Evaluating Job Scheduling Techniques for Highly Parallel Computers

Takashi Suzuoka[1,2] Jaspal Subhlok[2] Thomas Gross[2,3]

August 1995

CMU-CS-95-149

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Modern parallel systems with n processor nodes can be used by jobs requesting *up to n* processors. If less than n processors are requested, multiple jobs can be run at the same time, allowing a larger number of users to use the system. One of the challenges for the operating system is to give reasonable service to a diverse group of requests. A single 1-node job that is running for a long time may effectively block the whole machine if the next job requests all n processors. To date various policies have been proposed for the scheduling of highly parallel computers. But as the users of current systems (e.g., a Paragon or T3D) know, these policies work far from perfect. Furthermore there is no literature that reports how such systems are really used, and as a result, there are no established evaluation techniques for scheduling policies. In this paper, we report on the measurement of the usage of a 96-node Intel Paragon used in a combined research, development and production environment at a supercomputing center. We define a simple metric to evaluate different scheduling strategies, and show that a set of simple techniques can significantly improve the usage of the machine. We believe that these results are of prime interest to supercomputer installations that wish to improve the quality of service they provide to the users.

This research was sponsored by the Advanced Research Projects Agency/CSTO, monitored by SPAWAR under contract N00039-93-C-0152. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either express or implied, of ARPA or the US Government.

[1] Toshiba Corporation, 1, Komukai, Toshiba-Cho, Saiwai-Ku, Kawasaki 210, Japan
[2] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213
[3] Institut fuer Computersysteme, ETH Zuerich, 8092 Zuerich, Switzerland
Corresponding Author: Jaspal Subhlok (`jass@cs.cmu.edu`)

Keywords: Job scheduling, supercomputer usage, turnaround time , fairness, performance modeling, performance evaluation

1 Introduction

The central 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 to improve the overall system performance, which in turn is closely related to the turnaround time for users. In this research we address how large parallel computers are used and show how their usage can be improved with better job scheduling.

User jobs arriving in a parallel system require different numbers of processors. One simple way to manage processor allocation is to divide the processors of the machine into fixed partitions and maintain different queues for each partition. The obvious limitation of such a method is that efficiency is poor if the job mix is not in tune with the way the processors are partitioned. Further, this places an artificial limit on the largest job, in terms of number of processors, that can be executed.

Large commercial parallel systems (e.g., IBM SP2, Intel Paragon, and Cray T3D) either support or can support dynamic partitions and time sharing. These features provide the flexibility necessary to support a wide variety of scheduling options. Most commercial systems use a scheduler based on Network Queuing System (NQS), which allows the system manager to define the scheduling policy based on a set of parameters. Thus most commercial systems can support a variety of scheduling policies, but our experience shows that the actual scheduling policies used are not necessarily the best way to use the machine. The techniques that we present and validate in this paper can be used to improve the utilization of most highly parallel machines in a straightforward way.

The choice of a scheduling policy is difficult for two main reasons; the best policy is dependent on the job mix, and the scheduler has to 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. The most important aspects aren the distribution of jobs in terms of the number of processors and the execution time, and the rate at which the jobs are submitted for scheduling. Since the job mix can change dynamically and typical job queue characteristics are not well understood, this is an important issue. We have used data from an actual supercomputer installation for maximum realism in experimentation. We also present results for a range of job submission rates.

The scheduler must achieve both efficient use of the machine and fairness in scheduling, which can be conflicting yet interrelated objectives. We have defined concrete criteria for efficiency and fairness, evaluate all the techniques using both criteria, and examine the tradeoffs between them.

This paper is organized as follows. Section 2 defines our criterion for measuring the merits of a scheduling policy. Section 3 describes the framework of our research. Section 4 presents data on the usage of an Intel Paragon supercomputer, and discusses how it influences the scheduling choices. Section 5 studies the effectiveness of a set of simple scheduling optimizations. Section 6 contains conclusions.

2 A performance model for scheduling

The scheduler's main objective is to schedule jobs for efficient performance and fairness. However, the success 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 state and briefly justify the criteria that we use for performance and fairness.

2.1 Turnaround time

We use average turnaround time weighted by job size (TT) to evaluate processing performance. It is defined as follows:

$$TT = \sum P_i T_i / \sum P_i$$

P_i : number of processors that execute job #i

T_i : turnaround time for job #i

The user of a parallel computer is most interested in the turnaround time, not the execution time, waiting time, or queue length, since that is what affects the user directly. Hence turnaround time is a natural criterion for performance.

The reason for weighting turnaround time with job size is more subtle. It is possible to get good average turnaround time by making the larger jobs wait longer, since they require more resources. Such a bias is not directly captured by our fairness criterion (discussed next), but can be corrected by weighting the turnaround time with the job size. Hence we have factored this aspect of fairness into the performance criterion.

2.2 Fairness

The scheduler can execute jobs in an order that is different from the *first-come first-served* manner to improve the throughput, and this leads to unfairness. We use the average of the absolute value of the **s**kipping **c**ount(SC) as a measure of fairness, where the skipping count is the difference between the arrival number and the execution number of a job. For example, if a job is executed 3 jobs earlier (later) than the order in which it arrives, the skipping count is -3(+3). The smaller the value of the skipping count, the fairer the system.

The average skipping count gives a global measure of fairness. It is also important to control how many jobs can get ahead of a particular job, thus ensuring that a job is not indefinitely blocked. In most scheduling schemes, the system managers can control this **M**aximum **A**llowable **S**kipping **C**ount(MASC) and thus control fairness.

3 Scheduling model

We state the system support that is required for our job scheduling mechanisms and explain the basic framework. The support required is minimal and available on most current commercial parallel machines.

3.1 Partitioning

We assume that the parallel machine supports dynamic partitioning, possibly with the restriction that jobs be allocated a power of 2 processors. This can support a two dimensional mesh architecture where each job is assigned a square block of the mesh.

Commercial parallel machines may have less or no restrictions on how jobs are assigned to processors. For example, the Intel Paragon allows a job to be assigned to any group of processors. However, we allow assignments only of power of 2 processors (assigning extra processors if a job does not need a power of 2 processors) for a variety of reasons. First, such a scheme is applicable to a broader class of machines, since it allows for common restrictions. Second, such assignments usually lead to more contiguity which can be important for performance. Finally, a class of scheduling policies discussed in the literature give good results only when a power of 2 processors are used.

Our method of processor assignment is a variation of the buddy system [LC91a, LC91b]. It assigns either the smallest power of 2 processors needed to satisfy a job, or all the processors. Specifically, it allocates n processors to a job requesting r processors such that:

$$n = \begin{cases} 2^i & \ldots \quad \text{if } m \geq i \text{ and } 2^i \geq r > 2^{i-1} \\ N & \ldots \quad \text{if } N \geq r > 2^m \end{cases}$$

where

N = total number of processors in the system

m = maximum number that satisfies $N \geq 2^m$

A job cannot specify its shape but it is possible to allocate physically contiguous processors to a job if the system is an n-dimensional mesh. For example, a 2-D mesh connected system with 16 processors can map a job set requiring 1, 1, 2, 4, and 8 in many ways, two of which are show in Figure 1. The important point is that a contiguous block of processors will be available if enough processors are available to satisfy a request.

The scheme works even when the system has a non power of 2 processors, by decomposing it into subsystems with power of 2 processors. For example, a 100 processors system will be divided into 3 groups of 64, 32, and 4 processors respectively. As long as job sizes are less than or equal to 64, the system behaves as if it has 3 processor groups. When a job requests more than 64 processors, the system will provide all processors when they become free.

3.2 Memory usage

Our current model ignores the memory requirements of a job for scheduling decisions. That is, our model does not include the sharing of a node by two jobs T1 and T2 such that both T1 and T2 fit into memory, although we examine

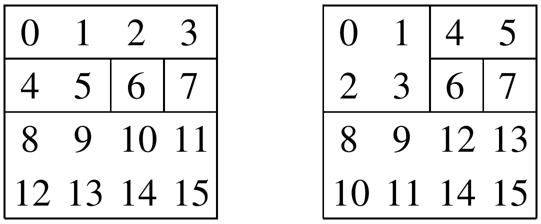

Figure 1: Examples of mapping of processors

the potential benefits of time sharing without regard to memory requirements. We believe time-sharing is an important aspect of scheduling, but ignore memory requirements for now for the following reasons: (1) current runtime systems often schedule only one job per processor, (2) the memory requirements of most jobs are unknown and are difficult to measure, and (3) the memory requirements of jobs often exceed physical memory and handling of virtual memory in a scheduling model is not well understood. It is our experience that one of the reasons to use a parallel system is the large pool of available memory, so for the time being, we do not see a need to investigate this aspect further.

3.3 Incoming jobs

We make the following realistic assumptions about the incoming jobs:

- The system does not know when a job comes and how many processors it needs, until it actually arrives.
- The system does not know when a job finishes execution until it does finish execution, and that all jobs eventually finish execution. We do examine the benefits of estimated job execution time if they were available.
- A job requests a fixed number of processors on arrival, and does not request additional processors at runtime.

For evaluating the benefits of time sharing, we assume that there is sufficient memory available for any scheduling, the context of the whole machine is switched at the same time i.e. gang scheduling [FR92], context switching is free, and that the policy for assigning time quantums to jobs can be set by the system manager. While some of these assumptions may be only be an approximation of reality, they are used only to get an idea of the benefits of time sharing.

4 Measurements of supercomputer usage

To develop and validate practical scheduling algorithms for parallel supercomputers, it is extremely important to understand how these computers are used. For this purpose, we processed and studied the log files of a 96 processor Intel Paragon containing usage information for the machine for several months[1]. Figure 2 shows the distribution of the jobs in terms of the number of processors needed to execute them and the actual execution time.

The data displayed in the figure gave us several important insights into the nature of supercomputer usage. We now discuss some of the interesting aspects of this data set.

We observe that a large fraction of the jobs run for a relatively short time, and a large fraction of those require relatively few processors. It reinforces our belief that supercomputers are used for a variety of different kinds of jobs, not simply by large jobs that require the whole machine. The data points out the importance of using a scheduling method that tries to avoid the situation where one or more small jobs are blocked by one, or more, long running large jobs. However, the data is not close to any of the common distributions like uniform, binomial, or Poisson that are often used to validate scheduling algorithms. Hence, it is important to use actual usage data for validation.

Another interesting observation is that the job sizes tend to be clustered around "power of 2" numbers of processors, with the exception that a large number of jobs needed all 96 processors that were available. Around two thirds of all jobs used 96 or power of 2 processors. On further examination we discovered that jobs that used a non power of 2 processors used very little time, i.e. almost all the time was spent executing jobs that used a power of 2 or 96 processors. We show the actual measurement in Table 4. The total time is weighted by the number of processors used, i.e ($\sum$ *number-of-processors* $\times$ *execution-time-in-seconds*), to get a fair measurement of machine usage.

[1] We thank Michael Vollmer of Intel SSD for collecting the information at the Paragon installed at ETH Zurich, Switzerland.

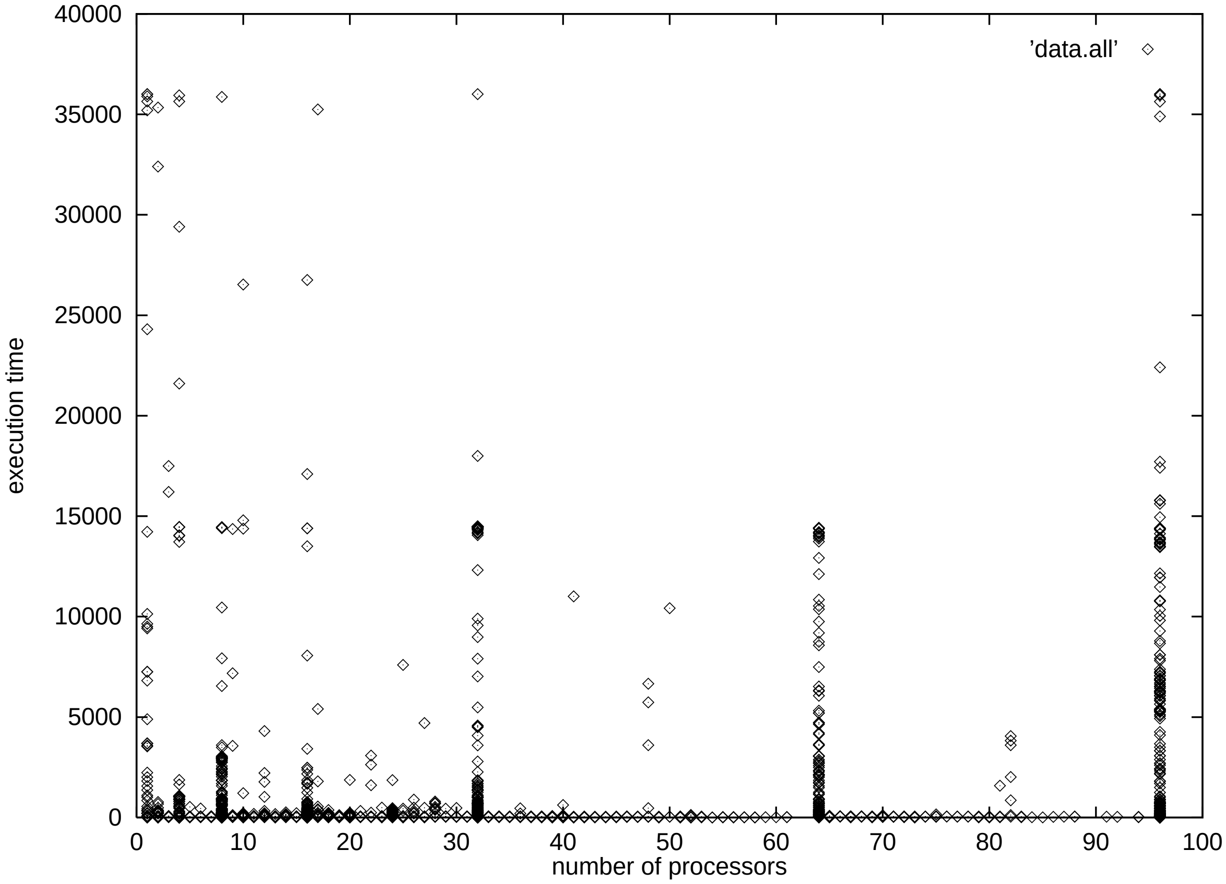

Figure 2: Distribution of number of processors and execution time for usage of 96-processor Paragon at ETH Zurich

processors	jobs	total time	
power of 2	880	50988978	(32.0%)
96	210	101822304	(63.8%)
others	633	6741028	(4.2%)

Table 1: Relation between number of processors and execution time

The dominant use of power of 2 numbers of processors is useful information because a large number of scheduling techniques discussed in literature assume that all jobs must execute on a number of processors that is a power of 2 [LC91a, CT94, DB93]. So these techniques are broadly applicable but they must be modified to allow the use of all processors when that number is not a power of 2.

Finally, we argue that usage of supercomputers indeed has a pattern and does not fluctuate wildly. For this purpose, we measured the usage patterns in two different periods (Fall 94 on the left and Spring 95 on the right), and the data is shown in Figure 3.

A visual comparison shows that the data does not differ dramatically, although some differences are visible. For example, the data is clustered more strongly around power of 2 numbers of processors in the figure on the right [2]. The relative stability of the usage data is important since it shows that there are patterns in the usage of supercomputers, and scheduling algorithms do not have to aim at a moving target. However, we wish to point out that more data from

[2]We conjecture that this is a more normal pattern of usage and the figure on the left is influenced by certain individuals repeatedly executing a small job with different number of processors for measuring scalability.

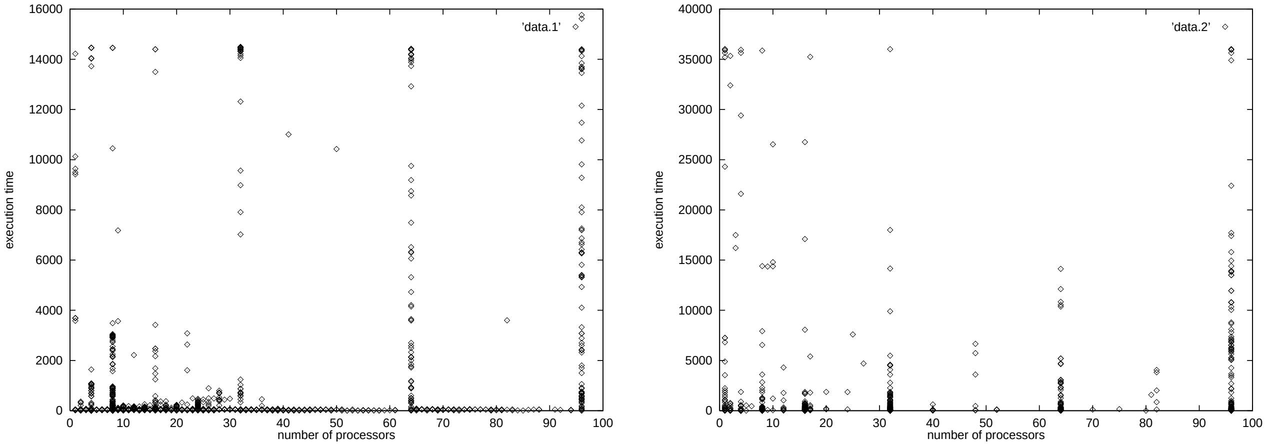

Figure 3: Paragon usage in Fall 94 (left) and Spring 95 (right)

multiple supercomputer installations is needed to make stronger statements about the way the supercomputers are used.

5 Scheduling optimizations

We now examine the benefits of a set of simple scheduling optimizations using the Paragon job mix discussed in the previous section. Our reference scheduling policy is a simple Space Sharing Scheme (SSS) in which the jobs are scheduled in order as soon as enough processors are available. The criteria for comparing scheduling schemes are average turnaround time for performance, and average skipping count for fairness.

5.1 Sorting the waiting queue

A small job can prevent a larger job from executing by blocking enough processors, and this leads to a low overall machine utilization. If the waiting queue is frequently sorted in the decreasing order of job size, the occurrence and the performance impact of the above scenario is minimized. This optimization was proposed by Li and Cheng [LC91a].

We simulated the scheduling of jobs using a sorted waiting queue and compared it with a simple SSS scheme. The results are shown in Figure 4. We observe a significant improvement in the average turnaround time, and the improvement is more when the job invocation ratio is higher. A higher job invocation ratio implies a longer waiting queue and therefore the impact of sorting the queue is more.

The main drawback of sorting is that it causes unfairness. A small job may keep getting passed by larger jobs, and may have to wait for an arbitrarily long time to be scheduled. Figure 5 shows that the average skipping count, which is our measure of (un)fairness, is significant, particularly for higher rates of job invocation. The worst case unfairness behavior can be easily controlled by specifying a *maximum skipping count*(MASC), which is the maximum number of jobs that are allowed to pass a particular job in the queue. The scheduling algorithm can be trivially modified to enforce the maximum skipping count. The impact on turnaround time of forcing a fixed maximum skipping counts is shown in Figure 6. Unrestricted sorting corresponds to an infinite queue length in the figure. We observe that when the maximum skipping count is restricted to 10, we still get a large fraction of the benefits of sorting. However, when the maximum skipping count is restricted to 5, the benefits of sorting are negligible. We conclude that the benefits of sorting depend directly on how much unfairness is permitted and we can get significant benefits for moderate maximum skipping counts which represent moderate fairness.

5.2 Back-filling

When a large job is blocked at the head of the waiting queue because enough processors are not available, it may be possible to execute a smaller job in the queue without affecting the time when the larger job gets scheduled. This technique is called *back-filing* and is in use at the IBM SP 2 installation at the Argonne National Laboratory [Lab94].

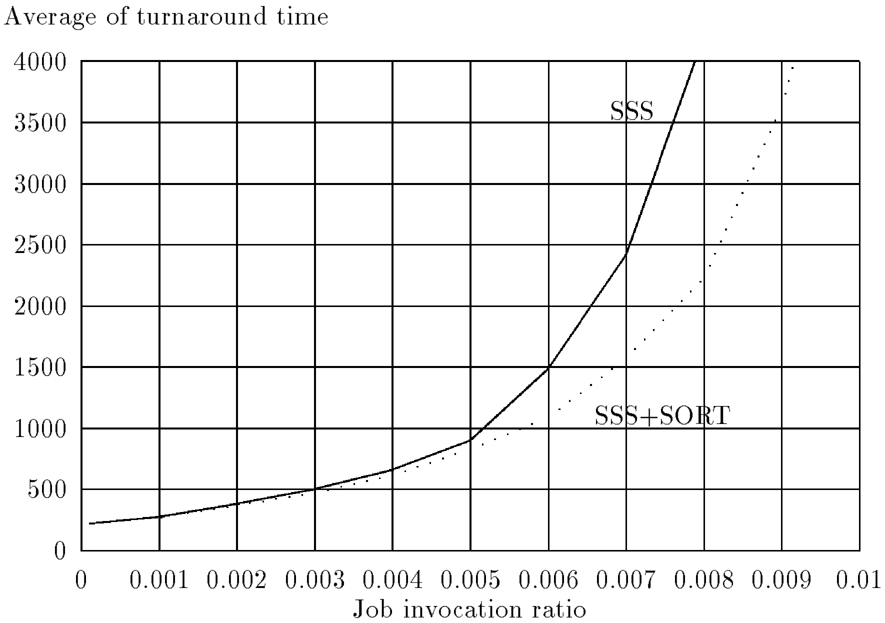

Figure 4: Impact of using a sorted waiting queue on performance

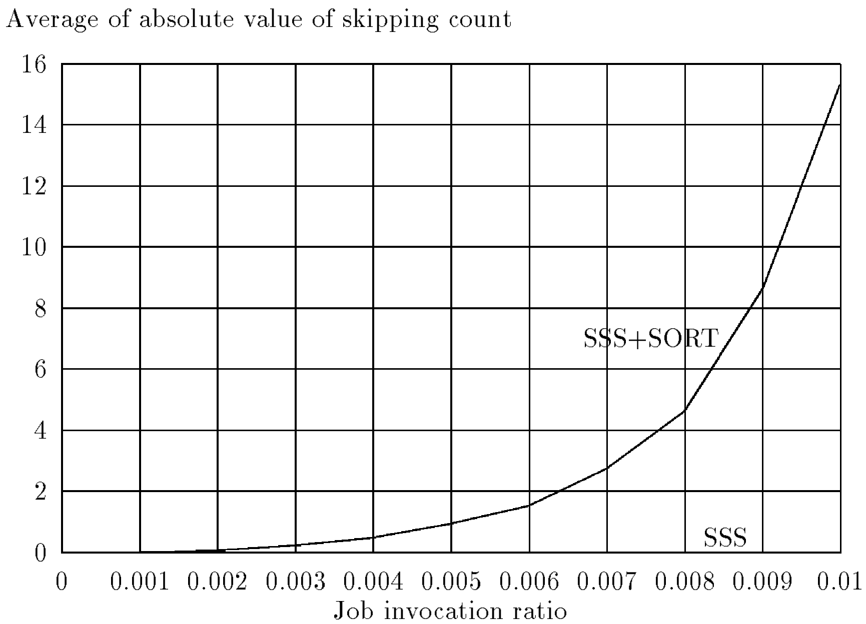

Figure 5: Impact of using a sorted waiting queue on fairness

Back-filling requires that the exact execution times of all jobs be known ahead of time. In a simple back-filling system, the jobs in the waiting queue are examined starting from the head of the queue, and any job that is guaranteed not to change the times at which the jobs ahead of it will get scheduled, is scheduled immediately. This form of back-filling never *increases* the turnaround time of any job.

We used back-filling for scheduling our job mix assuming that the exact execution times for all jobs were known ahead of time, and the results are shown in Figure 7. The figure shows that back-filling leads to a considerable

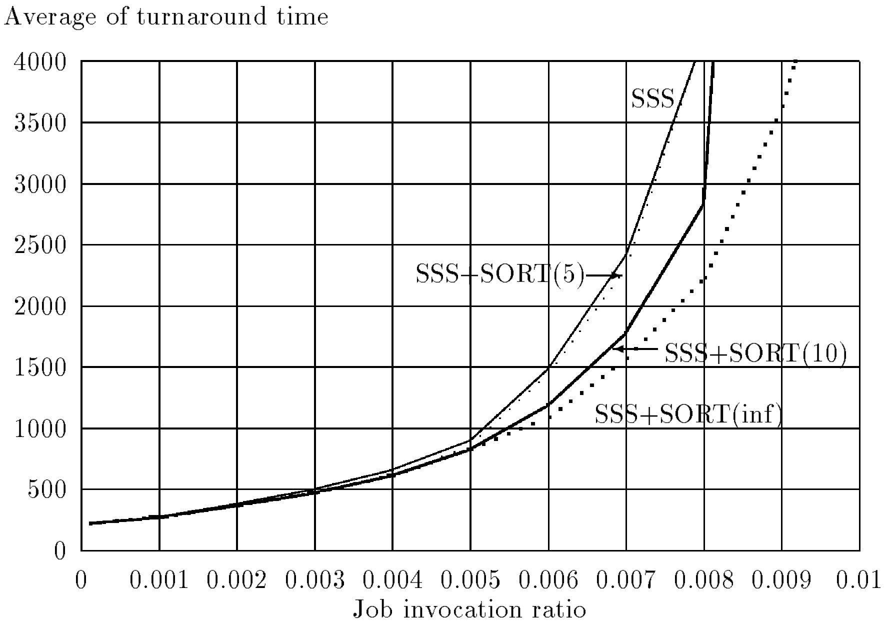

Figure 6: Impact of a fixed maximum skipping count on performance with sorting

improvement in average turnaround time with no loss of fairness[3]. Figure 7 also shows the average turnaround time using sorting and backfilling, which is considerably improved over those obtained by the two techniques individually. This shows that these two techniques can be used together effectively.

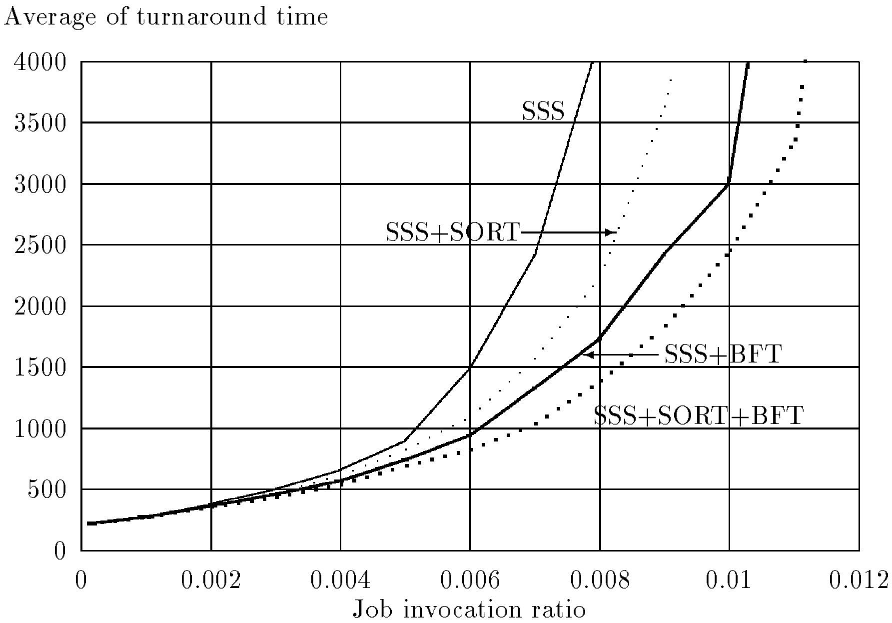

Figure 7: Impact of back-filling and sorting on performance

The main drawback of back-filling is that it is virtually impossible to predict the precise execution time of jobs.

[3] In this scenario measuring fairness by skipping count is misleading since moving jobs ahead has no impact on jobs that are skipped.

However, back-filling can be used with an approximate notion of execution time. In particular, we measured that the improvement in turnaround time was the almost the same when the estimated execution time had an average error of 50%. However, if the simple scheme discussed above is used with inexact execution times, perfect fairness is not guaranteed. Some jobs may get scheduled later than they would have without back-filling, which may or may not be acceptable. Schemes that would ensure fairness in the presence of inaccuracies in estimated execution time are beyond the scope of this paper. We conclude that back-filling is a useful technique, but using it in a practical environment involves a variety of issues that have to be sorted out depending on the policies of a supercomputer installation.

5.3 Time Sharing

A more general approach to scheduling allows time sharing, that is multiple jobs may be assigned to the same set of processors and the operating system assigns time quantums to the scheduled jobs. The time quantum assigned to an individual job is related to how many processors are busy when that job is executed. A shorter time quantum is assigned when only a few processors are busy, and vice-versa, so that the overall machine utilization is maximized.

It is difficult to compare a space sharing system with a time sharing system without the knowledge of the memory requirements of jobs and the context switching overheads. This information is difficult to measure, and was not available to us for the data that we have used. We simulated the execution of the job suite assuming that sufficient memory was available to schedule all the executing jobs, and assuming no context switching overhead. The job in the front of the queue is scheduled if that would lead to an improvement of machine utilization at that instance. For example, if there is only one job executing on all processors and the job at the head of the queue requires a subset of the processors, then that job will not be scheduled since that would decrease the machine utilization. Note that this scheme does not necessarily improve overall machine utilization, since that criterion is checked only at the point of scheduling a new job.

The average turnaround time with this time sharing scheme (TSS) is compared to that with space sharing in Figure 8. We observe that there is considerable improvement of performance with time sharing. In related ongoing research, we have discovered that sorting and back-filling optimizations discussed in earlier sections can be modified for use with a time sharing system and yield significant benefits. Time sharing is an important approach and we expect to examine its value in more detail in future work.

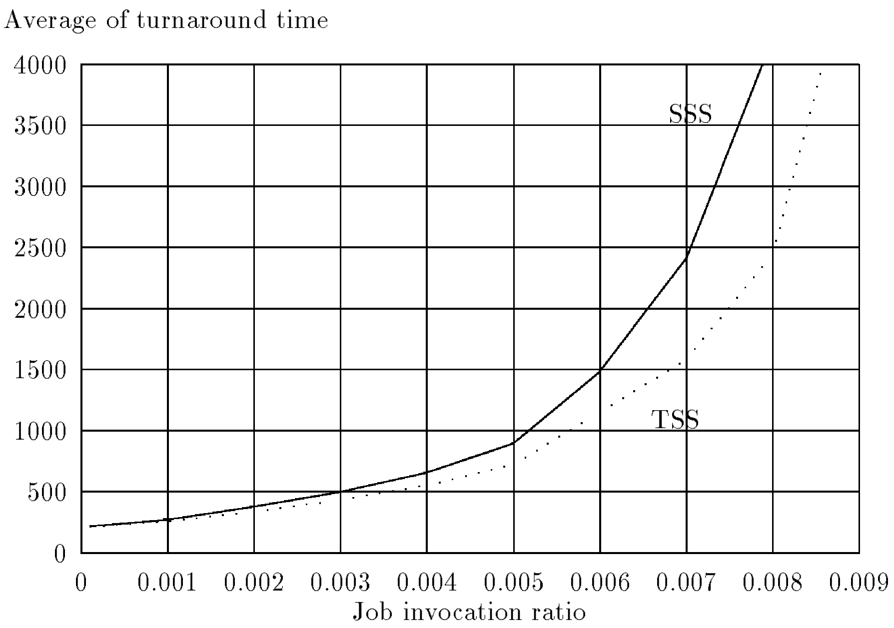

Figure 8: Comparison of time sharing and space sharing schemes

6 Conclusions

We have analyzed job arrival and execution data from a supercomputer installation and used it to determine the value of a set of scheduling mechanisms. While many scheduling policies and mechanisms have been proposed in the literature, their validation is based on simple assumptions about the characteristics of jobs that are executed on parallel machines, which are often unrealistic. Our entire experimentation is based on the data collected over several months on a 96 processor Intel Paragon installation. We show that a set of fairly simple mechanisms can significantly improve the average turnaround time for parallel machines.

We believe that it is extremely important to understand the characteristics of jobs that enter a parallel system in order to optimize scheduling, and the only way to determine that is to collect and analyze actual usage information, which can be hard to find. We have performed this study with he help of a supercomputing center at ETH Zurich. We believe that this is only a first step, and hope that other supercomputer installations and researchers would participate in research that would lead to better understanding and improvements in supercomputer usage.

References

[CT94] Po-Jen Chuang and Nian-Feng Tzeng. Allocating precise submeshes in mesh connected systems. *IEEE trans on Parallel and distributed systems*, Vol. 5(2):pp. 211–217, Feb. 1994.

[DB93] Jianxun Ding and Laxmi N. Bhuyan. An adaptive submesh allocation strategy for two-dimensional mesh connected systems. *Proc. of International Conference on Parallel Processing*, pages pp. 193–200, 1993.

[FR92] Dror G. Feitelson and Larry Rudolph. Gang scheduling performance benefits for fine-grain synchronizatio. *J. Parallel Distrib. Comput*, pages pp. 306–318, 1992.

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

[Lab94] Argonne National Laboratory. *Current Queueing Algorithm*. mosaic http: //www.mcs.anl.gov/home/rayl/scheduler/node19.html, Sep. 17, 15:56:16 CDT 1994.

[LC91a] Kequin Li and Kam-Hoi Cheng. Job scheduling in a partitionable mesh using a two-dimensional buddy system partitioning scheme. *IEEE trans. on Parallel and Distributed Systems*, Vol. 2(4):pp. 413–422, 1991.

[LC91b] Kequin Li and Kam-Hoi Cheng. A two dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system. *Journal of Parallel and Distributed Computing*, (12):pp. 79–83, 1991.

[LLNW94] Wanqian Liu, Virginia Lo, Bill Nitzberg, and Kurt Windisch. Non-contiguous processor allocation algorithms for distributed memory multicomputers. *Proc. of Supercomputing*, pages pp. 227–236, Nov. 1994.

[Ous82] John K. Ousterhout. Scheduling techniques for concurrent systems. *Proc. of 3rd International Conf. on Distributed Computing systems*, pages pp. 22–30, 1982.

[STH+94] Kuniyasu Suzaki, Hitoshi Tanuma, Satoshi Hirano, Yuuji Ichisugi, and Michiharu Tsukamoto(in Japanese). A TSS using partitioning algorithm "2D buddy" or "adaptive scan" for mesh-connected parallel computers. *IPSJ OS*, 65-18:pp. 137–144, July 1994.