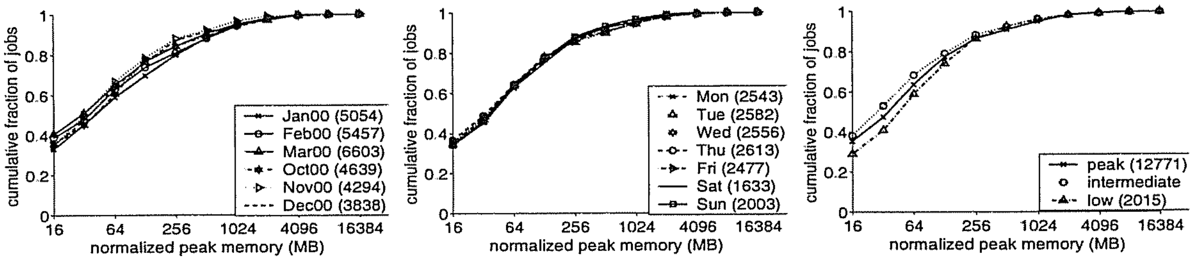
Figure 43(b) shows that the distribution of peak memory usage is similar across different days of the week. Figures 43(a) and (c) show that there is some variability in the distributions of peak memory usage over different months and over different arrival rate periods. In particular, jobs submitted during intermediate arrival rate periods tend to have somewhat lower peak memory usage. Similarly, Figures 44(a)-(c) show the variation in the distribution of normalized peak memory usage over different months (graph(a)) and over different arrival rate periods (graph (c)) and the similarity over different days of the week (graph (b)). In the remainder of this section we focus on further characterizing representative peak memory usage for jobs submitted during weekday peak arrival rate periods in October through December 2000. Parameters of the representative distributions for these periods could be adjusted to reflect the observed variations for different months or for the intermediate arrival rate period.

Similar to previous workloads [Fei97a, SSN99], Figures 44(a)-(c) shows that a large fraction (i.e., approximately 50%) of the O2K jobs use a very small (i.e., under 32 MB) peak memory usage per processor. On the other hand, another significant fraction (about 5%) of the jobs on



(a) Each Month (Weekday Peak Periods) (b) Each Day of the Week (Peak Periods Oct-Dec 2000) (c) Each Arrival Period (Weekdays Oct-Dec 2000)

Figure 43. Variations in Distribution of Peak Memory Used



(a) Each Month (Weekday Peak Periods) (b) Each Day of the Week (Peak Periods Oct-Dec 2000) (c) Each Arrival Period (Weekdays Oct-Dec 2000)

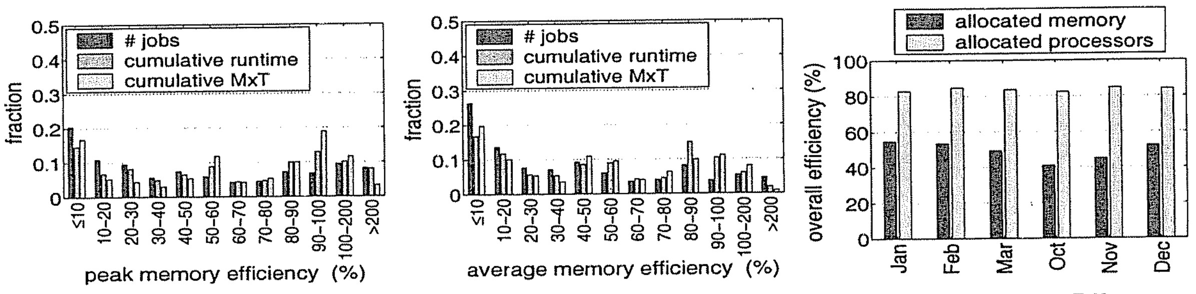
Figure 44. Variations in Distribution of Normalized Peak Memory Used

(Normalized Peak Memory = Peak Memory Used/Number of Requested Processors)

the O2K have normalized peak memory usage greater than 1 GB per processor.

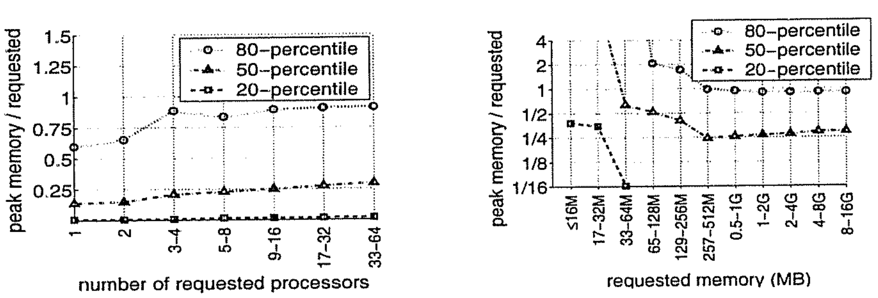
Figure 45 shows that there is a large discrepancy between requested memory and memory utilized per job. In particular, Figures 45(a) and (b) show that 15-20% of the jobs have peak or average memory usage higher than their requested memory; for the remaining jobs, the respective ratio of peak or average memory usage to the requested memory is distributed fairly uniformly over each 10% range from 10-100%. A similar result was reported for a CM-5 workload in [Fei97a]. Figure 45(c) shows that average memory efficiency (i.e., the utilization on the memory allocated) each month in in the range of 40-55%, wheres the overall utilization on the processors allocated each month is around 80%. As with the large discrepancy in requested runtime on the O2K, the scheduling performance may be improved if memory requests were more accurate.

Peak memory usage is an important job characteristic in that it defines the amount of



(a) Peak Memory Efficiency (b) Average Memory Efficiency (c) Overall Average Efficiency

Figure 45. Memory Efficiency



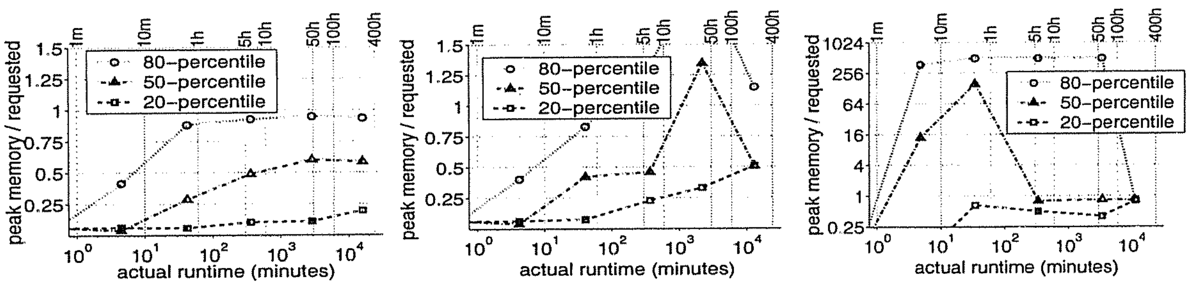
(a) Ranges of Requested Processors (b) Ranges of Requested Memory

Figure 46. Peak Memory/Requested Memory For Requested Processors and Memory

memory that must be allocated to the job (in a shared memory system such at the O2K) in order to guarantee that the job doesn't experience any memory interference during execution.

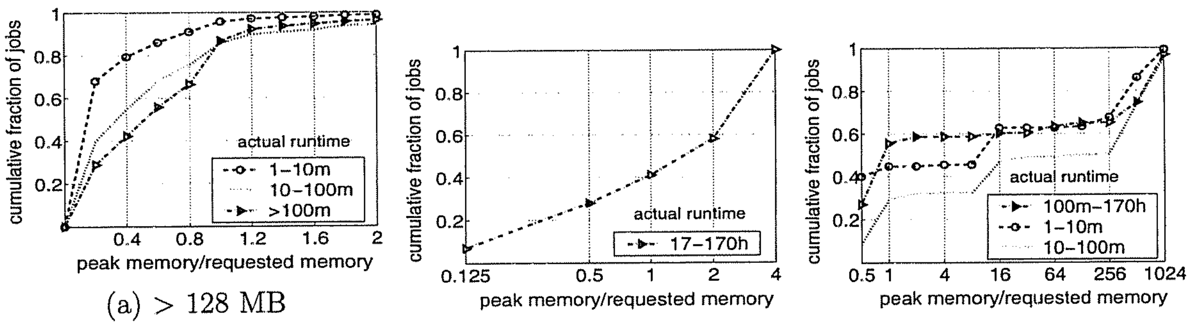
To further characterize peak memory usage for the purpose of creating synthetic workloads, we analyze the correlations between peak memory efficiency and other job characteristics. To that end, Figures 46 - 47 plot the percentiles of the peak memory efficiency per job, for ranges of requested number of processors, requested memory, and actual runtime.

Figure 46(a) shows that the peak memory efficiency per job is fairly insensitive to the requested number of processors. An implication of this result is that peak memory usage is positively correlated with the number of requested processors, since requested memory and requested processors are positively correlated (see Figure 28(c)). Similarly, there is a negative correlation between the normalized peak memory usage and the number of requested processors (see Figures 28(d)). In contrast, in the CM-5 [Fei97a] workload, the jobs with a larger number of processors not only use a larger amount of memory, but also a larger amount of per-processor



(a) M > 128 MB (b) M: 33-128 MB (c) M ≤ 32 MB

Figure 47. Peak Memory/Requested Memory For Actual Runtime



(a) > 128 MB
33-128 MB & 0-17h
33-128 MB & >170h

(b) 33-128 MB

(c) ≤ 32 MB

Figure 48. Conditional Distributions of Peak Memory/Requested Memory Given Requested Memory and Actual Runtime

(Weekday Peak Periods, October-December 2000)

memory.

Figure 46(b) shows that the distribution of peak memory efficiency is significantly different for jobs that request fewer than 128 MB of memory than for jobs that request more than 128 MB of memory. From this figure, noting that very few jobs request 33-64 MB of memory and that the 80-percentile value for these memory requests is statistically unreliable based on the small number of jobs, we partition the jobs into three classes according to their memory request (i.e., ≤ 32 MB, 33-128 MB, and > 128 MB) and provide the percentiles of peak memory efficiency as a function of actual job runtime for each class in Figures 47(a) - (c).[6]

Note that all jobs that have runtime under one minute use a very small fraction of their

[6]The ranges of actual runtime over which the percentiles in Figure 47(a)-(c) are computed are: 0-1 minute, 1-10 minutes, 10-100 minutes, 1.7-17 hours, 17-170 hours, and above 170 hours.

Table 11. Mean and CV of Conditional Distributions of Peak Memory / Requested Memory

Given Conditions		Peak Memory/Requested Memory					
		≤ 1			> 1		
Requested Memory	Actual Runtime	Mean	CV	#Jobs	Mean	CV	#Jobs
M > 128 MB	1m	0.00	13.03	890	-	-	0
	1-10m	0.17	1.52	1961	1.83	0.61	86
	10-100m	0.33	0.91	2449	3.88	1.56	407
	> 100m	0.44	0.75	4172	2.94	2.41	640
33 ≤ M < 128 MB	17-170hrs	0.39	0.63	193	1.97	0.20	277
M ≤ 32MB	1-10m	0.13	1.95	65	313.39	1.06	80
	10-100m	0.58	0.41	69	454.70	1.04	168
	100m-170h	0.59	0.30	35	515.85	0.70	28

(October - December 2000)

memory request; in fact, nearly all such jobs use less than 64 MB of memory.

The distribution of peak memory used as a fraction of requested memory is very similar for runtime greater than 10 minutes and memory request greater than 32 MB, except for jobs with memory request of 33-128 MB and actual runtime in the range of 1,000-10,000 minutes (i.e., approximately 17-170 hours). Thus, Figure 48 provides the requisite distributions for generating peak memory as a fraction of memory requested after requested memory and actual runtime have been generated as described earlier in this paper. Table 11 provides the mean and CV of the ratio of peak memory to the requested memory for each conditional distribution in Figure 48.

4.5.3 Summary of Job Resource Usage

To generate actual runtime and peak memory usage to complete a synthetic workload that is representative of the O2K weekday peak arrival rate workloads, these characteristics are obtained from the distributions in Figure 42 and 48, respectively. Processor utilization can also be generated from the distribution in Figure 24. To create a synthetic workload for intermediate arrival rate periods, the peak memory usage might be adjusted slightly as shown in Figure 43.

Low arrival rate periods have approximately the same distributions of actual runtime and peak memory usage as weekday peak arrival rate periods.

Note that, depending on the purpose of the synthetic workload, any of the characteristics that are not needed can be ignored.

4.6 Summary of Workload Characterization

This chapter provides a characterization of the large production parallel workload submitted to the NCSA O2K over two three month periods. This characterization is more complete than previous parallel workload characterizations in that new characteristics are provided (e.g., distributions of requested runtime, processor and memory utilizations, distribution of requested memory over more flexible range of possible requests), correlations among the characteristics are more fully explored, and conditional distributions are provided for generating synthetic workloads that include the observed correlations in the O2K workload. Another key difference in this analysis as compared with prior work is that job characteristics are provided for jobs that are submitted during periods of approximately stationary job arrival rate. From these characteristics we determined that the jobs submitted in different months or in different arrival rate periods are statistically very similar. The roadmaps for generating similar synthetic workloads are summarized in Sections 4.4.4 and 4.5.3.

Interesting characteristics of the O2K workload include: (a) the fifteen largest jobs in a typical month have average running time of over 200 hours and use an average of 4000-8000 processor hours, (b) most requested runtimes are default values (i.e., 5,50,200, or 400 hours), (c) whether or not a default runtime is requested, over half the jobs have actual runtime less than 20% of the requested value, and (d) overall utilization of allocated processors is approximately 80% whereas overall utilization of allocated memory is closer to 50%.

Some of the O2K workload characteristics that differ from previous workloads (e.g., the CV of job interarrival time equal to 1-2 instead of 2.5-6) are most likely due to measuring the O2K characteristics during periods of stationarity. Other differences (e.g., longer runtimes and larger memory requests) are most likely due to general trends in production parallel systems. Still other differences (e.g., lack of correlation between requested runtime and requested number of processors, or the large number of jobs with very inaccurate requested runtime) may either be due to the trend toward more widespread use of parallel/distributed computing, or may instead be reflective of the O2K usage and environment. Characterization of further modern production parallel/distributed computing workloads are needed to distinguish the trends from the environment-specific results.

Chapter 5

Performance Evaluation of Backfill Policies

This chapter presents results of our evaluation for the backfill policies by simulation using the O2K job traces that were characterized in Chapter 4.

As discussed in Chapter 3, our baseline policy is NCSA-LSF*. The key questions addressed in this chapter are:

- What is the relative performance of priority backfill policies versus NCSA-LSF* for the O2K?
- What is the 'best' priority backfill policy for the O2K?
- Can adding an immediate service with limited preemption for the backfill policies significantly improve the turnaround time for the short jobs, without impacting the performance of the other jobs?

As part of determining the best priority function for the backfill policies, we study the impact of the number of reservations and fixed versus dynamic-job reservation on the performance of backfill policies.

This chapter is organized as follows. Section 5.1 provides the definition of the priority functions used in each priority backfill policy evaluated. Section 5.2 compares NCSA-LSF* against FCFS-backfill and a new priority backfill policy, i.e., Priority-backfill that gives some priority to short jobs. We show that these two backfill policies significantly improve NCSA-LSF*. Based on our simulation results, a priority backfill policy, NCSA-MS*, which uses a slightly different priority function than in Priority-backfill replaced NCSA-LSF* for dispatching jobs on the O2K. The performance of NCSA-MS* is similar to that of Priority-backfill, which will be shown in Section 5.3. Section 5.3 also studies the impact of applying the weight for the requested processors and memory on Priority-backfill. In addition, an implementation error in NCSA-MS* was found recently; the error causes the reservations not able to be rescheduled sooner when a job departs earlier than expected. The impact of this error on the performance of NCSA-MS* is also evaluated in that section. Section 5.4 compares alternative priority backfill policies, i.e., Priority-backfill, SJF-backfill and LXF&W-backfill, each giving priority to short jobs. The effect of fixed-job and multiple reservations for backfill policies are also evaluated. Section 5.5 studies the potential benefit of adding a limited preemptive immediate service for backfill policies to improve the turnaround time for short jobs. Section 5.6 summarizes and concludes the performance evaluation results for the backfill policies.

5.1 Definition of Alternative Priority Functions

This section provides the definitions of the priority functions used in each priority backfill policy evaluated in this chapter. They are three new policies: Priority-backfill, Priority(P&M)-backfill, and LXF&W-backfill; and two previously proposed FCFS-backfill and SJF-backfill. In addition, this section also provides the definition of the priority measures in the Maui Scheduler configured for MHPCC SP/2 (i.e., MHPCC-MS*) and NCSA O2K (NCSA-MS*).

Table 12 provides the definition of the priority weights used in each policy (except for SJF-backfill). The definition of each job measure for which the priority weight applies is given in Table 13.

FCFS-backfill prioritizes jobs in order of job arrival time, or equivalently giving weight to job wait time only (i.e., $W_w = 1$ and all other weights are zero). SJF-backfill gives the highest

Table 12. Job Metrics and Weights for Backfill Policies

Weight Symbol	Weight Value						Job Measure
	FCFS -backfill	MHPCC -MS*	Priority -backfill	Priority (P&M) -backfill	NCSA -MS*	LXF&W -backfill	
W_w	1	60	1	1	240	0.02	J_w
W_x	-	10	5	5	1500	1	J_x
W_p	-	8	0.2	0	0	0	J_p
W_m	-	-	0	0	0	0	J_m
W_{pe}	-	-	0	0.2	100	0	$\max\{J_p, J_m\}$

Table 13. Definition of Job Measures

Job Measure	Defition
J_w	current job wait time in hours
J_x	estimated current job expansion $= \frac{J_w + \textit{requested runtime in hours}}{\textit{requested runtime in hours}}$
J_p	number of requested processors, J_p
J_m	requested memory in MB/358.4*, J_m

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

priority to the job that has the shortest requested runtime, which significantly improves the short jobs (thus the mean response time), but has the problem of starving long jobs.

The other priority backfill policies defined in Table 12 give priority to short jobs, but not as much as in SJF-backfill. MHPCC-MS* dynamically computes the priority of the jobs as a function of a weighted sum of the job waiting time, the estimated current job expansion factor, and the number of requested processors [1] as shown in Table 12. Note that under MHPCC-MS*, the priority of each job increases as the job waits in the queue because of a non-zero weight for the job waiting time (i.e., $W_w > 0$). Furthermore, the priority of the jobs with shorter requested runtime increases faster than that of the other jobs as the jobs wait in the queue, because of a non-zero weight for the estimated current job expansion factor (i.e., $W_x > 0$).

Priority-backfill is derived from MHPCC-MS*, except that a relatively higher weight (i.e.,

[1]MHPCC-MS* uses several additional job measures related to the site-specific political priority and the fairness, such as the user fairness, group fairness, and an urgency factor. Since these measures are not related to policy performance per se, they are not used in our evaluation.

W_x) is applied for the estimated current job expansion factor, in order to give higher priority to shorter jobs than in MHPCC-MS*. In our preliminary evaluation, we find that Priority-backfill has higher performance for the O2K workloads.

Since both processors and memory need to be scheduled on O2K, we also evaluate a variant of Priority-backfill; the Priority(P&M)-backfill policy is the same as Priority-backfill, except that it applies the weight for the larger value of the requested number of processors and normalized requested memory, rather than the number of requested processors (i.e., $W_{pe} > 0$ but $W_p = 0$).

NCSA-MS* also applies the weight for the larger value of the number of requested processors and the normalized requested memory, as in Priority(P&M)-backfill. The differences are that NCSA-MS* uses a slightly higher weight (i.e., W_{pe}) for this job resource request measure, but a slightly lower weight (i.e., W_w) for the current job waiting time, relative to the weight (i.e., W_x) for the estimated current job expansion factor. Note that the effect of increasing W_{pe} and decreasing W_w may offset each other to some degree, since it is likely for the jobs that request a large resource (processors and/or memory) to incur a longer wait in the queue than the jobs with a smaller resource request.

The motivations for LXF&W-backfill are: to give even higher priority to shorter jobs than that in Priority-backfill and to treat jobs that have the same job expansion factor equally, an idea similar to that in uniprocessor processor sharing as discussed in Section 1.1. Further motivations of these policies are to have simpler priority function than in the above priority backfill policies, in order to more easily tune the performance and to avoid user gaming for getting better service. Under LXF&W-backfill, the relative priority of the waiting jobs with shorter requested runtimes increases faster as they waits in the queue, compared to that in Priority-backfill, Priority(P&M)-backfill, and NCSA-MS*.

Except as noted, the default reservations used by all the other backfill policies evaluated are dynamic-job and one reservation. Fixed and multiple reservations are evaluated for SJF-backfill to avoid the starvation problem.

NCSA-MS* is configured with fixed-job and two reservations (increased to six reservations since May 2001). The current implementation of NCSA-MS* on the O2K uses non-sliding rather than sliding reservation. This error in the implementation was only discovered recently (June 2001) and will be corrected in a future release of the scheduler. However, we find that non-sliding reservation has minimal impact on the performance of NCSA-MS*, to be shown in Section 5.3.3.

5.2 Performance of Priority-backfill, FCFS-backfill, and NCSA-LSF*

Section 3.6 shows that NCSA-LSF* has poor performance and it is difficult to tune the parameters. This section compares Priority-backfill and FCFS-backfill against NCSA-LSF*, with a goal of providing NCSA useful information for determining whether backfill policies significantly improve NCSA-LSF* for the O2K workload. NCSA-LSF* and backfill policies have not been compared in previous work.

Since the LSF job classes do not have any meaning for the backfill policies, instead of showing the performance measures for each job class (as that for NCSA-LSF* in Section 3.6), it is more useful to provide performance measures versus job sizes (i.e., actual runtime, requested number of processors, and requested memory) for evaluating backfill policies and other policies that do not use LSF job class for scheduling.

Figure 49 plots the average waiting time versus actual job runtime for Priority-backfill, FCFS-backfill, and NCSA-LSF* for each of the six months studied.

The results show that Priority-backfill and FCFS-backfill have very similar average waiting time versus actual runtime for each month (except a slight difference in February 2000 in Figure 49(e)). This suggests that perhaps Priority-backfill does not give enough priority to short jobs, because of a small but significant weight for job waiting time and a non-zero weight

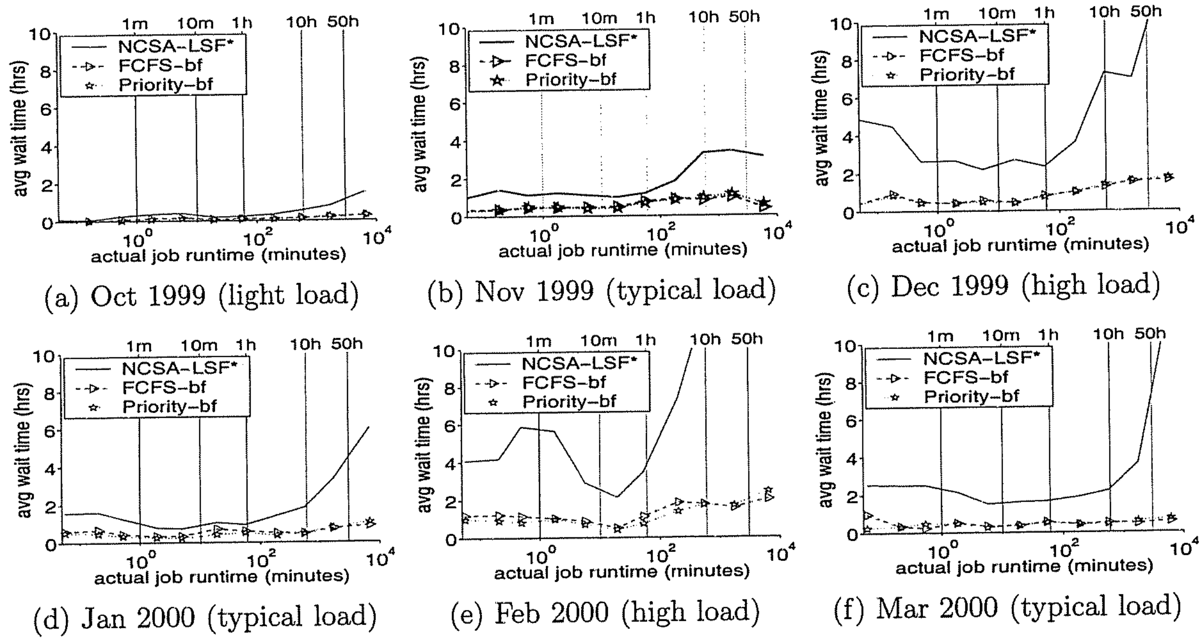


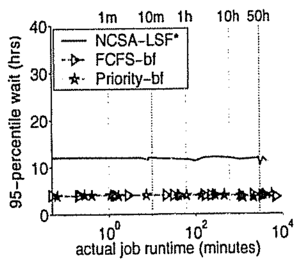
Figure 49. Variation in Monthly Average Wait of Priority-backfill, FCFS-backfill and NCSA-LSF*

for requested number of processors. However, later in Section 6.2.3, we will show this is mostly due to large discrepancy between requested and actual runtimes in the workload.

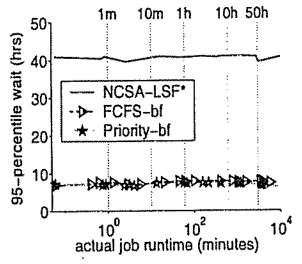
Figure 49 also show that the average waiting time under both Priority-backfill and FCFS-backfill is significantly lower than that for NCSA-LSF* over the entire range of actual runtime, for three months, including the two high-load months (i.e., December 1999 and February 2000 in Figure 49(c) and (e)) and March 2000 (Figure 49(f)). For the other three months, Priority-backfill and FCFS-backfill significantly improve the average waiting time of NCSA-LSF* for jobs that have long running time (> 10 hours).

Another observation is that unlike NCSA-LSF*, the average waiting time versus actual runtime under Priority-backfill and FCFS-backfill does not vary dramatically from month to month. Instead, there are small increases for the longest running jobs for the two high-load months (in December 1999 and February 2000 in Figure 49(c) and (e)).

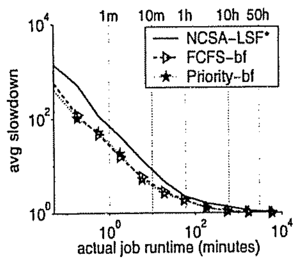
Figure 50 provides further comparison results for Priority-backfill, FCFS-backfill, and NCSA-LSF*. Figures 50(a)-(d) show that Priority-backfill and FCFS-backfill have significantly lower



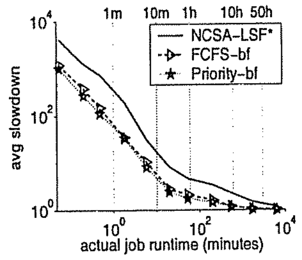
(a) 95th Percentile Wait Nov 1999 (typical load)



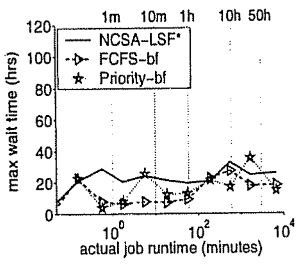
(b) 95th Percentile Wait Feb 2000 (high load)



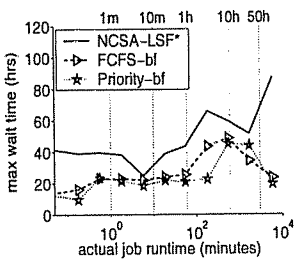
(c) Avg Slowdown Nov 1999 (typical load)



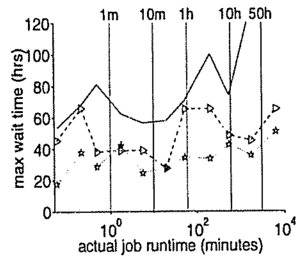
(d) Avg Slowdown Feb 2000 (high load)



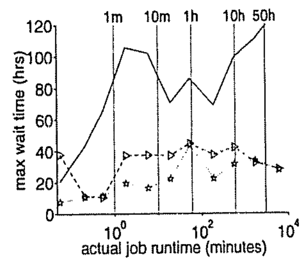
(e) Max Wait Oct 1999 (light load)



(f) Max Wait Nov 1999 (typical load)



(g) Max Wait Feb 2000 (high load)



(h) Max Wait March 2000 (typical load)

Figure 50. Further Comparisons of Priority-backfill, FCFS-backfill and NCSA-LSF*

95th percentile waiting time and average slowdown versus the actual runtime than that of NCSA-LSF* (shown for one typical-load month November 1999 and one high-load month February 2000 in (a)-(d)); the improvement is larger for the two high-load months.

Figures 50(e)-(h) plot the maximum waiting time versus actual runtime for each given month. Priority-backfill and FCFS-backfill have significantly lower maximum waiting time than that of NCSA-LSF* each month, except for the light-load month October 1999 (Figure 50(e)).

The only difference between Priority-backfill and FCFS-backfill is in their maximum wait time. As shown in Figures 50(e)-(h), Priority-backfill has slightly to considerably lower maximum wait for most of actual runtime for four months (shown in graphs (g)-(h) for February and March 2000; the difference in December 1999 and January 2000 is smaller than that in (h)). On the other hand, FCFS-backfill has somewhat lower maximum wait than that of Priority-backfill for the only light-load month, i.e., October 1999 (Figure 50 (e)). For November 1999, they have fairly similar maximum wait time (Figure 50 (f)).

Based on the predicted performance improvement of Priority-backfill over NCSA-LSF*, a priority backfill policy, i.e., NCSA-MS*, that uses a fairly similar relative priority measure (defined in Table 12) as that in Priority-backfill has been used on the O2K since July 2000. They have very similar performance except for the two high-load months, discussed in the next section.

5.3 Priority-backfill, Priority(P&M)-backfill, and NCSA-MS*

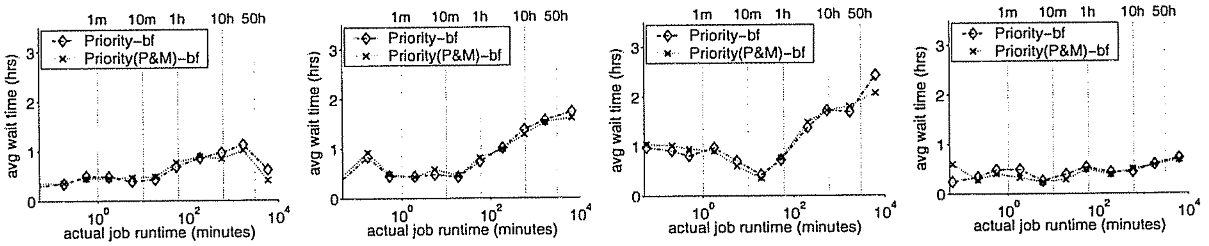
First, Section 5.3.1, compares Priority-backfill and Priority(P&M)-backfill that differ only in the requested resource measure they use; the former uses the requested processors, while the latter uses the larger value of requested processors and normalized requested memory. Section 5.3.2 compares Priority-backfill with NCSA-MS* to show that these policies have similar performance. Section 5.3.3 studies whether the performance of NCSA-MS* is affected because of the error of disallowing the scheduled start times to be rescheduled sooner when a job departs earlier than expected.

5.3.1 Impact of Processor and Memory Weights for Priority-backfill

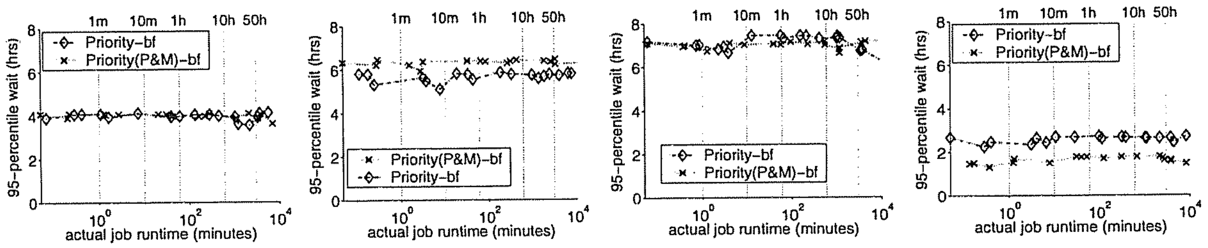
This section compares Priority-backfill and Priority(P&M)-backfill to study whether a weight to the requested memory as in Priority(P&M)-backfill can improve the performance of Priority-backfill.

Priority-backfill and Priority(P&M)-backfill have almost identical average waiting time over the entire range of actual runtime for each month (shown for four months in Figures 51(a)-(d). The remaining two months are shown in Figures 91(a)-(b) in Appendix F).

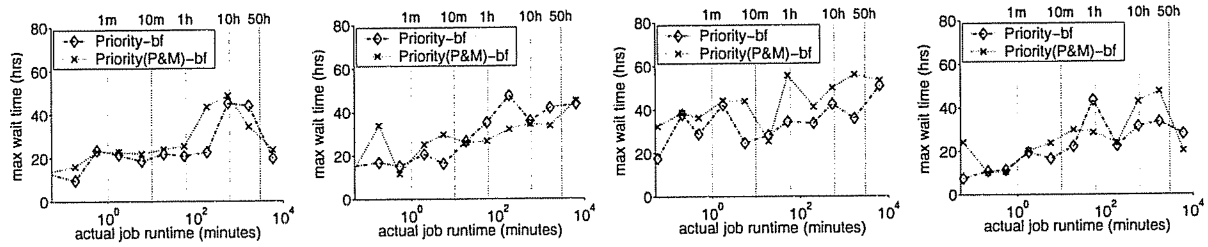
Priority-backfill has lower 95th percentile wait time in one high cpu load month (i.e., December 1999 shown in Figure 51(f)), while Priority(P&M)-backfill has lower 95th percentile wait time in a typical cpu load month, March 2000 (Figure 51(h)), in which the memory load



(a) Avg Wait (Nov99) (b) Avg Wait (Dec99) (c) Avg Wait (Feb00) (d) Avg Wait (Mar00)



(e) 95-percentile Wait (Nov 1999) (typical load) (f) 95-percentile Wait (Dec 1999) (high cpu-load) (g) 95-percentile Wait (Feb 2000) (high cpu-load) (h) 95-percentile Wait (March 2000) (high memory-load)

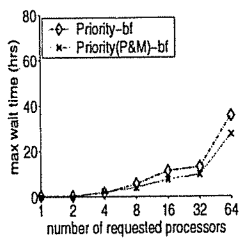


(i) Max Wait (Nov99) (j) Max Wait (Dec99) (k) Max Wait (Feb00) (l) Max Wait (Mar00)

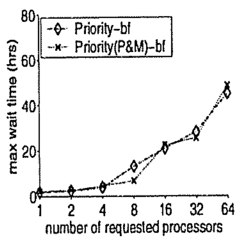
Figure 51. Performance of Priority-backfill and Priority(P&M)-backfill (Versus Actual Job Runtime)

is slightly higher than that in the other five months. In the other four months, they have almost identical 95th percentile wait time versus actual runtime (shown for November 1999 in Figure 51(e) and February 2000 in Figure 51(h). The results for the remaining two months, i.e., October 1999 and January 2000, are shown in Figures 91(c)-(d) in Appendix F).

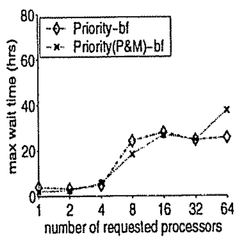
Figure 91(i)-(j) and (l) show that Priority-backfill and Priority(P&M)-backfill have more or less similar maximum wait for the three given months, including the heavy-load month in December 1999 (Figure 91(j)). They also have similar maximum wait for October 1999 and January 2000, shown in Figures 91(e)-(f) in Appendix F. Figure 91(k) shows that for the high



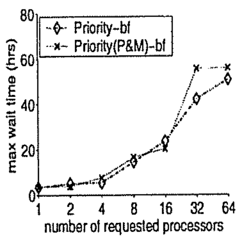
(a) #Processors Oct 1999



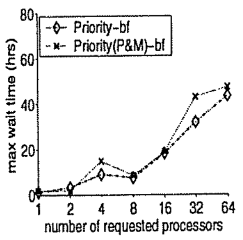
(b) #Processors Nov 1999



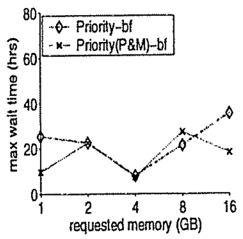
(c) #Processors Jan 2000



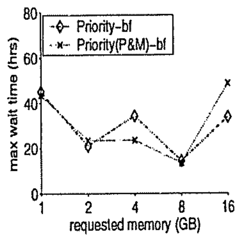
(d) #Processors Feb 2000



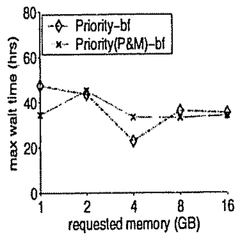
(e) #Processors March 2000



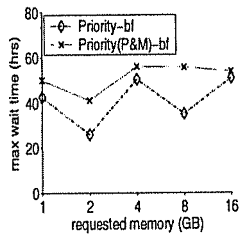
(f) Requested Memory Oct 1999



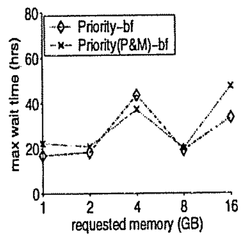
(g) Requested Memory Nov 1999



(h) Requested Memory Dec 1999



(i) Requested Memory Feb 2000



(j) Requested Memory March 2000

Figure 52. Comparisons of Priority-backfill and Priority(P&M)-backfill (Maximum Wait Versus Requested Processors and Memory)

cpu-load month (February 2000), Priority-backfill has lower maximum wait time over certain ranges of actual runtime.

Since the difference in Priority-backfill and Priority(P&M)-backfill is in the weight applied for the job resource requests, Figure 52 further studies how the performance of these two policies is different versus the requested number of processors and requested memory.

For the light-load month (i.e., October 1999), Figures 52(a) shows that Priority-backfill has slightly worse maximum wait for jobs that request > 8 processors, than that under Priority(P&M)-backfill. For November 1999, Figures 52(b) show that they have similar maximum waiting time verus requested number of processors. For December 1999, they have nearly identical maximum wait time verus requested number of processors (shown in Figure 91(g) in Appendix F).

Figures 52(c)-(e) show that Priority-backfill has slightly lower maximum waiting time for jobs that request over 32 processors in January 2000, 17-32 processors for February 2000, and > 16 processors for March 2000, respectively.

A key observation from Figure 52(a)-(e), is that the maximum waiting time approximately increases with the number of processors under both Priority-backfill and Priority(P&M)-backfill.

Figure 52(f)-(j) plots the maximum waiting time versus requested memory for each of the given five months. In contrast to the results in Figure 52(a)-(e), the maximum waiting time is rather insensitive to the requested memory. The key conclusion is that applying weight for the requested memory in addition to the number of requested processors as in Priority(P&M)-backfill does not improve Priority-backfill for the O2K workload, because processor resources are more constrained than memory resource.

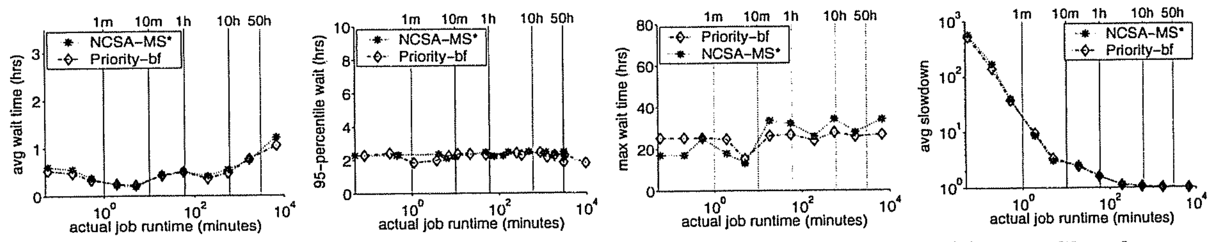
5.3.2 Priority-backfill Versus NCSA-MS*

Note that NCSA-MS* evaluated in this section assumes a correct implementation with reservation, i.e., the reservations are rescheduled to occur sooner if possible when a job departs earlier than estimated. Recall that Priority-backfill uses dynamic-job and single reservation, while NCSA-MS* uses fixed-job and two reservations, as described in Section 5.1.

Figure 53 shows that Priority-backfill and NCSA-MS* have very similar average, 95-percentile, and maximum waiting time, and average slowdown over the entire range of actual runtime, in the typical-load month (shown for January 2000, similar for the light-load and the other two typical-load months, i.e., October, November 1999 and March 2000).

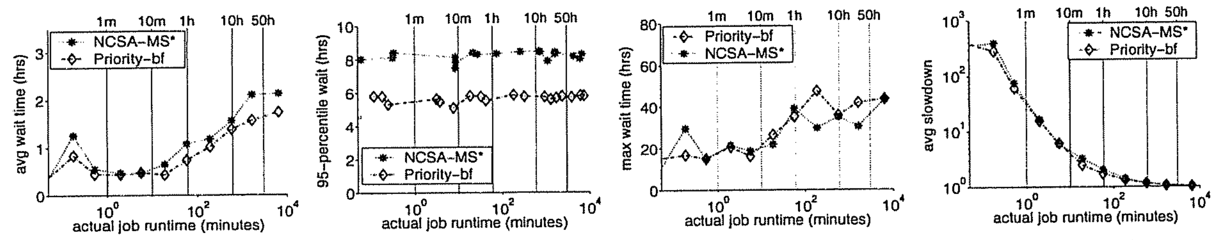
For the two high-load months, December 1999 and February 2000, Figures 54-55 show that Priority-backfill has a lower 95th percentile waiting time and slightly lower average waiting time, but slightly worse maximum waiting time.

Thus, Priority-backfill and NCSA-MS* have the same performance, except for the two high-load months. The difference in the high-load months is due to the slight difference in the priority measures as well as the difference in the reservations used (i.e., dynamic-job versus fixed-job and single versus multiple reservations). As discussed in Section 3.2.1, a fixed-job reservation may reduce the maximum wait than that of dynamic-job reservation, and multiple



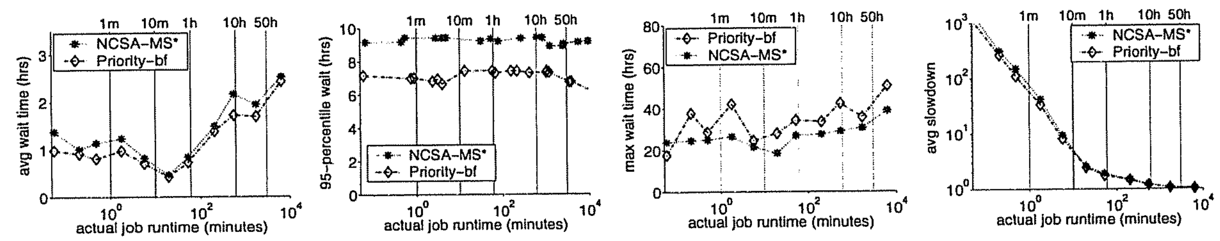
(a) Avg Wait (b) 95th Percentile Wait (c) Max Wait (f) Avg Slowdown

Figure 53. Typical-Load Performance of Priority-backfill versus NCSA-MS* (January 2000)



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

Figure 54. High-Load Performance of Priority-backfill versus NCSA-MS* (December 1999)



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

Figure 55. High-Load Performance of Priority-backfill versus NCSA-MS* (February 2000)

reservations may further reduce the maximum wait. However, the penalty with fixed-job and multiple reservations is higher 95th percentile waiting time, as shown for the comparison results for Priority-backfill and NCSA-MS* in the two high-load months in Figures 54-55.

5.3.3 Sliding Versus Non-sliding Reservations for NCSA-MS*

As discussed in Section 5.1, it was found recently that NCSA-MS* has an implementation error that does not allow the reservations to slide. This section studies how much the non-sliding

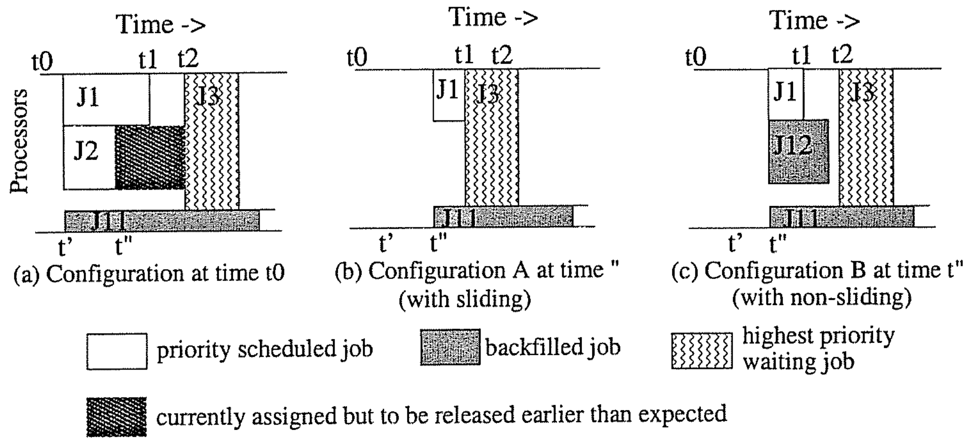

Time ->
t0
t1
t2
Processors
J1
J2
J3
J11
J12
t'
t"
(a) Configuration at time t0
(b) Configuration A at time "
(with sliding)
(c) Configuration B at time t"
(with non-sliding)
priority scheduled job
backfilled job
highest priority waiting job
currently assigned but to be released earlier than expected

Figure 56. Example Non-Sliding Reservation Upon a Job Departure

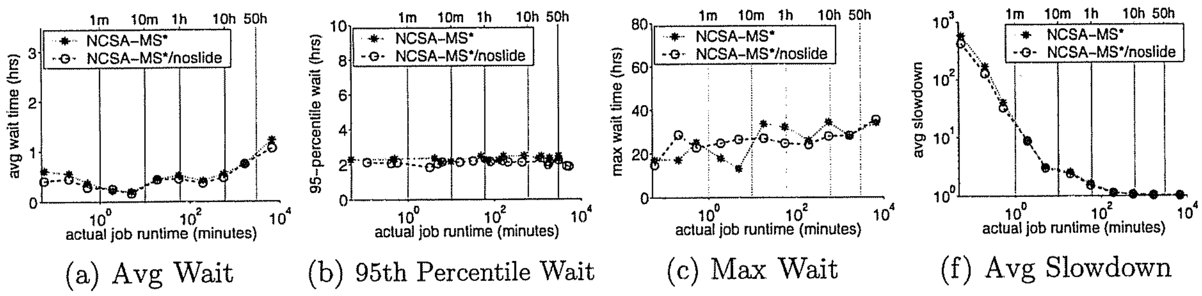
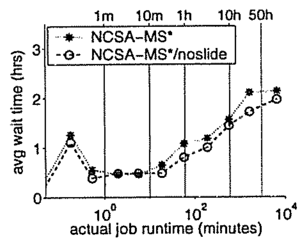


Figure 57. Impact of Non-Sliding Reservations for NCSA-MS* in Typical-Load Months
(January 2000)

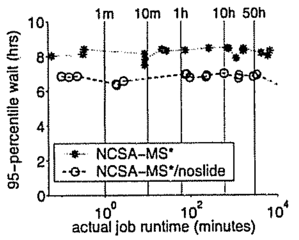
reservation impacts the performance of the NCSA-MS* scheduling algorithm.

An example of non-sliding reservation is illustrated in Figure 56. In Figure 56(a), job J2 is scheduled at time t' and is expected to depart at time t2, but it will actually depart at t" (earlier than expected). With a sliding reservation as shown in Figure 56(b), when job J2 departs, the scheduled start time for J3 is rescheduled to occur sooner at t1. On the other hand, with a non-sliding reservation as shown in Figure 56(c), at time t" when J2 departs, the existing scheduled start time for J3 does not slide, i.e., J3 is still expected to start at time t2, while job J12 is backfilled because it fits in the gap opened up by the departure J2 and is expected to complete before t2.

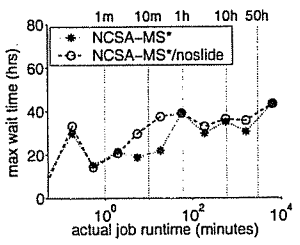
For each of the four typical-load months (shown for January 2000 in Figure 57), NCSA-MS* with sliding and with non-sliding reservations have nearly identical average and 95th percentile



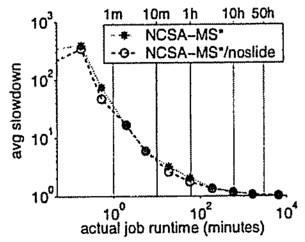
(a) Avg Wait vs. Actual Runtime



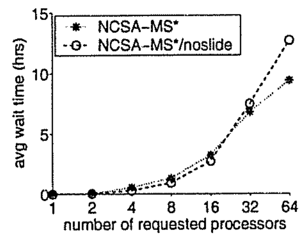
(b) 95th Percentile Wait vs. Actual Runtime



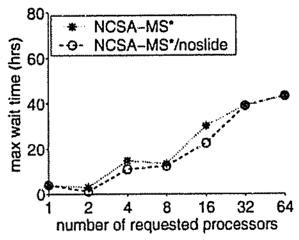
(c) Max Wait vs. Actual Runtime



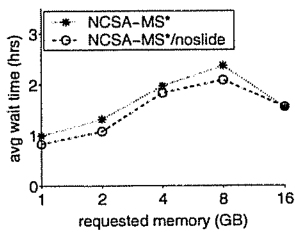
(d) Avg Slowdown vs. Actual Runtime



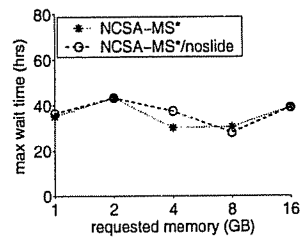
(e) Avg Wait vs. Requested Proc.



(f) Max Wait vs. Requested Proc.



(g) Avg Wait vs. Requested Mem.



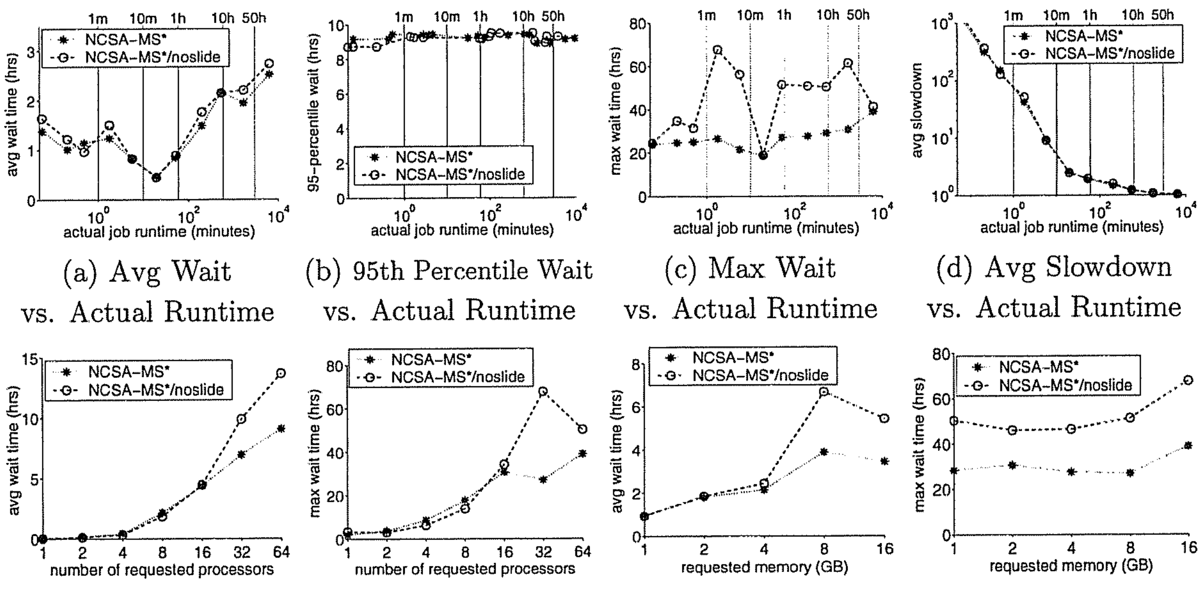
(h) Max Wait vs. Requested Mem.

Figure 58. Impact of Non-Sliding Reservations for NCSA-MS* in a High-Load Month (December 1999)

wait versus actual runtime, and a similar maximum wait versus actual runtime.

The results for the two high-load months (i.e., December 1999 and February 2000) are shown in Figures 58-59. In December 1999, NCSA-MS* with non-sliding reservation has lower 95th percentile waiting time (Figure 58(b)) but very similar performance for all other measures studied (Figures 58(a) and (c)-(h)), compared to NCSA-MS* with sliding reservation.

For the other high-load month, i.e., February 2000, Figures 59(a), (b) and (d) show that NCSA-MS* with non-sliding reservation and NCSA-MS* with sliding reservation have the same average and 95th percentile wait versus actual runtime, and average slowdown versus actual runtime, respectively. However, NCSA-MS* with non-sliding reservation is significantly worse with respect to the other performance measures, i.e., maximum waiting time for most ranges of actual runtime (in Figure 59(c)), average and maximum waiting time for requested processors > 16 (Figures 59(e)-(f)), average waiting time for requested memory > 4 GB (Figures 59(g)), and maximum waiting time versus the entire range of requested memory (Figures 59(h)).



(a) Avg Wait vs. Actual Runtime (b) 95th Percentile Wait vs. Actual Runtime (c) Max Wait vs. Actual Runtime (d) Avg Slowdown vs. Actual Runtime

(e) Avg Wait vs. Requested Proc. (f) Max Wait vs. Requested Proc. (g) Avg Wait vs. Requested Mem. (h) Max Wait vs. Requested Mem.

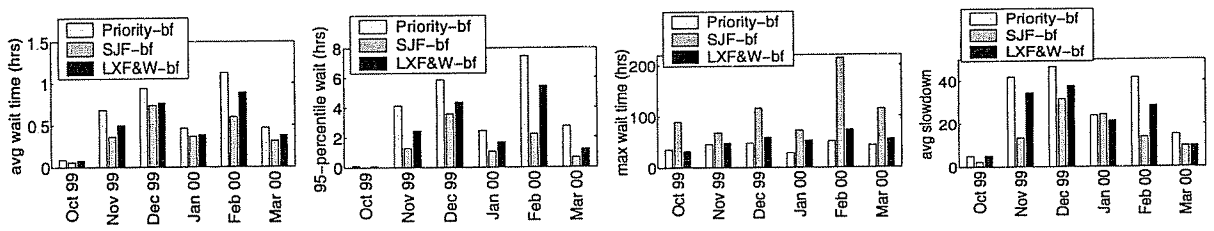
Figure 59. Impact of Non-Sliding Reservations for NCSA-MS* in a High-Load Month (February 2000)

The increase in the maximum wait with non-sliding reservation in February 2000 (Figure 59(c), (f), and (h)) and the decrease in the 95th percentile wait in December 1999 (Figure 58(c)) are due to the effect of providing more processing power for backfilling lower-priority jobs by disallowing the scheduled start time for jobs with reservation to occur sooner.

Non-sliding reservation is non-intuitive and lack of motivation (and in fact is an error in the NCSA-MS* implementation). Nevertheless, it has minimal impact on the performance for most months studied.

5.4 Performance of Alternative Priority Measures

Since Priority-backfill does not improve the waiting time of jobs compared to FCFS-backfill, we next evaluate SJF-backfill and LXF&W-backfill (defined in Section 5.1), both give a higher relative priority to short jobs than that in Priority-backfill. SJF-backfill has been shown to have higher performance than FCFS-backfill policies in previous work [ZK99, PK00]; however, it has



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

Figure 60. Overall Performance of Alternative Priority Measures

the potential starvation problem. LXF&W-backfill is a new priority backfill policy proposed in this thesis to favor shorter jobs without the starvation problem.

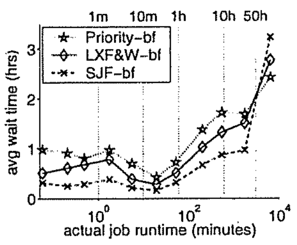
Section 5.4.1 compares the relative performance of Priority-backfill, LXF&W-backfill, and SJF-backfill, assuming the reservation is given to the job that has dynamically the highest priority. The impact of fixed-job and multiple reservations, which can be used as another mechanism to reduce the maximum waiting time as discussed in Section 3.2.1, are evaluated in Section 5.4.2. Based on the results, the best of each priority backfill is chosen and compared with each other again in Section 5.4.3.

5.4.1 SJF-backfill and LXF&W-backfill Versus Priority-backfill

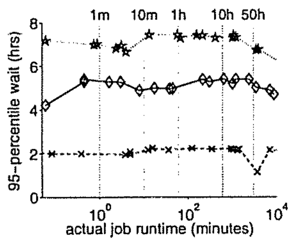
This section compares SJF-backfill and LXF&W-backfill against Priority-backfill, assuming the reservation is always given to the job that has dynamically the highest priority.

As shown in Figures 60(a)-(b), SJF-backfill has lower overall average wait and especially the 95th percentile waiting time than that of LXF&W-backfill and Priority-backfill. However, Figures 60(c) shows that SFJ-backfill has very poor maximum wait, especially for the high-load month in February 2000 (about 200 hours under SJF-backfill, compared to under 80 hours in both Priority-backfill and LXF&W-backfill). The results show that SFJ-backfill has the starvation problem.

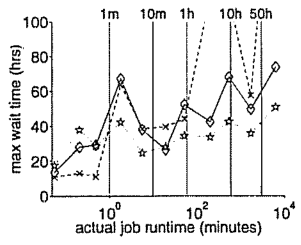
Figures 60(a)-(d) show that LXF&W-backfill has significantly lower overall average and 95th percentile wait time, average slowdown and a comparable maximum waiting time, compared to



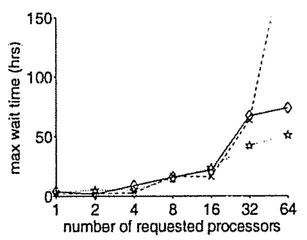
(a) Avg Wait vs. Actual Runtime



(b) 95th Percentile Wait vs. Actual Runtime



(c) Max Wait vs. Actual Runtime



(d) Max Wait vs. #Processors

Figure 61. Detailed Performance of Alternative Priority Measures (February 2000)

Priority-backfill.

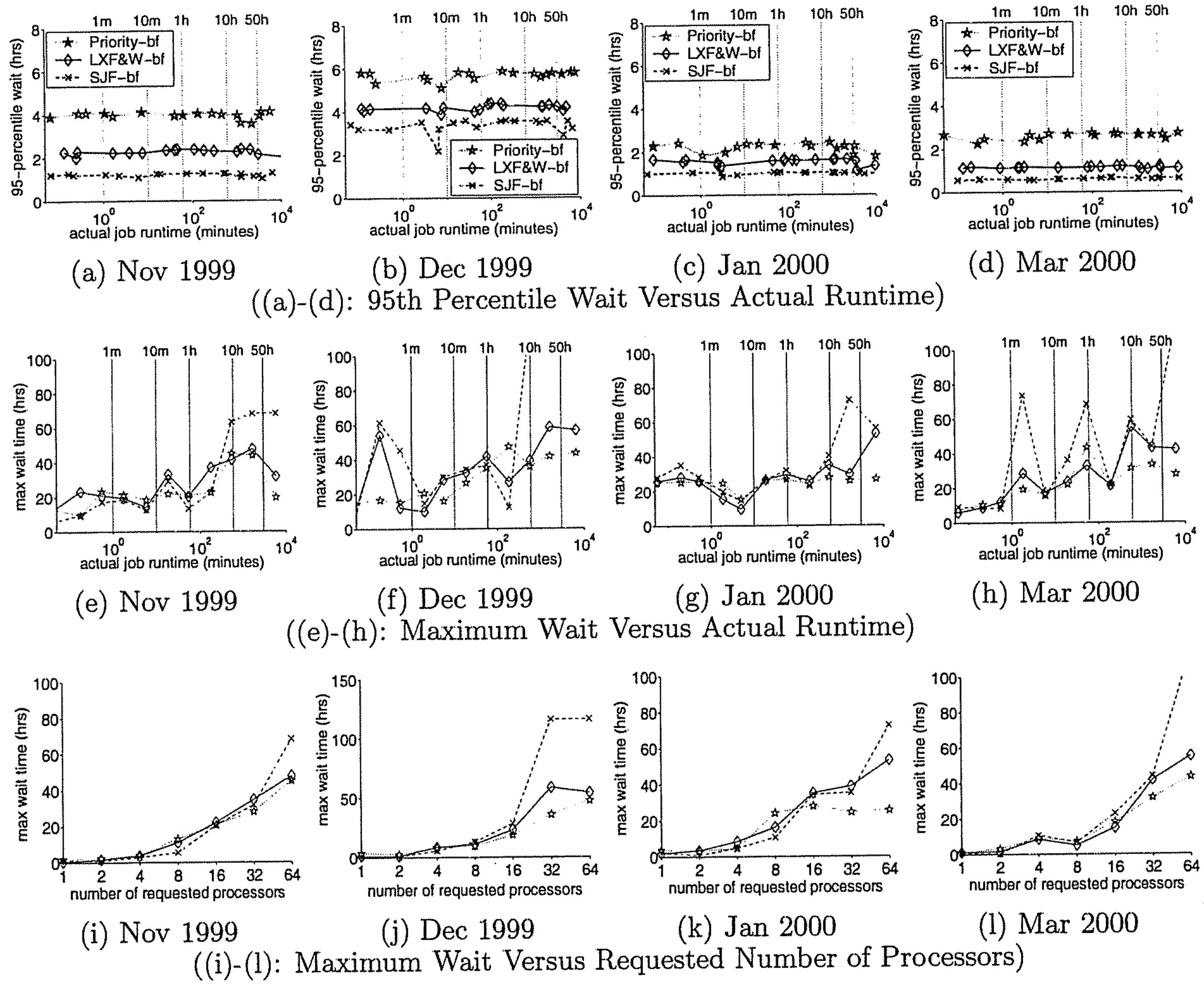
Figures 61(a)-(g) further show the detail performance of the three priority backfill for February 2000, in which month SJF-backfill has significantly lower average and 95th wait but very poor maximum wait, compared to LXF&W-backfill and Priority-backfill.

Figures 61(a)-(b) show that the lower average wait under SJF-backfill applies to the jobs for the entire range of actual runtime except for > 50 hours, and the lower 95th percentile wait under SJF-backfill applies to the entire range of actual runtime. Figure 61(c) shows that SJF-backfill has much worse maximum waiting time for most of the actual runtime ranges > 1 hour. Figures 61(d) show that the worst maximum wait time under SJF-backfill occurs to jobs that request > 32 processors. Thus, SJF-backfill has the problem of starving long and large-processor jobs.

Figures 62(a)-(d) plots that the 95th percentile wait time versus actual runtime for each of the four other months (except for the light-load month, October 1999). The difference in the 95th percentile wait time between LXF&W-backfill and SJF-backfill is significantly smaller than that in February 2000 Figures 61(b).

For each of these four months, Figures 62(e)-(h) plots the maximum wait versus actual runtime, and Figures 62(i)-(l) plots the maximum wait versus requested number of processors.

The key conclusion from these results Figures 60-62(a)-(l)) is that LXF&W-backfill has significantly lower 95th percentile over the entire range of actual runtime for each month (except



(a) Nov 1999 (b) Dec 1999 (c) Jan 2000 (d) Mar 2000
((a)-(d): 95th Percentile Wait Versus Actual Runtime)

(e) Nov 1999 (f) Dec 1999 (g) Jan 2000 (h) Mar 2000
((e)-(h): Maximum Wait Versus Actual Runtime)

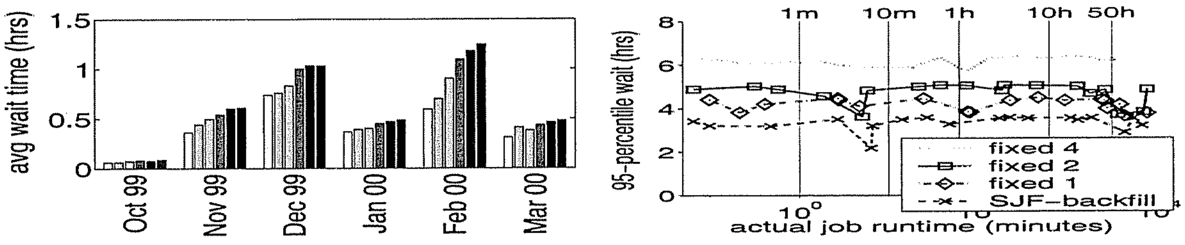
(i) Nov 1999 (j) Dec 1999 (k) Jan 2000 (l) Mar 2000
((i)-(l): Maximum Wait Versus Requested Number of Processors)

Figure 62. Detailed Performance of Alternative Priority Measures (For Other Months)

it's similar for the light-load month October 1999) and significantly lower overall average waiting time (Figure 60(a)), and comparable maximum wait time, compared to Priority-backfill, Thus, LXF&W-backfill is the preferred policy among the three priority backfill studied, for the O2K workload.

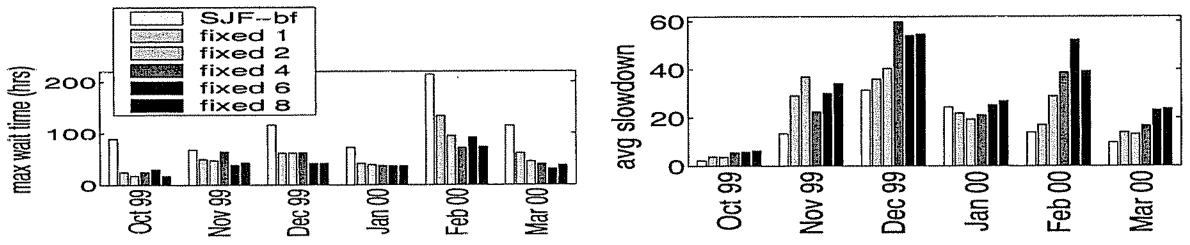
5.4.2 Improving SJF-backfill Using Different Reservation Rules

As shown in the previous section, SJF-backfill has the starvation problem. In this section, we study whether using fixed-job reservation and multiple reservations can improve the maximum



(a) Avg Wait

(b) 95th Percentile Wait vs. Actual Runtime (December 1999)



(c) Max Wait

(d) Avg Slowdown

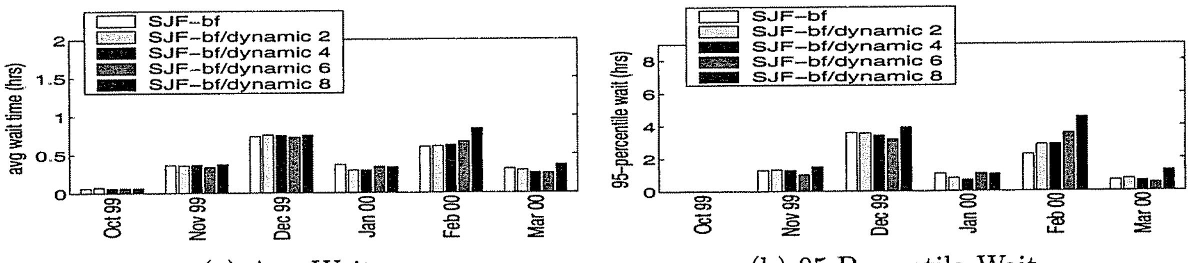
Figure 63. Impact of Fixed and Multiple Reservations for SJF-backfill

wait time in SJF-backfill, and whether the improved SJF-backfill can outperform LXF&W-backfill, the best backfill policy identified in the previous section.

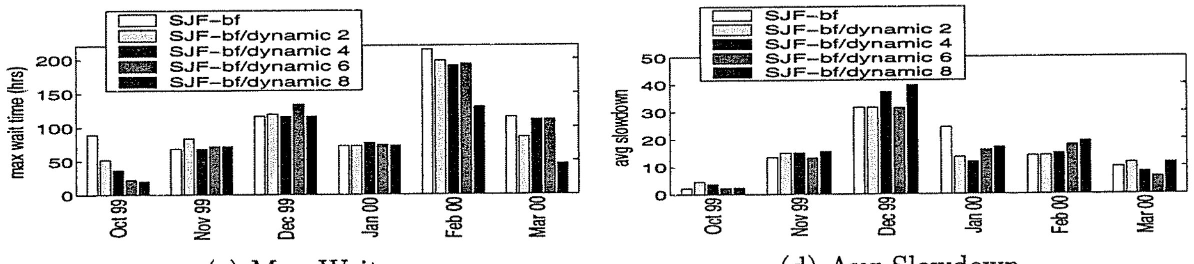
Figure 63 show the performance of SJF-backfill using fixed-job reservation with 1, 2, 4, 6, and 8 reservations.

As shown in Figure 63, using fixed-job and one reservation significantly reduces the maximum waiting time (by 20-70%) for each month (Figure 63(c)), but considerably increases other performance measures studied in most months. Using fixed-job and two reservations further significantly reduces the maximum waiting time for February 2000 (Figure 63(c)). Further increasing the number of reservations from two makes no improvement for the maximum waiting time, and in fact sometimes increases the maximum waiting time and also other performance measures. For example, in February 2000, fixed-job and six reservations has similar maximum wait but worse average and 95th wait, and worse average slowdown compared to that of fixed-job and two reservations.

In contrast to the effect of fixed-job and one or two reservations, Figure 64(c) show that with dynamic-job reservation, using multiple reservations has much smaller impact on the maximum



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



(c) Max Wait (d) Avg Slowdown

Figure 64. Impact of Number of Reservations for SJF-backfill with Dynamic-job Reservation

wait time of SJF-backfill.

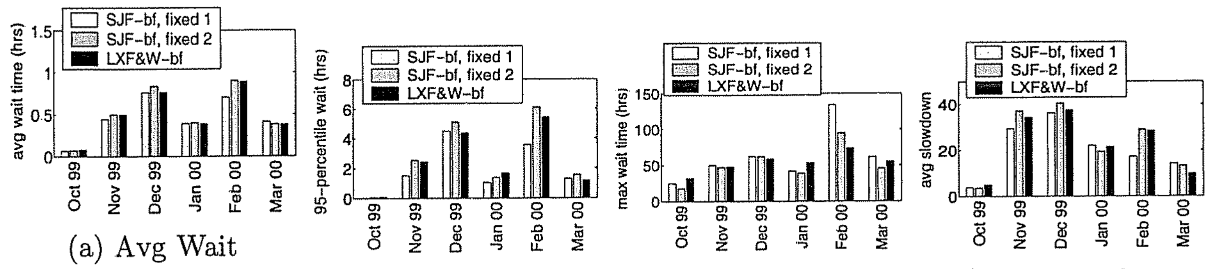
Thus, the key conclusion is that SJF-backfill can be improved by fixed-job and one or two reservations. In the next section, they will be compared with LXF&W-backfill and further discussed.

The impact of using fixed-job and using dynamic-job with multiple reservations on LXF&W-backfill and Priority-backfill is similar to that for using fixed-jobs and one or multiple reservations on SJF-backfill. The results for LXF&W-backfill are be provided in Figures 92-93 of Appendix F.

5.4.3 Re-evaluating Improved SJF-backfill with LXF&W-backfill

Figure 65 compares the improved SJF-backfill (using fixed-job with one and two reservations) against LXF&W-backfill. The Priority-backfill is not included, since it is worse than LXF&W-backfill.

Figure 65(c) shows SJF-backfill using fixed-job with one reservation still appears to have the potential starvation problem. Specifically, the maximum wait for SJF-backfill with fixed-job



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

Figure 65. Re-evaluation of Improved SJF-backfill Versus LXF&W-backfill

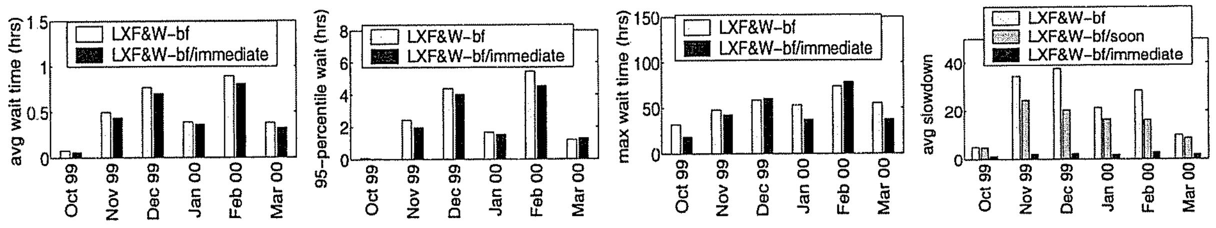
and one reservation is almost 50% (or almost 70 hours) worse than that of LXF&W-backfill for February 2000.

SJF-backfill using fixed-job with two reservations has more comparable maximum wait to that of LXF&W-backfill for February 2000 (Figure 65(c)). However, SJF-backfill using fixed-job with two reservations has worse overall average and 95th percentile wait time than that of LXF&W-backfill for most months, including February 2000. (as shown in Figure 65(a),(b) and (d)).

Thus, the estimated current job expansion factor with a small weight for the job wait time, used in LXF&W-backfill, appears to be the best priority function to use among the functions evaluated in this work.

5.5 LXF&W-backfill with 1-Minute Preemptive Immediate Service

As can be seen in the previous two sections, very short running jobs incur similar 95th (several hours) and maximum waiting time (tens of hours) as that for the other jobs under the priority backfill policies. To improve the turnaround for very short jobs, we propose a limited preemption option that gives a one-minute quantum of immediate service and up to 1 GB of initial memory to each new job that can't be started due to insufficient resources. The definition of limited preemption is given in Section 3.2.2. The values of one-minute cpu quantum and up to 1 GB of



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

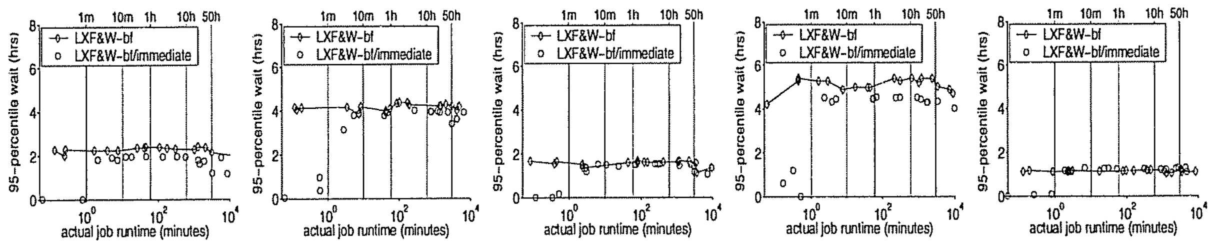
Figure 66. Overall Performance of LXF&W-backfill with 1-Minute Immediate Service

memory are motivated by the characteristics of the O2K workload, in that a significant fraction (15-20%) of the jobs terminate in under one minute, including jobs that request over a long runtime (> 50 hours) (Section 4.5.1), and nearly all of these jobs use much less than 1 GB of memory (Section 4.5.2).

Figure 66 evaluates the performance of LXF&W-backfill with limited preemption (i.e., LXF&W-bf/immediate). Also included in graph (d) is a non-preemptive version of test run option (i.e., LXF&W-bf/soon), which does not preempt any executing jobs for new jobs, but gives a 1-minute quantum and up to 1 GB of memory (as in the preemptive immediate service option) to each new job as soon as after its arrival if there are enough free processors and memory. The idea of the non-preemptive test run is proposed in [PK00] and shown to improve the average slowdown (by up to 50%) for FCFS-backfill and SJF-backfill. The non-preemptive test run evaluated in their paper is different from that evaluated in this section in that (1) the policies evaluated in [PK00] schedule the processors but not the memory (thus the initial memory is not a parameter); (2) they use a larger quantum, i.e., 15 minutes, which is heuristically chosen but not based on the workload characteristics.

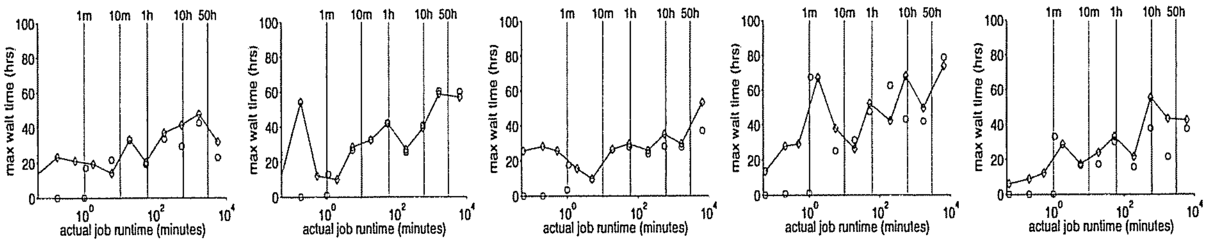
Figure 66(a)-(c) show that preemptive immediate service slightly improves the overall average and maximum wait time for most months, and slightly improves the 95th percentile wait time for three months (i.e., November and December 1999 and February 2000).

Figure 66(d) shows that preemptive immediate service dramatically reduces the average slowdown for each month. Furthermore, preemptive immediate service is much more effective



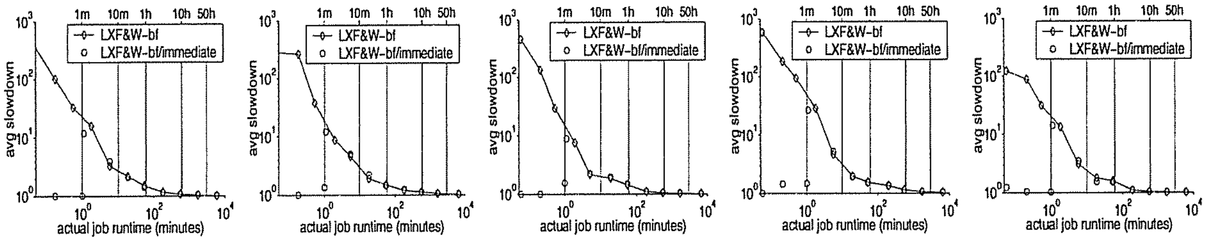
(a) Nov 1999 (b) Dec 1999 (c) Jan 2000 (d) Feb 2000 (e) Mar 2000

((a)-(e): 95th Percentile Wait Versus Actual Runtime)



(f) Nov 1999 (g) Dec 1999 (h) Jan 2000 (i) Feb 2000 (j) Mar 2000

((f)-(j): Maximum Wait Versus Actual Runtime)



(k) Nov 1999 (l) Dec 1999 (m) Jan 2000 (n) Feb 2000 (o) Mar 2000

((k)-(o): Average Slowdown Versus Actual Runtime)

Figure 67. Detailed Performance of LXF&W-backfill with 1-Minute Immediate Service

than non-preemptive option in reducing the average slowdown.

To examine how the performance of jobs that run for over 1 minute is impacted by the immediate service, Figure 67(a)-(e) plot the 95th percentile wait time versus actual runtime, Figure 67(f)-(j) plot the maximum wait time versus actual runtime, and Figure 67(k)-(o) plot the average slowdown versus actual runtime, for each month (except the light-load month, October 1999). The key conclusion from these results is that adding preemptive 1-minute immediate service dramatically improves the wait time and slowdown for jobs terminate under 1 minute, without adversely impacting the performance of the other jobs.

Note that about 1000-2000 jobs (at least 15% of all job arrivals) for each month (except the light load month October 1999) receive an immediate service; about 10-15% of these jobs actually complete in their initial quantum and memory. Among the jobs that receive immediate service, 100-200 jobs each month (except October 1999) require preempting the processors of the executing jobs; these jobs will not receive an immediate service under non-preemptive version.

5.6 Summary of Performance of Backfill Policies

- FCFS-backfill and Priority-backfill have almost identical performance, except that Priority-backfill has significantly lower maximum wait time in a few months.

- Furthermore, both Priority-backfill and FCFS-backfill have significantly lower average, 95th percentile, and maximum wait time and average slowdown over most ranges of actual runtime than that of highly-tuned NCSA-LSF*.

- Whether giving priority to the requested processors only (as in Priority-backfill) or both the requested processors and requested memory (as in Priority(P&M)-backfill and NCSA-MS*), the maximum waiting time approximately increases as the number of requested processors increases, but is relatively insensitive to the requested memory. This is because the processors are more constrained than the memory in the O2K workload.

- The current estimated expansion factor plus a relatively very small weight for the current job wait time as in LXF&W-backfill provides the starvation-free measure that gives priority to short jobs and significantly improves both Priority-backfill and FCFS-backfill. Specifically, LXF&W-backfill has at least 20% lower average and 95th percentile wait and at least 20% lower average slowdown, while comparable maximum wait time. Furthermore, the LXF&W priority measure is intuitive and based on fundamental scheduling policy results; and easy to be tune (only the weight for the job wait time); and discourages users gaming for better service.

- Adding preemptive 1-minute immediate service for backfill policies not only dramatically improves the performance for the jobs under 1 minute without impacting the performance of the other jobs for each month, and also slightly improves the 95th percentile wait time for all jobs in three out of six months studied.

- Preemptive 1-minute immediate service is much more effective than non-preemptive option in improving the turnaround for jobs under 1 minute.

Chapter 6

Further Policy Evaluation

This chapter examines further policies that require greater system support. One is the use of more accurate requested runtime for backfill policies. The other is EQspatial, which is compared against LXF&W-backfill with immediate service, the best priority backfill function studied in Chapter 5.

As shown in the previous section, adding a preemptive 1-minute immediate service significantly improves the performance of the jobs that complete in 1 minute. With support of dynamic processor repartitioning, the preemptive dynamic equal space partitioning policy (i.e., EQspatial) are expected to further improve jobs that are longer than 1 minute. Note that if most jobs have high processor efficiency on the number of processors allocated (and requested), such as that in the O2K workload (as shown in Figures 24 and 45), Gang Scheduling is expected to have similar performance as that of EQspatial for workloads. In addition, EQspatial can be modified to accommodate power-of-two numbers of processor allocation requirements (i.e., Folding, as discussed in Section 3.2.3). Thus, the performance of EQspatial is representative for Folding and Gang scheduling.

In the absence of preemption and/or support for dynamic processor repartitioning, improving the accuracy of the requested runtime may improve backfill policies, as backfill policies use requested runtimes to make scheduling decisions. The benefit for priority backfill policies that give priority to jobs with short requested runtime (such as LXF&W-backfill and SJF-backfill) may be larger than backfill policies that do not give priority to short jobs (such as FCFS-backfill).

In Section 6.1, we evaluate a dynamic space sharing policy, EQspatial-5m, designed to have a minimal processor repartitioning overhead (defined in Section 3.2.3), to examine the performance gain of equal space partitioning over LXF&W-backfill with preemptive 1-minute immediate service. Section 6.2 investigates the potential benefit of using improved runtime estimates for backfill policies. Section 6.3 summarized the results in this chapter.

6.1 Performance of Dynamic Equal Space Partitioning

This section evaluate the performance of EQspatial. The implication of the EQspatial for Gang scheduling is commented with data measured from EQspatial results.

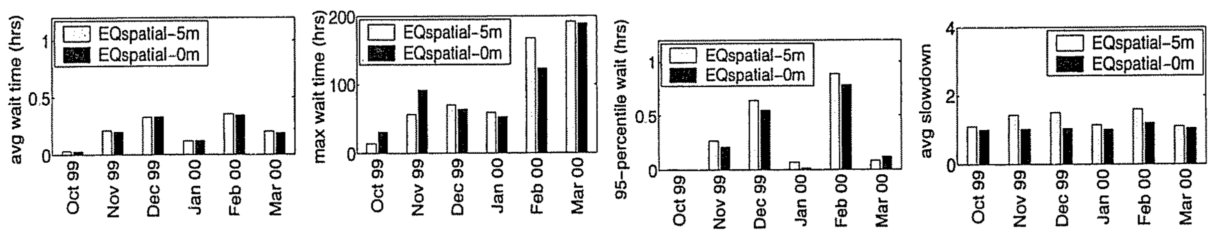
6.1.1 EQspatial-5m Versus LXF&W-backfill with 1-Minute Immediate Service

To reduce the processor repartitioning overhead, we simulate EQspatial with MinInterval equal to 5 minutes (i.e., EQspatial-5m). Figures 68(a)-(d) compare the overall performance measures for EQspatial-5m with the ideal EQspatial (i.e., MinInterval = 0 minute). Figures 68(e)-(g) further compare the performance versus actual runtime of these two policies for the high-load month (February 2000). These graphs show that EQspatial-5m has similar average and maximum wait time, and comparable 95th percentile wait and average slowdown, compared to EQspatial-0m,

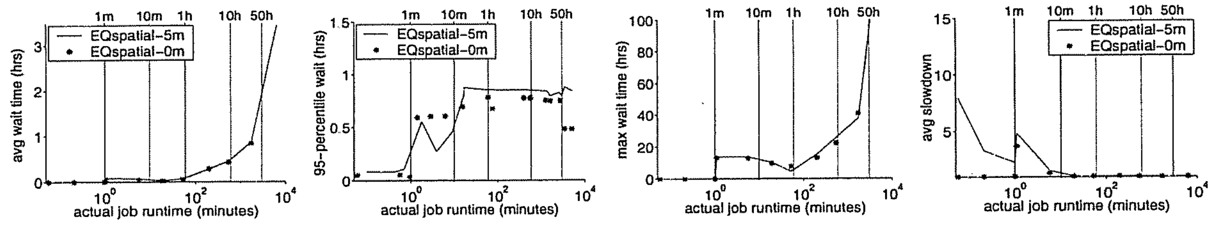
More than 99% of the arriving jobs in each workload are placed on a host for full execution immediately upon arrival, which indicates that the processor and memory usage on each host is reasonably well-balanced by the heuristic host placement policy defined in Section 3.2.3.

Figure 69 compares EQspatial-5m against the LXF&W-backfill with immediate service. Also included is EQspatial-5m/>50hr, which will be explained later.

Figures 69(a), (b) and (d) show that EQspatial-5m significantly improves the overall average wait time (by over 50%) and overall average slowdown (by over 30% except for the



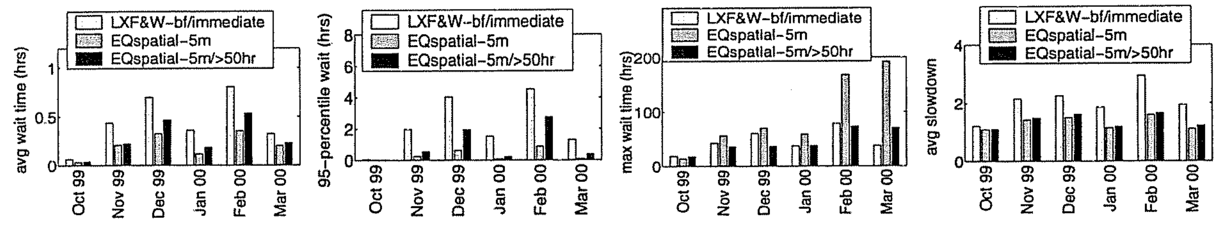
(a) Avg Wait (b) Max Wait (c) 95-th Percentile Wait (d) Avg Slowdown



(e) Avg Wait (Feb 2000) (f) 95-percentile Wait (Feb 2000) (g) Max Wait (Feb 2000) (h) Avg Slowdown (Feb 2000)

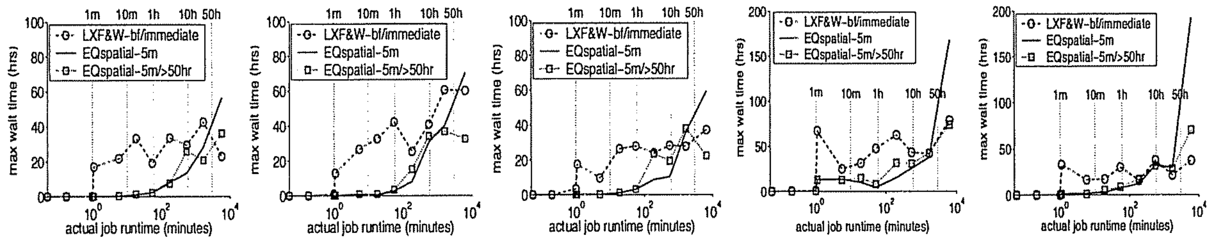
Figure 68. EQspatial-5m vs EQspatial-0m

light-load month, October 1999), and dramatically improve the 95th percentile wait in each month for LXF&W-backfill with 1-minute immediate service. However, Figure 69(c) shows that EQspatial-5m has very poor maximum waiting time in February and March 2000. Plots of maximum wait time versus actual runtime and number of requested processors for each month, in Figures 70-71 respectively, show that jobs that have long running time (> 50 hours) and a large number of processors (> 32) in February and March 2000 have very poor wait under



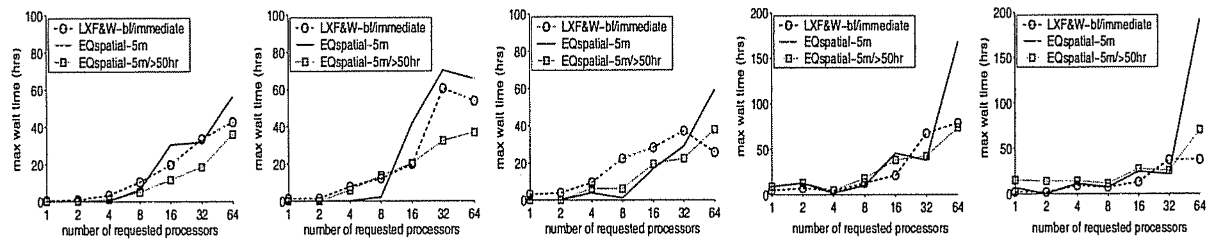
(a) Avge Wait (b) 95-Percentile Wait (c) Max Wait (d) Avg Slowdown

Figure 69. Overall Performance of EQspatial Versus LXF&W-backfill with Immediate Service



(a) Nov. 1999 (b) December 1999 (c) January 2000 (d) February 2000 (e) March 2000

Figure 70. EQspatial Versus LXF&W-backfill with Immediate Service: Maximum Wait vs. Actual Runtime

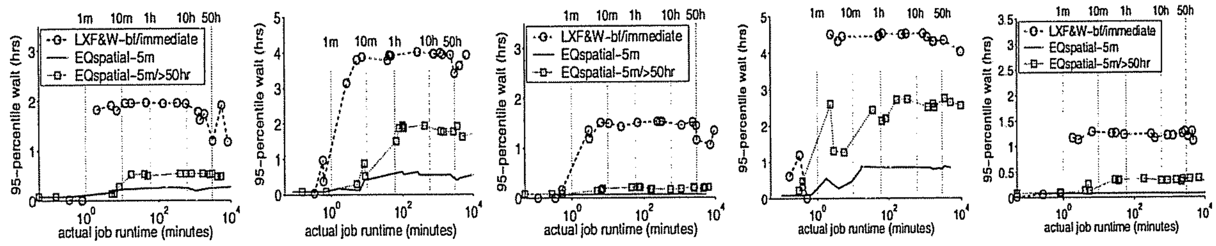


(a) Nov. 1999 (b) December 1999 (c) January 2000 (d) February 2000 (e) March 2000

Figure 71. EQspatial Versus LXF&W-backfill with Immediate Service: Maximum Wait vs. Number of Requested Processors

EQspatial-5m. Thus, it may require to improve the maximum wait time of the jobs that have a long runtime and a large number of processors under EQspatial-5m.

EQspatial-5m/>50hr is designed to improve the worst wait under EQspatial. Under EQspatial-5m/>50hr, 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 to run on. As shown in Figure 69(c) and Figures 70-71, EQspatial-5m/>50hr effectively improve the maximum wait of the largest jobs under EQspatial-5m, and Figures 69(a) and (d) show that EQspatial-5m/>50hr has similar overall average wait and slowdown as that of EQspatial-5m. Although EQspatial-5m has significantly lower 95th percentile wait for the two heavy-load months (Dec 1999 and Feb 2000) as shown in Figure 69(b), EQspatial-5m/>50hour may be preferred because of a significantly lower maximum wait for the largest jobs.



(a) Nov. 1999 (b) December 1999 (c) January 2000 (d) February 2000 (e) March 2000

Figure 72. EQspatial Versus LXF&W-backfill with Immediate Service: 95th Percentile Wait vs. Actual Runtime

Figures 72(a)-(e) further show the 95th percentile wait under EQspatial-5m/>50hr, EQspatial-5m, and LXF&W-backfill with immediate service for each month (except the light-load month). The graphs shows both EQspatial-5m/>50hr and EQspatial-5m significantly improve the 95th percentile wait for job > 1 minute under LXF&W-backfill for each month. Nevertheless, considering the effort required in implementing EQspatial-5m, LXF&W-backfill with immediate service is perhaps surprisingly competitive with respect to the overall average wait (Figures 69(a)) and overall average slowdown (Figures 69(d)).

6.1.2 Implication of EQS Results for EQT

To schedule the O2K workload, both EQspatial and Gang have the same consideration for balancing the load across the hosts and providing an equal allocation to each job. If a Gang Scheduler is used, we can use a similar host placement policy as in EQspatial to choose the host to schedule each job for balancing the load across hosts. Since the O2K workload has high (80%) utilization on the processors allocated (Figure 45 of Section 4.5.2), jobs are expected to have similar execution efficiency on the processors allocated under Gang and EQspatial. Thus, the main difference between EQspatial and Gang for the O2K workload occurs during the period of time when the total requested number of processors on a host is larger than the total number of processors available on the host. To examine how often this occurs, Table 14 shows for each month (except for the light-load month), the fraction of the time the total requested processors of all scheduled jobs on each of the eight O2K hosts is in the range of $\leq$ p, p+1 to 2p, and

Table 14. Total Number of Requested Processors on Each Host Under EQspatial

Month	Host	Total Processors Available	Total Requested Processors		
			$\leq$ P	P+1 - 2P	2P+1 - 3P
Nov 1999	eir	(128)	57	42	0
	hod1	(128)	60	39	0
	huldra	(128)	56	43	0
	jord1	(128)	62	37	0
	mimir	(128)	62	37	0
	modi2	(64)	100	0	0
	nerthus	(128)	50	49	0
	saga1	(128)	57	42	0
Dec 1999	eir	(128)	39	60	0
	hod1	(128)	41	58	0
	huldra	(128)	68	31	0
	jord1	(128)	55	44	0
	mimir	(128)	51	48	0
	modi2	(64)	100	0	0
	nerthus	(128)	38	61	0
	saga1	(128)	46	53	0
Jan 2000	eir	(128)	60	39	0
	hod1	(128)	48	51	0
	huldra	(128)	76	23	0
	jord1	(128)	68	31	0
	mimir	(128)	68	31	0
	modi2	(64)	100	0	0
	nerthus	(128)	61	38	0
	saga1	(128)	71	28	0
Feb 2000	eir	(128)	38	61	0
	hod1	(128)	34	64	1
	huldra	(128)	48	51	0
	jord1	(128)	45	54	0
	mimir	(128)	51	48	0
	modi2	(64)	99	0	0
	nerthus	(128)	47	52	0
	saga1	(128)	45	54	0
Mar 2000	eir	(128)	42	56	1
	hod1	(128)	54	42	2
	huldra	(128)	69	30	0
	jord1	(128)	79	20	0
	mimir	(128)	71	28	0
	modi2	(64)	100	0	0
	nerthus	(128)	51	48	0
	saga1	(128)	74	25	0

2p+1 to 3p, where p is the total number of processors available on the host. Note the total number of requested processors on each host is never greater than 3p.

As shown in Table 14, a significant fraction of the time (40-50% for the two high-load months and 50-70% for other months, except it is 100% the smallest host, modi2), the total number of requested processors of all scheduled jobs on each host is no more than the total number of the processors available, which means each job gets as many processors as it requests and thus the EQspatial policy equals to the Gang Scheduling policy in this case. In nearly all the remaining time, the total number of requested processors of all scheduled jobs on each host is no more than twice the total number of processors available, i.e., two time slies are sufficient to share the host among the jobs. Since there are almost never more than two time slices on each host, the performance of Gang is expected to be comparable to EQspatial and has the potential to significantly improve LXF&W-backfill for the O2K workload.

6.2 Potential Benefit of More Accurate Requested Runtime for Backfill Policies

Since backfill policies use requested runtimes for scheduling, a natural question to ask is: whether more accurate requested runtimes can improve the performance of backfill policies. The results in a previous study for FCFS-backfill suggest that the improvement is very limited [FW98]. The question is whether there is a significantly larger benefit of using more accurate requested runtimes for priority backfill policies that give priority to short jobs (such as in Priority-backfill, SJF-backfill, and LXF&W-backfill) than that for FCFS-backfill.

To evaluate the potential benefit of more accurate requested runtimes for backfill policies, in addition to the ideal case where the requested runtime equal to actual runtimes, two scenarios of more accurate requested runtimes are evaluated for backfill policies. Section 6.2.1 defines the two scenarios for more accurate requested runtimes. Section 6.2.2 presents the results for FCFS-backfill. The results are compared against the previous results. Section 6.2.3 presents the results for the priority backfill policies studied, i.e., Priority-backfill, SJF-backfill, and LXF&W-backfill. Section 6.2.5 presents the results of using more accurate requested runtimes for LXF&W-backfill with preemptive immediate service. A summary is provided in Section 6.3.

6.2.1 Two Possible Scenarios For More Accurate Requested Runtimes

Table 15 defines two scenarios (i.e., A and B) of more accurate but imperfect requested runtimes.

Table 15. Two Scenarios for More Accurate Requested Runtime

Scenario	Figure Label	Computed Requested Runtime	
A	T+k%	$\min\{(1 + k\%) \times \text{T}, \text{R}\}$,	for any T
B	T+k% | >10m,10%R	$\min\{(1 + k\%) \times \text{T}, \text{R}\}$,	if T > 10m or > 10% × R
		R,	otherwise

R: O2K requested runtime; T: actual runtime

In Scenario A, all jobs have improved requested runtimes. Specifically, for each job that requests a runtime that is greater than 1 + k% of the actual runtime of the job, the improved requested runtime for this job is computed to be 1 + k% of the actual runtime of the job. For example, suppose k = 20, if a job requests 50 hours, but actually terminates in 10 hours, the improved requested runtime for this job in Scenario A is computed as 12 hours (i.e., 1 + 20% of 10 hours). For all other jobs (i.e., which have the requested runtime ≤ 1 + k% of the actual runtime), the O2K (i.e., user-supplied) requested runtimes are used. Three values of k are simulated. They are 20, 50, and 100. A small value of k (such as 20) is more interesting for this analysis, since we are investigating the benefit of *more accurate* runtime for backfill policies. The other two larger values (50 and 100) are used to examine whether the value of k has a significant impact on the performance. Note that Scenario A is similar to the model of requested runtimes used in previous two papers [FW98, ZFMS00] (except that they model the value of k by the uniform distribution rather than deterministic). The mean value of k used is as large as 30,000 in [FW98] and from 50 up to 1000 in [ZFMS00]. These two papers use such large values for k, because their goal is to study whether using overestimated runtimes for

FCFS-backfill results in worse performance, compared to that if the actual runtimes are used; and whether the performance degrades more as k increases.

In Scenario B, all jobs, except for 'short' jobs, have improved requested runtimes. The motivation for this scenario is that these short jobs possibly prematurely terminate due to unexpected errors in the codes or in the configurations for the execution, thus it's difficult to improve their requested runtimes. In our simulation, jobs that run for under 10 minutes and use under 10% of the requested runtime are considered to prematurely terminate; they do not have improved requested runtimes. All other jobs have improved requested runtimes computed to be the smaller value of 120% of actual runtime and the O2K requested runtime (as that in Scenario A with $k = 20$). Thus, Scenario B is more pessimistic but perhaps more realistic than Scenario A. Note the threshold of runtime used to classify the jobs in Scenario B is only a heuristic used to give some idea of how the performance is affected if some short jobs do not have improved requested runtimes.

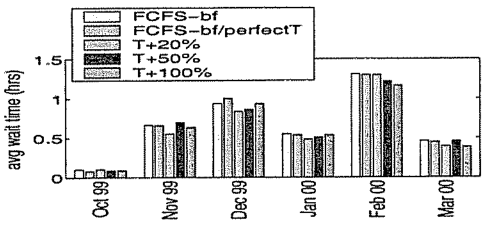
Note that about 25-30% of the jobs each month terminate in under 10 minutes and the majority of these jobs have a requested runtime greater than the actual runtime by over 20%. Scenario B use the O2K requested runtimes for these short jobs.

6.2.2 Impact of More Accurate Requested Runtime for FCFS-backfill

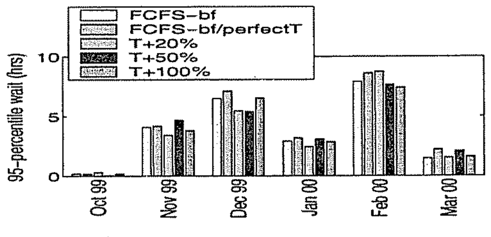
This section evaluates the potential benefit of using more accurate requested runtimes for FCFS-backfill, including using actual runtimes and Scenario A requested runtimes.

Figures 73(a) and (b) show that using perfect runtime information for FCFS-backfill does not improve the overall average nor the 95th percentile wait time. In fact, using actual runtimes increases the 95th percentile waiting time (although only slightly) for several months (December 1999 and February and March 2000, shown in Figure 73(b)).

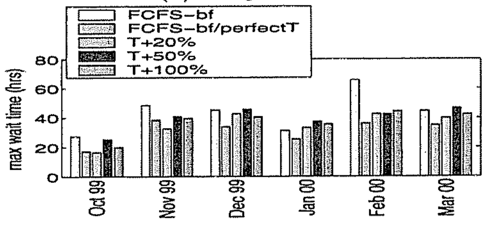
On the other hand, Figures 73(c) and (d) show that using actual runtimes for FCFS-backfill improves the maximum wait time by 20-30% each month, and improves the overall average



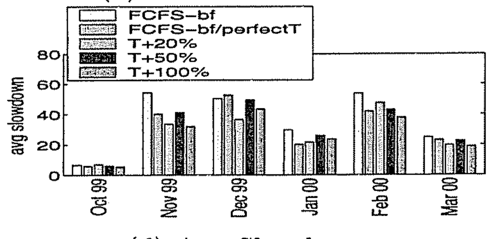
(a) Avg Wait



(b) 95th Percentile Wait



(c) Max Wait



(d) Avg Slowdown

Figure 73. Overall Performance of Using More Accurate Requested Runtime for FCFS-backfill

slowdown by up to 25%, which is only as much as that reported in previous results [FW98].

Figures 73(a)-(d) also show that using Scenario A requested runtimes for FCFS-backfill (whether the requested runtimes are up to 20%, 50%, or 100% longer than the actual runtimes) result in similar performance to that of using actual runtimes, except that Scenario A with T + 20% has slightly better average and 95th percentile wait time in four months (Figures 73(a) and (b)) and better average slowdown for two months (Figure 73(d)). This is in agreement with the previous results [FW98, ZFMS00], which shows that using modeled overestimated runtimes have similar or slightly better average waiting time or slowdown than using actual runtimes.

The reason Scenario A requested runtimes result in similar or slightly better performance than using actual runtimes is because increasing the requested runtimes proportional to actual runtimes as in Scenario A results in longer estimated completed time for currently executing jobs (thus increased chances to backfill longer jobs) but also longer estimated requested runtimes for waiting jobs (thus requiring a longer backfilled window). The effects on both executing and waiting jobs cancel out on each other to some degree; however, since the difference between the requested and actual runtimes for longer executing jobs is larger than that for shorter executing jobs in Scenario A, using Scenario A can be actually beneficial for shorter jobs.

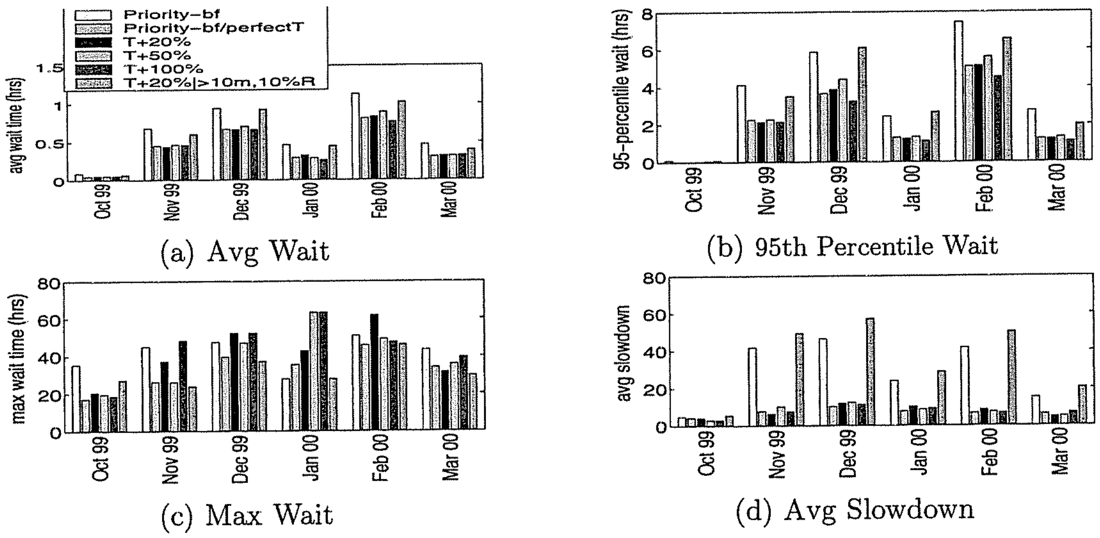


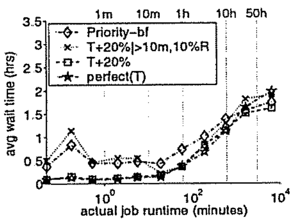
Figure 74. Overall Performance of More Accurate Requested Runtime for Priority-backfill

Thus, the conclusion is that if the actual runtimes are known or the requested runtimes are approximately proportional to the actual runtimes, the performance of FCFS-backfill can be improved, but the improvement is very limited. For this reason and because Scenario B is not expected to improve the performance of Scenario A, Scenario B is not evaluated for FCFS-backfill.

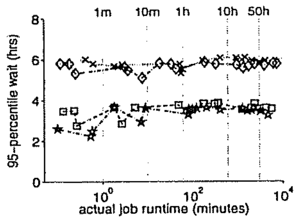
6.2.3 Potential Benefit of More Accurate Requested Runtime for Priority backfill Policies

This section examines the impact of using more accurate requested runtimes for priority backfill policies evaluated, including the actual runtimes, Scenario A and Scenario B.

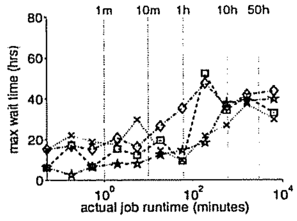
In contrast to the results for FCFS-backfill (Figures 73 in the previous section), Figures 74(a)-(d) show that the benefit of using actual runtimes and using Scenario A requested runtimes is much larger for Priority-backfill than for FCFS-backfill. In particular, using actual runtimes or Scenario A improved requested runtimes for Priority-backfill not only dramatically reduces the overall average slowdown (by 50-75% for most months, shown in Figure 74(d)), but also improves the overall average and 95th percentile wait time by over 25% each month (shown



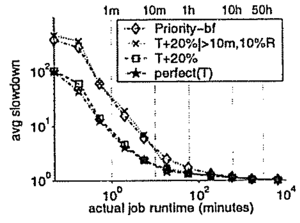
(a) Avg Wait (Dec 1999)



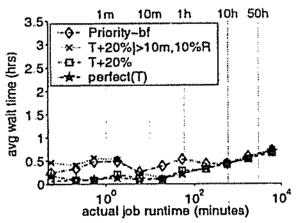
(b) 95th Percentile Wait (Dec 1999)



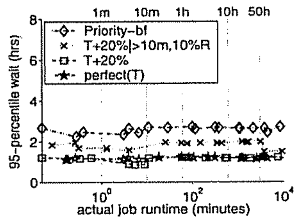
(c) Max Wait (Dec 1999)



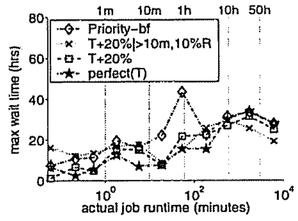
(d) Avg Slowdown (Dec 1999)



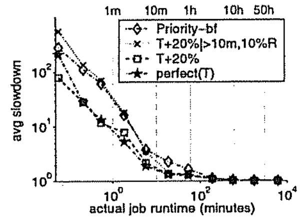
(e) Avg Wait (March 2000)



(f) 95th Percentile Wait (March 2000)



(g) Max Wait (March 2000)



(h) Avg Slowdown (March 2000)

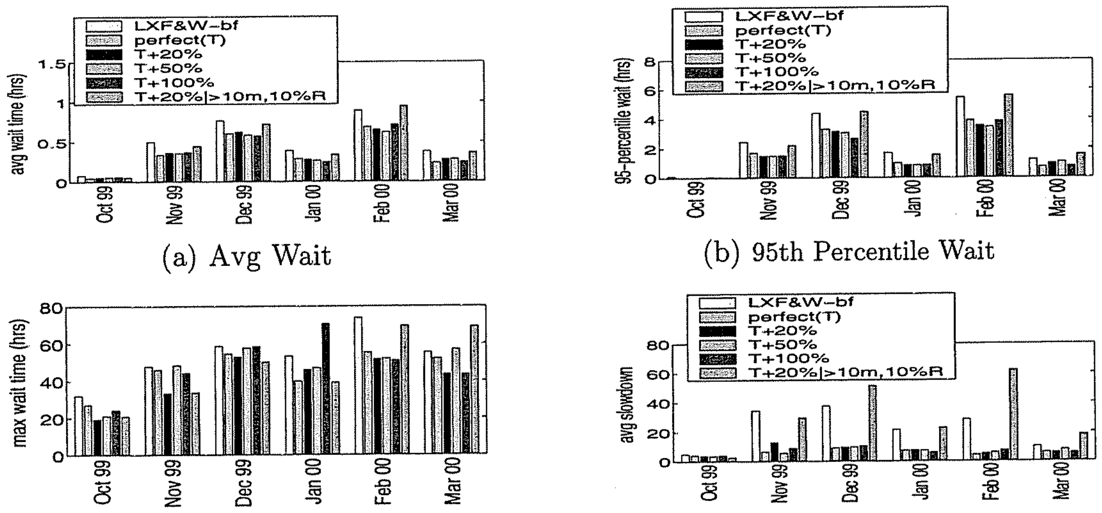
Figure 75. Performance vs. Actual Runtime Using More Accurate Requested Runtime for Priority-backfill

in Figures 74(a) and (b), respectively).

However, as also shown in Figures 74(a)-(d), using Scenario B requested runtimes for Priority-backfill only slightly improves the overall average, 95th percentile, and maximum wait time for most months, and increases the overall average slowdown by up to 20%.

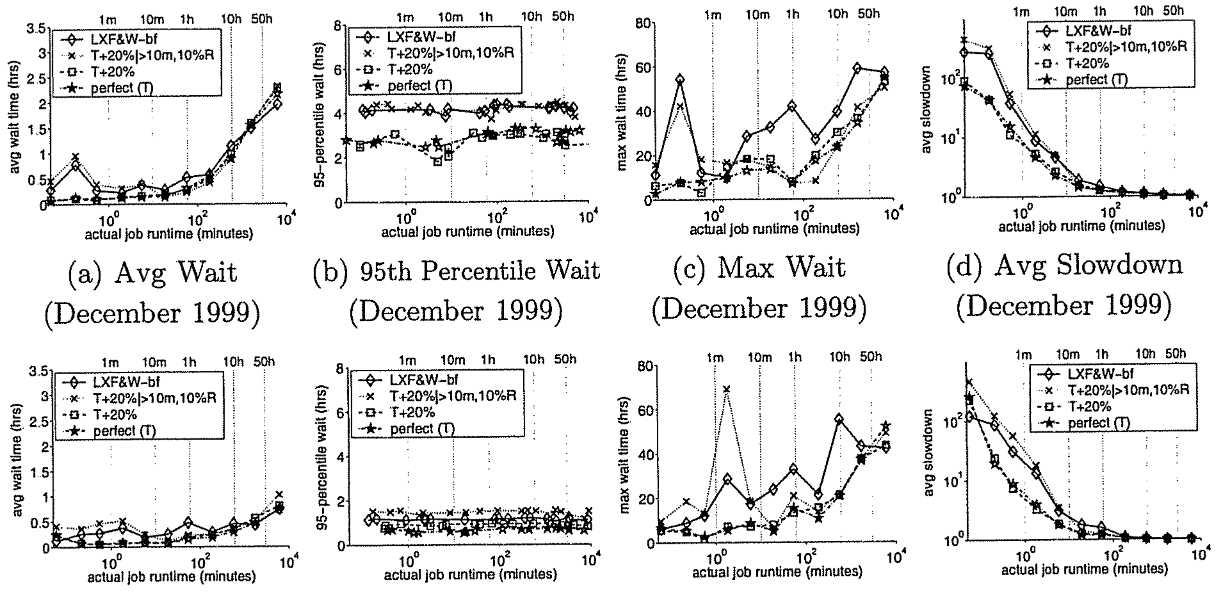
To show why Scenario B has worse average slowdown, Figure 75 plots the performance versus actual runtime for Priority-backfill using different scenarios of requested runtimes. It shows that prematurely terminated jobs (< 10 minutes) in Scenario B may incur poor average and maximum wait time and average slowdown longer jobs have more accurate requested runtime. Nevertheless, the performance of longer jobs (that have improved requested runtimes) in Scenario B is improved.

The impact of using actual runtimes and Scenario A and B on the other two priority backfill policies, LXF&W-backfill and SJF-backfill, is similar to that for Priority-backfill. The results for LXF&W-backfill are shown in Figure 76 for the overall performance and in Figure 77 for the performance versus actual runtime for two given months. The results for SJF-backfill are



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

Figure 76. Overall Performance of Using More Accurate Requested Runtime for LXF&W-backfill



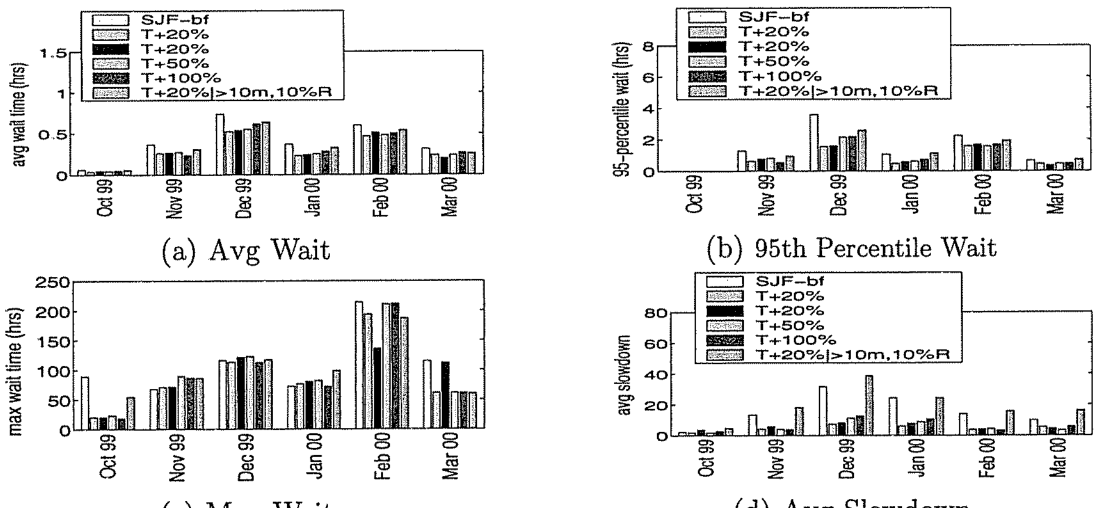
(a) Avg Wait (December 1999) (b) 95th Percentile Wait (December 1999) (c) Max Wait (December 1999) (d) Avg Slowdown (December 1999)

(e) Avg Wait (March 2000) (f) 95th Percentile Wait (March 2000) (g) Max Wait (March 2000) (h) Avg Slowdown (March 2000)

Figure 77. Detail Performance of Using More Accurate Requested Runtime for LXF&W-backfill

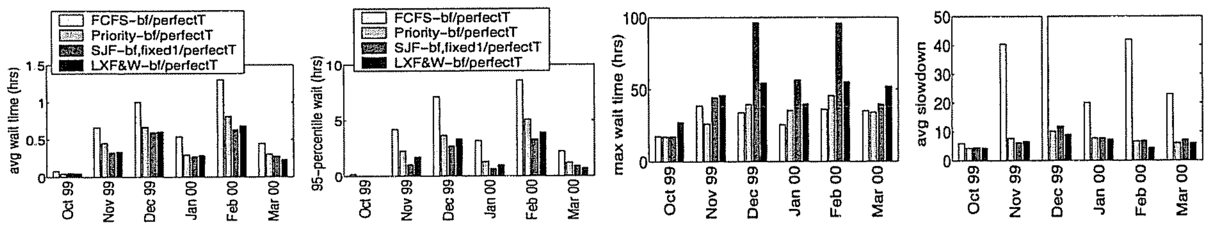
shown in Figure 78 for the overall performance.

The conclusions in this section are: (1) the benefit of using more accurate requested runtimes is much more significant for priority backfill policies that give priority to short jobs than that



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

Figure 78. Overall Performance of Using More Accurate Requested Runtime for SJF-backfill



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

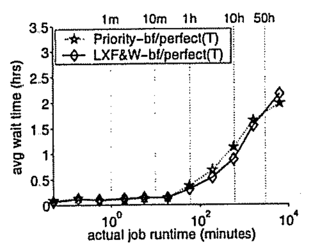
Figure 79. Overall Performance of Alternative Priority Backfill Policies Using Actual Runtimes

for FCFS-backfill; (2) however, jobs that prematurely terminate due to unexpected errors may incur poor turnaround if other jobs have improved requested runtimes.

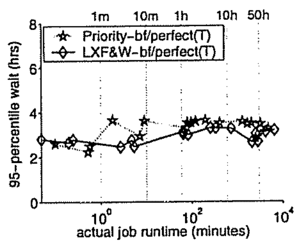
6.2.4 Alternative Priority Backfill Policies Using Actual Runtimes

Since the improvement of using more accurate requested runtimes is much larger for priority backfill policies that favor short jobs than that for FCFS-backfill, next, we re-evaluates FCFS-backfill, Priority-backfill, and LXF&W-backfill, using actual runtimes.

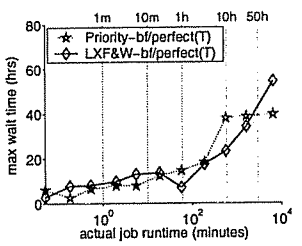
Recall that that using inaccurate requested runtimes given in the O2K workload, FCFS-backfill and Priority-backfill have similar performance (except for maximum wait time) versus



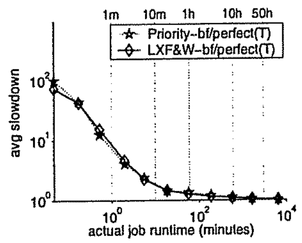
(a) Avg Wait (Dec 1999)



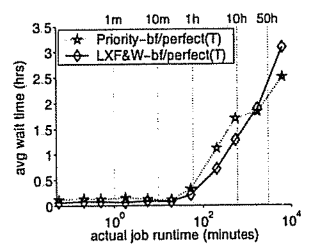
(b) 95th Percentile Wait (Dec 1999)



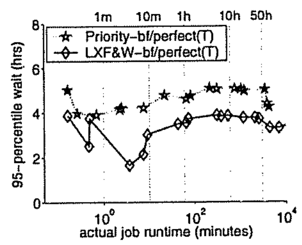
(c) Max Wait (Dec 1999)



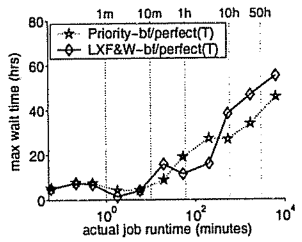
(d) Avg Slowdown (Dec 1999)



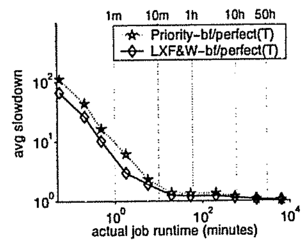
(e) Avg Wait (February 2000)



(f) 95th Percentile Wait (February 2000)



(g) Max Wait (February 2000)



(h) Avg Slowdown (February 2000)

Figure 80. Performance vs. Actual Runtime of Alternative Priority Backfill Policies Using Actual Runtimes

actual runtime for each month, as shown in Section 5.2. Figure 79 shows that if actual runtimes are known or all jobs (including short jobs) have improved requested runtimes, Priority-backfill significantly outperforms FCFS-backfill: significantly lower overall average and 95th percentile wait time and average slowdown (in Figures 79(a), (b), and (d)), and comparable maximum wait time (in Figure 79(c)).

Figure 79 also shows that LXF&W-backfill still has lower average and 95th percentile wait time than that of Priority-backfill if actual runtimes are used, but the improvement is much smaller than that if the O2K requested runtimes are used. Figure 80 further shows the performance versus actual runtime for Priority-backfill and LXF&W-backfill (for the case if the actual runtimes are known). It shows that LXF&W-backfill has considerably lower 95th percentile wait time than Priority-backfill for most ranges of actual runtime for the two heavy-load months (December 1999 and February 2000 in Figures 80(b) and (f) respectively), but Priority-backfill is very comparable for other measures.

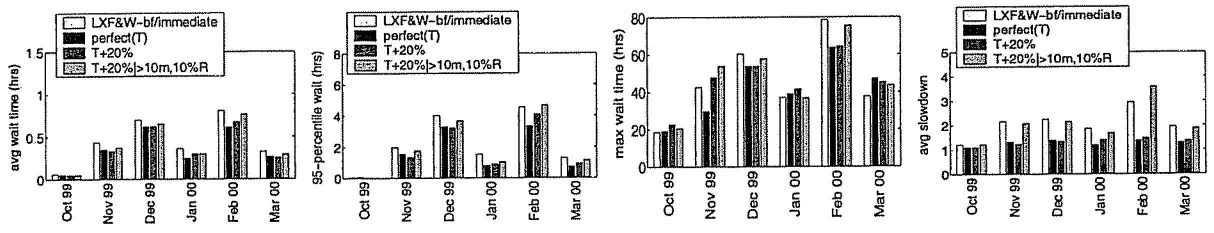
The key conclusion is that if the actual runtimes are known to the schedulers, Priority-backfill significantly outperform FCFS-backfill for all performance measures. Another conclusion is that LXF&W-backfill still outperforms Priority-backfill if the actual runtimes are known (or the requested runtimes are proportional to the actual runtimes), but the improvement is not as much as that if the requested runtimes given in the O2K workload are used.

6.2.5 Impact of More Accurate Requested Runtime For LXF&W-backfill with Immediate Service

This section provides the results for using more accurate requested runtimes for LXF&W-backfill with 1-minute immediate service, the best priority backfill policy.

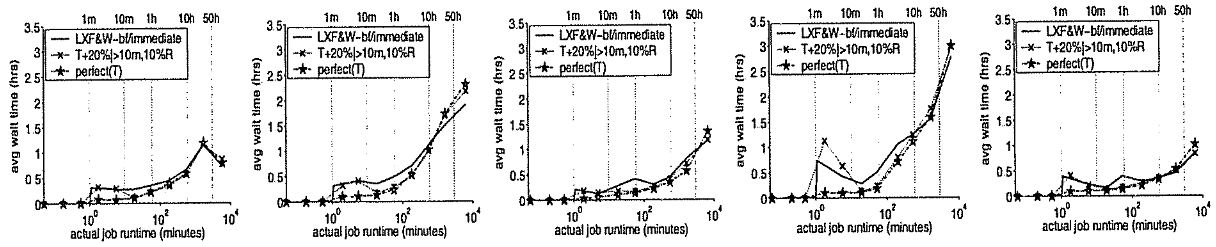
As shown in Figures 81(a)-(d), using actual runtimes improves LXF&W-backfill with 1-minute immediate service, but the improvement is significantly smaller than that for LXF&W-backfill. In particular, the improvement in the average slowdown for LXF&W-backfill with immediate service is 25% for most months and no more than 50% (Figures 81(d)), compared to more than 50% improvement for LXF&W-backfill in most months (Figures 76(d)). This is because the dramatical improvement in the average slowdown under LXF&W-backfill is mainly due to the improvement for 1-minute jobs, which has good turnaround under LXF&W-backfill with 1-minute immediate service whether the actual runtimes or the O2K requested runtimes are used. For the same reason, Scenario B has a smaller impact on the average slowdown for LXF&W-backfill with immediate service. In particular, Scenario B does not impact the performance for LXF&W-backfill, except a slight increase in the averge slowdown for one month (February 2000).

Figures 82(a)-(e) provides the average wait time versus actual runtime for LXF&W-backfill with 1-minute immediate service using the O2K requested runtimes, the actual runtimes, and Scenario B requested runtimes for each month (except the light-load month). The graphs show that due to the 1-minute quantum, the jobs that terminate in under 1-minute have good



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

Figure 81. Impact of More Accurate Requested Runtime on LXF&W-backfill with Preemptive Immediate Service



(a) Nov 1999 (b) Dec 1999 (c) Jan 2000 (d) Feb 2000 (e) March 2000

Figure 82. Impact of More Accurate Requested Runtime on LXF&W-backfill with Preemptive Immediate Service: Average Wait Time versus Actual Runtime Each Month

turnaround in any case. However, jobs that are longer than 1 minute but shorter than 10 minutes have significantly higher average wait time in Scenario B for one high-load month (February 2000, shown in Figure 82(d)).

The results for LXF&W-backfill with immediate service suggest that using a larger quantum for immediate service (e.g., 10 minutes) can improve jobs that prematurely terminate, even if other longer jobs have more accurate requested runtimes. However, a longer quantum may impact the performance of longer running jobs and may create the problem of memory pressure. Further investigation of the impact of a longer quantum for immediate service is necessary to determine whether it can improve the overall performance.

6.3 Summary of Further Policy Evaluation

In this chapter, we evaluate the performance of EQspatial against LXF&W-backfill with immediate service, and evaluate the potential benefit of using more accurate requested runtimes for backfill policies. Our results for FCFS-backfill using perfect runtime and Scenario A requested runtimes are consistent with previous results.

The key results are: (1) EQspatial has the potential to significantly further improve the performance of LXF&W-backfill with immediate service, especially for the 95th percentile waiting time, for the jobs that don't complete in the 1-minute quantum under LXF&W-backfill with 1-minute immediate service; (2) It may be important to modify EQspatial to reduce the maximum waiting time of the jobs that have long running time and a large requested number of processors in the challenging production workloads. A proposed simple modification, that allocates each long running jobs (> 50 hours) as many processors as requested, effectively improves the maximum wait time of EQspatial and results in comparable maximum wait and still significantly lower average and 95th percentile wait time than that of LXF&W-backfill with immediate service for most ranges of runtime; (3) The benefit of more accurate requested runtime is much larger for priority backfill policies that favor short jobs than that for FCFS-backfill. In particular, if the actual runtimes are known or the requested runtimes are proportional to the actual runtime, Priority-backfill significantly improves the average and 95th percentile wait time (by at least 25%) and average slowdown of FCFS-backfill; whereas, using the inaccurate requested runtimes given in the O2K workload, these two policies have similar performance (except for the maximum wait time) for each month studied; (4) Jobs that request a long runtime but prematurely terminate may incur poor turnaround, if other jobs have more accurate requested runtimes. Preemptive immediate service can help the short jobs that abort within the initial quantum. To help the other short jobs that prematurely terminate but run longer than the initial quantum, users are encouraged to request a short runtime to test-run each new configuration for a job, before submitting the job with a longer runtime.

Chapter 7

Conclusions and Future Work

This thesis has provided new and improved parallel job scheduling policies as well as a better understanding of relative scheduling policy performance for challenging production parallel workloads, such as the workload that runs on the NCSA Origin 2000 system. The design of new policies was guided by fundamental job scheduling principles and the observed workload characteristics, with two additional goals; namely, easy to tune and avoiding user gaming for better service. A more complete understanding of relative policy performance was obtained by using more extensive performance measures for policy comparisons (e.g., average and 95th percentile waiting time as a function of actual runtime) and by performing comparisons among a more extensive set of policies than in previous work.

Due to the absence of a reliable synthetic workload model in the literature, the performance evaluations and comparisons were conducted using trace-driven simulation with six one-month O2K job traces. A key contribution of this thesis is that we developed a model for creating synthetic workloads that have the observed *relationships* among key job characteristics in the workload as well as the observed distributions for each characteristic. The model uses conditional distributions to capture the relationships among job characteristics. The requisite conditional distributions are identified by plotting the percentiles of a given distribution that might need conditioning as a function of the values of the candidate conditioning variables. We applied the approach to obtain a procedure for generating synthetic workloads using conditional distributions that faithfully represent the O2K workload.

As a result of workload characterization, the O2K workload during the year 2000 is completely characterized with respect to the range of policies that were evaluated. The characterization aided the design of new and improved scheduling policies for the workload, and provides data for comparison against other system workloads. Thus, the results of the scheduling policy evaluation for the O2K workload may be applied to other systems that have similar workloads.

Section 7.1 summarizes the interesting characteristics of the O2K workload. Section 7.2 concludes the results of policy comparisons. Section 7.3 provides possible directions for future work on workload characterizations on the job scheduling policies.

7.1 Workload Characterization

The O2K workload is a large-scale production workload in that the O2K system has more processors and memory than previously studied systems, jobs request a large fraction (80-90%) of the processing cycles each month, and the longest O2K job runs up to 400 hours (compared to only a few tens of hours on previously studied production parallel scientific workloads). In addition, the O2K system is a shared-memory system, while the previous workloads studied are for distributed-memory systems. Despite the differences in the systems and system scale, for the characteristics that are reported in previous workload studies, there are many similarities and only a few differences in the characteristics of the O2K workload. In addition, this thesis has reported many new workload characterization results due to new workload measures and analysis.

The similarities in the O2K workload to previous workloads include the following. A large fraction of jobs are serial and most jobs request power of two numbers of processors. A large fraction (50%) of the O2K jobs use a small fraction (< 15%) of the memory available per processor. The total memory usage has a positive correlation with the number of requested processors. There is a large discrepancy between peak memory usage and requested memory. In particular, 15-20% of the jobs use more memory than requested. For the remaining jobs, the ratio of the peak memory usage to requested memory is approximately uniformly distributed over each 10% interval from 0 to 100%. Many jobs have a significant discrepancy between the requested and actual runtimes. The actual runtime can be modeled by the Weibull distribution, which is simpler but similar to the piece-wise log-uniform distribution suggested for previous workloads. Jobs that are very short (under 1 minute) use much less than 1 GB of memory in total.

A few characteristics in the O2K workload are different from several previously studied workloads. First, the coefficient of variation (CV) of the job interarrival time, during each period of relatively stable arrival rate and excluding system downtimes, is in the range of 1 - 2 for the O2K workload. This is similar to that reported for the daytime workload on an SP/2, but smaller than the CV of 2.5 - 6 that has been reported for several other workloads (iPSC/860, Paragon, SP/2). One possible reason for the higher CV in many previous workloads is that the CV was computed for daytime and nighttime periods over several months (possibly also including system downtimes during which no arrivals occur). Second, there is a negative correlation between normalized memory usage (i.e., peak memory usage divided by number of processors) and the number of processors requested in the O2K workload, while a positive correlation between these two job measures is reported for a CM-5 workload. In addition, we found a complex correlation between actual runtime and the number of requested processors in the O2K workload, which is similar to an SP/2 workload; in contrast, a positive correlation between these two job measures is reported for an iPSC/860 and a Paragon workload.

Regarding the discrepancy between the requested runtime and actual runtime, there is a significantly higher fraction of the O2K jobs making very poor runtime estimations than that in an SP/2 workload; namely, at least 40% of the O2K jobs for any default requested runtime class, compared to only 10% of all SP/2 jobs, use under 10% of the runtimes requested. This could be due to larger default requested runtimes defined on the O2K (i.e., 5, 50, 200, and 400 hours).

The new workload measures and analysis provided in this thesis include: the characteristics of the fifteen largest jobs each month, comparisons of the workload mix from month to month, more complete analysis of number of job arrivals for each hour and each day, including identifying periods of stable arrival rate, comparisons of the characteristics of jobs submitted from period to period, the distribution of requested runtime, the distribution of requested memory for a shared-memory system, the distribution of processor efficiency per job (i.e., the utilization on the requested processors of the job), the distribution of average memory efficiency per job (i.e., the average memory usage divided by the memory requested of the job), the overall utilization on the processors and memory allocated each month, new conditional distributions that capture the relationships among key job characteristics, i.e., the distributions of requested memory, conditioned on requested number of processors; the distributions of requested runtime, conditioned on requested number of processors and requested memory; the distributions of the ratio of the actual to the requested runtime, conditioned on the requested runtime, the requested number of processors, and the requested memory; and the distributions of the ratio of the peak memory usage to requested memory, conditioned on the actual runtime and requested number of processors. In addition, the distribution of actual runtime for each requested runtime class is also provided. Finally, we provide a procedure for generating synthetic workloads that have the observed distributions and correlation between key job characteristics in the O2K workload; the procedure is summarized in Sections 4.4.4 and 4.5.3.

Interesting results from the new analyses, which impact scheduler design, include:

- In a typical month, the fifteen largest jobs each month have a runtime over 200 hours and use 4000 - 8000 processor-hours on average, which is at least an order of magnitude larger than the average over all jobs in each month. In addition, at least two-thirds of these largest jobs each month achieve 80-100% utilization of the requested (and allocated) processors.

- A large fraction (60%) of all jobs, including long and/or large-processor jobs, achieve over

80% utilization of the processors allocated, which leads to high overall utilization (i.e., 80%) on the processors allocated each month.

- The overall utilization on the memory allocated each month is 40-55%, which is significantly lower than the utilization on the processors each month.
- There is a large discrepancy between actual and requested runtime for every requested runtime class. In particular, 10% of the jobs that request over 5 hours of runtime terminate in under 1 minute. This is possibly due to unexpected errors in the programs, yet it is surprising that 10% of the jobs that request over 5 hours would terminate due to unanticipated errors. Furthermore, over 20% of the jobs that request 200 or more hours of runtime run for over 10 hours but terminate in under 50 hours. Thus, it appears that a significant fraction of inaccurate requested runtimes are due to misestimation of the runtime. We note that the scheduler performance may be improved if the requested runtimes could be more accurate.

Other new workload results that are also important for scheduling include the following. First, the O2K workload mix did not change over time, even after the job scheduler used on the system changed (from NCSA-LSF* to a priority backfill policy). Second, there are three weekday periods (i.e., peak, intermediate, and low) and two weekend periods (i.e., intermediate and low) of approximately stable arrival rate per hour. We notice from the data provided in several previous papers that similar periods of stable job arrival rate per hour also exists on several other systems (namely, iPSC/860, SP/2, and the Lewis network of workstations at NASA), but these periods were not identified or used in the previous work. Third, allowing statistical fluctuation in small samples, the jobs submitted during peak arrival rate periods have similar characteristics to the jobs submitted during non-peak periods of arrival rate. The job characteristics are also similar for different months and for different days of the week. Fourth, although the total requested memory is positively correlated with the number of requested processors, the normalized requested memory is negatively correlated with the number of requested processors. Finally, a significant fraction (15-20%) of the jobs request a larger fraction of system memory than processors. Such jobs would not run efficiently on a distributed-memory system.

7.2 Job Scheduling Policies

This thesis has proposed new and improved policies and has provided more extensive evaluations and comparisons of new and previous policies, with a goal of providing useful information to help NCSA or other production facilities to choose an appropriate scheduling policy for their workloads.

To gain more complete understanding of policy performance, more extensive performance measures were used for evaluating job scheduling policies. In addition to the overall average wait and average slowdown as commonly used in previous policy comparisons, we provided average, 95th, and maximum wait time versus actual runtime, number of requested processors, and requested memory.

The new and previous policies quantitatively compared are: (1) the highly tuned static job-class priority NCSA-LSF* scheduler, (2) two new and simplified priority backfill policies: Priority-backfill and LXF&W-backfill that give priority to short jobs without starving long jobs, (3) previously proposed FCFS-backfill and SJF-backfill, (4) varying numbers of reservations for priority backfill policies, (5) limited preemption for backfill policies to improve the turnaround for very short jobs, (6) the EQspatial spatial equi-partitioning policy modified to reduce the maximum wait for large and long running jobs. (7) the impact of more accurate requested runtime for Priority-backfill, LXF&W-backfill, and FCFS-backfill. The results for the impact on FCFS-backfill are compared against prior work.

The key conclusions of job scheduling policies in this thesis are:

- Static job-class priority scheduling such as that in NCSA-LSF*, does not appear to perform well, because limits are required to prevent starvation. This causes significant wait times for higher-priority jobs when there are more of them to be scheduled. Furthermore, the scheduling performance is sensitive to specific order of job arrivals and it is difficult to tune the job limits.

- The benefit of more accurate requested runtime is much larger for priority backfill policies that favor short jobs than that for FCFS-backfill.

- The current estimated job expansion factor plus a relatively very small weight for the current job waiting time, used in LXF&W-backfill, provides a *starvation-free* priority measure for backfill policies that favors short jobs and performs significantly better than FCFS-backfill and the other priority backfill policies studied. Furthermore, the LXF&W priority measure has three additional advantages: (1) intuitive and based on fundamental scheduling policy results, (2) easy to experimentally tune (i.e., the weight for the job wait time) due to its simplicity, and (3) avoids gaming for better service.

- With an appropriate initial quantum and memory and selection of jobs for preemption that have not recently been preempted, adding a preemptive immediate service for priority backfill policies effectively provides immediate turnaround for jobs that complete in the initial quantum and memory, without impacting the performance of other jobs. The initial quantum and memory are based on measured distribution of actual runtime and the measured memory usage of short jobs.

- The dynamic equal spatial partitioning policy (EQspatial) has the potential to significantly improve priority backfill policies with limited preemption for immediate service, but it may be important to reduce the maximum waiting time of long and large-processor jobs under EQspatial.

  A proposed simple modification that allocates as many processors as requested to each long running job ($>$ 50 hours) effectively reduces the maximum wait time of EQspatial to be comparable to the best priority backfill policy studied, i.e., LXF&W-backfill with immediate service, and has significantly better performance (e.g., 50-75% lower 95th percentile wait time for most ranges of runtime).

Further simulation results show that (1) for the O2K workload, giving priority to large-memory jobs in addition to large-processor jobs does not improve the performance of large-memory jobs, because the processor resources are more constrained than the memory on the current O2K system. (2) Jobs that request a long runtime but prematurely terminate may incur poor turnaround, if other jobs provide more accurate requested runtimes. Preemptive immediate service can help the short jobs that abort within the initial quantum. To help the other short jobs that prematurely terminate, users may request a short runtime (e.g., 10 minutes) to test-run each new configuration for a job before submitting the job with a longer runtime.

7.3 Future Work

7.3.1 Future Work for Workload Characterization

Possible directions for future work on the study of the workloads and the workload model include: (1) analysis of related job arrivals in the O2K workload, i.e., the patterns of repeated submissions of similar applications; (2) analysis of job I/O demands; (3) analysis of the variation in the actual memory usage relative to the memory requested during the execution of each job; (4) analysis of the workload submitted during weekdays in which there is a higher number of job arrivals per day (i.e., > 500) than that in a typical weekday (i.e., 350-400/day); (5) analysis of whether statistical fluctuation in finite sample sizes accounts for small variation observed among periods of job arrival rate, daily, and monthly workloads on the O2K; (6) analysis of the jobs that request a dedicated host on the O2K; (7) implementation of a synthetic workload generator that uses the workload distributions provided in Chapter 4, with the flexibility of varying the workload parameters within expected or hypothetical range; and (8) analysis of the workloads on future cluster of computers (e.g., the 960-node IA32 Linux cluster at NCSA) and on more widely distributed environment (e.g., Condor or Globus).

7.3.2 Future Work for Policy Evaluation

Possible directions for future work on job scheduling policies include (1) further evaluation of EQspatial, using EQspatial host placement to choose the host for scheduling a job and using gang scheduling for sharing the processors among the jobs on each host. (2) further refinements of LXF&W-backfill with an initial quantum larger than 1 minute to possibly help jobs that prematurely terminate but run longer than 1 minute. Note that a larger quantum may require a larger initial memory ($>$ 1 GB) and a higher threshold of non-interrupted execution time (e.g., 100 minutes as opposed to 10 minutes for 1-minute quantum) before an executing job is eligible to be preempted. (3) evaluation of nonpreemptive policies with initial adaptive processor allocations (e.g., ASP-max) for the O2K. Such policies have the ability to adapt to increased load by reducing the processor allocations; however, a minimum processor allocations may be required to avoid long turnaround for large-processor jobs. (4) comparing policies using synthetic workloads generated from the workload model provided in Chapter 4.

Bibliography

[AAN+98] P. Agnihotri, V. K. Agarwala, J. J. Nucciarone, K. M. Morooney, and C. Das. The penn state computing condominium scheduling system. In Proc. 1998 ACM/IEEE Supercomputing Conf., Orlando, November 1998.

[Aid00] K. Aida. Effect of job size characteristics on job scheduling performance. In Proc. 6th Workshop on Job Scheduling Strategies for Parallel Processing, pages 1–17, Cancun, May 2000. Lecture Notes in Comp. Sci. Vol. 1911, Springer-Verlag.

[All90] A. O. Allen. Probability, Statistics, and Queueing Theory with Computer Science Applications. Academic Press, 2nd edition, 1990.

[BF00] A. Batat and D. G. Feitelson. Gang scheduling with memory considerations. In Proc. Int'l. Parallel and Distributed Processing Symp., May 2000.

[BJK+95] R. Blumofe, C. Joerg, B. Kuszmaul, C. Leiserson, K. Randall, and Y. Zhou. Cilk: An efficient multithreaded runtime system. In Proc. 5th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, pages 207–216, Santa Barbara, June 1995.

[BL97] R. Blumofe and C. Leiserson. Adaptive and reliable parallel computing on networks of workstations. In USENIX 1997 Annual Technical Conf. on UNIX and Advanced Computing Systems, Anaheim, CA, 1997.

[CMV94] S.-H. Chiang, R. K. Mansharamani, and M. K. Vernon. Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. In Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, pages 33–44, Nashville, May 1994.

[CNSW97] A. Chowdhury, L. D. Nicklas, S. K. Setia, and E. L. White. Supporting dynamic space-sharing on clusters of non-dedicated workstations. In Proc. 17th Int'l. Conf. on Distributed Computing Systems, 1997.

[DAC96] A. C. Dusseau, R. H. Arpaci, and D. E. Culler. Effective distributed scheduling of parallel workloads. In Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, Philadelphia, May 1996.

[DF99] A. B. Downey and D. G. Feitelson. The elusive goal of workload characterization. Performance Evaluation Review, 26(4):14–29, March 1999.

[Dow97a] A. B. Downey. A parallel workload model and its implications for processor allocation. In Proc. 6th IEEE Symp. on High Performance Distributed Computing, Portland, August 1997.

[Dow97b] A. B. Downey. Predicting queue times on space-sharing parallel computers. In Proc. 3rd Workshop on Job Scheduling Strategies for Parallel Processing, Geneva, April 1997. Lecture Notes in Comp. Sci. Vol. 1291, Springer-Verlag.

[DY98] S. P. Dandamudi and H. Yu. Performance sensitivity of space-sharing processor scheduling in distributed-memory multicomputers. In Proc. 12th Int'l. Parallel Processing Symp., pages 403–409, Orlando, March 1998.

[EZL89] D. L. Eager, J. Zahorjan, and E. D. Lzzowaka. Speedup versus efficiency in parallel systems. IEEE Transactions on Computers, pages 408–423, March 1989.

[Fei96] Dror G. Feitelson. Packing schemes for gang scheduling. In Proc. 2nd Workshop on Job Scheduling Strategies for Parallel Processing, pages 89–110, Honolulu, April 1996. Lecture Notes in Comp. Sci. Vol. 1162, Springer-Verlag.

[Fei97a] D. G. Feitelson. Memory usage in the LANL CM-5 workload. In Proc. 3rd Workshop on Job Scheduling Strategies for Parallel Processing, pages 78–94, Geneva, April 1997. Lecture Notes in Comp. Sci. Vol. 1291, Springer-Verlag.

[Fei97b] Dror G. Feitelson. Job scheduling in multiprogrammed parallel systems. Technical report, IBM T. J. Watson Research Center, August 1997.

[FJ97] Dror G. Feitelson and Morris A. Jette. Improved utilization and responsiveness with gang scheduling. In Proc. 3rd Workshop on Job Scheduling Strategies for Parallel Processing, Geneva, April 1997. Lecture Notes in Comp. Sci. Vol. 1291, Springer-Verlag.

[FJM+99] H. Franke, J. Jann, J. Moreira, P. Pattnaik, and M. Jette. An evaluation of parallel job scheduling for ASCI Blue-Pacific. In Proc. 1999 ACM/IEEE Supercomputing Conf., Portland, November 1999.

[FN95] D. .G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In Proc. 1st Workshop on Job Scheduling Strategies for Parallel Processing, pages 337–360, Santa Barbara, April 1995. Lecture Notes in Comp. Sci. Vol. 949, Springer-Verlag.

[FR90] D. G. Feitelson and L. Rudolph. Distributed hierarchical control for parallel processing. Computer, 23(5):65–77, May 1990.

[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.

[Gib97a] R. Gibbons. A historical application profiler for use by parallel schedulers. Master's thesis, Univ. of Toronto, Ontario, 1997.

[Gib97b] 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.

[GL99] G. Ghare and S. T. Leutenegger. The effect of correlating quantum allocation and job size for gang scheduling. In Proc. 5th Workshop on Job Scheduling Strategies for Parallel Processing, pages 91–110, San Juan, April 1999. Lecture Notes in Comp. Sci. Vol. 1659, Springer-Verlag.

[GST91] D. Ghosal, G. Serazzi, and S. K. Tripathi. The processor working set and its use in scheduling multiprocessor systems. IEEE Trans. on Software Engineering, 17(5):443–453, May 1991.

[GTU91] A. Gupta, A. Tucker, and S. Urushibara. The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications. In Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, pages 120–132, San Diego, May 1991.

[HB00] M. Harchol-Balter. Task assignment with unknown duration. In Proc. Int'l. Conf. on Distributed Computing Systems, Taipei, Taiwan, April 2000.

[HBD96] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. In Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, May 1996.

[Hot96] S. Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. In Proc. 2nd Workshop on Job Scheduling Strategies for Parallel Processing, pages 27–40, Honolulu, April 1996. Lecture Notes in Comp. Sci. Vol. 1162, Springer-Verlag.

[HSO96] S. Hotovy, D. Schneider, and T. O'Donnell. Analysis of the early workload on the Cornell Theory Center IBM SP2. In Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, pages 272–273, Philadelphia, May 1996.

[IPS$^+$97] N. Islam, A. Prodromidis, M. S. Squillante, A. S. Gopal, and L. L. Fong. Extensible resource scheduling for parallel scientific applications. In Proc. Eighth SIAM Conf. on Parallel Processing for Scientific Computing, March 1997.

[IPSG97] N. Islam, A. Prodromidis, M. S. Squillante, and A. S. Gopal. Extensible resource scheduling for parallel scientific applications. In Proc. 17th Int'l. Conf. on Distributed Computing Systems, pages 561–658, Minneapolis, March 1997.

[JEM97] V. K. Naik J. E. Moreira. Dynamic resource management on distributed systems using reconfigurable applications. Technical Report RC 20890, IBM, 1997.

[JJS01] D. B. Jackson, H. L. Jackson, and Q. O. Snell. Simulation based HPC workload analysis. In Proc. Int'l. Parallel and Distributed Processing Symp. (IPDPS) 2001, San Francisco, April 2001.

[JKK92] N. L. Johnson, S. Kotz, and A. W. Kemp. Univariate Discrete Distributions. Wiley, 2nd edition, 1992.

[JN99] J. P. Jones and B. Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In Proc. 5th Workshop on Job Scheduling Strategies for Parallel Processing, pages 1–16, San Juan, April 1999. Lecture Notes in Comp. Sci. Vol. 1659, Springer-Verlag.

[JPF$^+$97] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan. Modeling of workload in MPPs. In Proc. 3rd Workshop on Job Scheduling Strategies for Parallel Processing, Geneva, April 1997. Lecture Notes in Comp. Sci. Vol. 1291, Springer-Verlag.

[Kle76] L. Kleinrock. Queueing Systems. Wiley, New York, 1976.

[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.

[LKK99] William Leinberger, George Karypis, and Vipin Kumar. Job scheduling in the presence of multiple resource requirements. In Proc. 1999 ACM/IEEE Supercomputing Conf., Portland, November 1999.

[LV90] S. T. Leutenneger and M. K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, pages 226–236, Boulder, May 1990.

[Mat] MathWorks, Natick, MA. Matlab. http://www.mathworks.com/.

[Mau] Maui High Performance Computing Center, Maui, Hawaii. Maui Scheduler. http://www.mhpcc.edu/maui/.

[MCF$^+$98] J. E. Moreira, W. Chan, L. L. Fong, H. Franke, and M. Jette. An infrastructure for efficient parallel job execution in terascale computing environments. In Proc. 1998 ACM/IEEE Supercomputing Conf., Orlando, November 1998.

[MEB91] S. Majumdar, D. L. Eager, and R. B. Bunt. Characterisation of programs for scheduling in multiprogrammed parallel systems. Performace Evaluation, pages 109–130, 1991.

[MHP] Maui High Performance Computing Center. http://www.mhpcc.edu/.

[MNK96] J. E. Moreira, V. K. Naik, and R. B. Konuru. A programming environment for dynamic resource allocation and data distribution. In Proc. Ninth Int'l Workshop on Languages and Compilers for Parallel Computing, 1996.

[MVZ93] 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.

[MZ94] Cathy McCann and John Zahorjan. Processor allocation policies for message-passing parallel computers. In Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, pages 19–32, Nashville, May 1994.

[MZ95] Cathy McCann and John Zahorjan. Scheduling memory constrained jobs on distributed memory parallel computers. In Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, pages 208–219, Ottawa, May 1995.

[NSS93a] V. K. Naik, S. K. Setia, and M. S. Squillante. Performance analysis of job scheduling policies in parallel supercomputing environments. In Proc. 1993 ACM/IEEE Supercomputing Conf., pages 824–833, Portland, November 1993.

[NSS93b] V. K. Naik, S. K. Setia, and M. S. Squillante. Scheduling of large scientific applications on distributed memory multiprocessor systems. In Proc. 6th SIAM Conf. on Parallel Processing for Scientific Computing, pages 913–922, Norfolk, VA., March 1993.

[Ous82] J. K. Ousterhout. Scheduling techniques for concurrent systems. In Proc. 3th Int'l. Conf. on Distributed Computing Systems, 1982.

[PD96] J. Padhye and L. Dowdy. Dynamic vesus adaptive processor allocation policies for message passing parallel computers: an empirical comparison. In Proc. 2nd Workshop on Job Scheduling Strategies for Parallel Processing, pages 165–181, Honolulu, April 1996. Lecture Notes in Comp. Sci. Vol. 1162, Springer-Verlag.

[PF00] F. Petrini and W.-C. Feng. Time-sharing parallel jobs in the presence of multiple resource requirements. In Proc. 6th Workshop on Job Scheduling Strategies for Parallel Processing, pages 113–136, Cancun, May 2000. Lecture Notes in Comp. Sci. Vol. 1911, Springer-Verlag.

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

[PL95] J. Pruyne and M. Livny. Parallel processing on dynamic resources with carmi. In Proc. 1st Workshop on Job Scheduling Strategies for Parallel Processing, Santa Barbara, April 1995. Lecture Notes in Comp. Sci. Vol. 949, Springer-Verlag.

[Pla] Platform Computing Corporation, North York, Canada. LSF Scheduler. http://www.platform.com/.

[PS97] Eric W. Parsons and Kenneth C. Sevcik. Implementing multiprocessor scheduling disciplines. In Proc. 3rd Workshop on Job Scheduling Strategies for Parallel Processing, pages 166–192, Geneva, April 1997.

[RSD$^+$94] E. Rosti, E. Smirni, L. W. Dowdy, G. Serazzi, and B. M. Carlson. Robust partitioning policies of multiprocessor systems. Performance Evaluation (Special issue on the performance modeling of parallel processing systems), 19(2/3):141–165, March 1994.

[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.

[SE97] C. Severance and R. Enbody. Comparing gang scheduling with dynamic space sharing on symmetric multiprocessors using automatic self-allocating threads (asat). In Proc. 11th Int'l. Parallel Processing Symp., pages 288–292, Geneva, 1997.

[Set97] S. K. Setia. Trace-driven analysis of migration-based gang scheduling policies for parallel computers. In Proc. Int'l. Conf. Parallel Processing, pages 489–492, 1997.

[Sev89] K. C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. In Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, pages 171–180, May 1989.

[SGS96] J. Subhlok, T. Gross, and T. Suzuoka. Impact of job mix on optimizations for space sharing schedulers. In Proc. 1996 ACM/IEEE Supercomputing Conf., Pittsburgh, November 1996.

[SPWC98] P. G. Sobalvarro, S. Pakin, W. E. Weihl, and A. A. Chien. Dynamic coscheduling of communicating processes on workstation clusters. In Proc. 12th Int'l. Parallel Processing Symp., Orlando, March 1998.

[SRDS93] E. Smirni, E. Rosti, L. W. Dowdy, and G. Serazzi. Evaluation of multiprocessor allocation policies. Technical report, Vanderbilt University, Nashvile, 1993.

[SS99] F. A. Silva and I. D. Scherson. Concurrent gang: Towards a flexible and scalable gang scheduler. In Proc. 11th Symp. on Computer Architecture and High Performance Computing, Natal, Brazil, September 1999.

[SSN99] 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, March 1999.

[SST93] S. Setia, M. S. Squillante, and S. Tripathi. Processor scheduling on multiprogrammed, distributed memory parallel computers. In Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, 1993.

[ST91] S. Setia and S. Tripathi. An analysis of several processor partitioning policies for parallel computers. Technical Report CS-TR-2684, University of Maryland, May 1991.

[ST93] S. Setia and S. Tripathi. A comparative analysis of static processor partitioning policies for parallel computers. In Proc. Int'l. Workshop on Modeling and Simulation of Computer and Telecommunications Systems (MASCOTS), January 1993.

[SW95] P. G. Sobalvarro and W. E. Weihl. Demand-based coscheduling of parallel jobs on multiprogrammed multiprocessors. In Proc. 1st Workshop on Job Scheduling Strategies for Parallel Processing, pages 106–126, Santa Barbara, April 1995. Lecture Notes in Comp. Sci. Vol. 949, Springer-Verlag.

[SYZ99] M. S. Squillante, D. D. Yao, and L. Zhang. The impact of job arrival patterns on parallel scheduling. Performance Evaluation Review, 26(4):52–59, March 1999.

[TF99] David Talby and Dror G. Feitelson. Supporting priorities and improving utilization of the IBM SP2 scheduler using slack-based backfilling. In Proc. 13th Int'l. Parallel Processing Symp., pages 513–517, San Juan, April 1999.

[TG89] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In Proc. 12th ACM Symp. on Operating Systems Principles, pages 159–166, Litchfield Pk., December 1989.

[WLF$^+$96] K. Windisch, V. Lo, D. Feitelson, B. Nitzberg, and R. Moore. A comparison of workload traces from two production parallel machines. In Proc. 6th Symp. on the Frontiers of Massively Parallel Computation, pages 319–326, October 1996.

[WMKS96] M. Wan, R. Moore, G. Kremenek, and K. Steube. A batch scheduler for the Intel Paragon MPP system with a non-contiguous node allocation algorithm. In Proc. 2nd Workshop on Job Scheduling Strategies for Parallel Processing, pages 48–64, Honolulu, April 1996. Lecture Notes in Comp. Sci. Vol. 1162, Springer-Verlag.

[WPS97] F. Wang, M. Papaefthymiou, and M. Squillante. Performance evaluation of gang scheduling for parallel and distributed multiprogramming. In Proc. 3rd Workshop on Job Scheduling Strategies for Parallel Processing, Geneva, April 1997. Lecture Notes in Comp. Sci. Vol. 1291, Springer-Verlag.

[YJ01] A. B. Yoo and M. A. Jette. A dynamic coscheduling technique for symmetric multiprocessor clusters. In Proc. 7th Workshop on Job Scheduling Strategies for Parallel Processing, pages 36–47, Cambridge, MA., June 2001.

[YL95] K. K. Yue and D. J. Lilja. Loop-level process control: An effective processor allocation policy for multiprogrammed shared-memory multiprocessors. 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.

[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.

[ZJWB98] B. B. Zhou, C. W. Johnson, D. Walsh, and R. P. Brent. Job packing and re-packing schemes for enhancing the performance of gang scheduling. In Proc. 4th

Workshop on Job Scheduling Strategies for Parallel Processing, pages 129–143, Orlando, March 1998. Lecture Notes in Comp. Sci. Vol. 1459, Springer-Verlag.

[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.

[ZM90] J. Zahorjan and C. McCann. Processor scheduling in shared memory multiprocessors. In Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, pages 214–225, May 1990.

Appendix A

NCSA O2K Logs and Daemons

This appendix describes the information in the job logs on the O2K and how we extract the information from the logs.

A.1 The O2K Logs

Three types of job logs on the O2K are used for extracting the job information for analysis in this thesis. Table 16 summarizes the fields in the log that are used in our analysis. The fields marked by an asterisk (*) are further explained below. The actual log entries will be shown at the end of this section.

Each job may be suspended and resumed waiting for several times. Each of such events with the time when it occurs is recorded in the LSF event log. In our simulation, the submit time for such a job is the last time it is ready in the queue, but not the time it is submitted by the user. Only a very small fraction (< ?%) of the jobs have been suspended while waiting.

Very infrequently a job is suspended during execution. The total suspended time is recorded in the JMD log. The actual runtime of the job does not include the time the job has been suspended during execution.

To record the memory and cpu load usage, the JMD process for each job polls the information of the resource usage of the job every 30 seconds. The value of the cpu load is from 0 to the maximum number of processors; it is defined to be the total cpu time used divided by 30 seconds. The peak usage of memory and cpu load so far is updated when the peak changes. The frequency of the usage is reported in pairs of the value of the usage and time (in seconds).

Table 16. The Content of O2K Logs

Log	Field	Comments
LSF Accounting Log	job ID	assigned by LSF
	job name	often incomplete
	user name	the submitter
	job submission time	
	job dispatch time	
	job complete time	
	queue name	e.g., vst-mj
	resource requests string	#processors, memory & runtime can be derived
	execution host name	where the job was executed
LSF Event Log	suspended (while waiting)*	
	resumes pending*	
Per-Job JMD Log	total cpu time	
	total suspended time*	during execution
	memory used*	both peak and the frequency are recorded
	cpu load used*	
	killed time & reason*	if the job is killed

The values of the usage are chunked into bins, with the chunk size of 100 MB for the memory usage and 1 for the cpu load, e.g., a memory usage in the range of 500 ± 50 MB is counted for the bin with a value of 500 MB. As an example of the frequency of memory usage, "100 830 2000 540 4000 36000" means the job used 100 ± 50 MB for 830 seconds, 2000 ± 50 MB for 540 seconds, and 4000 ± 50 MB for 36000 seconds. The averaged memory usage reported in Section 4.5.2 is computed as the average usage over time, e.g., in the above example, the average memory usage for the job is 100 × 830 + 2000 × 540 + 4000 × 36000 divided by (830 + 540 + 36000).

The information of each job is extracted from these three job logs. and stored as one entry in the log for the month the job is submitted to. These new per-month logs are used for our later analysis and simulation.

Below further shows the actual data recorded in the three logs for an example job (with id 385173). The identity of the user and the project name is replaced by "xxx".
This job has the following entry in the LSF accounting log:

```
"JOB_FINISH" "3.2" 946539696 385173 23533 33816723 1 946481108 0 0 946486336 "xxx"
"st_mj" "rusage[npe=15:nmem=2000] order[npe] span[hosts=1]" "" "" "modi4"
"/scratch-res4/xxx" "" "htjob.o" "" "946481108.385173" 0 1 "huldra" 32 60.0
"1.0-0.15-300" "#!/bin/csh;#;#BSUB -n 15 # Specify 4 threads/processes;#;#BSUB -P
xxx # Charge job to project abc (recommended for users; # with multiple
projects);#BSUB -o htjob.o # Store the stan" 794591.549313 1339.050120 59808 0 -1 0 0
5683 62 0 7246 459 -1 282 399 41 10908 14 -1 "" "xxx" 2048 1 "" "" 0 887264 5806016
```

These are the log entries of the job in the LSF event logs (across two log files) in the order they are recorded, for the job submission, executing, status checked, and complete events.

1.
```
"JOB_NEW" "3.2" 946481108 385173 23533 33816723 1 946481108 0 0 -65535 0 0 "xxx"
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 "" 60.00 23 "st_mj" "rusage[npe=15:nmem=2000]
order[npe] span[hosts=1]" "modi4" "/scratch-res4/xxx/xxx/xxx/xxx/xxx" "" ""
```

```
"htjob.o" "" "/u/ncsa/xxx" "946481108.385173" 0 "" "" "1.0-0.15-300"
"#!/bin/csh;#;#BSUB -n 15 # Specify 4 threads/processes;#;#BSUB -P xxx #
Charge job to project abc (recommended for users; # with multiple projects);#BSUB -o
htjob.o # Store the stan" 0 "" "xxx" 1 "SGI02K" "" "" "" "" 16 0
```

2. "JOB_START" "3.2" 946486336 385173 4 0 0 60.0 1 "huldra" "" "" 0 "" 0

3. "JOB_START_ACCEPT" "3.2" 946486336 385173 3049622 3049622 0

4. "JOB_EXECUTE" "3.2" 946486336 385173 23533 3049622 "/scratch-res4/xxx" "/u/ncsa/xxx" "xxx" 3049622 0

5. "JOB_STATUS" "3.2" 946539696 385173 32 0 0 795930.6250 946539696 1 794591.549313 1339.050120 59808 0 -1 0 0 5683 62 0 7246 459 -1 282 399 41 10908 14 -1 2048 0

6. "JOB_CLEAN" "3.2" 946543473 385173 0

This is the actual JMD log file for the job:

```
IOChunk=100
IObChunk=10
IObread=1671219200
IObreadRateUse=0 53381
IObwrite=217267200
IObwriteRateUse=0 53381
IOread=136237269140
IOreadRateUse=0 53381
IOwrite=1436173386816
IOwriteRateUse=0 53381
cpuTime=795870.4709928
detectTime=946486359
```

```
domain=ncsa.uiuc.edu

finishTime=946539743

host=huldra

hostFactor=1

jmdGapTime=33

jmdVersion=1.171

jobId=LSBATCH:385173

jobName=1.0-0.15-300

limitCpuTime=2160000

limitMem=2048000

limitNpe=15

loadChunk=1

loadUse=12 36 15 53238 0 71 1 36

memChunk=100

memUse=800 36 900 53238 0 107

npeUse=15 53274 0 71 1 36

peakLoad=14.9028616902195

peakMem=880592

peakNpe=15

project=xxx

queue=st_mj

queueFactor=1.0

runTime=53407

scratchDir=/scratch/LSBATCH/385173.29Dec1999105216

startTime=946486336

submitTime=946481108
```

user=xxx

A.2 Processes For Monitoring O2K Jobs

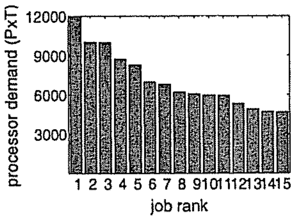
I've developed three processes that run on the interactive host (i.e., modi4) to monitor the system. The function of each process is briefly described below. The information produced by these processes is helpful for studying scheduler anomalies and check for the information of system downtime.

- A process that takes a snapshot of the scheduling state every two hours, including the currently executing jobs and the jobs in the waiting queues
- A process that records the current cpu load of each host every 1 minute in average (the actual interval between two recordings is generated by an exponential distribution with an average of 1 minute to simulate a random process).
- A process that checks the time stamps of the LSF configuration files every half hour, and if any configuration file has been changed, the new configuration is saved so that we have a copy of the LSF configuration after each change.

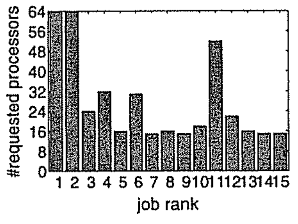
These processes die when modi4 is shutting down for whatever reasons. Upon such an event, an email notification from modi4 is sent to a list of people, including me. I restart the processes later when modi4 comes back on line.

Appendix B

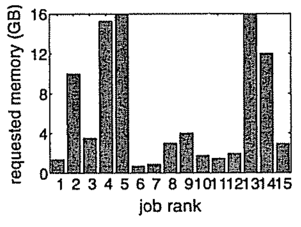
Characteristics of Fifteen Largest Jobs For Additional Months



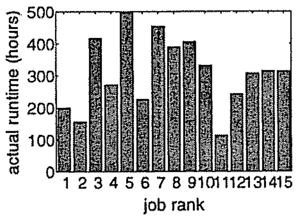
(a) P×T hours



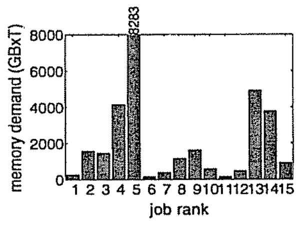
(b) Requested Processors



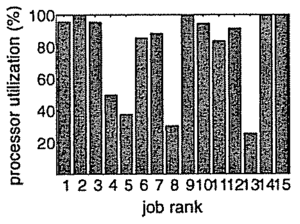
(c) Requested Memory



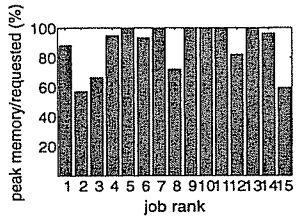
(d) Actual Runtime



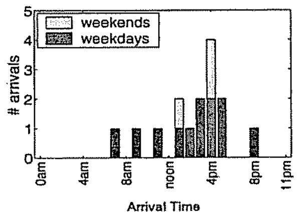
(e) M×T hours



(f) Processor Utilization

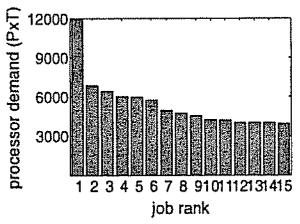


(g) Peak Memory Efficiency

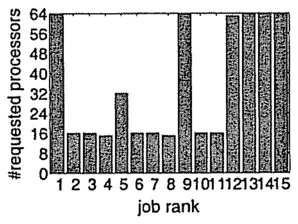


(h) Arrival Times

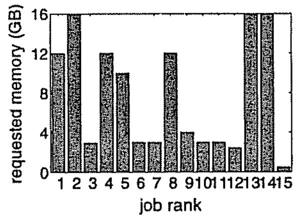
Figure 83. Characteristics of the Fifteen Largest Jobs (January 2000)



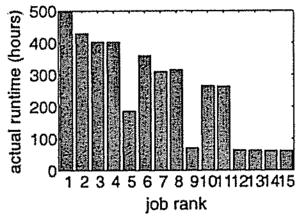
(a) P×T hours



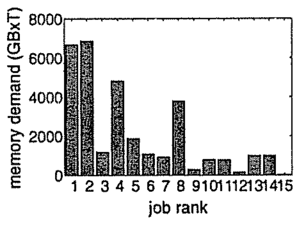
(b) Requested Processors



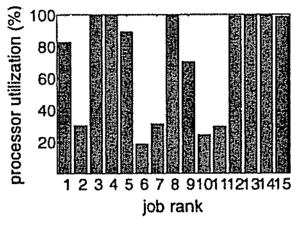
(c) Requested Memory



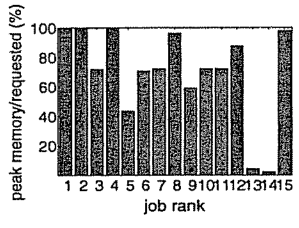
(d) Actual Runtime



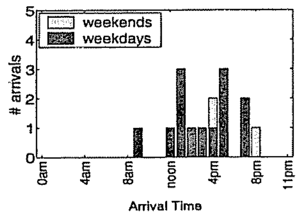
(e) M×T hours



(f) Processor Utilization

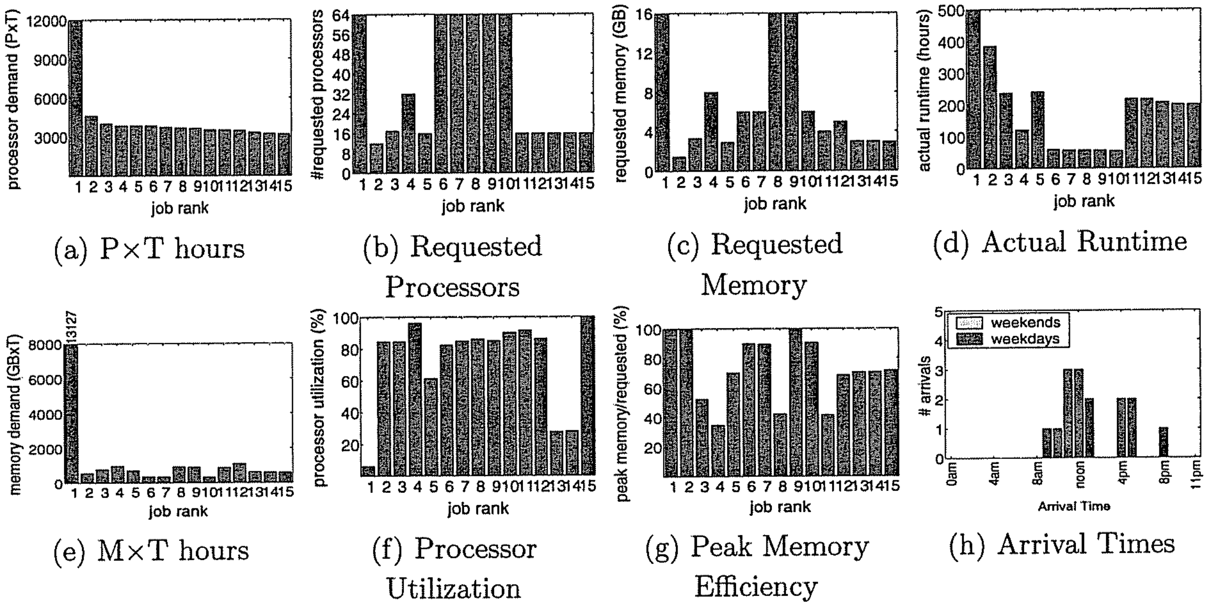


(g) Peak Memory Efficiency



(h) Arrival Times

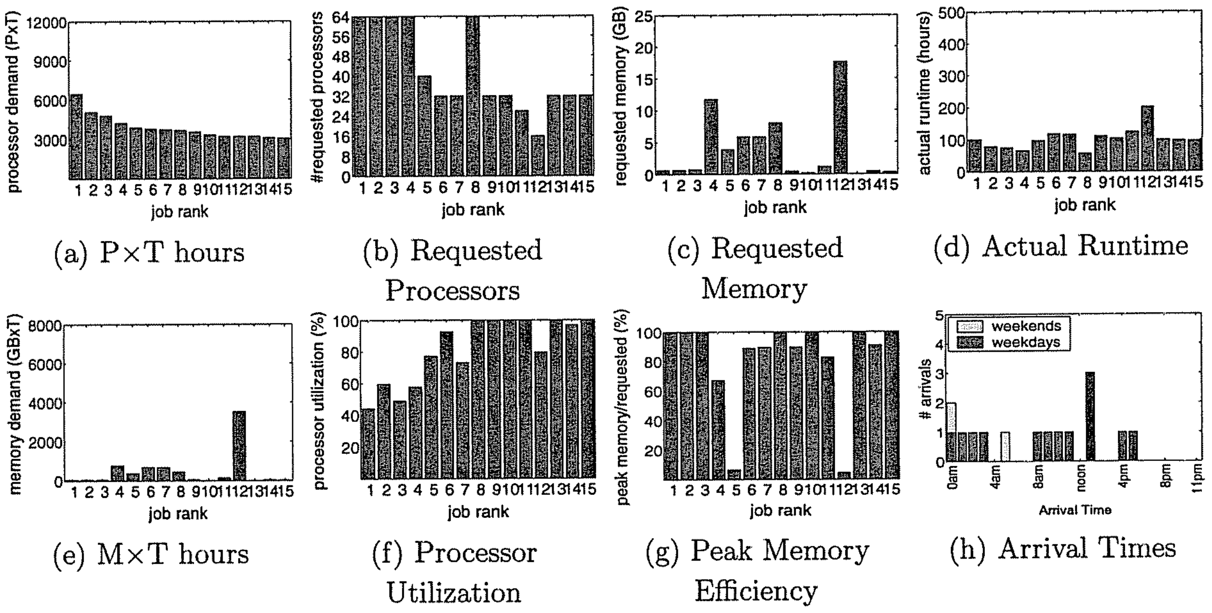
Figure 84. Characteristics of the Fifteen Largest Jobs (Feburary 2000)



(a) P×T hours (b) Requested Processors (c) Requested Memory (d) Actual Runtime

(e) M×T hours (f) Processor Utilization (g) Peak Memory Efficiency (h) Arrival Times

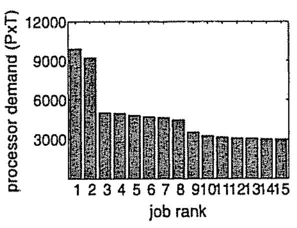
Figure 85. Characteristics of the Fifteen Largest Jobs (March 2000)



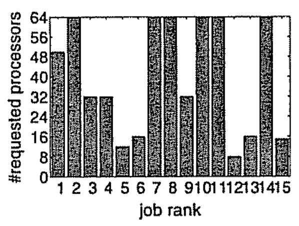
(a) P×T hours (b) Requested Processors (c) Requested Memory (d) Actual Runtime

(e) M×T hours (f) Processor Utilization (g) Peak Memory Efficiency (h) Arrival Times

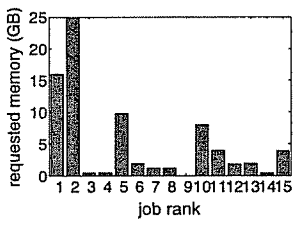
Figure 86. Characteristics of the Fifteen Largest Jobs (October 2000)



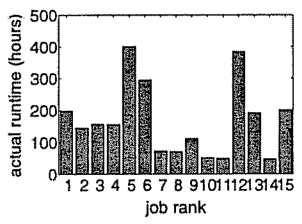
(a) P×T hours



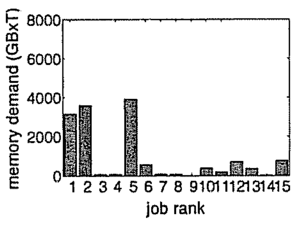
(b) Requested Processors



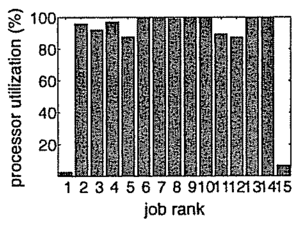
(c) Requested Memory



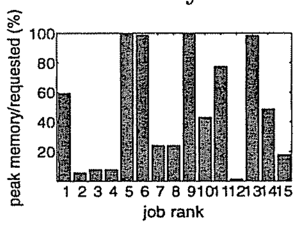
(d) Actual Runtime



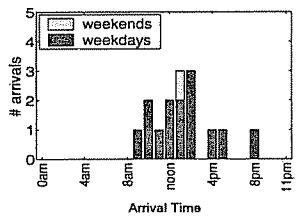
(e) M×T hours



(f) Processor Utilization

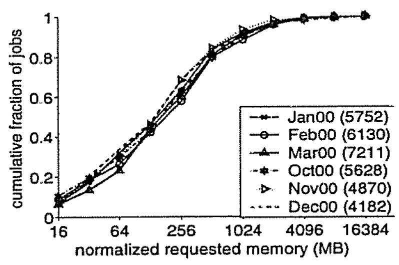


(g) Peak Memory Efficiency

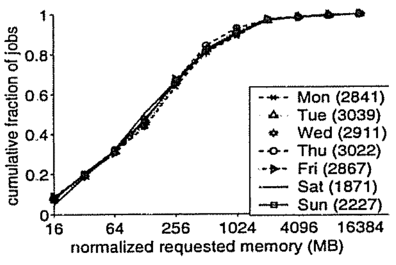


(h) Arrival Times

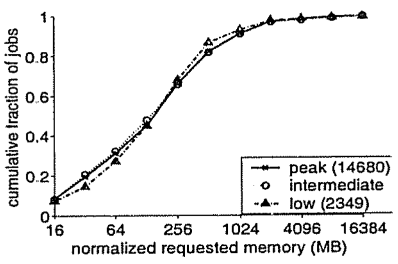
Figure 87. Characteristics of the Fifteen Largest Jobs (November 2000)



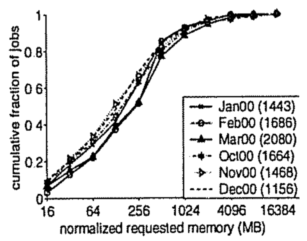
(a) Each Month Peak Period



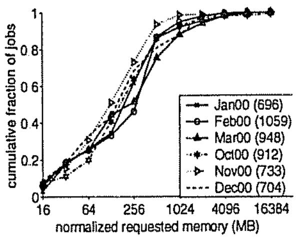
(b) Each Day of the Week Peak Period Oct-Dec 2000



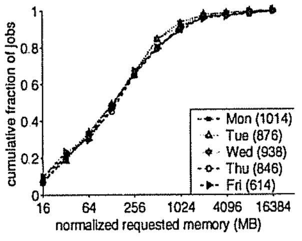
(c) Each Arrival Period Weekdays Oct-Dec 2000



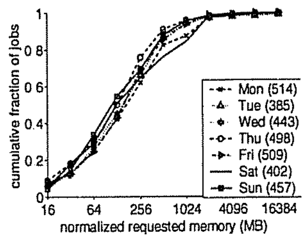
(d) Intermdiate Each Month



(e) Low Each Month



(f) Intermediate Period Each Day of Week (Oct-Dec 2000)



(g) Low Period Each Day of Week (Oct-Dec 2000)

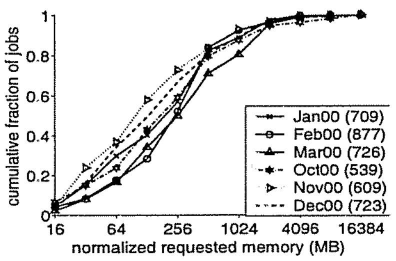
Figure 88. Variation in Distribution of Normalized Requested Memory During Weekdays

(Normalized Requested Memory = Requested Memory/#Requested Processors)

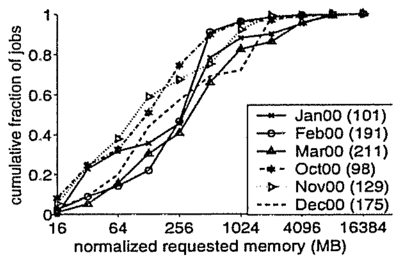
Appendix C

Additional Workload Characterization Graphs

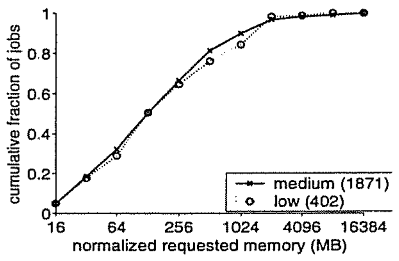
Figures 88-89 provide additional graphs to show the variation of the distribution of normalized requested memory across different periods.



(a) Intermediate Period Weekends Each Month



(b) Low Period Weekends Each Month



(c) Each Weekend Period Oct-Dec 2000

Figure 89. Variation in Distribution of Normalized Requested Memory During Weekends

(Normalized Requested Memory = Requested Memory/#Requested Processors)

Appendix D

Anomalies in the LSF Scehduler

As mentioned in Section 3.6, the LSF scheduler sometimes schedule the jobs in violation of the rules, which has caused some discrepancy in our simulated results and the results analyzed from the log.

Specifically, we found three types of scheduling rules that have been violated under the LSF scheduler. Type 1 is the rule for the class priority, e.g., the mj jobs are started on a host that currently has the priority for the lj jobs and there are lj jobs that are waiting in the queue and can be scheduled. Type 2 is the rule for the class limits (i.e., per-host mj and lj limits and system-wide mt and lt limits), e.g., the mj jobs are started on a host on which the mj job limit has been reached. Type 3 is the rule that ensures the processors on each host are not oversubscribed. Some jobs have been scheduled in violation of this rule. Table 17 provides an example of the number of jobs that have been scheduled in violation of each rule, during each given period.

Table 17. Scheduling Violations Under the LSF Scheduler

	#Jobs Scheduled in Violation of Rules							
	Class Priority		Class Limit				Processor	Memory
Period	sj	mj	mj	lj	mt	lt	Limit	Limit
6/1-6/7	79	2	6	0	37	3	0	0
6/8-6/14	110	0	14	9	0	4	0	1
6/24-7/6	55	5	1	0	0	0	1	0
7/7-7/12	11	1	2	0	1	0	2	0

Appendix E

NCSA-LSF* Tuning Results

This appendix provides example LSF tuning results or suggestions based on our simulation study.

E.1 Tuning Per-Queue & Per-Host Job Limits

Prior to June 1998, there were limits on the number of running jobs in each queue and on each host. Table 18 shows Our study showed that a large fraction of the waiting time is due to per-host limit and per-queue limit (for popular queues), shown for an example week in March 1998. We thus suggested to remove these job limits, which were implemented immediately.

Table 18. Job Waiting Time Due To Each Job Limit Under Early LSF

Queue	Number	Wait Time (hrs)		Job Limit		
	Jobs	Avg	Sum	User	Queue	Host
st_sj	**251**	1.6	397.0	3.8%	**62.2%**	14.9%
st_mj	94	0.9	81.5	5.0%	0.0%	**51.5%**
mt_sj	**110**	7.4	809.1	8.6%	**65.7%**	10.3%
mt_mj	59	7.8	458.3	0.0%	0.3%	**67.0%**
mt_lj	12	6.3	75.6	0.0%	0.0%	**59.8%**
lt_sj	19	10.6	200.7	0.0%	1.0%	**56.6%**
lt_mj	40	25.7	1026.6	8.2%	9.8%	26.0%
lt_lj	10	36.6	366.5	0.0%	0.0%	16.3%
All	897	3.9	3510.3	6.0%	25.2%	28.2%

(Period for analysis: Week 1, March 1998)

E.2 Tuning Class Job Limits

E.3 Suggestions for Immediate Service

Table 19 shows the 10-th, 20-th, and 50-th percentiles of waiting time for each LSF queue that have an average cpu time limit of over 5 hours and that for all space shared jobs for an example week in March 1998. The key observation is that a significant fraction of these jobs used a very small total cpu time (< 1 minute). Thus, we suggested to give each new job a short immediate service if it can't start right away. The idea is the same as the immediate service we propose for policies with backfill in Section 5.5. However, the preemption has never been implemented on the O2K so far.

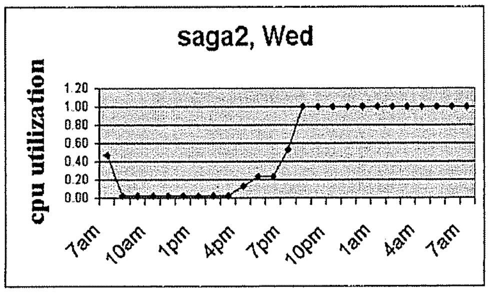
Table 19. Per-Job Total CPU Time In LSF Queues

		Total CPU time (hours) Percentiles		
Queue	**#Jobs**	10-th	20-th	50-th
st_sj	249	0.0	0.0	3.1
st_mj	93	0.0	0.3	4.7
mt_mj	59	0.1	4.2	23.6
mt_lj	12	0.0	0.0	0.5
lt_sj	19	0.0	1.4	28.8
lt_lj	10	0.0	0.0	5.0

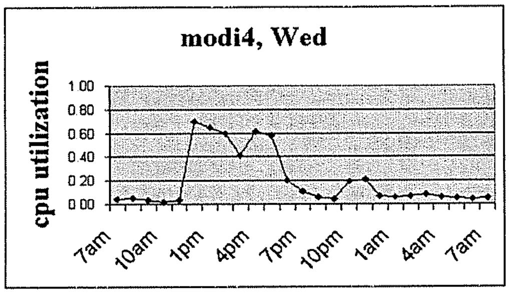
(Period for analysis: Week 2, March 1998)

E.4 Suggestions for Utilizing Dedicated and Interactive Hosts

Figure 90 shows the cpu utilization on a then dedicated host (saga2) and the interactive host, i.e., modi4 for an example weekday day in May 1998. It shows that these hosts have low utilization sometimes. Thus, we suggest that these hosts can run space-shared jobs when it's nearly idle. The space-shared vst classes were created to run for short jobs (under 5 hours) and were allowed to use dedicated hosts if no dedicated jobs are available for scheduling.



(a) Dedicated Host



(b) Interactive Host

Figure 90. CPU Utilizaation on LSF Dedicated and Interactive Hosts

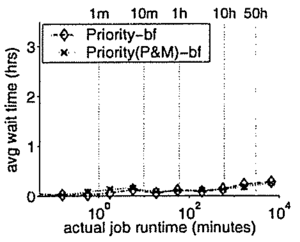
(an example day in May 1998)

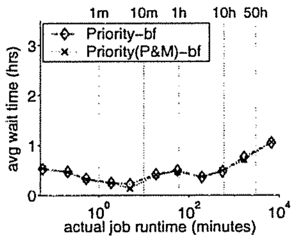
Appendix F

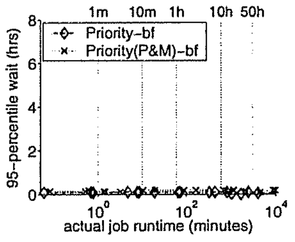
Additional Performance Results for Backfill Policies

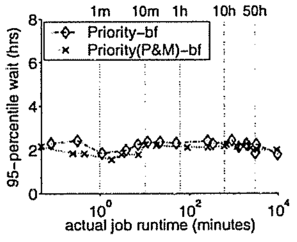
Figure 91 shows the average, 95th percentile, and maximum waiting time versus actual runtime for the two typical load months (October 1999 and January 2000) that are not shown in Figure 91.

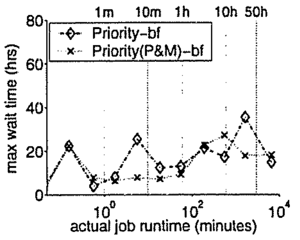
Figures 92-93 provide the results for the impact of number of reservations and fixed-job versus dynamic-job reservation on the performance of LXF&W-backfill.

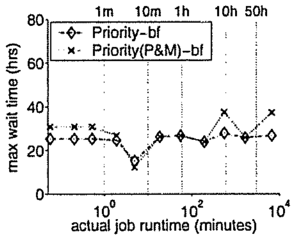

(a) Avg Wait (Oct 1999) (typical load)

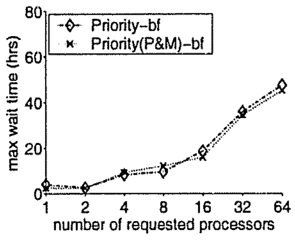

(b) Avg Wait (Jan 2000) (typical load)


(c) 95-percentile Wait (Oct 1999) (typical load)


(d) 95-percentile Wait (Jan 2000) (typical load)


(e) Max Wait (Oct 1999) (typical load)


(f) Max Wait (Jan 2000) (typical load)


(g) #Processors Dec 1999

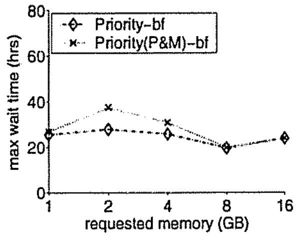
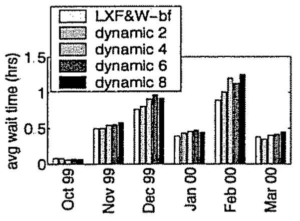
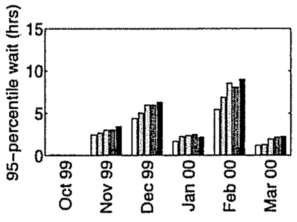
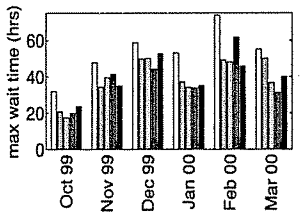

(h) Requested Memory Jan 2000

Figure 91. Performance of Priority-backfill and Priority(P&M)-backfill versus Actual Runtime: Additional Results


(a) Avg Wait


(b) 95th Percentile Wait


(c) Max Wait

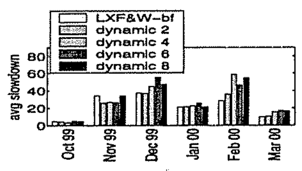
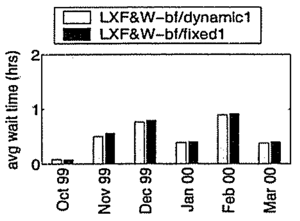
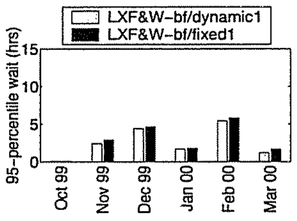
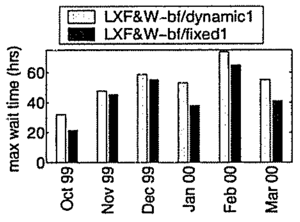

(d) Avg Slowdown

Figure 92. Impact of Number of Reservations On LXF&W-backfill (Dynamic-job)


(a) Avg Wait


(b) 95th Percentile Wait


(c) Max Wait

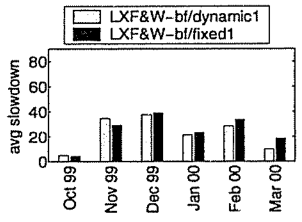

(d) Avg Slowdown

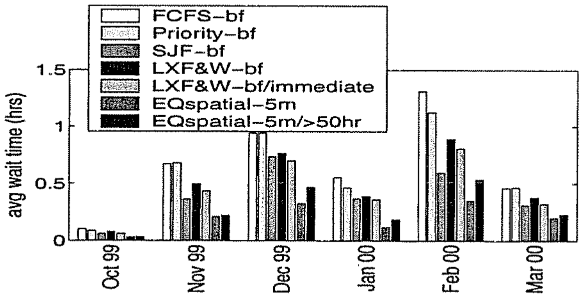
Figure 93. Fixed vs. Dynamic Reservation for LXF&W-backfill

(One Reservation)

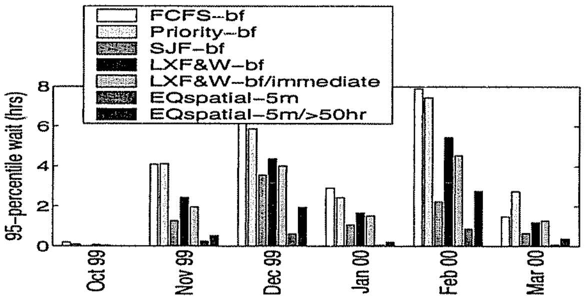
Appendix G

Performance Comparisons of Key Policies

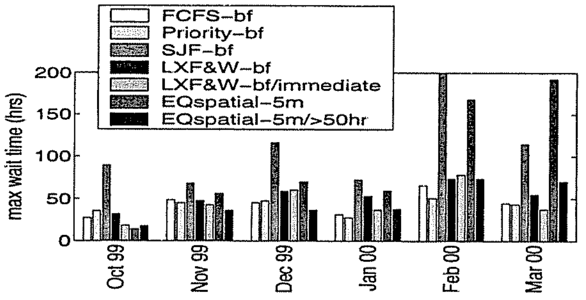
Figure 94 summarizes the performance of the key policies evaluated in Chapters 5-6, using O2K requested runtime. They are two previously proposed FCFS-backfill and SJF-backfill; two new priority backfill policies: Priority-backfill and LXF&W-backfill; LXF&W-backfill with immediate service; and dynamic EQspatial-5m and EQspatial-5m/>50hr.



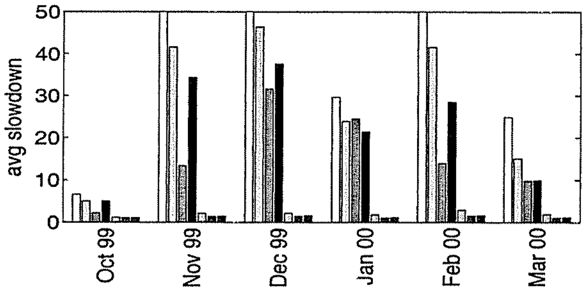
(a) Avg Wait



(b) 95th Percentile Wait



(c) Max Wait



(d) Avg Slowdown

Figure 94. Overall Performance of Priority Backfill and EQspatial Policies