# An Evaluation of Preemption Strategies for Parallel Job Scheduling

Rajkumar Kettimuthu, Vijay Subramani, Srividya Srinivasan, Thiagaraja B Gopalasamy, D K Panda and P Sadayappan

## Abstract

Although theoretical results have been established regarding the utility of pre-emptive scheduling in reducing average job turn-around time, job suspension/restart is not much used in practice at supercomputer centers for parallel job scheduling. A number of questions remain unanswered regarding the practical utility of pre-emptive scheduling. We explore this issue through a simulation-based study, using job logs from a supercomputer center. We develop an adaptive and tunable selective-suspension strategy, and demonstrate its effectiveness. We also present new insights into the effect of pre-emptive scheduling on different job classes.

## 1. Introduction

Although theoretical results on the effect of pre-emptive scheduling strategies in reducing average job turn-around time have been well established, pre-emptive scheduling is not currently being used for scheduling parallel jobs at supercomputer centers. Compared to the large number of studies that have investigated non-preemptive scheduling of scheduling parallel jobs, little research has been reported on empirical evaluation of preemptive scheduling strategies using real job logs [2,3,7,9].

The basic idea behind preemptive scheduling is simple: if a long running job is temporarily suspended and a waiting short job is allowed to run to completion first, the wait time of the short job is significantly decreased, without much fractional increase in the turn-around time of the long job. Consider a long job with runtime Tl. If after time t, a short job arrives with runtime Ts. If the short job were run after completion of the long job, the average job turnaround time would be (Tl + (Tl+Ts-t))/2, or Tl + (Ts-t)/2. Instead, if the long job were suspended when the short job arrived, the turnaround times of the short and long jobs would be Ts and (Ts+Tl) respectively, giving an average of Ts + Tl/2. The average turnaround time with suspension is less if Ts < Tl-t, i.e. the remaining runtime of the running job is greater than the runtime of the waiting job. However a simple preemptive scheduling strategy (that suspends a running job whenever a waiting job's wait time goes above the wait time of the running job) may not always improve the average turnaround time. Consider a situation where a job with runtime T starts execution, and immediately another job with the same runtime arrives in the queue. Very soon, the wait time of the waiting job will go above that of the running job, causing the first job to be suspended to allow the other job to run. But the situation would reverse very soon after the waiting job starts running. The two jobs would keep alternating, with both finally completing around time 2T (assuming negligible overhead for job suspension/restart). The turnaround time for each job, and therefore the average turnaround time would thus be 2T. Without preemption, the average turnaround time would have been (T+2T)/2, i.e. 1.5T. These simple examples show that the benefit of preemptive scheduling can be very dependent on the job mix being scheduled. Therefore it is important to perform evaluations of preemptive scheduling schemes using realistic job mixes derived from actual job logs from supercomputer centers. The primary contributions of this paper are:

- The development of an adaptive, tunable selective-suspension strategy for pre-emptive scheduling of parallel jobs, and
- Characterization of the significant variability in the average job slowdown for different job categories, and

We study the effect of preemption on the performance of various categories of jobs using the Maui scheduler [16] in its simulation mode. The rest of the paper is organized follows. Section 2 presents some basic background on scheduling of parallel jobs. Section 3 discusses the simulation environment we use. In Sections 4, a basic preemptive scheduling scheme is evaluated. In Sections 5, 6 and 7, enhancements to the basic strategy are proposed and evaluated. In Section 8, we evaluate the impact of job-suspension overheads on pre-emptive scheduling. Section 9 reviews related work. Section 10 presents our conclusions.

## 2 Background

A number of approaches have been proposed in the past for scheduling parallel jobs on a cluster of workstations [11,12,13,14,15]. Scheduling is usually viewed using a 2D chart with time along the horizontal axis and the number of processors along the vertical axis. Each job can be thought of as a rectangle whose length is the user estimated run time and width is the number of processors required. The simplest way to schedule jobs is to use the First-Come-First-Served (FCFS) policy. This approach suffers from low system utilization. Backfilling [5,10] was proposed to improve the system utilization and has been implemented is several production schedulers [6,8]. Backfilling works by identifying “holes” in the 2D chart and moving forward smaller jobs that fit those holes. There are two common variations to backfilling - conservative and aggressive. In conservative backfilling, a smaller job is moved forward in the queue as long as it does not delay any previously queued job. In aggressive backfilling, a small job is allowed to leap forward as long as it does not delay the job at the head of the queue. The scheduler maintains the current list of running jobs along with their expected completion times and a list of queued jobs with the user estimated run times.

.

Pseudo-code for aggressive backfilling :

```
1. Given the current list of running jobs, find the earliest time that the job at the head
   of the queue can start executing.
2. Make a reservation for the job at the head of the queue at the calculated time.
```

3. Iterate over each job in the idle queue to find jobs to backfill. A job is eligible for backfilling if it will fit in to one of the existing holes in the schedule without delaying the job at the head of the queue.

Some of the common metrics used to evaluate the performance of scheduling schemes are the average turnaround time and the average bounded slowdown. We have used the bounded slowdown for our studies. The bounded slowdown of a job is defined as follows:

Bounded Slowdown = (Completion time - Queue time)/ Max(Running time, 10)

The threshold of 10 seconds is used to limit the influence of very short jobs on the metric.

Pre-emptive scheduling aims at providing lower delay to short jobs relative to long jobs. Since long jobs have greater tolerance to delays as compared to short jobs, we assign priorities to jobs that adequately capture this. A suitable metric for the priority is the *xfactor*, which increases rapidly for short jobs and gradually for long jobs.

xfactor = (Wait time + Estimated Run Time)/ Estimated Running Time

## 3. Simulation Environment

The Maui scheduler is a popular batch scheduler widely used in the HPC community. This scheduler allows configuration of parameters like Job Prioritization, Fair Share policies, and Backfill policies. The scheduler also supports 3 modes – Normal, Simulation and Test modes. The simulation mode was used for our studies. In the Simulation mode, the input to the scheduler is in the form of a workload trace file and a resource trace file. The workload trace file specifies for each job, parameters like queue time, start time, completion time, wall clock limit, number of nodes requested, the amount of memory it needs etc. The resource trace file models the resources in the system i.e. the number of nodes in the system, the amount of memory at each node etc. The EASY backfill policy [8] was used as the base scheduling scheme for this study.

**Workload Characterization**:

We studied the effect of suspension on the system performance under various loads, ranging from the actual load from the job trace to higher loads all the way up to system saturation. The impact of varying system load was modeled by simply changing the arrival times of the jobs in the workload file. From collection of workload logs available from Feitelson's archive [17, the CTC workload trace, a commonly used trace [4,5], was used to evaluate the proposed schemes. As with all job logs, the trace contained information about all jobs submitted to the system, including a number of jobs that abnormally aborted. It is common practice to perform simulation-based studies of job scheduling using all jobs from a trace. While this is desirable from the point of view of modeling an actual system workload, we believe that there is a serious problem that has been ignored by previous studies. Abnormally aborted jobs tend to excessively skew the average slowdown of jobs in a workload. Consider a job requesting a wall-clock limit of 24 hours, that is queued for 1 hour, and then aborts within one minute due to some fatal exception. The slowdown of this job would be computed to be 60, whereas the average slowdown of normally completing long jobs is typically under 2. If even 5% of the jobs have a high slowdown of 60, while 95% of the normally completing jobs have a slowdown of 2, the average slowdown over all jobs would be around 5. Now consider a scheme such as the speculative backfilling strategy evaluated in [4]. With this scheme, a job is given a free timeslot to execute in, even if that slot is considerably smaller than the requested wall-clock limit. Aborting jobs will quickly terminate, and since they did not have to be queued till an adequately long window was available, their slowdown would decrease dramatically with the speculative backfilling scheme. As a result, the average slowdown of the entire trace would now be close to 2, assuming that the slowdown of the normally completing jobs does not change significantly. A comparison of the average slowdowns would seem to indicate that the speculative backfill scheme results in a significant improvement in job slowdown from 5 to 2. However, under the above scenario, the change is due only to the change for the small fraction of aborted jobs, and not due to any benefits to the normal jobs. In order to avoid this problem when evaluating our proposed strategies, we first removed all the aborted jobs from the trace. Further, in order to avoid any effects due to user inaccuracy in estimation of job runtimes, we used the actual recorded runtimes as the basis for all scheduling decisions.

Table 1 summarizes a 3-week workload trace obtained from system logs starting July 1996, modified as stated above, to remove all aborted jobs. There were 430 processors in the system. Under normal load, with the standard non-preemptive aggressive backfilling strategy, using xfactor as the scheduling priority, the utilization was 46 percent. Traces corresponding to higher offered loads were created by compressing the arrival times of jobs without changing their runtimes. Thus a high load trace corresponding to double the actual load was created by changing all arrival times to half the actual value. This modified trace was used as the high load workload. Although it is known that user estimates are quite inaccurate in practice, as explained above, we first studied the effect of preemptive scheduling under the idealized assumption of exact estimation, before studying the effect of inaccuracies in user estimates of job run time. Also, we first studied the impact of pre-emption under the assumption that the overhead for the suspension and restart is negligible and then studied the influence of the overhead.

**Job classification**

Any analysis that is based on the aggregate slowdown of the system as a whole does not provide insights into the variability within different job categories. Therefore in our discussion, we classify the jobs into various categories based on the runtime and the number of processors requested, and analyze the slowdown for each category. The distribution of the jobs from the trace are shown in Table 1, on the basis of number of processors requested, the run-times for the jobs, as well as processor-time product.

| Job classification based on the Processor-Seconds utilized | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **0-10** | **11-100** | **$10^2$ -$10^3$** | **$10^3$- $10^4$** | **$10^4$ - $10^5$** | **$10^5$ - $10^6$** | **$10^6$ - $10^7$** | **> $10^7$** |
| #jobs | 0 | 355 | 981 | 1048 | 1292 | 518 | 69 | 2 |
| **percentage** | 0 | 8 | 23 | 25 | 30 | 12 | 2 | 0 |

| Job classification based on the number of processors | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3-4** | **5-8** | **9-16** | **17-32** | **33-64** | **>64** |
| #jobs | 1951 | 303 | 565 | 412 | 535 | 277 | 134 | 88 |
| **percentage** | 46 | 7 | 13 | 10 | 13 | 6 | 3 | 2 |

| Job classification based on the run time | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **0-1min** | **1-10min** | **10-60min** | **1-2hrs** | **2-4hrs** | **4-8hrs** | **8-16hrs** | **>24 hrs** |
| #jobs | 366 | 1186 | 814 | 466 | 505 | 321 | 447 | 160 |
| **percentage** | 9 | 28 | 19 | 11 | 12 | 8 | 10 | 4 |

**Table 1: Job Distribution in Workload Trace**

To analyze the performance of jobs of different sizes and lengths, jobs were classified into 16 categories: four categories based on their run time – very short, short, long and very long and four categories based on the number of processors requested – sequential, narrow, wide and very wide. The criteria used for job classifications is shown in Table 2.

| | **0-10Mins** | **10Min - 1Hr** | **1Hr - 8 Hr** | **> 8Hr** |
|---|---|---|---|---|
| 1 Processor | VS S | VS N | VS W | VS VW |
| 2-8 Processors | S S | S N | S W | S VW |
| 9-32 Processors | L S | L N | L W | L VW |
| >32 Processors | VL S | VL N | VL W | VL VW |

**Table 2:**
**Job Categories based on processors and time required**

The distribution of jobs in the trace, corresponding to the sixteen categories is given in Table 3.

| Job Categories | | | | |
|---|---|---|---|---|
| | **0-10Mins** | **10Min - 1Hr** | **1Hr - 8 Hr** | **> 8Hr** |
| 1 Processor | 581 | 699 | 221 | 51 |
| 2-8 Processors | 418 | 208 | 115 | 73 |
| 9-32 Processors | 571 | 266 | 400 | 55 |
| >32 Processors | 381 | 107 | 76 | 43 |

**Job Percentage**

|  | 0-10Mins | 10Min - 1Hr | 1Hr - 8 Hr | > 8Hr |
|---|---|---|---|---|
| 1 Processor | 14 | 16 | 5 | 1 |
| 2-8 Processors | 10 | 5 | 3 | 2 |
| 9-32 Processors | 13 | 6 | 9 | 1 |
| >32 Processors | 9 | 3 | 2 | 1 |

**Table 3:**
**Category based distribution of jobs**

## 4. A Selective Suspension Scheme

In a study presented at this year's IPDPS conference [3], a pre-emptive scheduling strategy called the "Immediate Service (IS)" scheme was evaluated. With this scheme, each arriving job was given an immediate time-slice of 10 minutes, by suspending one or more running jobs if needed. The selection of jobs for suspension was based on their instantaneous-xfactor, defined as (wait time + total accumulated run time)/ (total accumulated run time). Jobs with the lowest instantaneous-xfactor were suspended. Jobs that did not complete within their immediate service period were placed in the waiting queue, sorted by the instantaneous-xfactor. The instantaneous-xfactor was also used as the priority for backfill scheduling of queued jobs. The IS strategy was shown to significantly decrease the average job slowdown for the traces simulated. However, no information was provided on how different job categories were affected. As with the speculative backfilling scheme discussed earlier, the IS scheme can also be expected to provide significant improvement to the slowdown of aborted jobs in the trace. So it is unclear how much, if any, of the improvement in slowdown was experienced by the jobs that completed normally.

A potential shortcoming of the IS scheme is that its scheduling decisions are not in any way reflective of the expected runtime of a job. Thus a 20-minute job and a 24-hour job that arrive at the same time will both have the same priority after completing their initial immediate service period. Since the expected slowdown of the 20-minute job will rise at a much faster rate than that of the 24-hour job, it would be desirable to give higher priority to the 20-minute job.

We first propose a simple preemptive scheduling scheme, called the Selective Suspension (SS) scheme, which seeks to be discriminative in situations such as that exemplified above. With this scheme, an idle job can pre-empt a running job if its priority is sufficiently higher than the running job, and its required time is less than or equal to the remaining running time for the running job. The latter constraint is imposed in order to preserve any reservations beyond the completion point of the running job. An idle attempts to suspend a collection of running jobs so as to obtain enough free processors. In order to control the rate of suspensions, a suspension factor (SF) is used. This specifies the minimum ratio of the priority of a candidate idle job to the priority of a running job for preemption to occur. The priority used is the xfactor of the job. With a suspension factor of 1, a very large number of suspensions can occur and may lead to thrashing. Consider two jobs x and y of the same length that arrive at the same time. Assume that both of them require all the processors in the system. Initially, both of them will have a priority of 1 and say x gets in first. In the next scheduling iteration, y will have a higher priority than x and will suspend x. This will repeat at every iteration and both x and y will complete nearly at the same time. Thus, both of them will have a slowdown of 2. This will not happen with a suspension factor of 2 since job x would have completed by the time the priority of job y becomes twice the priority of job x. In order to avoid thrashing and to reduce the number of suspensions, we use different suspension factors between 2 and 10 in evaluating our schemes. Thus an idle job can preempt a running job only if its priority is at least SF times greater than the priority of the running job. All the idle jobs that are able to find the required number of processors by suspending lower priority running jobs are selected for execution by preempting the corresponding job. The scheduler periodically invokes the preemption routine.

**Pseudo code**:

```
1. Sort the list of running jobs in ascending order of priority and sort the idle jobs in
   descending order of priority.
2. For each job in the idle queue do the following
     a. Traverse the list of running jobs
     b. If the priority of the idle job < Suspension factor * Priority of the running
        job at the head of the run queue, then Exit.
     c. If a set of one or more jobs that satisfy the following criteria is found, then
        go to step 2d.
          i. Number of processors requested by the idle job <= Sum of the
             processors used by each job in the set
         ii. User estimated runtime of the idle job < Remaining runtime of
             each job in the set
        iii. Priority of the idle job > Suspension factor * Priority of each job in
             the set.
     d. Suspend the running job(s) selected in step 2c.
   End For
```

We compare the SS scheme run under various suspension factors with the No-Suspension (NS) scheme and the IS scheme. Fig1 shows the results under normal load and fig.2 shows the results under high load. The trends are similar under both the loads except for the fact that the effects are more pronounced with the high load. Considering first the normal-load case, we can see that the SS scheme provides significant improvement for the Very-Short (VS) and Short (S) length categories and Wide(W) and Very-Wide (VW) width categories. For example, for the VS-VW category, slowdown is reduced from 8 for the NS scheme to under 2 for SS with SF=2. For VS and S length categories, a decrease in SF results in lowered slowdown. This is because a lower SF increases the probability that a job in these categories will suspend a job in the Long (L) or Very-Long (VL) category. The same is also true for the L length category, but the effect of change in SF is less pronounced. For the VL length category, there is an opposite trend with decreasing SF, i.e. the slowdown worsens. This is due to the increasing probability that a Long job will be suspended by a job in a shorter category as SF decreases. In comparison to the base NS scheme, the SS scheme provides significant benefits for VS and S categories, a slight improvement for most of the Long categories (except Sequential), but is slightly worse for the VL categories. The same is true also for the IS scheme.

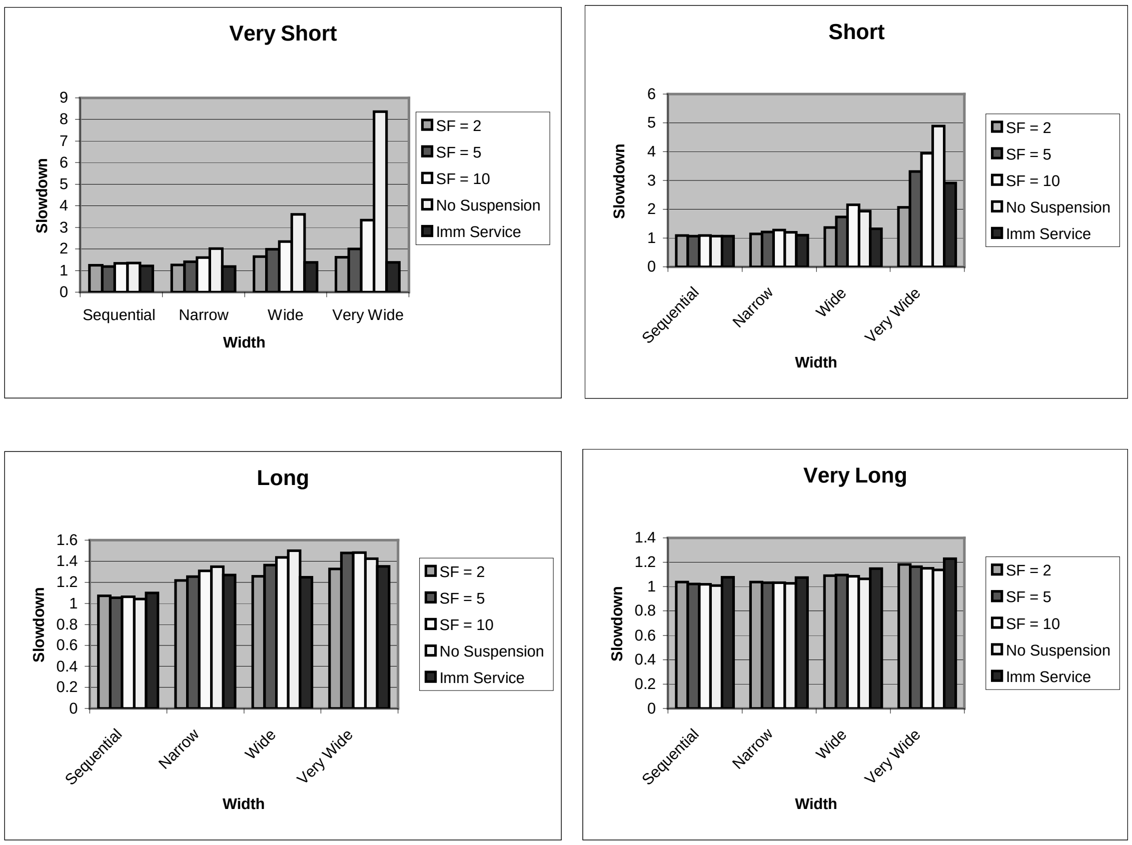

**Fig.1 : Selective Suspension with Reservation under Normal Load**

The performance of the IS scheme is very good for the normal-load case. It is slightly better than the SS scheme for the VS length category and slightly worse for the VL length category.

For the high-load case, the general trends with SS are similar to the normal-load case, but the effects are more pronounced. The factor of improvement for the VS-VW case is now over 5 with SF=2, but now the degree of worsening for the VL-Seq case is over 2 for all values of SF. The performance of IS however deteriorates considerably for the L and VL length categories. This is due to the fact that the number of suspensions with IS grows rather uncontrollably as load increases.

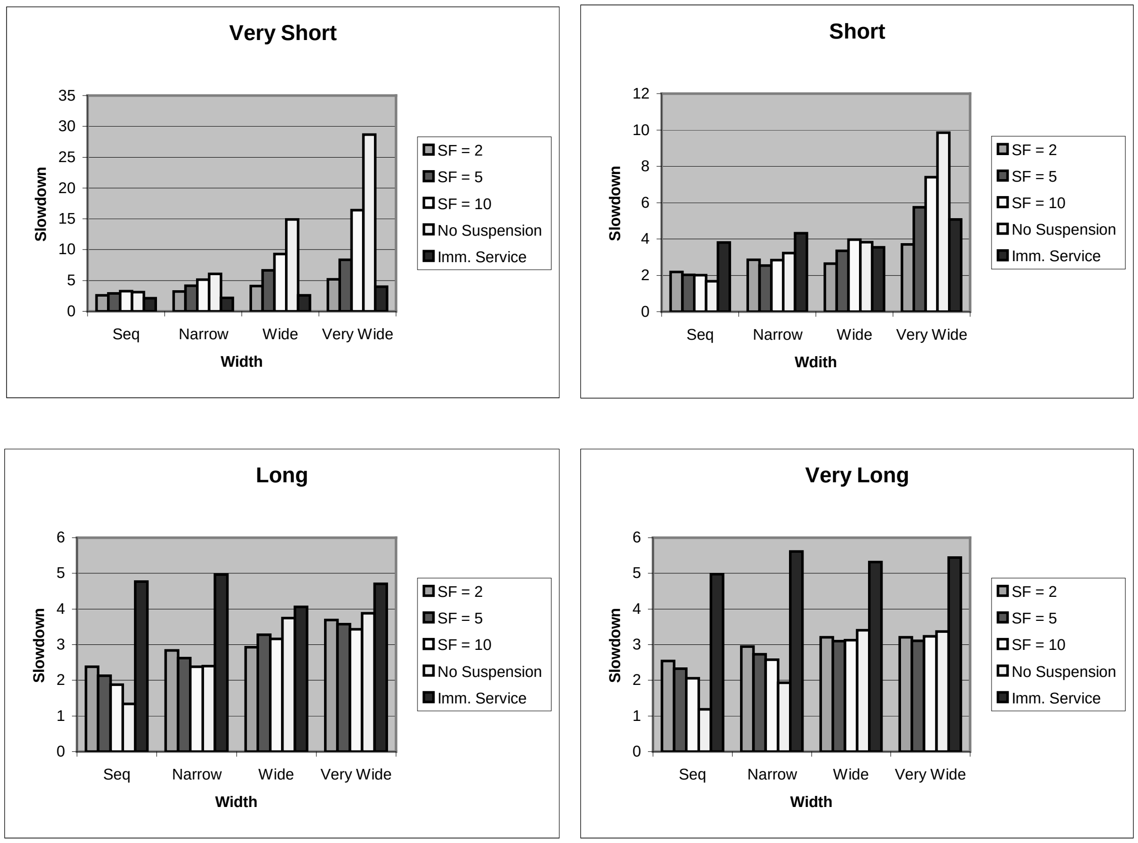

**Fig 2: Selective Suspension with Reservation under High Load**

A primary reason for the poor performance of SS for the VL categories and some L categories is that a long job with a high xfactor may have difficulty getting in because of the difficulty in finding running jobs with remaining runtime greater than its remaining runtime. This restriction for the preemption was imposed by the need to preserve reservations. All backfill scheduling schemes use job reservations for one or more jobs at the head of the idle queue as a means of guaranteeing finite progress and thus avoidance of starvation.

## 5. Selective Suspension with No Reservations (SSNR)

Since the SS strategy uses the expected slowdown as the priority mechanism, there is an automatic guarantee of freedom from starvation – ultimately any job's expected slowdown factor will get large enough that it will be able to preempt some running job and begin execution. Therefore, it is possible to run the backfill algorithm without the usual reservation guarantees. This allows us to remove the restriction that preempted jobs have longer remaining runtimes than suspending jobs. Due to space limitations, for the rest of the paper, we only present data for the high-load case. Simulations have been run for various loads including the normal trace load, and the trends are qualitatively similar, with the differences being magnified for the high-load case.

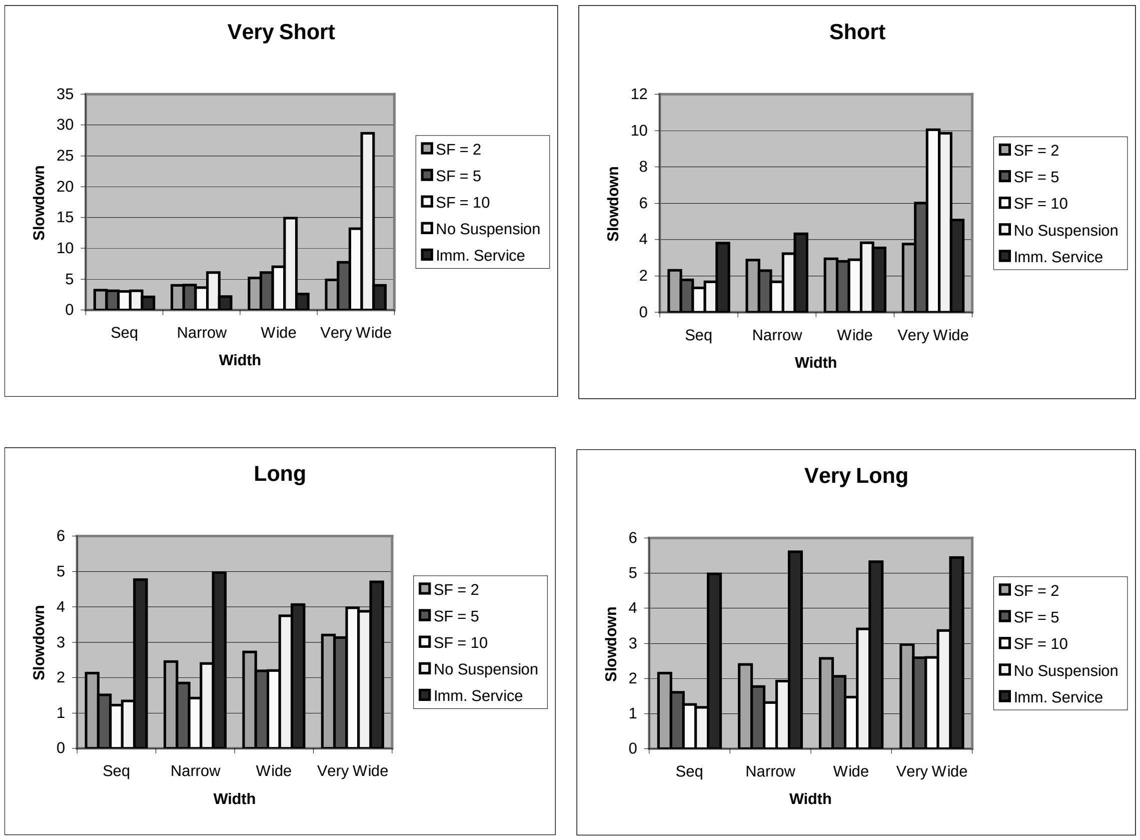

**Fig 3: Selective Suspension without Reservation under High Load**

From fig.3, it is clear that the removal of the reservation in the SS scheme helps the long jobs. There seems to be an overall improvement for most of the cases when compared to the SS scheme with reservations. This is because, with reservation the holes in the schedule are filled by backfilling and there is a good possibility for holes even after backfilling. When the reservation is removed, it is possible to do more effective backfilling, without leaving as many holes in the schedule. The trend across various suspension factors remains the same as that of the earlier scheme. Now, the suspension factor of 10 performs better than the no suspension scheme for all the Long categories except for VL-Seq, which too is very close to the base value. But with SF = 10, the slowdowns for some of the short categories are now more than 10. An SF of 2 does performs very well for the short categories but for some of the long categories, the slowdown is double that of the base value. An SF value of 5 performs quite well for the short cases without affecting the long ones much.

## 6. Adaptive Selective Suspension with No Reservations (ASSNR)

From the graphs from the previous sections, it can be observed that the job categories with the highest slowdowns (e.g. VS-VW) with the NS schedule achieve significant reductions in their slowdowns. However, even with the SS scheme, their slowdown is still considerably higher than most of the other job categories. This is because, job in some categories inherently have a higher probability of waiting longer in the queue than another job with comparable xfactor from another job category. For example, consider a VS-VW job needing 300 processors, and a VS-Seq job in the queue at the same time. If both jobs have the same xfactor, the probability that the VS-Seq finds a running job to suspend is higher than the probability that the VS-VW job finds enough lower-pririty running jobs to suspend. Therefore, the average slowdown of  the VS-VW category tends to be higher than the VS-Seq category. This suggests that an approach to redress this inequity is to selectively increase the priority of jobs in those categories that end up with high slowdowns. This is implemented by boosting a job's priority based on the previous history of job slowdowns from that category. In this scheme, the priority of a job is dynamically boosted by the current average slowdown of jobs in that category.
This reduces the slowdown of jobs in that category. If the average slowdown of the jobs in that category reduces sufficiently, the boost factor will decrease and the slowdowns increase. The boost factor thus changes dynamically until a steady state is reached. From fig.4, it can be observed that the slowdowns of the VS-VW jobs and S-VW jobs, which had the highest slowdowns, are reduced significantly compared to the previous SSNR scheme.

Boost Factor = Current Category-Slowdown ^ Adaptivity Factor

Priority = Xfactor * Boost Factor

The boost factor was set to some power of the current slowdown of the job category. We experimented with an adaptive Power Factor of 1 and 2. In both cases, there was an improvement in slowdowns of the job categories that had the highest slowdowns with the SSNR scheme, with the improvement for AF=2 being slightly higher than for AF=1. At the same time, the slowdowns of the categories with the lowest slowdowns with the SSNR scheme became slightly worse, with AF=2 causing a greater deterioration than AF=1. We only present data for the case of AF=1 here.

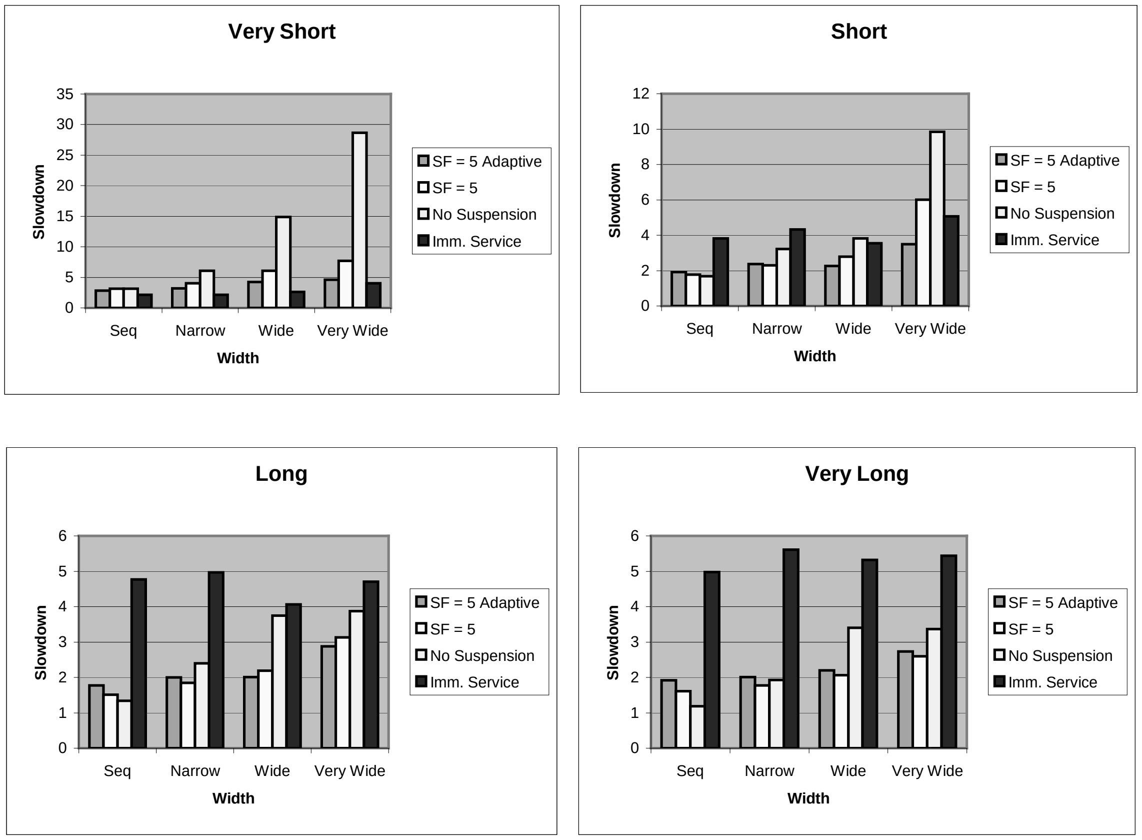

**Fig 4: Adaptive Selective Suspension**

## 7. Tuned Adaptive Selective Suspension with No Reservations (TASSNR)

With the schemes proposed so far, the implicit goal has been to minimize the slowdowns of all job categories to the extent possible. This assumes that in fact it is desirable to strive for equal slowdowns for all job categories. This is not necessarily the case in practice. It is very likely that supercomputer center policies are set up with the intent of ensuring that very long jobs have lower slowdowns than short jobs. This is because, a slowdown of 4 is much more likely to be tolerable for a 30 minute job than a 30-hour job. We next enhance the ASSNR scheme to incorporate selective tuning of slowdowns for specific job categories. This is done by appropriately scaling the current average slowdown of the targeted job categories by a tuning factor. The larger the tuning factor, the larger the boost to the priority of that job category. Thus, the the boost factor is now given by

Boost Factor = (Tuning Factor * Current Slowdown) ^ Adaptivity Factor

Suppose for example that a supercomputer center decides that the VS-VW category is especially important since many prototype runs for large production runs are tried out in that category. This might represent jobs that cannot be tested on smaller numbers of processors due to memory limitations. However, the total required runtimes are very short, and very quick turnaround is demanded. In such a case, a tuning factor such as 2 or 4 can be applied to the VS-VW job category. Fig.5 shows the impact of tuning. Note that the TASSNR scheme is being compared with the ASSNR scheme and not the SSNR scheme. It can be observed that the slowdown of the very short very wide jobs decrease by up to 40% without significant adverse effects on the other categories.

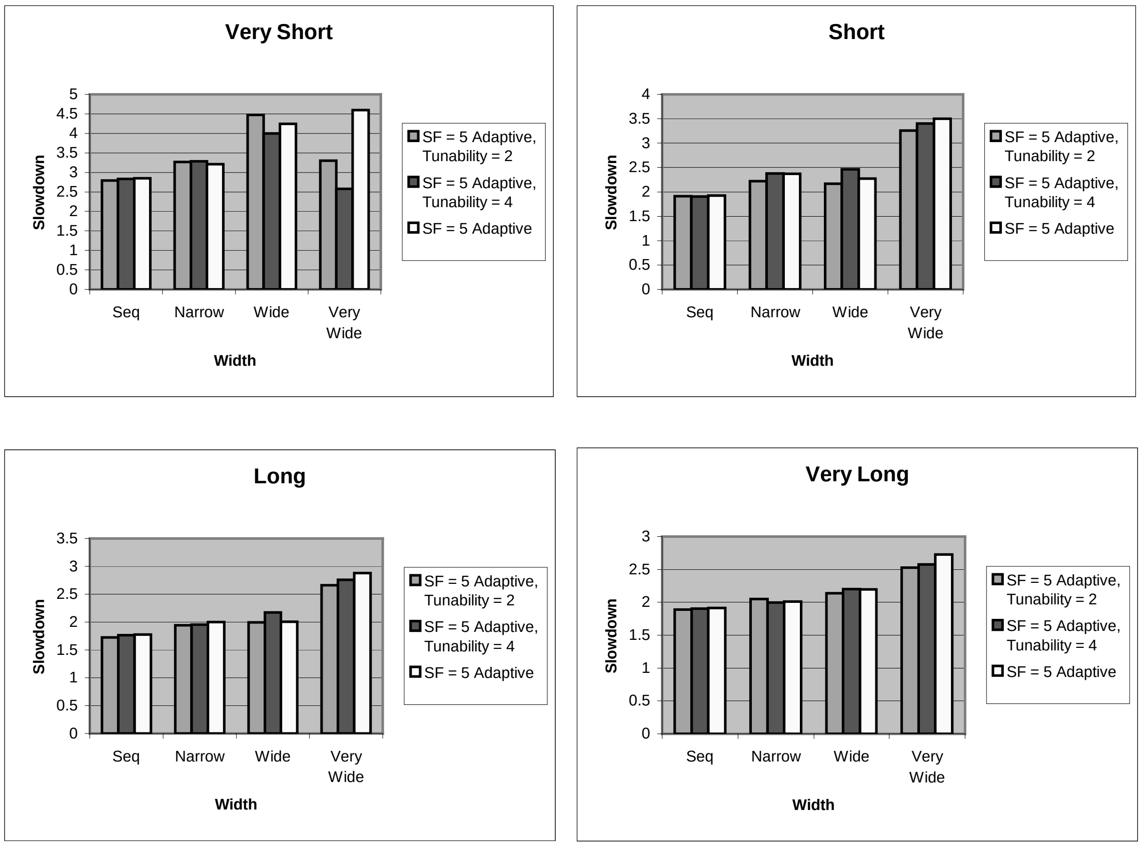

**Fig 5: Tuned Adaptive Selective Suspension**

## 8. Modeling of Job Suspension Overhead

We have so far assumed no overhead for pre-emption of jobs. In this section, we report on simulation results that incorporate overheads for job suspension. Since the job trace did not have information about job memory requirements, we considered the memory requirement of jobs to be random and uniformly distributed between 100MB and 1GB. The overhead for suspension is calculated as the time taken to write the main memory used by the job to the disk. Two memory transfer rates were considered, based on the following two extreme scenarios:

a) With a high-speed local disk for every node and every node consists of a single processor. In this case, the transfer rate for the processor was assumed to be 20 MBps.

b) With a commodity local disk for every node, with each node being a quad, the transfer rate per processor was assumed to be only 2 MBps.

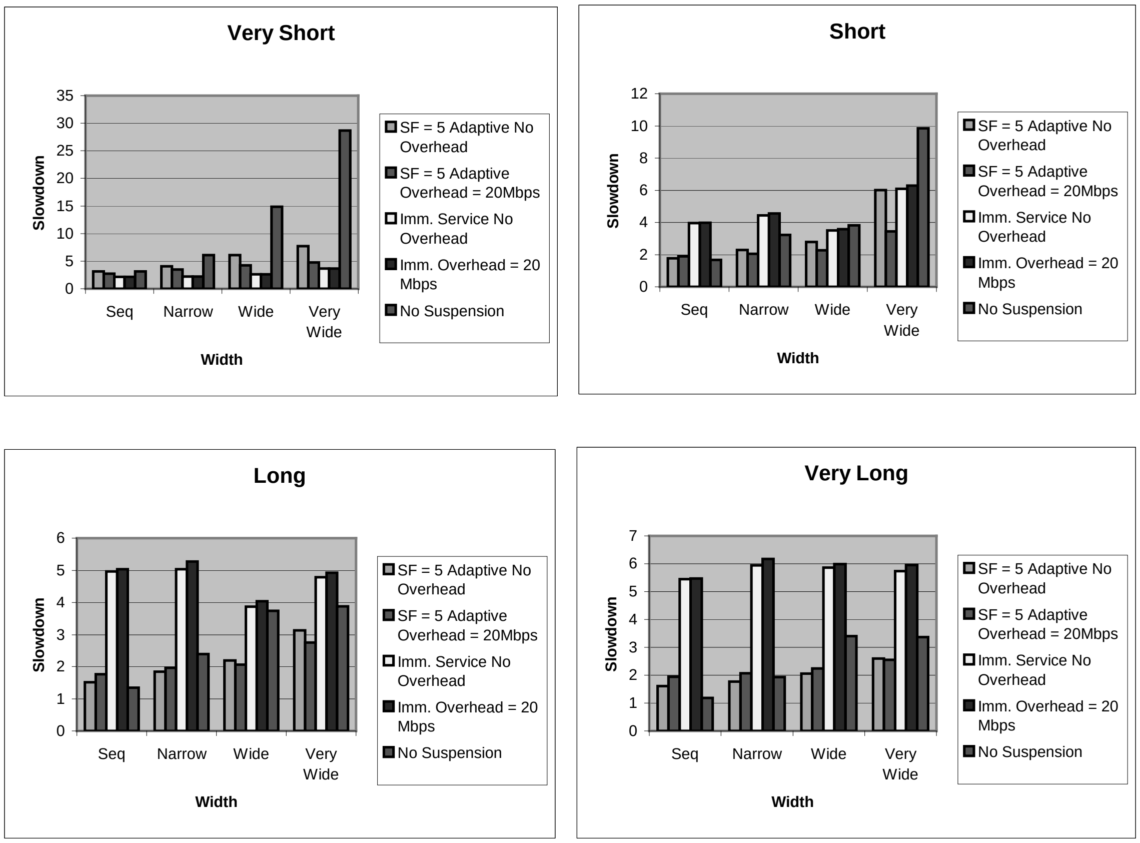

**Fig 6: Suspension Overhead of 20 Mbps**

From fig.7 it can be observed that the slowdowns for the IS go up considerably compared to ASSNR with an overhead of 2 Mbps. This is because the number of suspensions for IS is considerably larger than with ASSNR. Although he slowdowns for ASSNR also generally increase slightly when overhead is modeled, it is still considerably better than NS even with an assumed data rate of only 2 MBps.

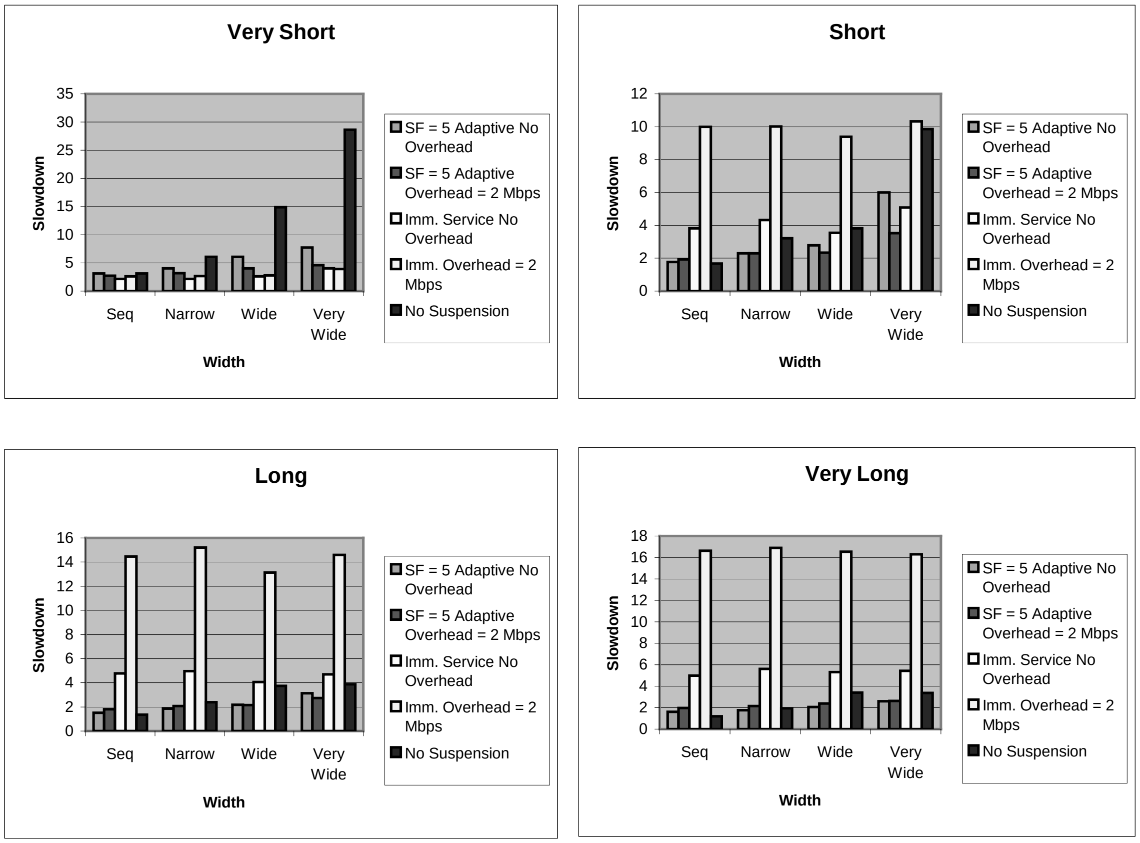

**Fig 7: Suspension Overhead of 2 Mbps**

## 9. Related Work

As stated earlier, although there is an abundance of work on trace based simulation studies of non-preemptive scheduling schemes, there is very little work on evaluation of pre-emptive scheduling strategies for scheduling of parallel jobs. Chiang and Vernon [3] evaluate a preemptive policy with limited preemption to provide immediate service to each arriving job. This scheme uses the current slowdown as opposed to the xfactor in choosing the job to suspend. Chiang et al [2] examine the run-to-completion policy with a suspension policy that allows a job to be suspended at most once. Both these approaches limit the number of suspensions while we use the suspension factor to control the rate of suspensions without limiting the number of times a job can be suspended. Deng et al [1] present a theoretical analysis of the performance of the DEQ scheme by doing a comparative analysis based on the mean completion time. Also in each of these papers the metric used to study the system performance is the overall mean response time, the overall average slowdown and the overall mean completion time respectively. In this paper we have studied performance with respect to different job categories rather than simply the overall average slowdown over all jobs.

## 8. Conclusions

In this paper, we have explored the issue of pre-emptive scheduling of parallel jobs, using a job trace from a supercomputer center. We have proposed a tunable, adaptive selective suspension scheme and demonstrated that it provides significant improvement in the average slowdown of several job categories. It was also shown to provide better slowdown for most job categories over a previously proposed Immediate Service scheme. We also modeled the effect of overheads for job suspension, showing that even under stringent assumptions about available bandwidth to disk, the proposed scheme provides significant benefits over non-preemptive scheduling and the Immediate Service strategy.

## References

[1] Xiaotic Deng, Nian Gu, Tim Brecht, KaiCheng Lu. Preemptive scheduling of parallel jobs on multiprocessors. In Seventh Annual ACM--SIAM Symposium on Discrete Algorithms, pages 159--167, Atlanta, Georgia, January 1996.

[2] Su-Hui Chiang, Rajesh K. Mansharamani, Mary K. Vernon. Use of application characteristics and limited preemption for run to completion parallel processor scheduling policies. Proc. ACM SIGMETRICS Conf. on measurement and modeling of computer systems, Nashville May 1994, pp.33-44.

[3] Su-Hui Chiang, Mary K. Vernon. Production job scheduling for parallel shared memory systems. IPDPS 2001.

[4] Dejan Perkovic, Peter J. Keleher. Randomization and speculation and adaptation in batch schedulers. Proc.2000 ACM/IEEE super computing conf. Dallas, Nov. 2000.

[5] Ahuva W. Mu'alem, Dror G Fietelson. Utilization, predictability, workloads and user runtime estimates in scheduling the IBM SP2 with backfilling. Proc. 12th Int'l. parallel processing symp. Orlando, March 1998, pp. 542-546.

[6] Intel Corp., iPSC/860 multi-user accounting, control and scheduling utilities manual. Order no. 312261-002, May 1992

[7] S.T. Leutenneger and M.K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. Proc. ACM SIGMETIRCS Conf. on measurement and modeling of computer systems. Boulder, May 1990, pp.226-236.

[8] J. Skovira, W. Chan, H. Zhou and K. Lifka. The EASY-Loadleveller API Project. Proc. 2nd Workshop on Job Scheuling Strategies for Parallel Processing, Hobolulu, Apr. 1996, pp. 41-47. Lecture Notes in Comp. Sci. Vol. 1162, springer-Verlag.

[9] E.W. Parsons and K.C. Sevcik. Implementing multiprocessor scheduling disciplines. Proc. 3rd workshop on job scheduling strategies for parallel processing, Geneva, Apr. 1997, pp. 166-192.

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

[11] O. Arndt, B. Friesleben, T. Keilmann, and F. Thilo. A Comparative study of online scheduling algorithms for networks of workstations. Cluster computing 2000.

[12] P. Holenarsipur, V. Yarmolenko, J. Duato, D. K. Panda and P. Sadayappan, ``Characterization and Enhancement of Static Mapping Heuristics for Heterogeneous Systems," Proceedings of Seventh Intl. Conf. on High Performance Computing, December 2000.

[13] K. Sevcik. Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems. To appear in Performance Evaluation, special issue on the performance modeling of parallel processing systems.

[14] J. Zahorjan and C. McCann. Processor Scheduling in Shared Memory Multiprocessors. Proc. 1990 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, Boulder, CO, May 1990, 214-225.

[15] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik and P. Wong. Theory and practice in parallel job scheduling, in job scheduling strategies for parallel processing, D.G. Feitelson and L. Rudolph, eds., Lecture Notes in Computer Science 949, Springer-Verlag, Berlin , NewYork, Heidelberg, 1995, pp.337-360.

[16] D. Jackson. Core Algorithms of the Maui Scheduler, Technical Document, 2001. www.supercluster.org

[17] D.G. Feitelson. Logs of Real Parallel Workloads from Production Systems. http://www.cs.huji.ac.il/labs/parallel/workload/logs.html