Impact of Job Mix on Optimizations for Space Sharing Schedulers

Jaspal Subhlok
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213
jass@cs.cmu.edu
http://www.cs.cmu.edu/~jass

Thomas Gross
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213
and
Institut fuer Computersysteme, ETH Zuerich, 8092 Zuerich, Switzerland

Takashi Suzuoka
Toshiba Corporation, 1, Komukai, Toshiba-Cho, Saiwai-Ku, Kawasaki 210, Japan

Abstract Modern parallel systems with N nodes can concurrently service multiple jobs requesting a total of up to to N nodes. One of the challenges for the operating system is to give reasonable service to a diverse group of jobs. A sequence of large jobs, each requiring over half of the available nodes, can reduce the machine utilization by up to 50%, but scheduling a long running job on the idle nodes may block the stream of large jobs. Various policies have been proposed for scheduling parallel computers, but as the users of current supercomputers know, these policies are far from perfect. This paper reports on the measurement of the usage of a 512-node IBM SP2 at Cornell Theory Center, a 96-node Intel Paragon at ETH Zurich, and a 512-node Cray T3D at Pittsburgh Supercomputing Center. We discuss the characteristics of the different workloads and examine their impact on job scheduling. We specifically show how two simple scheduling optimizations based on reordering the waiting queue can be used effectively to improve scheduling performance on real workloads. Supercomputer workloads from different installations exhibit some common characteristics, but they also differ in important ways We demonstrate how this knowledge can be exploited in the design and tuning of schedulers.

Keywords backfilling, job characteristics, job scheduling, scheduling optimizations, space sharing, space slicing, supercomputer workloads

Appears in Proceedings of Supercomputing '96
Further information and copies available from: http://www.cs.cmu.edu/~fx

1 Introduction

The focus of parallel processing research is fast execution of jobs. However, for expensive parallel supercomputers, a job often spends more time in the waiting queue than actually executing. Since the turnaround time of a job is the only figure of merit that matters to a user, it is important to not only improve the execution time of individual jobs but also the waiting time and the overall system performance. In this paper we analyze how large parallel computers are used and show how the service provided by them can be controlled and improved with simple job scheduling optimizations.

Large commercial parallel systems support dynamic partitions and time sharing, although not all sites choose to operate their system that way; instead, space sharing is the preeminent approach used today. Most commercial systems can support a variety of scheduling policies [Fei94] and they typically allow the system manager to tune a scheduling policy with a set of parameters. However, our experience shows that the actual scheduling policies used are not always the best way to use a machine. Often the nodes of a parallel machine are divided into fixed partitions for different kinds of jobs, which places artificial constraints on scheduling. The goal of this research is to evaluate different scheduling optimizations using actual workloads, thus forming a rational basis for designing and making the best use of scheduling systems. Our actual experimentation uses trace data from a 512-node IBM SP2 at Cornell Theory Center, a 96-node Intel Paragon at ETH Zurich, and a 512-node Cray T3D at Pittsburgh Supercomputing Center.

The choice of a scheduling policy is difficult for two main reasons; the best policy is dependent on the job mix, and the scheduler must balance efficient machine usage against fairness in scheduling. We discuss these two issues separately.

The job mix has a high impact on the effectiveness of a scheduling policy. Distribution of the number of nodes, the execution times, and the job arrival rates, are all important aspects of a job mix. Since the job mix can change dynamically, and typical job queue characteristics are not well understood, characterizing the job mix is a difficult issue. We use data from three supercomputer installations to identify the common features of job mixes, as well as to understand how they vary. We analyze the relationship between the job mix characteristics and the choice of scheduling mechanisms. We present results for the actual job submission rate, as well as for scenarios in which jobs are submitted uniformly faster or slower. By using actual traces to drive such scenarios, we can simulate heavier and lighter workloads than those actually observed, which approximate a change in the user population under the assumption that the user behavior, in terms of job mix, remains the same.

A scheduler must achieve efficient use of the machine and fairness in scheduling. In practice, these two objectives conflict yet they cannot be achieved independently. For example, a single job using all nodes and executing for a long time uses the machine optimally, but allocating all nodes to this job may not be the most reasonable way to schedule a machine if this choice results in many small jobs having to wait for a long time. We use separate criterion for efficiency and fairness to evaluate scheduling optimizations and examine the tradeoffs between them.

Scheduling optimizations evaluated in this paper are for a space sharing system where available nodes are divided among the executing jobs and a job continuously executes on a group of nodes until completion. Jobs do not share nodes and jobs are not swapped between executing and waiting states. The optimizations discussed are based on reordering the waiting queue for improving performance. Time sharing systems and systems that allow job swapping are certainly important but beyond the scope of this paper. We limit ourselves to discussing the added complexity of simulating time sharing with real traces.

This paper is organized as follows. Section 2 presents and characterizes data on the usage of 3 supercomputer installations. Section 3 discusses how job characteristics influence scheduling policy. Section 4 defines our criterion for measuring the merits of a scheduling policy and explains how workload traces can be used to evaluate scheduling optimizations. Section 5 studies the effectiveness of a set of simple scheduling optimizations on different job mixes. Section 6 contains conclusions.

2 Measurements of supercomputer usage

To develop and validate practical scheduling algorithms for parallel supercomputers, it is important to understand how these computers are used. For this purpose, we processed and studied the log files of three supercomputer installations: a 512-node IBM SP2 at Cornell Theory Center, a 96-node Intel Paragon at ETH Zurich[1], and a 512-node Cray T3D at Pittsburgh Supercomputing Center. These installations represent 3 major architectures for large parallel machines and are used by a large and diverse group of academic and commercial users.

The usage pattern that develops at a supercomputer installation depends on a large number of factors and may change over time. Technical factors include available memory, distribution of memory among nodes, scalability characteristics of applications, and stability of the machine. For example, some jobs are memory-bound and use the smallest number of nodes that satisfy memory requirements, and some jobs do not realize performance improvements beyond a certain number of nodes as communication and overheads begin to dominate. The usage is also influenced by administrative factors like the billing system and priority structure for different types of jobs and users. Finally, biases in the scheduling policy and the total amount of usage affect the nature of the job stream. For example, increased usage of the machine, or a change in scheduling policy, may increase the turnaround time for larger jobs, thus discouraging users from submitting larger jobs. However, the actual usage patterns are still the best target for evaluating scheduling optimizations, and in the rest of this paper we attempt to understand and use them without paying further attention to the reasons that may have led to a particular pattern.

We would like to point out that a full understanding of the usage of each of these parallel machines requires a detailed study (see, e.g., [FN95, HSO96, Hot96, WLF$^+$96]). However, the purpose of this section is to focus on those aspects of supercomputer usage that reveal patterns across multiple installations, with the objective of employing this commonality of behavior to select, develop, and validate scheduling algorithms and optimizations.

IBM SP2 at Cornell Theory Center was primarily using LoadLeveler for scheduling for the duration in which the data was collected, although it has since been replaced by EASY-LL scheduler [SCZL96]. The Intel Paragon at ETH primarily uses NQS for scheduling. The usage presented only represents the jobs scheduled by NQS and LoadLeveler respectively, since the usage data is obtained from the scheduler logs. While there are other ways of submitting jobs to these machines, it is known that most of the "production" usage of these machines is through these schedulers. The Cray T3D at Pittsburgh Supercomputing Center is scheduled with NQS running on the Cray C90 front end with additional scheduling control on the T3D. The data presented represents the entire

[1]This machine was recently upgraded to 150 nodes, but for this paper, we limit the analysis to the 96-node system.

usage of this machine.

The usage data for all machines represents 3 or more months of use. The Paragon and the SP2 allow a job to use any number of nodes, while the T3D allows jobs to use only power-of-2 nodes. The specific scheduling policies and parameters used on these machines (e.g., setting of priorities) are not presented as they are complex and not directly relevant to this research. We do not intend to compare specific scheduling policies used at different installations, but only to characterize the behavior of job streams, and use these characteristics to derive constraints and consideration for scheduling optimizations.

2.1 Job sizes and machine usage

We first examine the distribution of the jobs in terms of number of nodes and machine usage, for the three machines. In the rest of this paper we will use "small" and "large" to refer to the size of a job in terms of number of nodes, and "short" and "long" to refer to the execution time of a job.

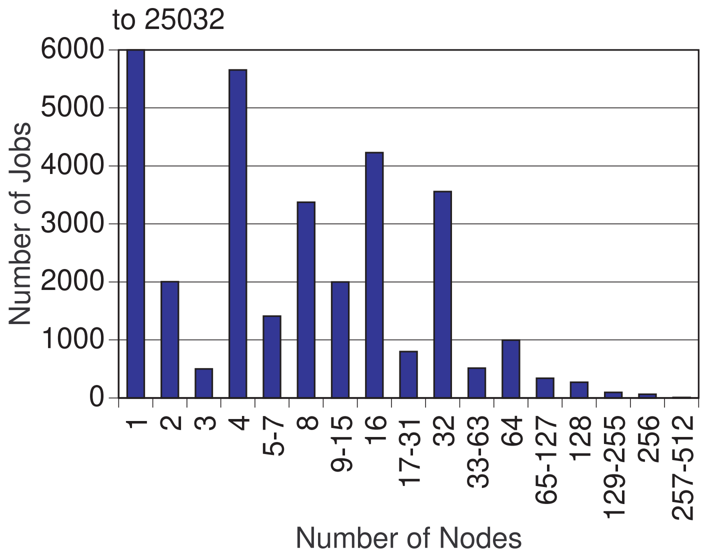
Figure 1 shows the distribution of the number of nodes for the job stream. Figure 2 shows machine usage for jobs of different sizes. In all cases, the number of nodes actually used to execute a job is the same as the number of nodes requested. As stated earlier, the SP2 and the Paragon allow from 1 to the number of available nodes to be requested, while the T3D allows only power-of-2-nodes. Further, due to architectural reasons, the T3D allocates a minimum of two nodes per job.

We first observe on Figure 1 that a fairly large number of jobs use a small number of nodes, although the percentages vary significantly. On the SP2 and the T3D, well over half the jobs use 4 or fewer nodes. The percentage is relatively small on the Paragon (possibly because many of the single node jobs do not use NQS). There are various reasons for a large number of small jobs, including sequential jobs that use the machine to take advantage of a large memory or a software package, debugging runs, and jobs run by the maintenance staff.

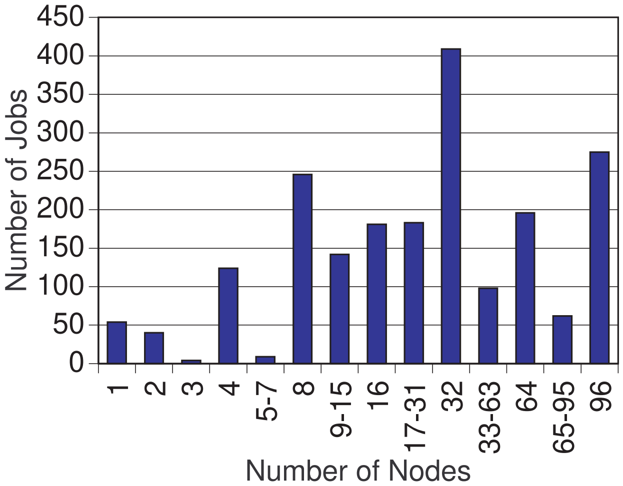
Figure 2 shows the machine usage (or cycles served) measured as the product of the execution time of the job and the number of nodes used by the job. This figure shows that the actual machine usage by small jobs is relatively small. On the Paragon and the T3D, machine usage by jobs using 4 or fewer nodes is negligible, while it is around 11 percent for the SP2 [2]. Similar results are reported for the usage of an Intel iPSC 860 machine at NASA [FN95]. The general conclusion is that a significant number of jobs may use only a few nodes, but these jobs consume a relatively small part of the compute resources.

Another interesting observation is that the numbers of nodes tend to be clustered around powers of 2 for the SP2 and the Paragon. (The T3D permits only power-of-2 numbers of nodes.) A special case is that a large number of jobs use all 96 nodes of the Paragon. For jobs sizes larger than 64, the usage of the SP2 is almost evenly divided between power-of-2 size jobs and other jobs. We speculate that the users tend to use a power of 2 number that matches closely to their needs, and this becomes more difficult for a much larger SP2 as compared to the smaller Paragon. The general conclusion is that the usage of the machine tends to be clustered at powers-of-2 numbers of nodes, although other jobs can account for a significant part of the usage.

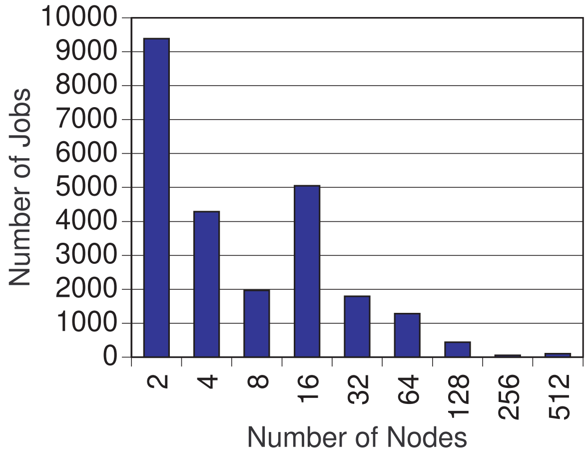
[2] The relatively heavy usage of the SP2 by single node jobs can be attributed to two factors. Many users use the SP2 for single node programs because of the large memories available on some of the SP2 nodes. Also, there is a range of software available on the SP2 at Cornell Theory Center which users may not be able to afford on their workstations.



512-node IBM SP2 (CTC)[3]



96-node Intel Paragon (ETH)

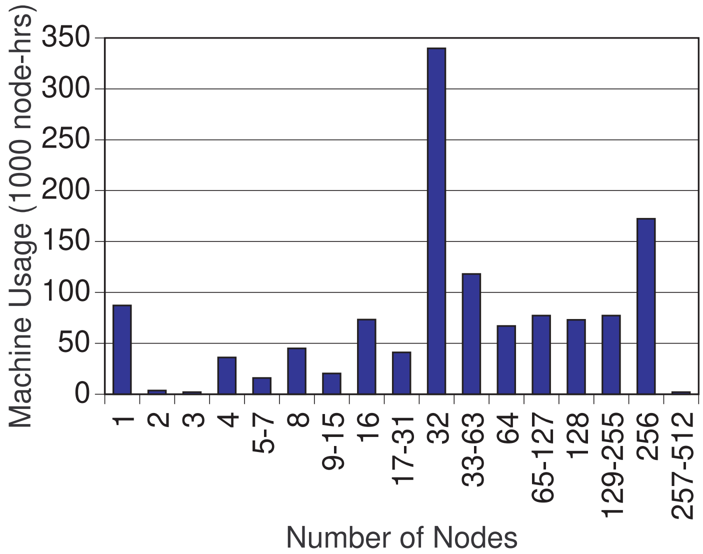


512-node Cray T3D (PSC)

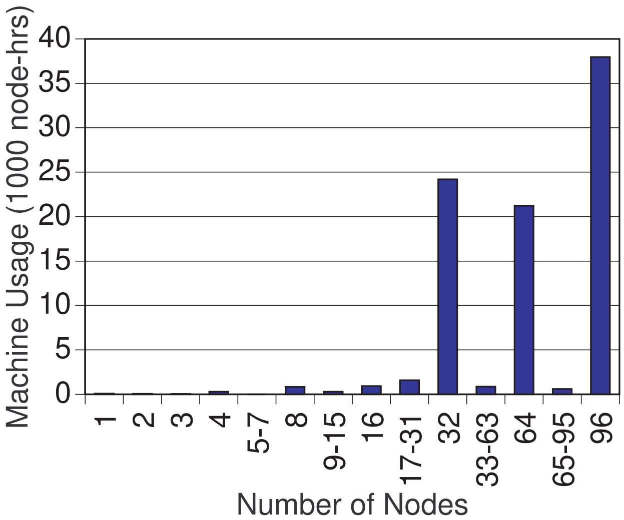
Figure 1: Distribution of job sizes.

Finally, the usage of the machines appears to be dominated by medium size jobs. On the Paragon, job sizes between 32 and 96 account for about 95 percent of the usage, while on the T3D, job sizes between 32 and 128 account for about 82 percent of the total usage. On the SP2, 32-node jobs account for about a third of the total usage, and the bulk of the usage is widely distributed in the range from 1 to 256 nodes[3]. Thus, these machines are primarily being used as shared resources by a small set of moderate sized jobs, rather than being dominated by large or small jobs.

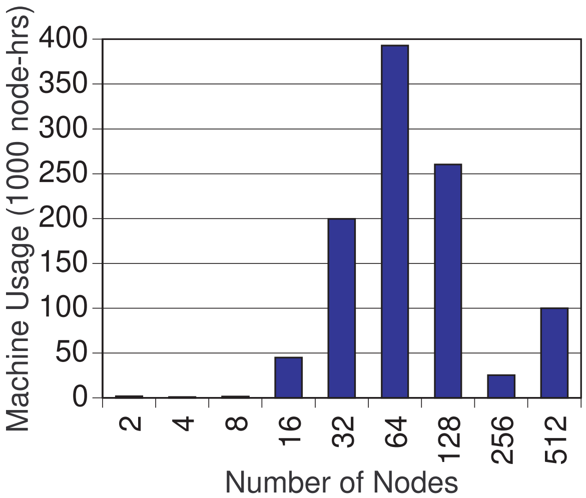
[3]We wish to point out that the usage reported for the SP2 at Cornell Theory Center does **not** reflect the current usage pattern at this installation. The CTC workload data was collected for the LoadLeveler scheduler and prior to the installation of the EASY-LL scheduler which provides backfilling. EASY-LL is much more effective for scheduling large jobs than LoadLeveler, and this encouraged the submission of larger jobs. Recent reports from this center clearly indicate that the SP2 is being used far more heavily by large (250+ node) jobs now and achieving very good machine utilization.



512-node IBM SP2 (CTC)[3]



96-node Intel Paragon (ETH)



512-node Cray T3D (PSC)

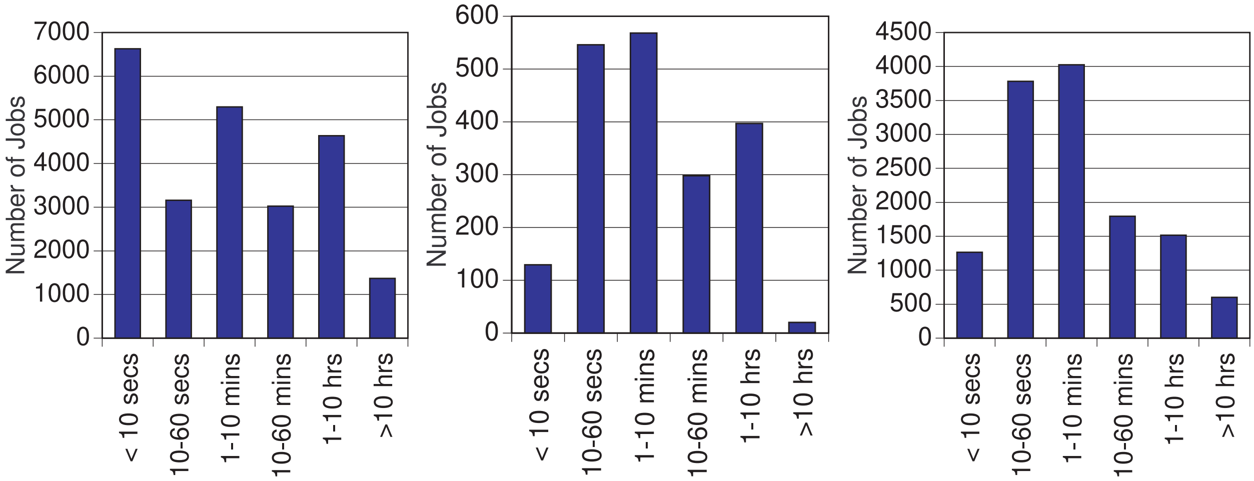
Figure 2: Machine usage for different job sizes.

2.2 Job execution times and machine usage

We next examine the distribution of the actual execution times of the jobs and see what fraction of the machine resources are used by “short running” and “long running” jobs. Figure 3 shows the distribution of the execution times for the job stream, while Figure 4 shows the machine usage by jobs in different ranges of execution times.

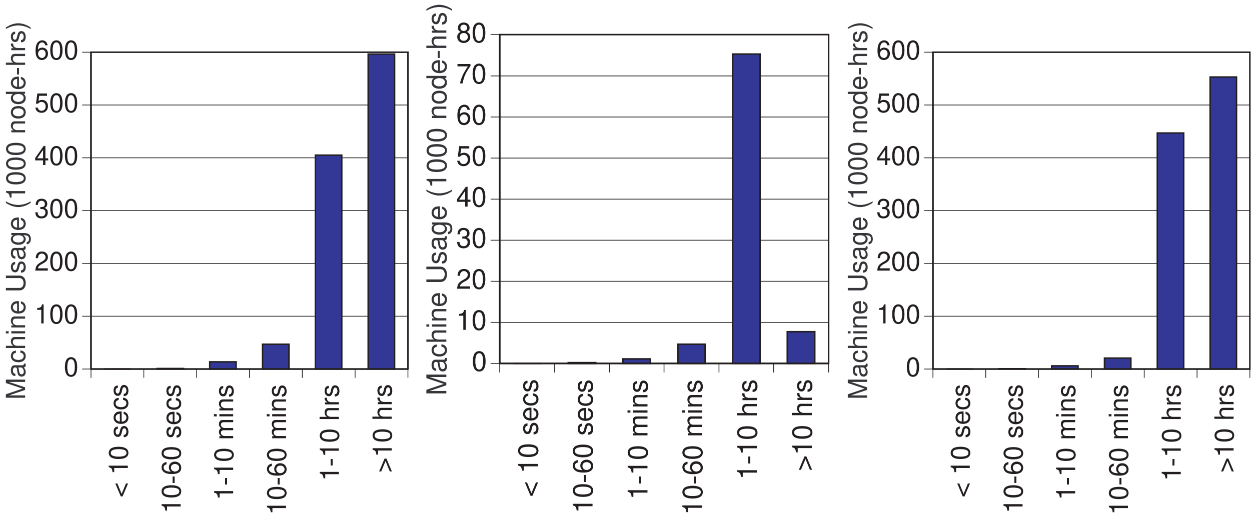
We note that on all machines, a large fraction of the jobs run for under a minute, and an overwhelming majority of the jobs run for less than 10 minutes. Reasons for a short execution time include test jobs and jobs that encounter a runtime error soon after startup. Also, a significant number of jobs run for over 1 hour. Approximately, the number of jobs appear to decrease exponentially with the execution time.

However, the machine usage plotted in Figure 4 shows that the small jobs use an extremely



512-node IBM SP2 (CTC) 96-node Intel Paragon (ETH) 512-node Cray T3D (PSC)

Figure 3: Distribution of job execution times.



512-node IBM SP2 (CTC) 96-node Intel Paragon (ETH) 512-node Cray T3D (PSC)

Figure 4: Machine usage for different job execution times.

small part of the overall machine resources. All jobs running for less than 1 minute use a negligible amount of machine resources, and jobs shorter than 10 minutes account for less than 2 percent of the machine resources. This is certainly not surprising since the machine usage is proportional to the running time. The general conclusion is that there is a large number of short running jobs, but the usage is completely dominated by the longer running jobs.

3 Implications of job characteristics for scheduling

We summarize the main observations about job characteristics from the previous section and then discuss how they impact scheduling decisions.

1. There may be a large number of number of jobs that use a few nodes but the corresponding machine usage is small. (It is somewhat higher for the SP2 at CTC than for the other systems.)

2. For the larger machines (SP2 and T3D), machine usage by large jobs (using more than a third of all nodes) is relatively small.

3. Well over half of the machine usage is by jobs that use a power-of-2 number of nodes. In the case of Paragon, which has a total of 96 nodes, there is also significant machine usage by 96-node jobs.

4. Well over half the jobs execute in less than 10 minutes, but these jobs account for less than 2 percent of machine usage. Medium to long running jobs dominate machine usage.

The fact that machine usage is dominated by medium size jobs, and not by jobs that need all or most of the nodes, underlines the importance of effective algorithms to share the machine in space sharing or time sharing mode.

The dominant use of power-of-2 numbers of nodes is useful information, because several scheduling techniques discussed in the literature make the assumption that jobs must execute on a power-of-2 number of nodes [LC91, CT94, DB93]. These techniques may be broadly applicable but they must allow for node numbers that are not powers of 2.

The fact that a large fraction of jobs use only a few nodes and run for a short time suggests that such jobs could be scheduled preferentially without significantly affecting other jobs. It also points to the importance of good execution time estimates to identify short running jobs during scheduling. This observation also suggests that time sharing between short running and long running jobs may be a good strategy. The summary of implications for the space and time sharing schemes is as follows:

- A pure space sharing scheme should pay special attention to the fact that many jobs use only a few nodes, and many jobs run for a short time. Such a scheme should avoid the situation where one small job blocks a large job from getting scheduled, leaving wasted nodes, as well as the situation where one large job prevents a large number of small jobs from being scheduled.

- A time sharing scheme should be effective in obtaining good turnaround times for short running jobs, which constitute an overwhelming majority of submitted jobs, without adversely affecting long running jobs significantly.

4 Evaluation of job scheduling

We state the criterion for evaluation of job scheduling and then discuss how workload traces can be used to evaluate scheduling algorithms in space sharing and time sharing environments.

4.1 Evaluation criterion

A scheduler's main objective is to schedule jobs for efficient performance and fairness. However, success or failure in meeting these objectives can be measured in several different ways. For example, performance can be measured as average turnaround time, average waiting time, or average queue length. We will not discuss the merits of these different criteria but merely state and briefly justify the criteria that we use for performance and fairness.

Performance

For this study we use average turnaround time as the primary measure of performance since this measure affects the user of a parallel computer most directly. The average waiting time is directly related to the average turnaround time (the difference being the average execution time) and is also shown and discussed when appropriate. We do not report system utilization even though it is of prime importance to supercomputer installations. Part of the reason for its omission is that system utilization is hard to measure accurately from traces in the presence of (partial and complete) system downtimes, and gaps and other inconsistencies in trace data. However, average turnaround time correlates well with system utilization in a normal operating environment.

Fairness

A scheduler can execute jobs in an order that is different from the first-come–first-served, to improve throughput, and such scheduler actions lead to unfairness. It is difficult to have a general criterion for fairness since a particular scheduling policy may favor and disfavor specific types of jobs. Instead of having a set criterion for fairness, we deal with it on a case-by-case basis. In general, if a scheduling policy favors certain kinds of jobs over others, we separately measure the performance of the different kinds of jobs to understand the extent of unfairness.

It is also important to be able to bound the maximum unfairness and ensure that a job is not indefinitely blocked. For this purpose, we define **M**aximum **A**llowable **S**kipping **C**ount (MASC), which is the maximum number of jobs that can pass a job in the waiting queue (i.e., if MASC=10 then at most 10 jobs arriving after a particular job can be scheduled before it). Most space sharing scheduling algorithms can be modified to operate with a given MASC, although at a potentially reduced performance. Whenever it is important, we perform simulations with different values of MASC to study the impact of fairness control over performance.

4.2 Simulation of space sharing scheduling

In pure space sharing scheduling, a job requests a fixed number of nodes on arrival, and the scheduler finds and allocates the nodes, possibly after the job has been in a waiting queue. The job then executes on the assigned nodes until completion. The nodes are not shared and the jobs are never suspended during execution.

To simulate space sharing scheduling algorithms and optimizations, we only need to know the arrival times and execution times of jobs. In most modern parallel machines, the physical location of the nodes on which a job executes does not significantly affect the execution, and hence the logged execution time can be taken as the execution time for simulations. All scheduling optimizations evaluated in this paper are for a space sharing model.

4.3 Simulation of time sharing scheduling

Accurately simulating scheduling policies in a time sharing environment is much harder than dealing with space sharing. Time sharing requires a knowledge of the memory requirements of the jobs and a knowledge of the context switching time. The memory usage is often not available in the job traces and it is difficult and cumbersome for users to provide a good estimate of memory requirements. The time to context switch between jobs in memory, and the time to store or restore the state of a job to/from secondary storage, is even harder to estimate. These problems can perhaps be resolved in a reasonable way with better monitoring, but we do not report any results from time sharing simulations.

5 Impact of scheduling optimizations

We examine the benefits of a set of simple space sharing scheduling optimizations based on rearrangement of the waiting queue. This section is based on the job arrival data for the IBM SP2 at Cornell Theory Center and the Intel Paragon at ETH. The purpose of these experiments is to provide guidance to the designer of a scheduling system about the value of these scheduling optimizations.

5.1 Simulation setup

We first describe our basic model for the incoming job stream. The system has no advance knowledge about when a job arrives and how many nodes it needs until the job actually arrives. A job requests a fixed number of nodes on arrival and does not request additional nodes at runtime. The system does not know when a job will finish execution, although we present results from a set of simulations that assume advance knowledge of execution time. This model of job streams is typical for supercomputer installations.

We use logs from the supercomputer installations to generate the job arrival stream. We include only jobs that were scheduled and completed normally. For a variety of reasons, the logs are not sufficient to completely recreate the load experienced by a particular installation. For example, the SP2 at the Cornell Theory Center was occasionally down for maintenance, and some of the nodes were reserved for special projects and thus not available for batch processing. Also, the users can explicitly request nodes with specific resources like large memories, but this aspect is ignored in our simulation. The Paragon at ETH is regularly switched to a mode where it accepts only interactive jobs that are not scheduled with NQS and therefore do not show up on the NQS logs. Also, some of the log files were missing or corrupted, leading to gaps in the simulation sequence.

We use the actual job stream, but simulate scheduling also for uniformly smaller and larger inter-arrival intervals. A smaller interval corresponds to a higher arrival rate. The figures in this section use a “relative” arrival interval where a value of 1 nominally corresponds to the arrival intervals of the actual trace. This setup models varying machine loads. We compare scheduling with different optimizations to a reference space sharing scheduling policy in which the jobs are scheduled in order as soon as enough nodes are available.

5.2 Sorting the waiting queue

Since a large number of small jobs (i.e., jobs requiring a small number of nodes) can be executed in the time it takes to execute a single large job, the average service to a diverse group of jobs can be improved by preferentially executing the smaller jobs. If the waiting queue is frequently sorted with the smaller jobs placed in the front, the smaller jobs are favored. This optimization can also improve the utilization of the machine when there is a large waiting queue, as smaller jobs are more likely to fit together on the machine. Note that this optimization is different from the sorting algorithm that places the larger jobs in front [LC91].

Our workloads exhibit a large number of small jobs in the job mix. Hence we expect sorting of job queue to improve the overall throughput and turnaround time. An important problem with sorting is that it is unfair to larger jobs: if the waiting queue is always sorted, then a large job can be blocked indefinitely. To prevent this kind of situation from occuring, a sorting algorithm can be modified to include a MASC (maximum skipping count), which is the maximum number of jobs that can overtake a job in the waiting queue.

We first study the benefits of sorting the waiting queue over a simple "first-come–first-served" scheduling for the SP2 workload. Figure 5 compares the average turnaround times without sorting, the average turnaround times with sorting, and the average turnaround times with constrained sorting where the maximum skipping count (MASC) is controlled to ensure a degree of fairness. We also plot the constant average execution time (labeled "No Wait"), so that the average waiting time can also be observed. (If the average execution time line is considered to be the horizontal axis, then the plot represents the average waiting time.)

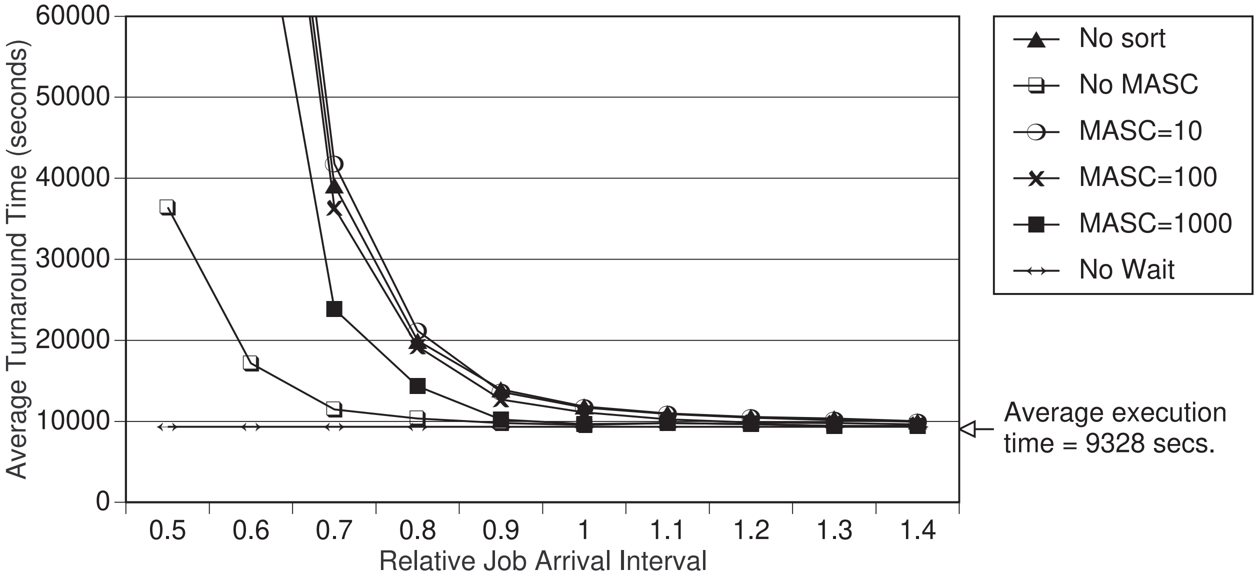


Figure 5: Impact of using a sorted waiting queue on performance for the SP2 workload.

We observe a significant improvement in the average turnaround time with unconstrained sorting for job arrival rates corresponding to high workloads (i.e., small values for the relative job arrival interval). The system is able to provide reasonable average turnaround time for much heavier workloads than would be possible without sorting. It is not surprising that there is a more pronounced improvement for higher job arrival rates, since the higher arrival rate implies longer waiting queues and therefore more opportunities for sorting.

Figure 5 also shows that limiting the maximum skipping count significantly compromises the benefits of sorting. Using MASC=10 actually causes the performance to be worse for a range of the data, and a MASC more than 100 must be used to get a significant benefit from sorting.

We now take a closer look at the sorting results. First we focus on the range of the curve that corresponds to a moderate load and small to moderate average waiting time, which we consider to be the most interesting part of the load spectrum. Since sorting intuitively favors smaller jobs, we categorize the jobs as small, medium, and large. Jobs using 16 or less nodes are considered small, jobs using 128 or more are considered large, and others are medium. About 87% of all jobs qualify as small while less than 1% are large. We examine the impact of sorting on small jobs in Figure 6, and on large jobs in Figure 7.

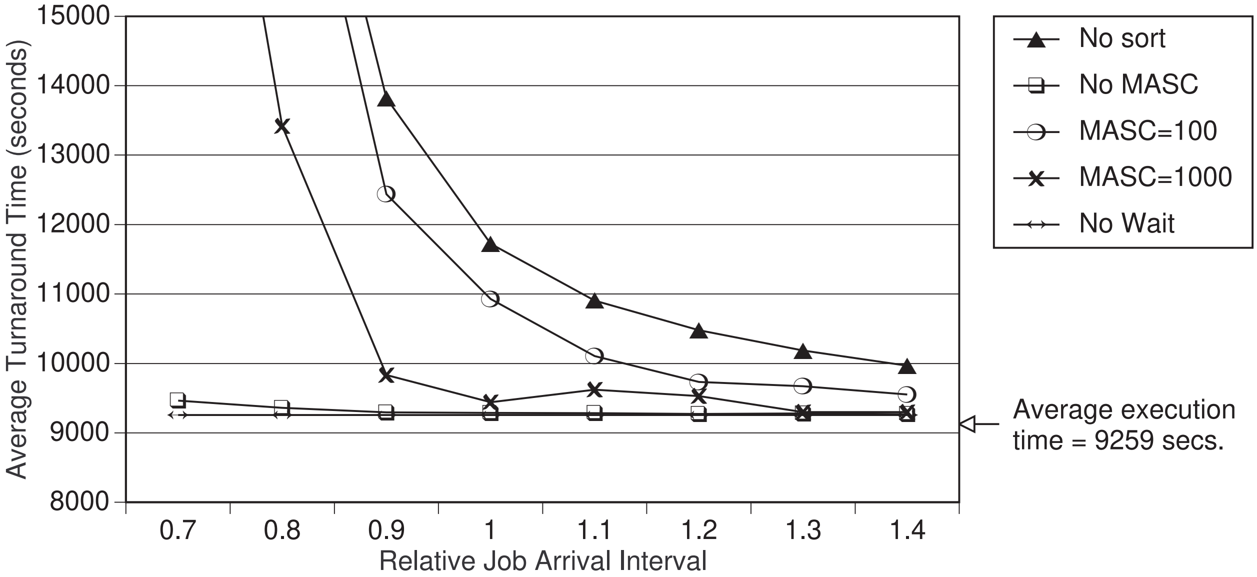


Figure 6: Impact of using a sorted waiting queue on small jobs. All jobs using 16 or less nodes are considered small. 87% of all jobs are small.

We observe that unrestricted sorting ("No MASC" curve) practically eliminates the waiting time for small jobs in this range. However, the average waiting time for large jobs increases rapidly for smaller job arrival intervals, and is a factor of 3 higher in the case of a relative job arrival interval of 1. We now focus on the curve for sorting with MASC=100. The average waiting time for small jobs appears to improve significantly, ranging from about 50% for slower arrival rates to about 30% for faster arrival rates. Conversely, the average waiting time for large jobs increases by about 50% to 25% in the same range. However, this appears to be a good tradeoff since over 87% of the jobs are small and less than 1% of the jobs are large.

The impact of sorting on the workload represented by the Paragon trace is shown in Figure 8. We notice that the improvement in turnaround time with unrestricted sorting is significant for heavier workloads but smaller than the improvement obtained for the SP2. Also, fixing a lower MASC has less impact on the benefits of sorting; well over half of the maximum possible improvement is obtained with MASC = 100 for most of the curve. The curve for MASC = 1000 is not shown since

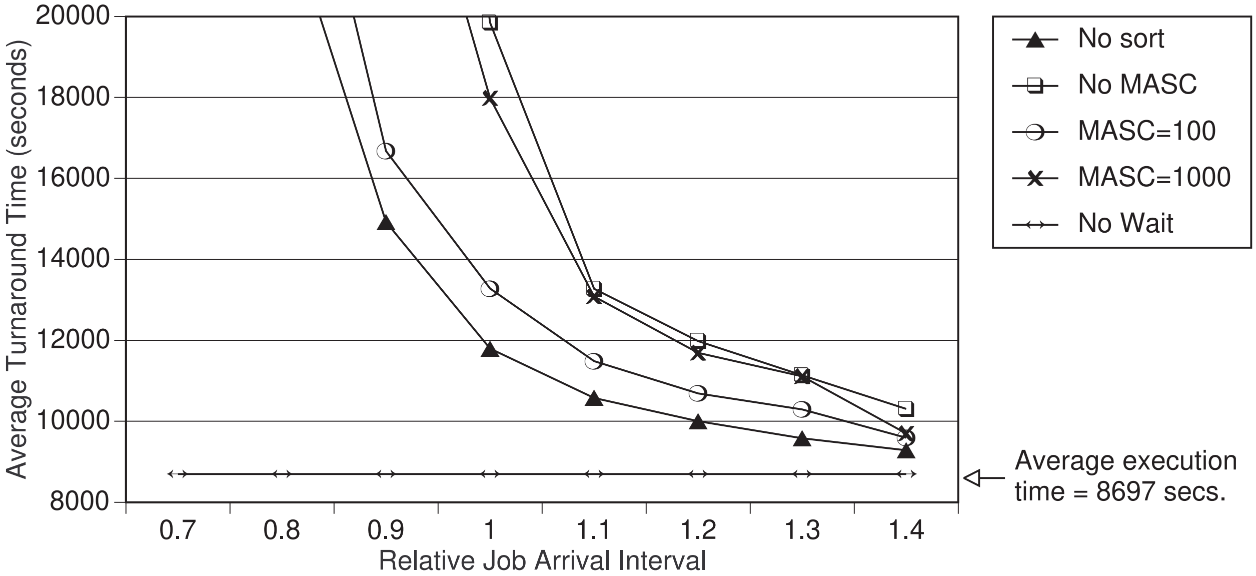


Figure 7: Impact of using a sorted waiting queue on large jobs. All jobs using 128 or more nodes are considered large. Less than 1% of all jobs are large.

it is identical to that for unrestricted sorting.

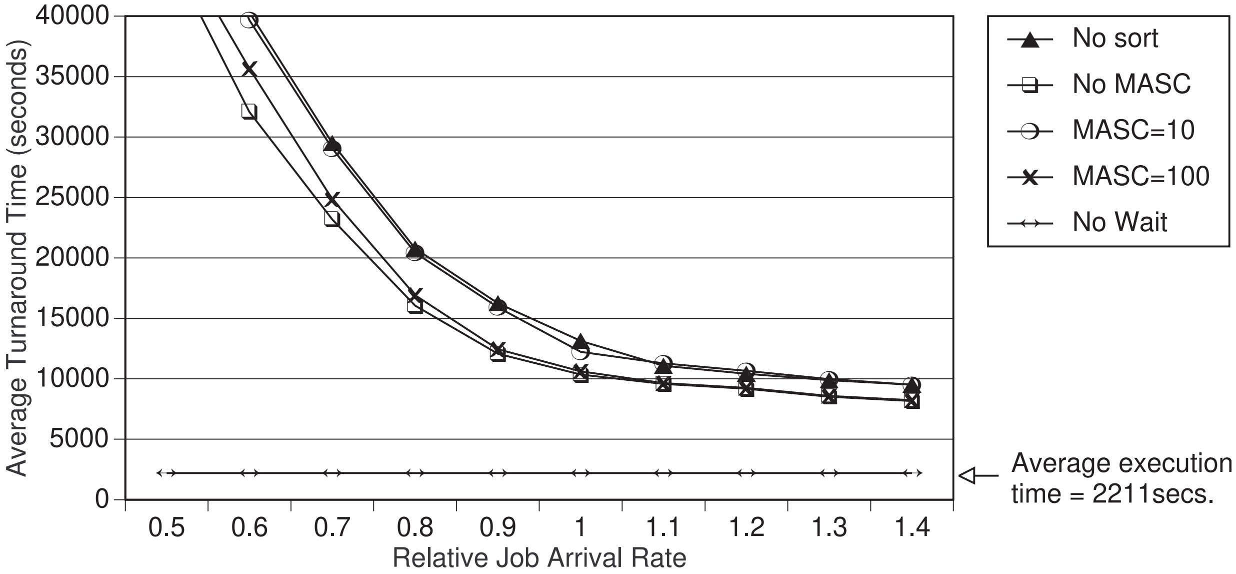


Figure 8: Impact of using a sorted waiting queue on performance for the Paragon workload.

5.3 Backfilling

When a large job is blocked at the head of the waiting queue because enough nodes are not yet available, it may be possible to execute a smaller job in the queue without affecting the start time of the large job. This technique is called “backfilling”. It has been in use at the IBM SP2 installation at the Argonne National Laboratory [LHR95] and is being installed with a new scheduler on the IBM SP2 at the Cornell Theory Center. Backfilling assumes that the execution times of jobs are known ahead of time. In a simple backfilling system, the jobs in the waiting queue are examined as candidates for backfilling starting from the head of the queue. The first job that can be scheduled without changing the times at which the jobs ahead of it will get scheduled, is moved to the head of the queue and scheduled immediately. This form of backfilling never "increases" the turnaround time of any job.

Our traces show that supercomputer workloads generally contain a large number of jobs that use only a few nodes and execute for a short time. It should be possible to promote many of these jobs in the waiting queue by backfilling without affecting other jobs. Hence we expect this technique to be effective in improving the performance of scheduling.

The effect of backfilling on the workloads from the traces from the SP2 and the Paragon are shown in Figure 9 and Figure 10, respectively. The figures show that backfilling leads to a small improvement in the average turnaround time for the SP2, and a somewhat larger improvement for the Paragon.

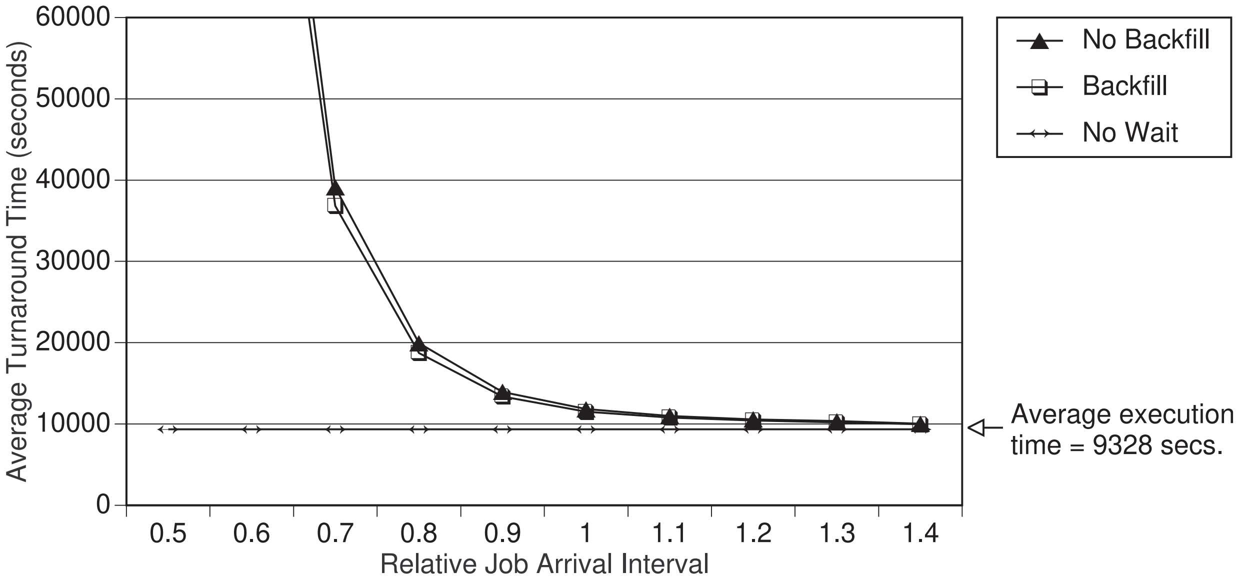


Figure 9: Impact of backfilling on performance for the SP2 workload.

Backfilling primarily improves the service for the jobs with relatively small machine usage. The change in the average turnaround time may not give a good indication of the importance of backfilling since the average is dominated by long running jobs which get smaller benefits from backfilling.

We now focus on the impact of backfilling on jobs with a low machine usage. For this purpose we define an "easy" job as a job that requires less than 4 node-hours of machine usage. By this definition, 69% of the jobs on the SP2 and 75% of the jobs on the Paragon are easy. We also look specifically at the improvement in the waiting time instead of turnaround time, since backfilling (or any other scheduling optimization) can improve only the waiting time and not the execution time. The percentage reduction in the average waiting time for easy jobs for the SP2 and the Paragon are shown in Figure 11.

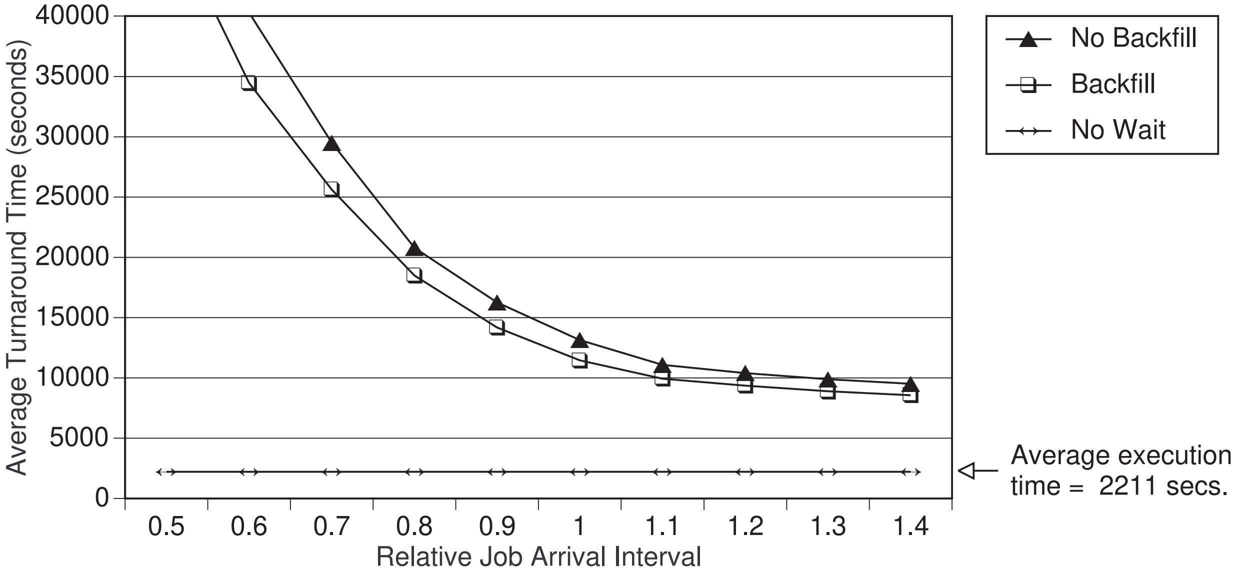


Figure 10: Impact of backfilling on performance for the Paragon workload.

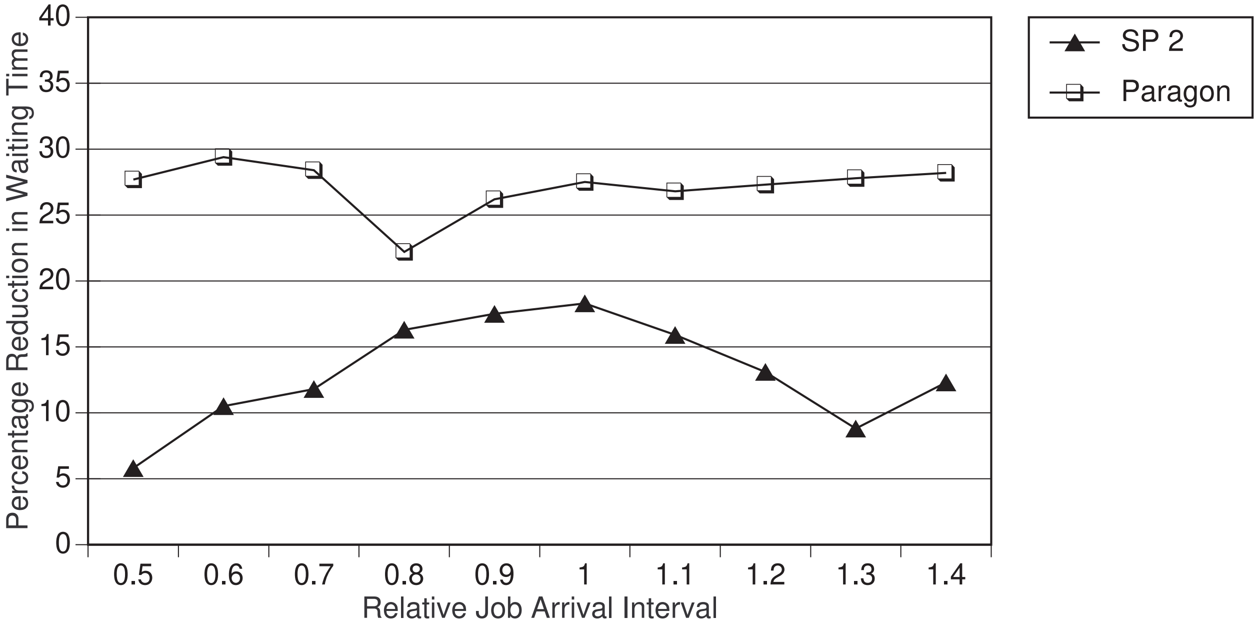


Figure 11: Impact of backfilling on jobs with small resource requirements, or "easy" jobs. A job using less than 4 node-hours of total computation is considered easy.

We note that backfilling leads to a clear improvement in the waiting time for easy jobs. For the SP2, the improvement is around 15% for the range of arrival intervals that are close to the actual job arrival rate. For the Paragon, the corresponding improvement is in the range of 25-30%. Note that this improvement is measured only for the majority of jobs considered easy, but all jobs potentially benefit from backfilling. We conclude that backfilling is an effective optimization for job scheduling as it significantly helps a large fraction of the jobs and does not adversely affect any job.

We should caution that these simulations assumed that the execution times of all jobs were known ahead of time, which is virtually impossible. However, in related work we have observed that reasonable approximations to the actual execution time can be effective in obtaining the benefits of backfilling. On a practical note, preliminary experience with EASY-LL scheduler [SCZL96] at Cornell Theory Center has been very positive, in part because the workload appears to adapt to make the best use of backfilling.

5.4 Relationship between job mix and optimization

We first analyze the impact of the two scheduling optimizations on the IBM SP2. The average turnaround time on this machine improves only slightly with backfilling and sorting with a small MASC. It improves dramatically with unrestricted sorting when the job arrival rate is high.

The workload on the 512-node IBM SP2 is dominated by jobs that have a small to medium size in relation to the total number of nodes. This mix makes it easier to utilize the machine efficiently, as smaller jobs pack better than large jobs. Therefore, the performance using a simple first-come–first-served scheduling discipline is expected to be fairly good which makes it harder to obtain further improvements. This is the reason that the benefits of sorting with a low MASC, as well as backfilling, are relatively small, despite the fact that a large number of small jobs can benefit from these optimizations.

When the job arrival rate is high for the SP2, use of unrestricted sorting results only in smaller jobs being scheduled, while the large jobs get indefinitely blocked. This behavior results in a dramatically improved turnaround time, mainly because delaying a few large jobs means that many small jobs can execute, which reduces the overall average waiting time.

In contrast, the workload on the 96-node Paragon is dominated by medium to large size jobs, which can be easily blocked in a simple first-come–first-served scheme even when several nodes are idle. This results in poor machine utilization, which in turn causes longer turnaround times. Sorting is beneficial as it prevents the smaller jobs from being blocked behind larger jobs. Backfilling is even more useful as it allows the smaller jobs to fit into holes that are created when a stream of larger jobs (but smaller than the full machine size) are to be scheduled in sequence. As stated earlier, backfilling is also helped by the fact that a large percentage of jobs use a small amount of machine resources and are therefore easy to promote by backfilling.

6 Concluding remarks

We have analyzed the job arrival and execution time data from 3 supercomputer installations and used these traces to understand common and distinguishing features of supercomputer workloads, and to drive simulations of space sharing scheduling policies. While many scheduling policies and optimizations have been discussed in the literature, their validation is based on simple assumptions about the arrival and execution time patterns of the job stream. Our entire experimentation is based on the job traces collected at three sites: a 512-node IBM SP2 at Cornell Theory Center, a 96-node Intel Paragon at ETH Zurich, and a 512-node Cray T3D at Pittsburgh Supercomputing Center. While there is no way to predict the workload that a parallel computer will encounter in any particular time period, it is fair to claim that the actual workloads observed on a variety of parallel machines over several months constitute a good model for validating the practical value of job scheduling optimizations.

The main contribution of this paper is to establish that the usage patterns of large parallel systems can be used to guide the design and evaluation of scheduling policies. We show that sorting the waiting queue to enable shorter waiting time for smaller jobs is effective, but a delicate balance must be found between fairness to larger jobs and overall performance. Backfilling can significantly improve the service for jobs that need a small amount of machine resources, without any cost to other jobs. However, backfilling requires that the execution times of jobs be known in advance, and this underlines the importance of obtaining good execution time estimates.

The main conclusion of this paper is not a definite recommendation about the best scheduling policy. The results of this paper, however, give a clear indication about the kinds of benefits that can be expected from sorting and backfilling if the broad characteristics of the job stream are known. Our results, along with a knowledge of expected workload characteristics, can also be used as a starting point for developing more sophisticated scheduling policies, or for choosing among a wide variety of known scheduling optimizations.

Installations of the current generation of large parallel machines are often partitioned to separately service jobs with different characteristics. However, the best possible use of a machine can be achieved only if the full machine is treated as a single shared resource, but it is a difficult challenge to manage a machine effectively and efficiently in this mode. This paper attempts to provide a rational basis for the design and evaluation of scheduling policies that manage all nodes of a parallel system. The results demonstrate that practical scheduling tradeoffs can be effectively managed by a single global scheduler.

7 Acknowledgements

We thank Michael Vollmer of Intel SSD and Bruno Loepfe of ETHZ for collecting the information at the Paragon installed at ETH Zurich, Switzerland. Donna Bergmark and Steve Hotovy at Cornell Theory Center collected and explained the trace data for the SP2. Chad Vizino, Sergiu Sanielevici, Janet Brown, and Juan Leon of Pittsburgh Supercomputing Center helped us collect and understand the usage information for the Cray T3D. The simulations used in the results were done by Kanda Runapongsa of CMU. Jonathan Betz and Rawesak Tanawongsuwan of CMU, and Joep Rottinghuis of ETH also contributed to the analysis of the usage logs. Finally, the referees made several good suggestions to improve the paper, although we could implement only some of them due to limited time.

This research is sponsored in part by the Advanced Research Projects Agency/CSTO monitored by SPAWAR under contract N00039-93-C-0152.

References

[CT94] P. Chuang and N. Tzeng. Allocating precise submeshes in mesh connected systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5(2):211–217, Feb. 1994.

[DB93] J. Ding and L. Bhuyan. An adaptive submesh allocation strategy for two-dimensional mesh connected systems. In *Proc. International Conference on Parallel Processing*, pages 193–200, St. Charles, IL, August 1993.

[Fei94] D. Feitelson. A survey of scheduling in multiprogrammed parallel systems. Technical Report Research Report RC 19790(87657), IBM T. J. Watson Research Center, 1994.

[FN95] D. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In *Proc. IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 215–227, April 1995.

[Hot96] S. Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. In *IPPS '96 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 272–273, April 1996.

[HSO96] S. Hotovy, D. Schneider, and T. O'Donnell. Analysis of the Early workload on the Cornell Theory Center IBM SP2. In *Proc. SIGMETRICS' 96*, pages 272–273, Philadelphia, PA, May 1996. Full version available as CTC Technical Report No. 234 at http://www.tc.cornell.edu/Research/tech.rep.html.

[IBM95] IBM. *IBM LoadLeveler Adminstration Guide*, August 1995.

[Int94] Intel. *Paragon User's Guide*, June 1994.

[LC91] K. Li and K. Cheng. Job scheduling in a partitionable mesh using a two-dimensional buddy system partitioning scheme. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2(4):pp. 413–422, 1991.

[LHR95] D. Lifka, M. Henderson, and K. Rayl. Users guide to the Argonne SP scheduling system. Technical Report ANL/MCS-TM-201, Mathematics and Computer Science Division, 1995.

[LLNW94] W. Liu, V. Lo, B. Nitzberg, and K. Windisch. Non-contiguous processor allocation algorithms for distributed memory multicomputers. In *Supercomputing '94*, pages 227–236, Washington, D.C, Nov 1994.

[Ous82] J. Ousterhout. Scheduling techniques for concurrent systems. In *Proc. 3rd International Conference on Distributed Computing systems*, pages 22–30, 1982.

[SCZL96] J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY - LoadLeveler API project. In *IPPS '96 Workshop on Job Scheduling Strategies for Parallel Processing*, April 1996.

[WLF$^+$96] K. Windisch, V. Lo, D. Feitelson, B. Nitzberg, and R. Moore. A comparison of workload traces from two production parallel machines. Technical Report CIS-TR-96-06, University of Oregon, April 1996.