Selective Preemption Strategies for Parallel Job Scheduling

A Thesis

Presented in Partial Fulfillment of the Requirements for
the Degree Master of Science in the
Graduate School of The Ohio State University

By

Rajkumar Kettimuthu, B.E.

* * * * *

The Ohio State University

2003

Master's Examination Committee:

P. Sadayappan, Adviser

Mario Lauria

Approved by

____

Adviser
Department of Computer
and Information Science

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 a tunable selective-suspension strategy, and demonstrate its effectiveness. We also present new insights into the effect of pre-emptive scheduling on different job classes and deal with the impact of suspensions on the worst-cases.

Dedicated to my parents

ACKNOWLEDGMENTS

I thank my adviser Dr. P Sadayappan for all the help, support and encouragement that he provided to me. He was a great source of inspiration for me. He was very helpful during the paper submissions. It was a great and wonderful experience working with him.

I thank Dr. Mario Lauria for serving on my thesis committee. He was very understanding and helped me a lot in many occasions.

I thank my friends Srividya and Vijay for helping me with my research. I thank my roommates Ramki, Chala, Niranjan and Gopal for their co-operation and understanding.

VITA

March 16, 1978	Born - Dharapuram, India
1999	B.E. Computer Science and Engineering
July 1999-August 2000	Network Engineer, Cisco's Offshore development Center, India.
Fall 2000-Winter 2001	Graduate Teaching Associate, Ohio State University.
Spring 2001-Spring 2002	Graduate Research Associate, Ohio State University.
Summer 2002-Fall 2002	Research Aide, Argonne National Laboratory.
Winter 2003-Present	Graduate Teaching Associate, Ohio State University.

PUBLICATIONS

Research Publications

R. Kettimuthu, V. Subramani, S. Srinivasan, T. B. Gopalsamy, D. K. Panda and P. Sadayappan "Selective Preemption Strategies for Parallel Job Scheduling". *Proceedings of the 2002 International Conference on Parallel Processing*, 602–610, Aug. 2002.

S. Srinivasan, R. Kettimuthu, V. Subramani and P. Sadayappan "Selective Reservation Strategies for Backfill Job Scheduling". *8th Workshop on Job Scheduling Strategies for Parallel Processing*, July 2002.

V. Subramani, R. Kettimuthu, S. Srinivasan and P. Sadayappan "Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests". *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, 359–366, July 2002.

S. Srinivasan, R. Kettimuthu, V. Subramani and P. Sadayappan "Characterization of Backfilling Strategies for Parallel Job Scheduling". *Proceedings of the ICPP-2002 Workshops*, 514–519, Aug. 2002.

V. Subramani, R. Kettimuthu, S. Srinivasan, J. Johnson and P. Sadayappan "Selective Buddy Allocation for Scheduling Parallel Jobs on Clusters". *Proceedings of the IEEE International Conference on Cluster Computing*, Sep. 2002.

S. Srinivasan, V. Subramani, R. Kettimuthu, P. Holenarsipur and P. Sadayappan "Effective Selection of Partition Sizes for Moldable Scheduling of Parallel Jobs". *Proceedings of the 9th International Conference on High Performance Computing*, Dec. 2002.

FIELDS OF STUDY

Major Field: Computer and Information Science

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Although theoretical results on the effect of pre-emptive scheduling strategies in reducing average job turn-around time have been well established [10], [12], [9], [30], [11] 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 parallel jobs [28], [19], [34], [7], [20], [23], [2], [37], [26], [14], [24] little research has been reported on empirical evaluation of preemptive scheduling strategies using real job logs [4], [5], [29], [25]. 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 T_l. If after time t, a short job arrives with runtime T_s. If the short job were run after completion of the long job, the average job turnaround time would be $\frac{(T_l+(T_l+T_s-t))}{2}$, or $T_l + \frac{(T_s-t)}{2}$. Instead, if the long job were suspended when the short job arrived, the turnaround times of the short and long jobs would be T_s and $(T_s + T_l)$ respectively, giving an average of $T_s + \frac{T_l}{2}$. The average turnaround time with suspension is less if $T_s < T_l - t$, i.e. the remaining runtime of the running job is greater than the runtime of the waiting job.

The suspension criterion has to be chosen carefully to ensure freedom from starvation. Also, the suspension scheme should bring down the average turnaround times without increasing the worst case turnaround times. Even though theoretical results [10], [12], [9], [30], [11] have established that preemption improves the average turnaround time, it is important to perform evaluations of preemptive scheduling schemes using realistic job mixes derived from actual job logs from supercomputer centers, to understand the effect of suspension on various categories of jobs. The primary contributions of this work are:

- The development of a selective-suspension strategy for pre-emptive scheduling of parallel jobs, and
- Characterization of the significant variability in the average job turnaround time for different job categories, and
- The study of the impact of suspension on the worst case turnaround times of various categories and development of a tunable scheme to improve worst case turnaround times.

1.1 Organization of Thesis

The thesis is organized as follows. Chapter 2 provides background on parallel job scheduling and motivation for doing preemptive job scheduling. Chapter 3 presents the proposed selective preemption strategies and evaluates their performance under the assumption of accurate estimation of job run times. Chapter 4 studies the impact of inaccuracies in user estimates of run time on the selective preemption strategies. It also discusses the overhead model for job suspension and restart and evaluates the proposed schemes in the presence of overhead. The performance of the selective preemption strategies under different load conditions is also presented in this chapter. Chapter 5 summarizes the results of this thesis and provides pointers to future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

Scheduling of parallel jobs is usually viewed in terms of a 2D chart with time along one axis and the number of processors along the other axis. Each job can be thought of as a rectangle whose width is the user estimated run time and height is the number of processors requested. Parallel job scheduling strategies have been widely studied in the past [1], [3], [8], [21], [33], [22], [6]. The simplest way to schedule jobs is to use the First-Come-First-Served (FCFS) policy. This approach suffers from low system utilization due to fragmentation of the available processors. Consider a scenario where there are a few jobs running in the system and the next queued job requires all the processors in the system. An FCFS scheduler would leave the currently free processors idle, even if there are queued jobs requiring only a small number of processors waiting. Some solutions to this problem are to use dynamic partitioning [27] or gang scheduling [15]. However, these schemes have practical limitations. An alternate approach to improving system utilization is backfilling.

2.1 Backfilling

Backfilling was developed for the IBM SP1 parallel supercomputer as part of the Extensible Argonne Scheduling sYstem (EASY) [26] and has been implemented in several production schedulers [18], [32]. Backfilling works by identifying "holes" in the 2D schedule and moving forward smaller jobs that fit those holes. With backfilling, users are required to provide an estimate of the length of the jobs submitted for execution. This information is used by the scheduler to predict when the next queued job will be able to run. Thus, a scheduler can determine whether a job is sufficiently small to run without causing delay to previously queued jobs.

It is desirable that a scheduler with backfilling support two conflicting goals. On the one hand, it is desirable to move as many short jobs forward, in order to improve utilization and responsiveness. On the other hand, it is also important to avoid starvation of large jobs, and in particular, to be able to predict when each job will run. There are two common variants to backfilling - conservative and aggressive(EASY) that attempt to balance these goals in different ways.

2.1.1 Conservative Backfilling

With conservative backfill, every job is given a reservation (start time guarantee) when it enters the system. A smaller job is allowed to backfill only if it does not delay any previously queued job.

Thus, when a new job arrives, the following allocation procedure is executed by a conservative backfill scheduler. Based on the current knowledge of the system state, the scheduler finds the earliest time at which a sufficient number of processors are available to run the job for a duration equal to the user estimated run time. This is called the anchor point. Thus the anchor point of a job is the earliest time at which the job can start without violating any previous commitments. The scheduler then updates the system state to reflect the allocation of processors to this job starting

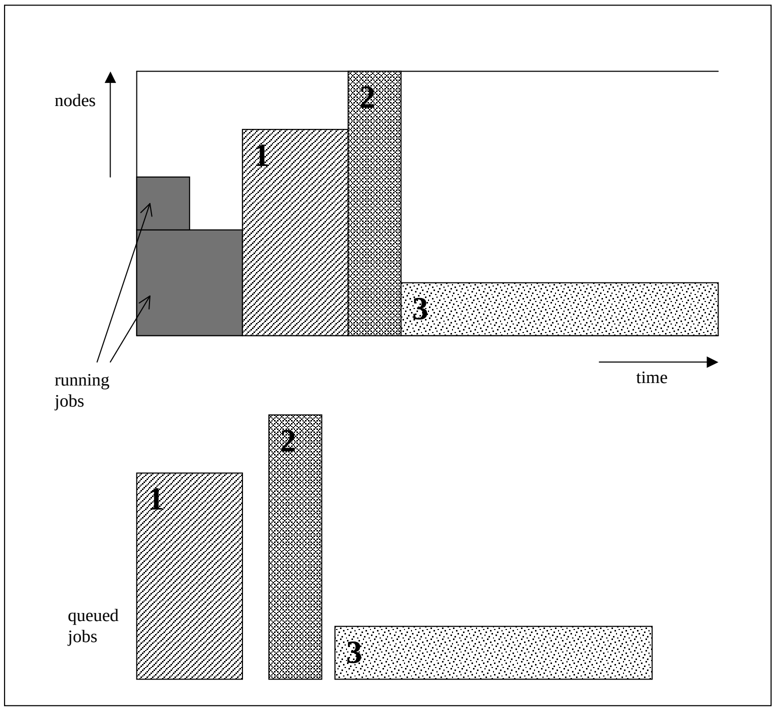

Figure 2.1: Conservative Backfilling.

from its anchor point. If the job's anchor point is the current time, the job is started immediately.

An example is given in Figure 2.1. The first job in the queue does not have enough processors to run, so a reservation is made for it at the anticipated termination time of the longer running job. Similarly, the second queued job is given a reservation at the anticipated termination time of the first queued job. Although enough processors are available for the third queued job to start immediately, it would delay the second job; therefore, the third job is given a reservation after the second queued job's anticipated termination time.

Thus in conservative backfill, jobs are assigned a start time when they are submitted, based on the current usage profile. But they may actually be able to run sooner because previous jobs terminated earlier than expected. In this scenario, the original schedule is compressed. This is done by releasing the existing reservations one by one, when a running job terminates, in the order of increasing reservation start time guarantees and attempting backfill for the released job. If due to early termination of some job, "holes" of the right size are created for this job, then it gets an earlier reservation. In the worst case, the released job would be reinserted in the same position it held previously.

Obviously there is no danger of starvation, as a reservation is made for each job when it is submitted.

2.1.2 Aggressive Backfilling

Conservative backfilling moves jobs forward only if they do not delay any previously queued job. Aggressive backfilling takes a more aggressive approach, and allows jobs to skip ahead provided they do not delay the job at the head of the queue. The objective is to improve the current utilization as much as possible, subject to some consideration for queue order. The price is that execution guarantees cannot be made, because it is impossible to predict how much each job will be delayed in the queue.

An aggressive backfill scheduler scans the queue of waiting jobs and allocates processors as requested. The scheduler gives a reservation guarantee to the first job in the queue that does not have enough processors to start. This reservation is given at the earliest time at which the required processors are expected to become free, based on the current system state. The scheduler then attempts to backfill the other queued jobs. To be eligible for backfill, a job must require no more than the currently

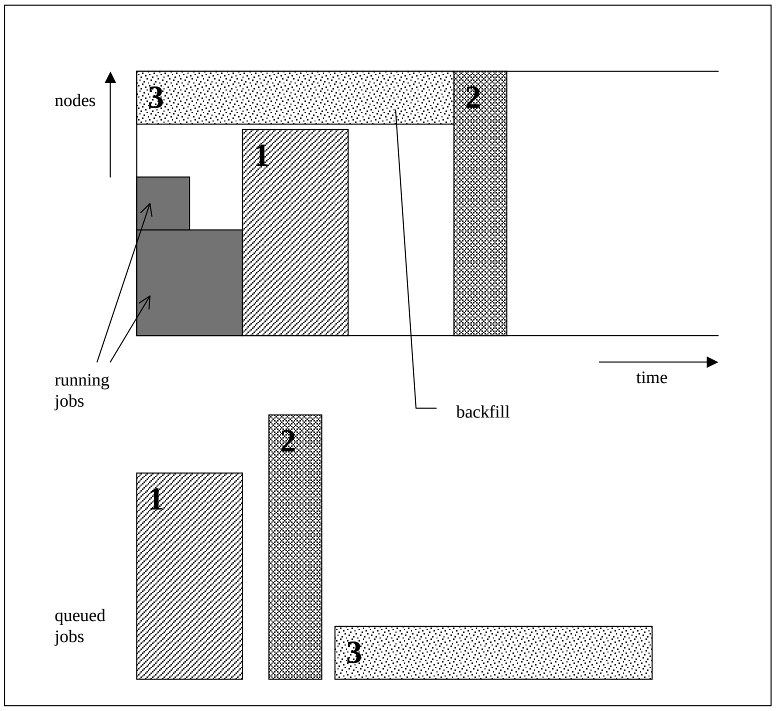

Figure 2.2: Aggressive Backfilling.

available processors, and in addition it must satisfy either of two conditions that guarantee that it will not delay the first job in the queue:

- it must terminate by the time the first queued job is scheduled to commence, or
- it must use no more nodes than that are free at the time the first queued job is scheduled to start.

Figure 2.2 shows an example.

2.2 Metrics

Some of the common metrics used to evaluate the performance of scheduling schemes are the average turnaround time and the average bounded slowdown. We use these metrics for our studies. The bounded slowdown [16] of a job is defined as follows:

Bounded Slowdown = (Wait time + Max(Run time, 10))/Max(Runtime, 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, our suspension criteria is based on xfactor, which increases rapidly for short jobs and gradually for long jobs.

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

2.3 Related Work

Although pre-emptive scheduling is universally used at the operating system level to multiplex processes on single-processor systems and shared-memory multi-processors, it is rarely used in parallel job scheduling. A large number of studies have addressed the problem of parallel job scheduling (see [16] for a survey of work on this topic), but most of them address non-preemptive scheduling strategies. Further, most of the work on pre-emptive scheduling of parallel jobs considers the jobs to be malleable [10], [29], [31], [35], i.e. the number of processors used to execute the job is permitted to vary dynamically over time.

In practice, parallel jobs submitted to supercomputer centers are generally rigid, i.e. the number of processors used to execute a job is fixed. Under this scenario, the various schemes proposed for a malleable job model are inapplicable. Very few studies have addressed pre-emptive scheduling under a model of rigid jobs, where the pre-emption is "local", i.e. the suspended job must be re-started on exactly the same set of processors on which they were suspended.

In [5], a pre-emptive scheduling strategy called the "Immediate Service (IS)" scheme was evaluated for the shared-memory systems. 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. The IS strategy was shown to significantly decrease the average job slowdown for the traces simulated. A potential shortcoming of the IS scheme is that its preemption decisions are not in any way reflective of the expected runtime of a job. The IS scheme can 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. However, no information was provided on how different job categories were affected. Chiang et al [4] 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. In [29], the design and implementation of a number of multiprocessor preemptive scheduling disciplines are discussed. They study the effect of preemption under the models of rigid, migratable and malleable jobs. They conclude that the preemption scheme that they propose may increase the response time for the model of rigid jobs.

So far, only very few simulation based studies have been done on preemption strategies for clusters. With no process migration, the distributed memory systems impose an additional constraint that the suspended jobs should get the same processors when it restarts. In this thesis, we propose tunable suspension strategies for parallel job scheduling in environments where process migration is not feasible.

2.4 Workload Characterization

We perform simulation studies using a locally developed simulator with workload logs from a supercomputer center. Most supercomputer centers keep a trace file as a record of the scheduling events that occur in the system. This file contains information about each job submitted and its actual execution. Typically the following data is recorded for each job

- Name of job, user name, etc
- Job submission time
- Job resources requested, like memory, processors, etc
- User estimated run time
- Time when job started execution
- Time when job finished execution

From the collection of workload logs available from Feitelson's archive [13], a subset of the CTC workload trace, was used to evaluate the various schemes. The trace was logged from a 430 node IBM SP2 at the Cornell Theory Center. Under normal load, with the standard non-preemptive aggressive backfilling strategy, using FCFS as the scheduling priority, the utilization was 51 percent. 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 accurate 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 job suspension and restart is negligible and then studied the influence of the overhead.

Any analysis that is based on the aggregate slowdown / turnaround time 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 / turnaround time for each category.

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(VS), Short(S), Long(L) and Very Long(VL) and four categories based on the number of processors requested - Sequential(Seq), Narrow(N), Wide(W) and Very Wide(VW). The criteria used for job classification is shown in Table 2.1. The distribution of jobs in the trace, corresponding to the sixteen categories is given in Table 2.2.

Table 2.3 shows the average slowdowns for the different job categories under non-preemptive aggressive backfilling strategy. The overall slowdown was 3.58. Even though the overall slowdown is low, from the table, it can be observed that some of

	1 Proc	2-8Procs	9-32Procs	>32 Procs
0-10min	VS Seq	VS N	VS W	VS VW
10min-1hr	S Seq	S N	S W	S VW
1hr-8hr	L Seq	L N	L W	L VW
>8hr	VL Seq	VL N	VL W	VL VW

Table 2.1: Job categorization criteria

	1 Proc	2-8 Procs	9-32 Procs	>32 Procs
0-10min	14%	8%	13%	9%
10min-1hr	18%	4%	6%	2%
1hr-8hr	6%	3%	9%	2%
>8hr	2%	2%	1%	1%

Table 2.2: Job distribution by category

	1 Proc	2-8 Procs	9-32 Procs	>32 Procs
0-10min	2.6	4.76	13.01	34.07
10min-1hr	1.26	1.76	3.04	7.14
1hr-8hr	1.13	1.43	1.88	1.63
>8hr	1.03	1.05	1.09	1.15

Table 2.3: Average slowdown for various categories with non preemtive scheduling

the Very Short categories have slowdowns as high as 34. Preemptive strategies aim at reducing the high average slowdowns for the short categories with little degradation to long jobs.

CHAPTER 3

SELECTIVE SUSPENSION

We first propose a preemptive scheduling scheme, called the Selective Suspension (SS) scheme, where an idle job can pre-empt a running job if its priority is sufficiently higher than the running job. An idle job 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 suspension threshold of a candidate idle job to the suspension threshold of a running job for preemption to occur. The suspension threshold used is the xfactor of the job.

3.1 Theoretical Analysis

Let T_1 and T_2 be two tasks submitted to the scheduler at the same time. Let both tasks be of the same length and require the entire system for execution, with the system being free when the two tasks are submitted. Let 's' be the suspension factor. Before starting, both tasks have a suspension threshold of 1. The suspension threshold of a task remains constant when the task executes and increases when the task waits. One of the two tasks, say T_1, will be started instantaneously. The other task, say T_2, waits until its suspension threshold τ_2 becomes 's' times the threshold of T_1 before it can preempt T_1. Now T_1 waits until its suspension thresold τ_1 becomes ‘s’ times τ_2 before it can preempt T_2. This occurs repeatedly. The optimal value for SF to restrict the number of repeated suspensions by two similar tasks arriving at the same time can be obtained as follows:

Let τ_w represent the suspension threshold of the waiting job and τ_r represent the suspension threshold of the running job.

Condition for the first suspension:

$\tau_w =$ s

The preemption swaps the running job and the waiting job. So after the preemption, $\tau_w = 1$ and $\tau_r =$ s.

Condition for second suspension:

$\tau_w = s * \tau_r$

$\tau_w = s^2$

Similarly, the condition for n^{th} suspension is $\tau_w = s^n$.

The lowest value of s for which at most n suspensions occur is given by,

$\tau_w = s^{n+1}$, when the running job completes.

When the running job completes,

$\tau_w = \frac{waittime+runtime}{runtime}$ i.e $\tau_w = 2$; since wait time of the waiting job = the run time of the running job

$s^{n+1} = 2$ and s = $2^{\frac{1}{n+1}}$

Thus, if the number of suspensions has to be 0, then s = 2. For at most 1 suspension, we get s as $\sqrt{2}$. With s=1, the number of suspensions is very large, only bounded by the granularity of the preemption routine. With all jobs having equal length, any suspension factor greater than 2 will not result in any suspension and will be same as the suspension factor 2. However, with jobs of varying length, the number of suspensions reduces with higher suspension factors. Thus, in order to avoid thrashing and to reduce the number of suspensions, we use different suspension factors between 1.5 and 5 in evaluating our schemes.

3.2 Preventing Starvation without Reservation Guarantees

An idle job can preempt a running job only if its threshold is at least SF times greater than the threshold of the running job. All the idle jobs that are able to find the required number of processors by suspending lower threshold running jobs are selected for execution by preempting the corresponding jobs. 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 thereby avoiding starvation. But start time guarantees do not make much significance in a preemptive context. Even if we give start time guarantees for the jobs in the idle queue, they are not guaranteed to run to completion. Since the SS strategy uses the expected slowdown (xfactor) as the suspension threshold, 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(s) and begin execution. Therefore, it is possible to run the backfill algorithm without the usual reservation guarantees. So, we remove the guarantees for all our preemption schemes.

Further, jobs 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 VW job needing 300 processors, and a Seq job in the queue at the same time. If both jobs have the same xfactor, the probability that the Seq job finds a running job to suspend is higher than the probability that the VW job finds enough lower threshold running jobs to suspend. Therefore, the average slowdown of the VW category will tend to be higher than the Seq category. To redress this inequity, we impose a restriction that the number of processors requested by a suspending job should be at least half of the number of nodes requested by the job that it suspends, thereby preventing the wide jobs from being suspended by the narrow jobs. The scheduler periodically (after every minute) invokes the preemption routine.

3.3 Algorithm

Let τ_i be suspension threshold for a task T_i which requests n_i processors. Let N_i represent the set of processors allocated to T_i. Let F_t represent the set of free processors and f_t represent the number of free processors at time 't' when the preemption is attempted.

Let candidates(T_i) represent the set of tasks that can be preempted by task T_i.

candidates(T_i) = $\{ T_j : \tau_i > \text{SF} * \tau_j$ and $\frac{n_j}{n_i} < 2 \}$

T_i can be scheduled by preempting one or more tasks in candidates(T_i) if and only if

$n_i \leq (\Sigma n_j + f_t) \; \forall T_j \; \epsilon$ candidates(T_i)

If T_i is itself a previously suspended task attempting reentry, the processor restriction also applies. So the above condition becomes:

candidates(T_i) = $\{ T_j : \tau_i > \text{SF} * \tau_j$ and $\frac{n_j}{n_i} < 2$ and $N_i \cap N_j \neq \emptyset \}$

T_i can be scheduled by preempting one or more tasks in candidates(T_i) if and only if

$n_i \leq (\Sigma n_j + f_t) \; \forall T_j \; \epsilon$ candidates(T_i)

and

$N_i \subseteq F_t \cup N_j \; \forall T_j \,\epsilon$ candidates(T_i)

For both of the above scenarios, T_i preempts tasks in candidates(T_i) as given by the following condition: The set of tasks suspended by T_i is

P = $\{T_j : T_j \,\epsilon$ candidates(T_i) and $n_i \leq f_t + \Sigma n_j\}$

and $((\Sigma n_k \; (\forall T_k \,\epsilon$ P$))$ - $(n_m : T_m \,\epsilon$ P$)) < n_i$

3.4 Results

We compare the SS scheme run under various suspension factors with the No-Suspension(NS) scheme with aggressive backfilling and the IS scheme. From Figure 3.1 and Figure 3.2, 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 34 for the NS scheme to under 3 for SS with SF=2 (turnaround time is also reduced from 4000 to 600). For VS and S length categories, lower SF results in lowered slowdown / turnaround time. 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 / turnaround time 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 No-Suspension(NS) scheme, the SS scheme provides significant benefits for VS and S categories, a slight improvement for most of the Long categories, but is slightly worse for the VL categories.

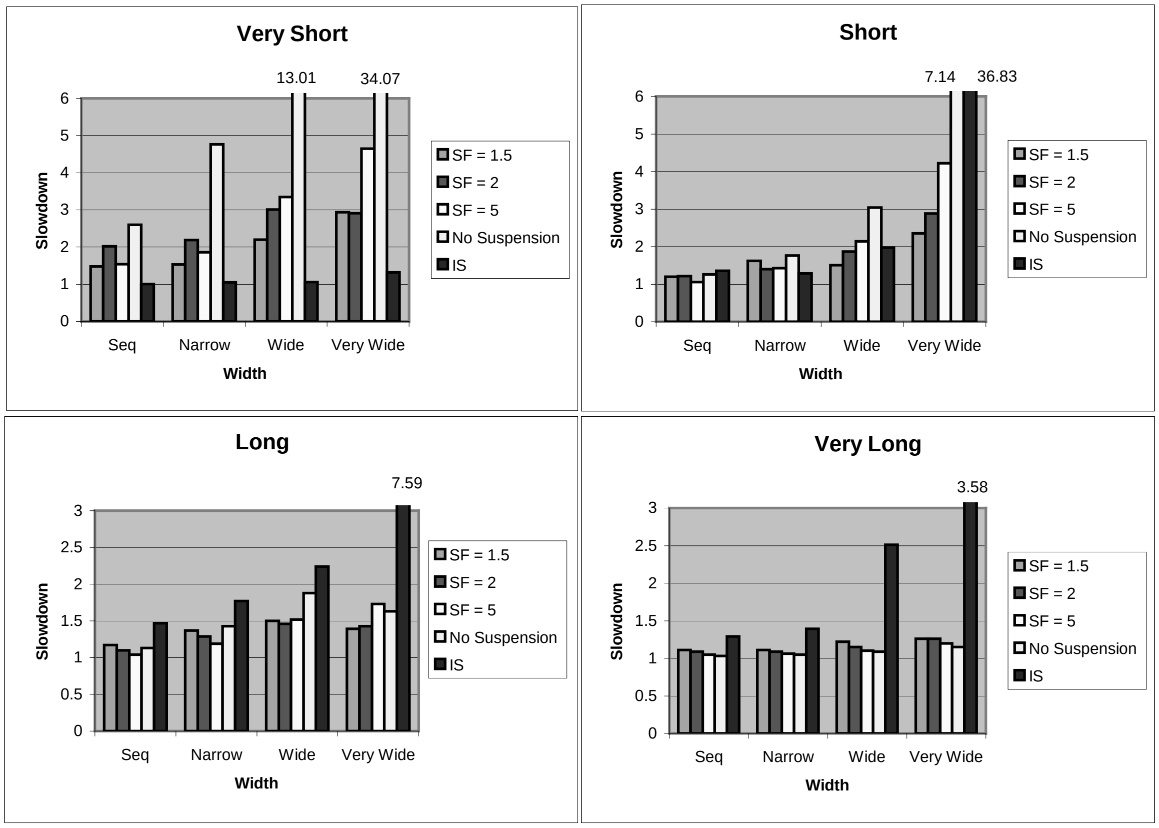

Figure 3.1: Comparison of the average slowdowns of the SS scheme with the NS and IS schemes. Compared to NS, SS provides significant benefit for the VS, S, W and VW categories, slight improvement for most of L categories, but a slight deterioration for the VL categories. Compared to IS, SS performs better for all the categories except for the VS categories.

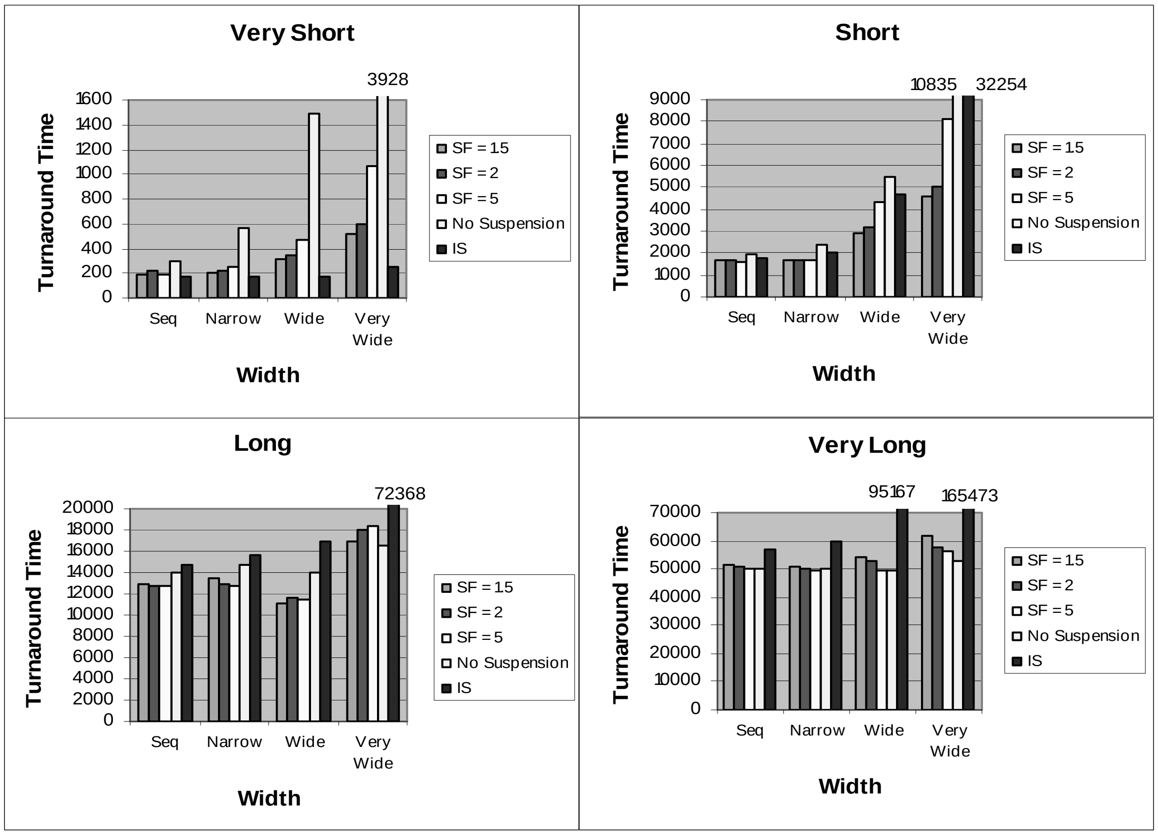

Figure 3.2: Comparison of the average turnaround times of the SS scheme with the NS and IS schemes. The trends observed here are similar to the the trends observed with the average slowdown metric.

The performance of the IS scheme is very good for the VS category. It is better than the SS scheme for the VS length category and worse for the other categories. Even though, the overall slowdown for IS, is considerably less than the No-Suspension scheme, it is not better than SS. Moreover, in IS the VW and VL categories get significantly worse.

3.5 Tunable Selective Suspension (TSS)

From the graphs from the previous section, it can be observed that the SS scheme significantly improves the average slowdown and turnaround time of various job categories. But from the practical point of view, the worst case slowdowns / turnaround times are much important. A scheme that improves the average case slowdowns / turnaround times for most of the categories and makes the worst case slowdown / turnaround time for the long categories worse, is not an acceptable scheme. For example, a delay of 1 hour for a 10 minute job (slowdown = 7) is tolerable whereas a slowdown of 7 for a 24 hour job is unacceptable.

In Figure 3.3, we compare the worst case slowdowns for SF=2 with the worst case slowdowns of the NS scheme and the IS scheme. It can be observed that the worst case slowdowns of the SS scheme is much better than the NS scheme in most of the cases. But the worst case slowdowns for some of the long categories is worse than the NS scheme. Even though the worst case for SS is less than that of NS, the absolute worst case slowdowns are much higher than the average slowdown for some of the short categories. For the IS scheme, the worst case slowdowns for the very short categories is less and is very high for the long ones, which is unacceptable. Figure 3.4 compares the worst case turnaround times for the SS scheme with worst

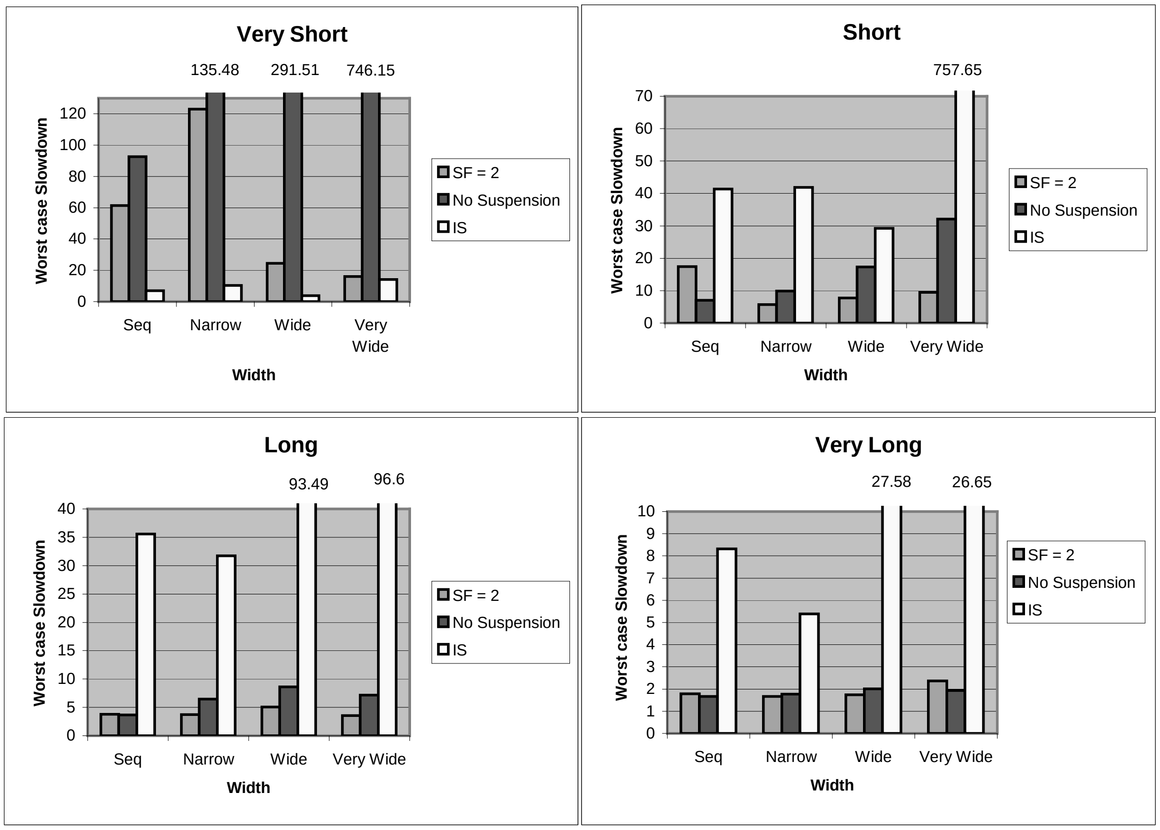

Figure 3.3: Comparison of the worstcase slowdowns of the SS scheme with the NS and IS schemes. SS is much better than NS for most of the categories and is slightly worse for some of the VL categories. Compared to IS, SS is much better for all the categories except for the VS categories.

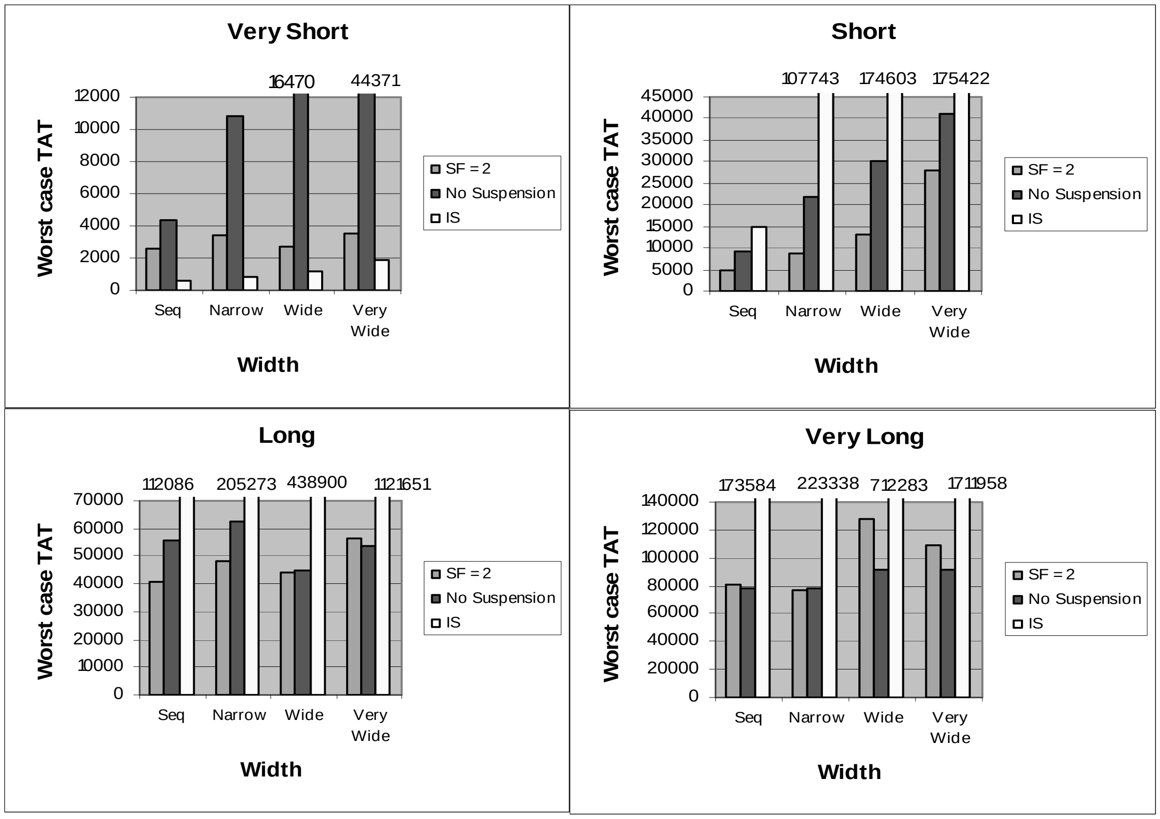

Figure 3.4: Comparison of the worstcase turnaround times of the SS scheme with the NS and IS schemes. SS is much better than NS for most of the categories and is slightly worse for some of the VL categories. Compared to IS, SS is much better for all the categories except for the VS categories.

case turnaround times of the NS scheme and the IS scheme. Even though the trends observed here are similar to that of worst case slowdowns, the categories where SS is the best with respect to worst case turnaround time are not same as the categories for which SS is the best with respect to worst case slowdowns. This is because a job with the worst case turnaround time need not be the one with worst case slowdown. We propose a tunable scheme to improve the worst case slowdowns / turnaround times without losing the improvement in the average slowdowns / turnaround times. This is done by controlling the variance in the slowdowns / turnaround times by associating a limit with each job. Preemption of a job is disabled when its threshold exceeds this limit. This limit is set to 1.5 times the average slowdown of the category that the job belongs to.

3.5.1 Control of variance

Task T_i preempts tasks in candidates(T_i) as given by the following condition:

The set of tasks suspended by T_i is

P = { T_j:T_j ϵ candidates(T_i) and $n_i \leq f_t$+Σn_j and $\tau_j \leq 1.5$ *SD_{avg}(category(T_j))}

and ((Σn_k ($\forall T_k$ ϵ P)) - (n_m : T_m ϵ P)) $< n_i$

where SD_{avg}(category(T_j)) represents the average slowdown for the job category to which T_j belongs.

3.5.2 Results

Figure 3.5 and Figure 3.6 show the result of this tunable scheme. It improves the worst case slowdowns for some long categories (VL W, VL VW, L N) and some short categories (VS Seq, VS N, S Seq) without affecting the worst case slowdowns of the other categories. It improves the worst case turnaround times for most of the

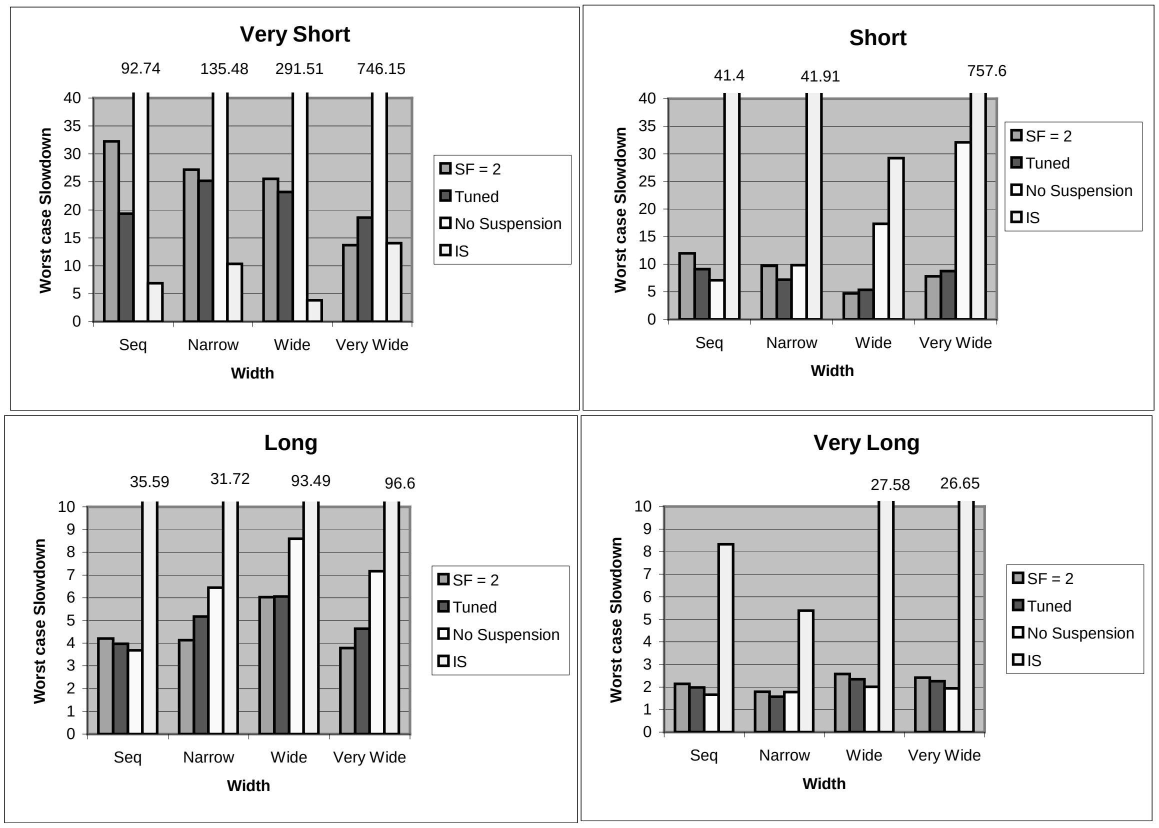

Figure 3.5: Comparison of the worstcase slowdowns of the TSS scheme with the SS, NS and IS schemes. TSS improves the worstcase slowdown for the VL categories and some of the S categories without adversely affecting the other categories.

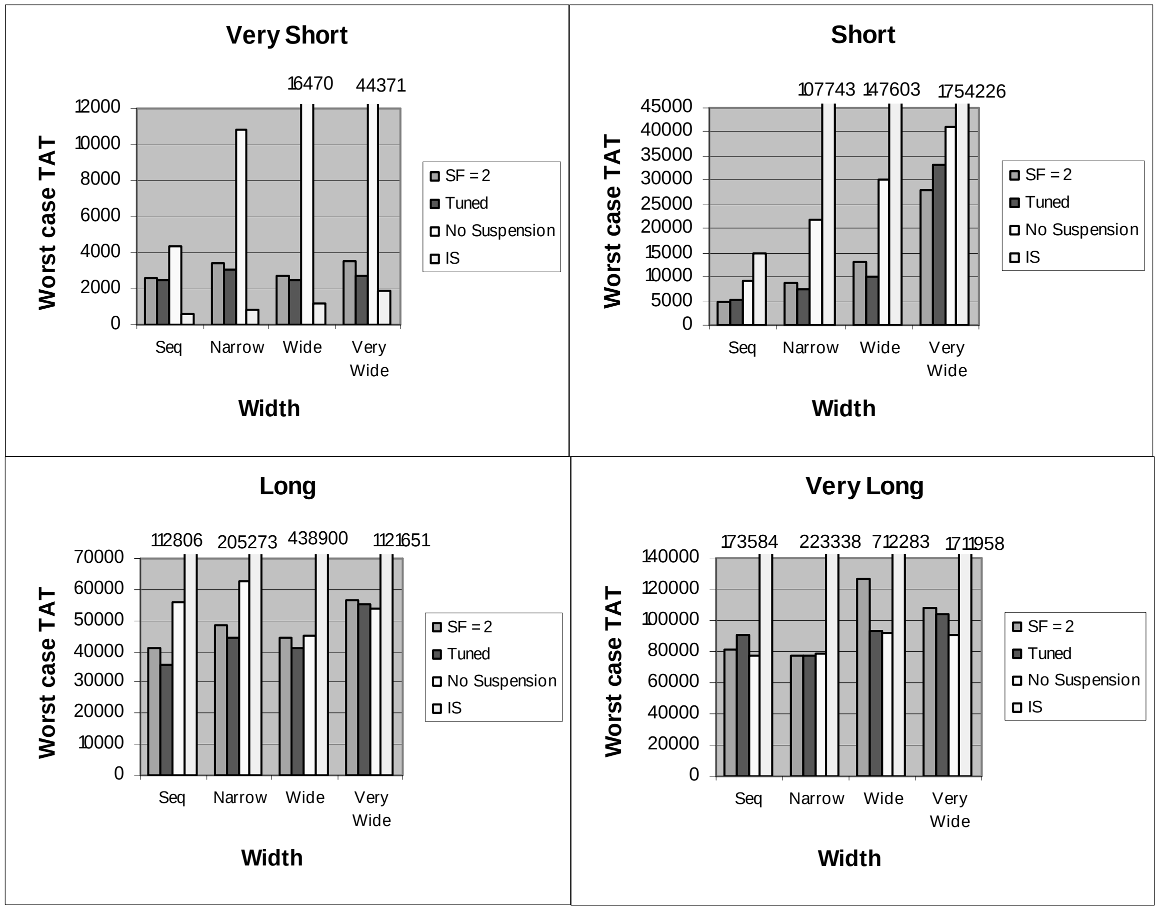

Figure 3.6: Comparison of the worstcase turnaround times of the TSS scheme with the SS, NS and IS schemes. TSS improves the worstcase turnaround time for most of the categories without adversely affecting the other categories.

categories without affecting the worst case turnaround times of the other categories. This scheme can also be applied to selectively tune the slowdowns / turnaround times for particular categories.

CHAPTER 4

IMPACT OF USER ESTIMATE INACCURACIES

We have so far assumed that the user estimates are perfect. Now, we consider the effect of user estimate inaccuracy on the proposed schemes. This is desirable from the point of view of modeling an actual system workload. In this context, we believe that there is a problem that has been ignored by previous studies, when analyzing the effect of over estimation on scheduling strategies. 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 [8]. 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 only due to the small fraction of aborted jobs, and not due to any benefits to the normal jobs. In order to avoid this problem, we group the jobs into two different estimation categories:

- Jobs that are well-estimated (the estimated time is not more than twice the actual runtime of that job) and
- Jobs that are poorly estimated jobs (the estimated runtime is more than twice the actual runtime).

Within each group, the jobs are further classified into 16 categories based on their actual run time and the number of processors requested.

It can be observed from Figure 4.1 that the Selective Suspension scheme improves the slowdowns for most of the categories without affecting the other categories. The slowdowns for the short and wide categories is quite high compared to the other categories and this is mainly because of the over estimation. Since the suspension threshold used by the SS scheme is xfactor, it favors the short jobs. But if a short job was badly estimated, it would be treated as a long job and its priority would increase only gradually. So, it will not be able to suspend running jobs easily and will end up with a high slowdown. This does not happen with IS because of the 10 minute time quantum for each arriving job irrespective of the estimated run time and therefore the slowdowns for the very short category (whose length is less than or equal to 10 minutes) is better in IS than other schemes. However, for the other categories, SS performs much better than IS.

Figure 4.2 compares the average turnaround times for the SS scheme with that of the NS and IS schemes. The improvement in performance for the short and wide categories is much less compared the improvement acheived with the accurate user estimate case. The argument provided above for the increase in slowdowns for the short and wide categories explains the reason for this. The seemingly long jobs (badly estimated short jobs) will not be able to suspend running jobs easily and have to wait in the queue for a longer time ending up with a high turnaround time.

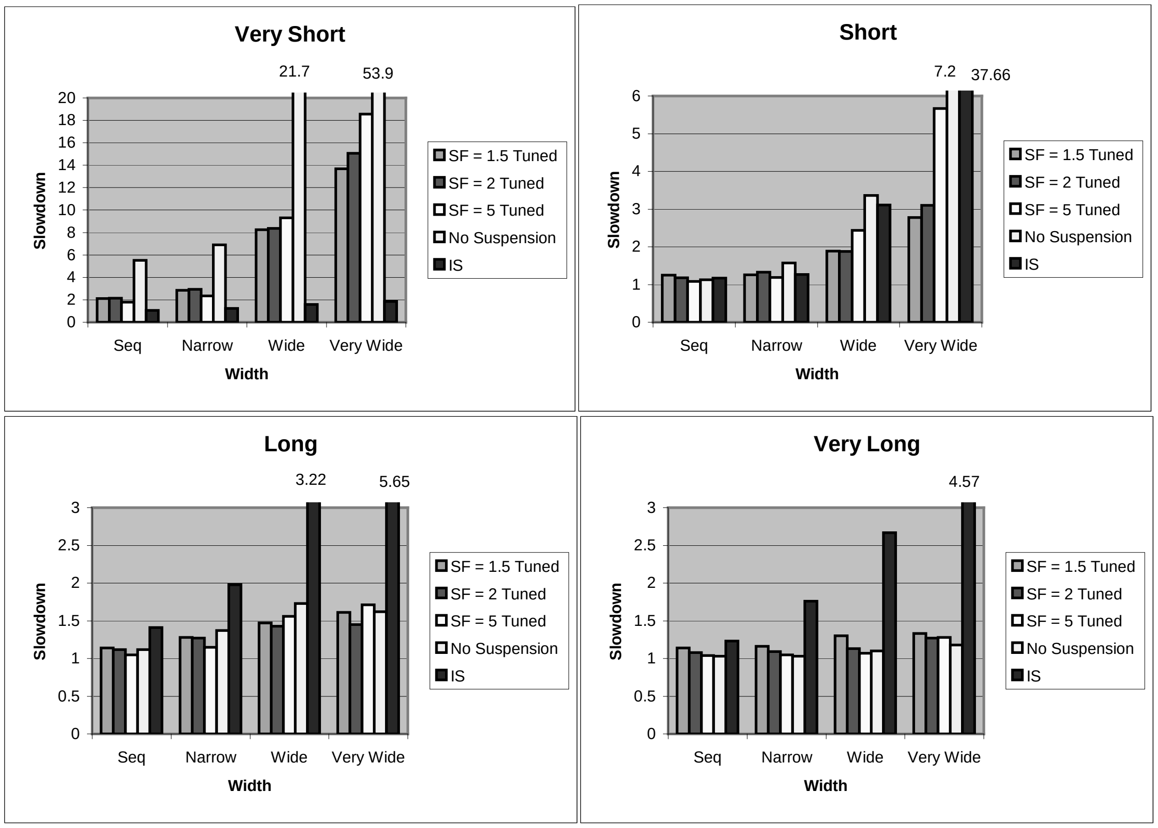

Figure 4.1: Comparison of the average slowdowns of the SS scheme with the NS and IS schemes with inaccurate user esitmates of runtime. Similar trends are observed here as for the case with accurate user estimates.

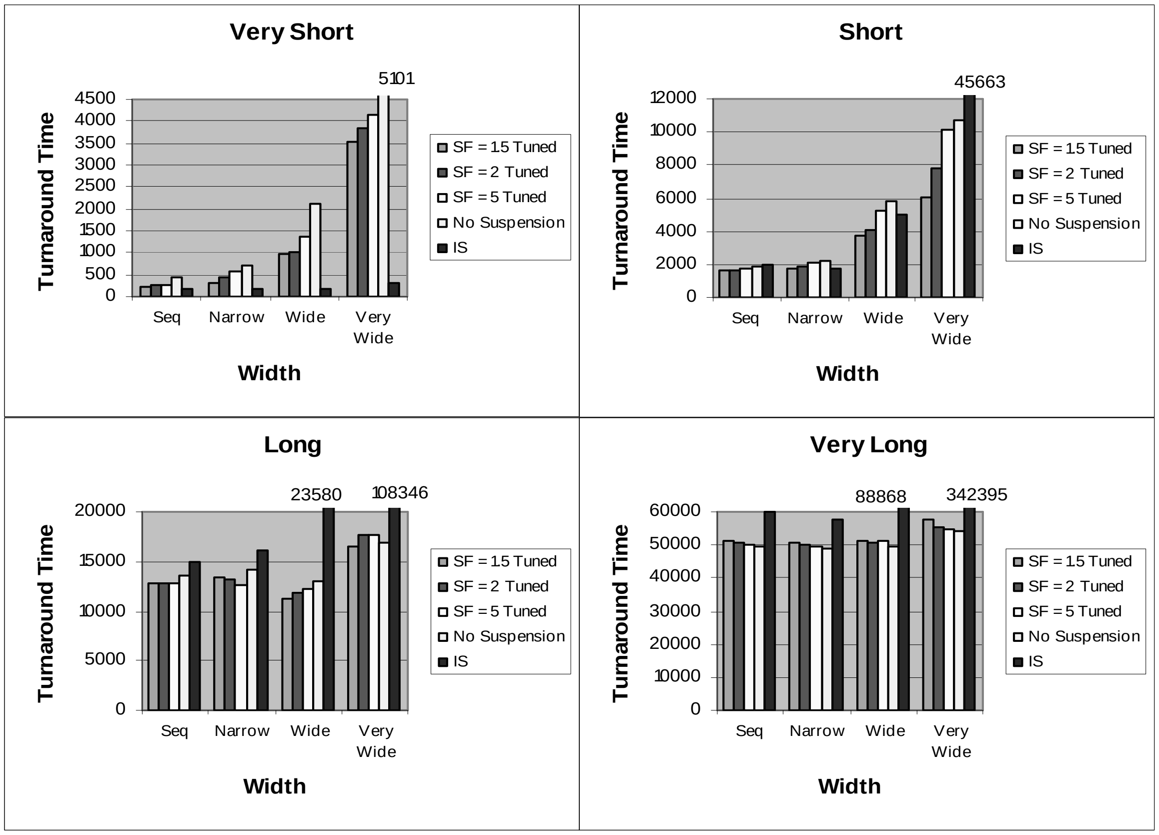

Figure 4.2: Comparison of the average turnaround times of the SS scheme with the NS and IS schemes with inaccurate user esitmates of runtime. The trends are similar to the trends observed with accurate user estimates.

From Figure 4.3 and Figure 4.4, it is evident that the higher slowdowns for the VS categories in SS is due to the poorly estimated jobs.

Figure 4.5 and Figure 4.6 show that the reduction in the percentage improvement of the average turnaround times for the short and wide categories in SS is due to the poorly estimated jobs. It can also be observed that, for the well estimated jobs, SS is better or comparable to IS for VS categories and SS outperforms IS in all other categories.

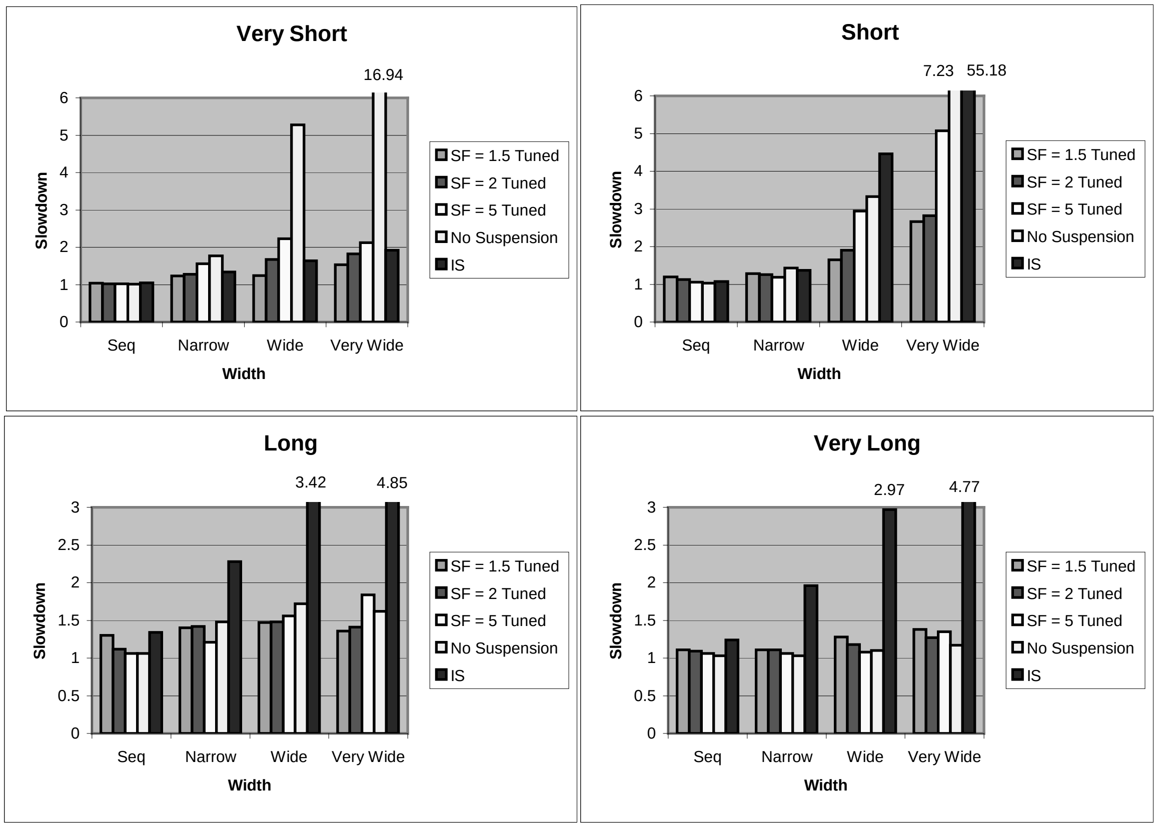

Figure 4.3: Comparison of the average slowdowns of the SS scheme with the NS and IS schemes for the well-estimated jobs. The trends are similar to that of the accurate user estimate case for the S, L and VL categories. The performance of the SS scheme for the VS categories is better or comparable to that of the IS scheme.

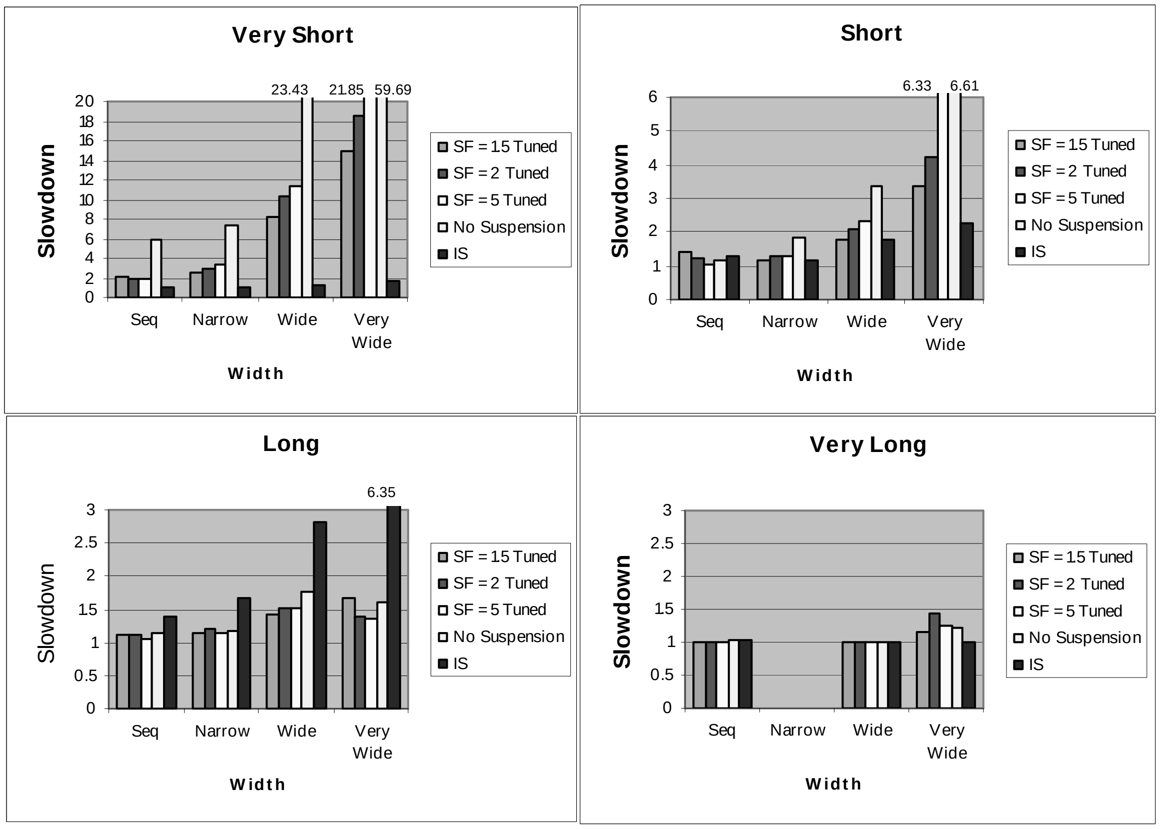

Figure 4.4: Comparison of the average slowdowns of the SS scheme with the NS and IS schemes for the poorly-estimated jobs. The trends are similar to that of the accurate user estimate case for the S, L and VL categories. The SS scheme tends to penalize poorly estimated jobs that belong to the VS category.

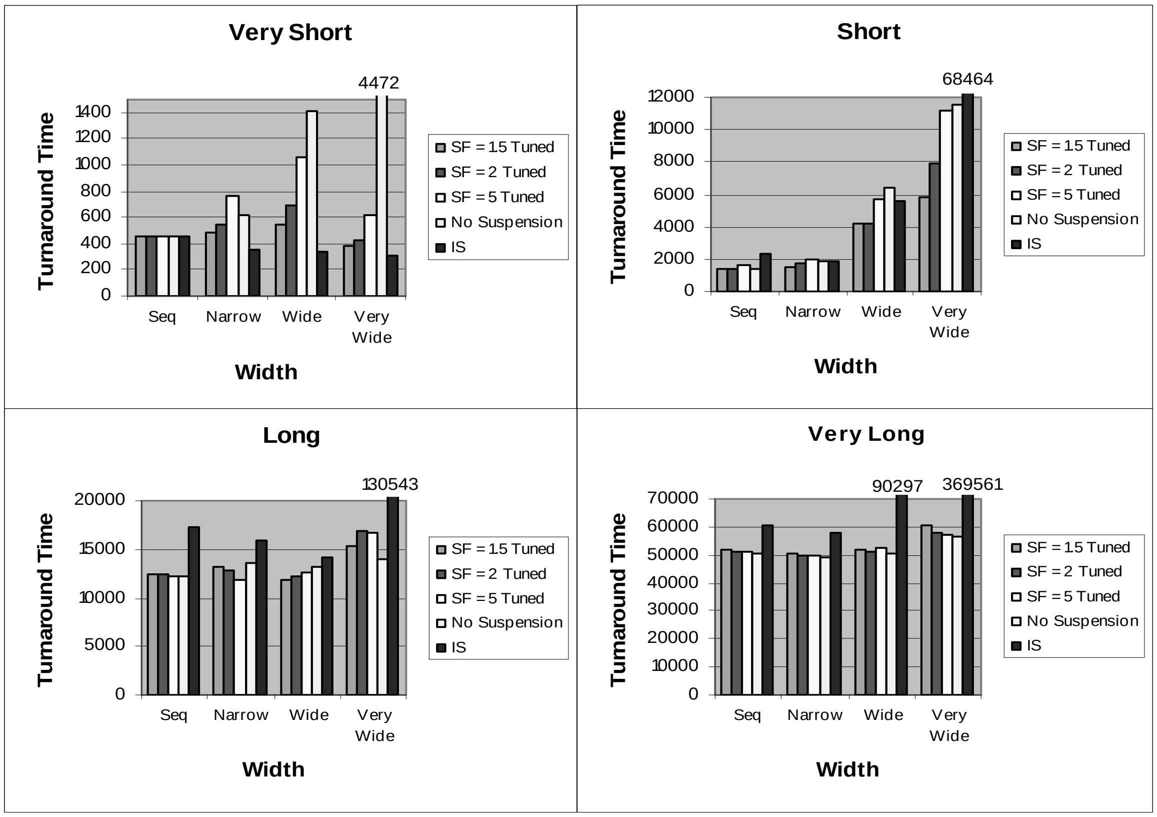

Figure 4.5: Comparison of the average turnaround times of the SS scheme with the NS and IS schemes for the well-estimated jobs. The performance of the SS scheme for the VS category is comparable to that of the IS scheme. For the other categories the trends are similar to that of the accurate user estimate case.

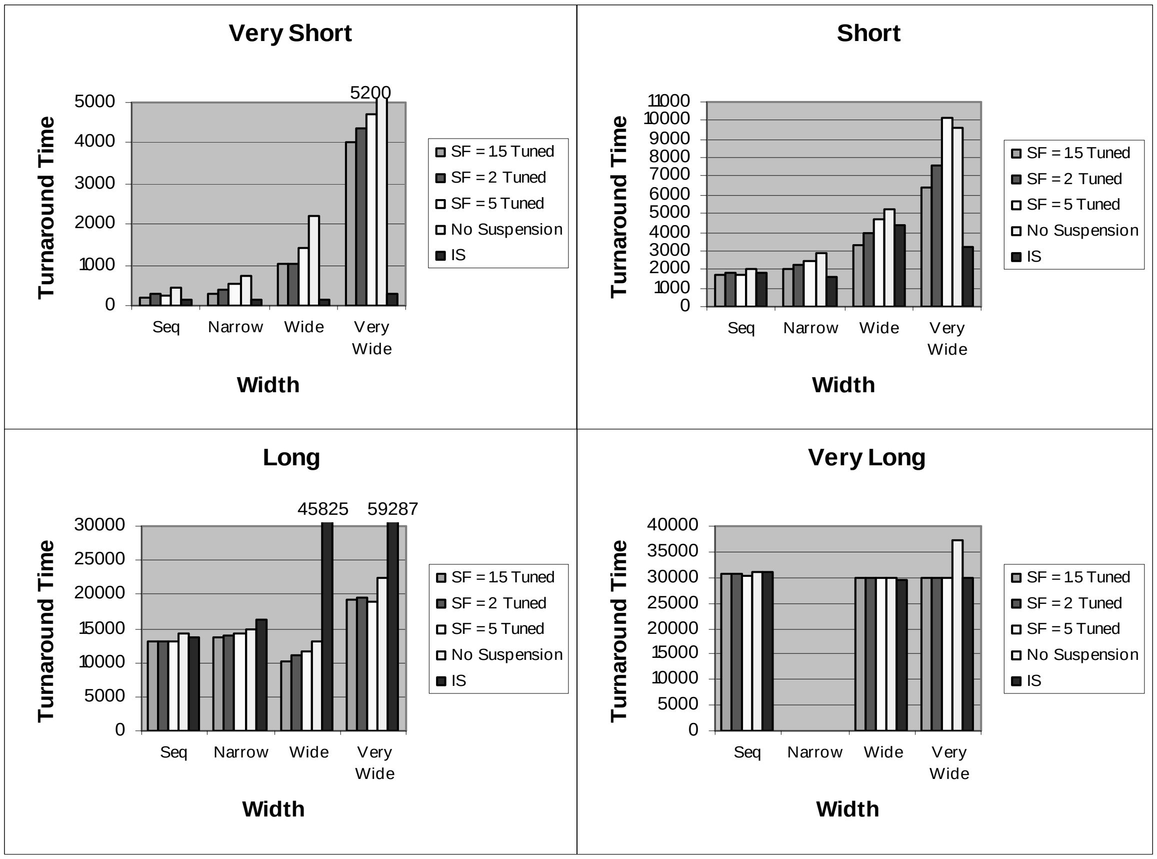

Figure 4.6: Comparison of the average turnaround times of the SS scheme with the NS and IS schemes for the poorly-estimated jobs. The SS scheme tend to penalize the poorly-estimated short jobs.

4.1 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. The memory transfer rate that we considered is based on the following scenario: with a commodity local disk for every node, with each node being a quad, the transfer rate per processor was assumed to be 2 MBps(corresponding to a disk bandwidth of 8 MBps).

Figure 4.7 and Figure 4.8 compare respectively the slowdowns and turnaround times of the proposed tunable scheme with NS and IS in the presence of overhead for job suspension/restart. It can be observed that overhead does not significantly affect the performance of the SS Scheme.

4.2 Load Variation

We have so far seen the performance of the Selective Suspension scheme under normal load. In this section, we present the performance of the SS scheme under different load conditions starting from the normal load (original trace) until the system reaches saturation. The different loads correspond to modification of the traces by dividing the arrival times of the jobs by suitable constants, keeping their run time the same as in the original trace.

For simplicity, we reduced the number of job categories from sixteen to four for the load variation studies: two categories based on their run time - Short(S) and

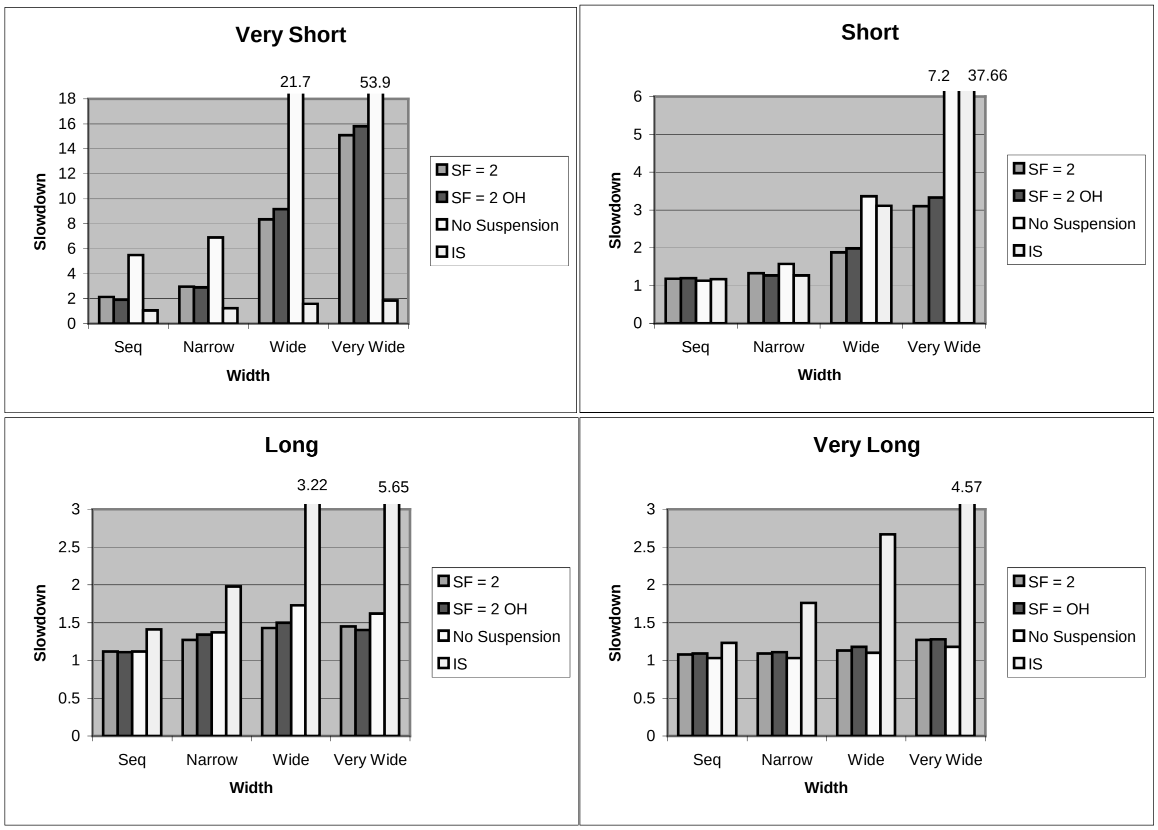

Figure 4.7: Comparison of the average slowdowns of the SS scheme with the IS and NS schemes, with modeling of overhead for suspending/restarting a job. The performance of the SS scheme with modeling of overhead is comparable to its performance in the absence of overhead.

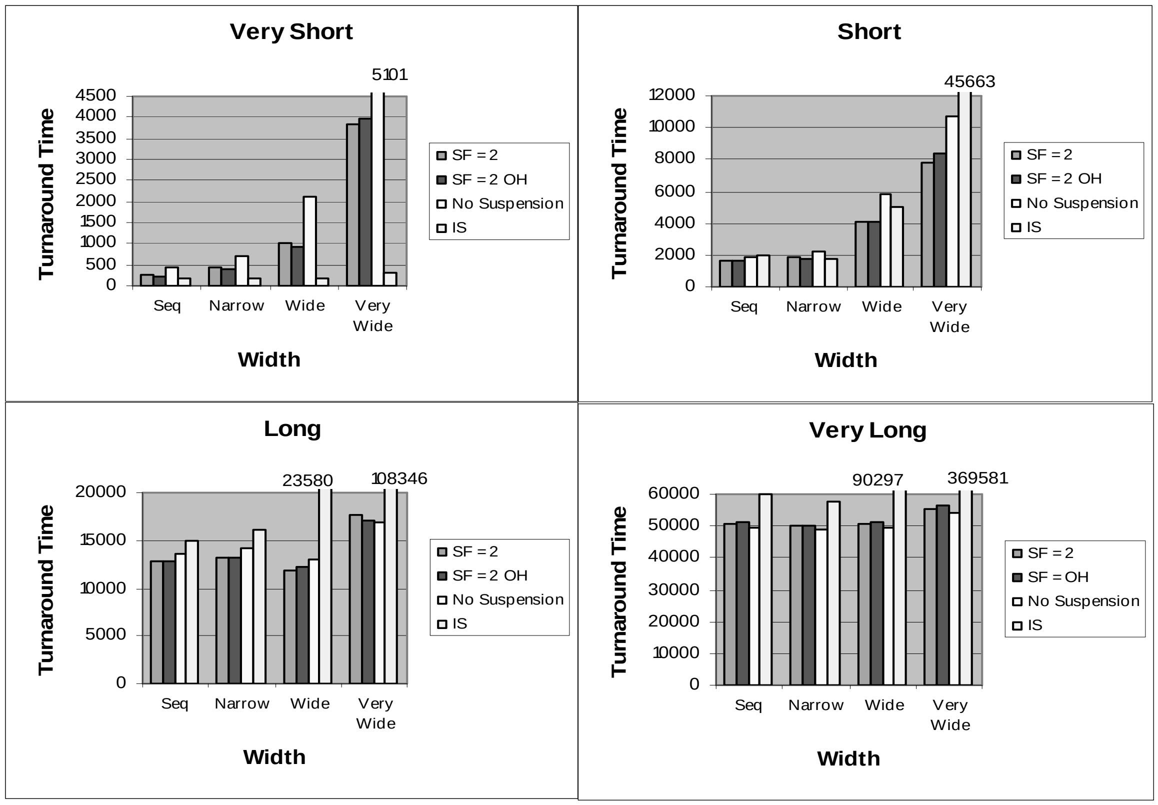

Figure 4.8: Comparison of the average turnaround times of the SS scheme with the IS and NS schemes, in the presence of overhead for suspending/restarting a job. The overhead does not introduce any significant change in the performance of the SS scheme.

	≤8 Processors	**>8 Processors**
≤1Hr	SN	SW
>1Hr	LN	LW

Table 4.1: Job categorization criteria for load variation studies

	≤8 Processors	**>8 Processors**
≤1Hr	44%	13%
>1Hr	30%	13%

Table 4.2: Job distribution by category for load variation studies

Long(L) and two categories based on the number of processors requested - Narrow(N) and Wide(W). The criteria used for job classifications is shown in Table 4.1. The distribution of jobs in the trace, corresponding to the four categories is given in Table 4.2.

Figure 4.9 shows the overall system utilization for different schemes under different load conditions. It can be observed that the SS scheme is able to achieve a better utilization than the NS scheme at higher loads whereas the overall system utilization is very low under the IS scheme. Also, there is no significant increase in the overall system utilization (for both the SS and NS schemes) when the load factor is increased beyond 1.6 which indicates that the system reaches saturation at a load factor of 1.6. We report the performance of the SS scheme for various load factors between 1.0 (normal) and 1.6.

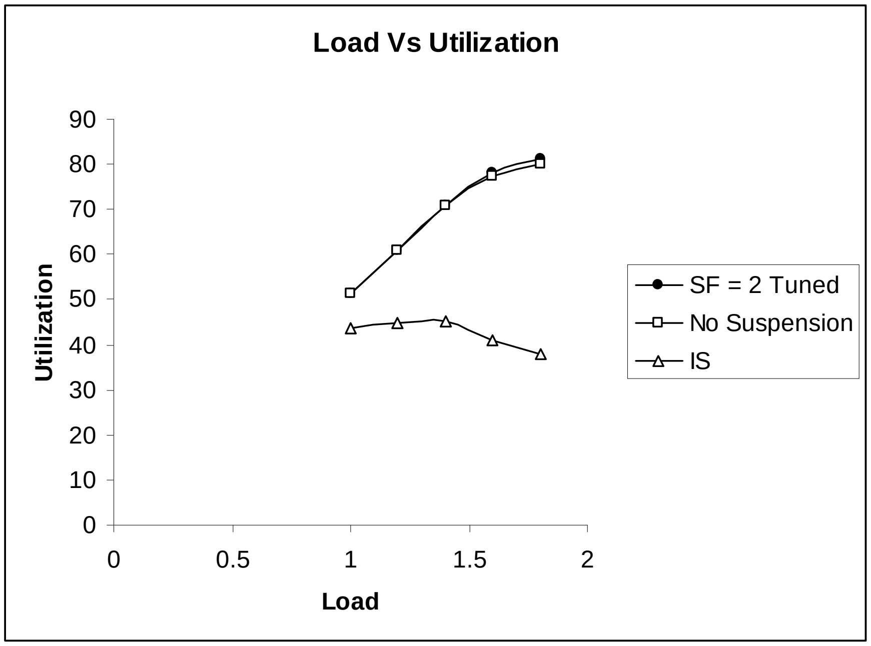

Figure 4.9: Comparison of overall system utilization of the SS scheme with the NS and IS schemes under different load conditions. The overall system utilization with the SS scheme is better or comparable to the NS scheme. IS is much worse in terms of overall system utilization.

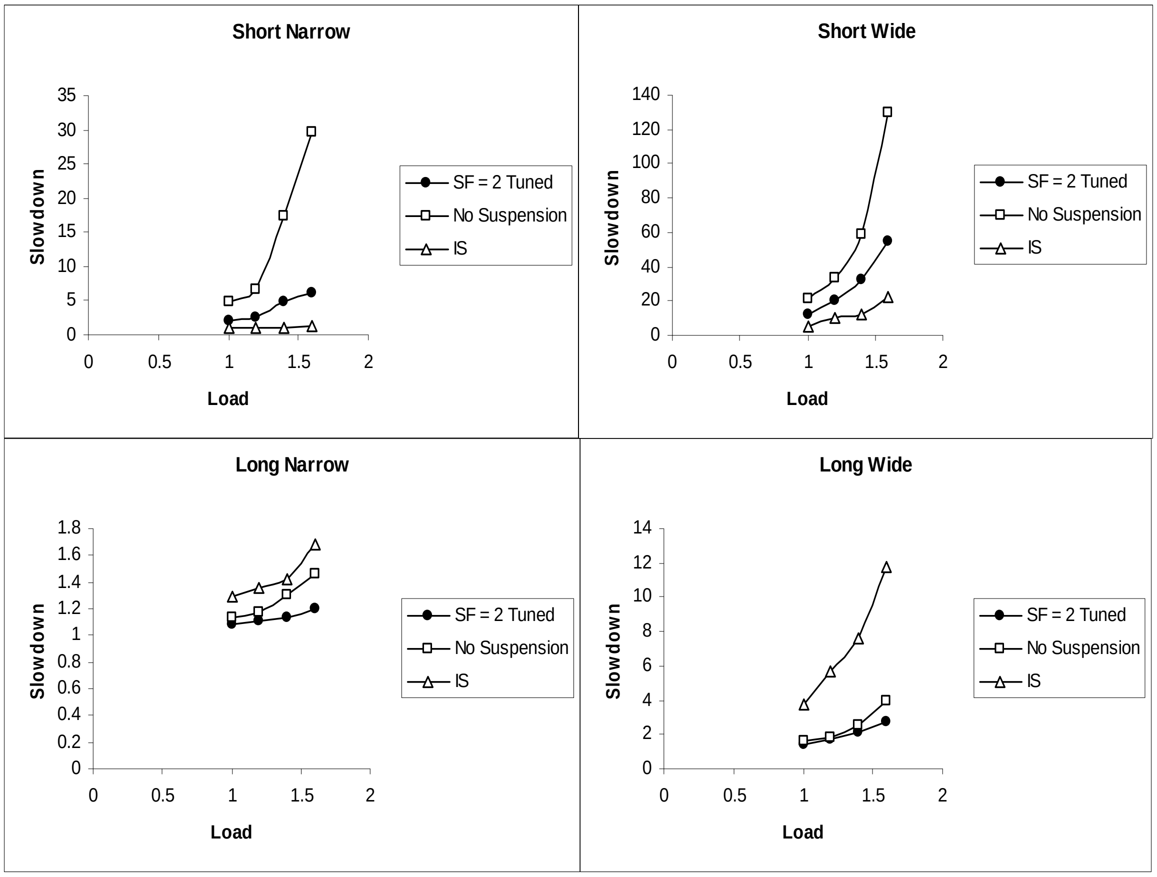

Figure 4.10: Comparison of the average slowdowns of the SS scheme with the IS and NS schemes under different load conditions. The improvements achieved by the SS scheme are more pronounced under high load.

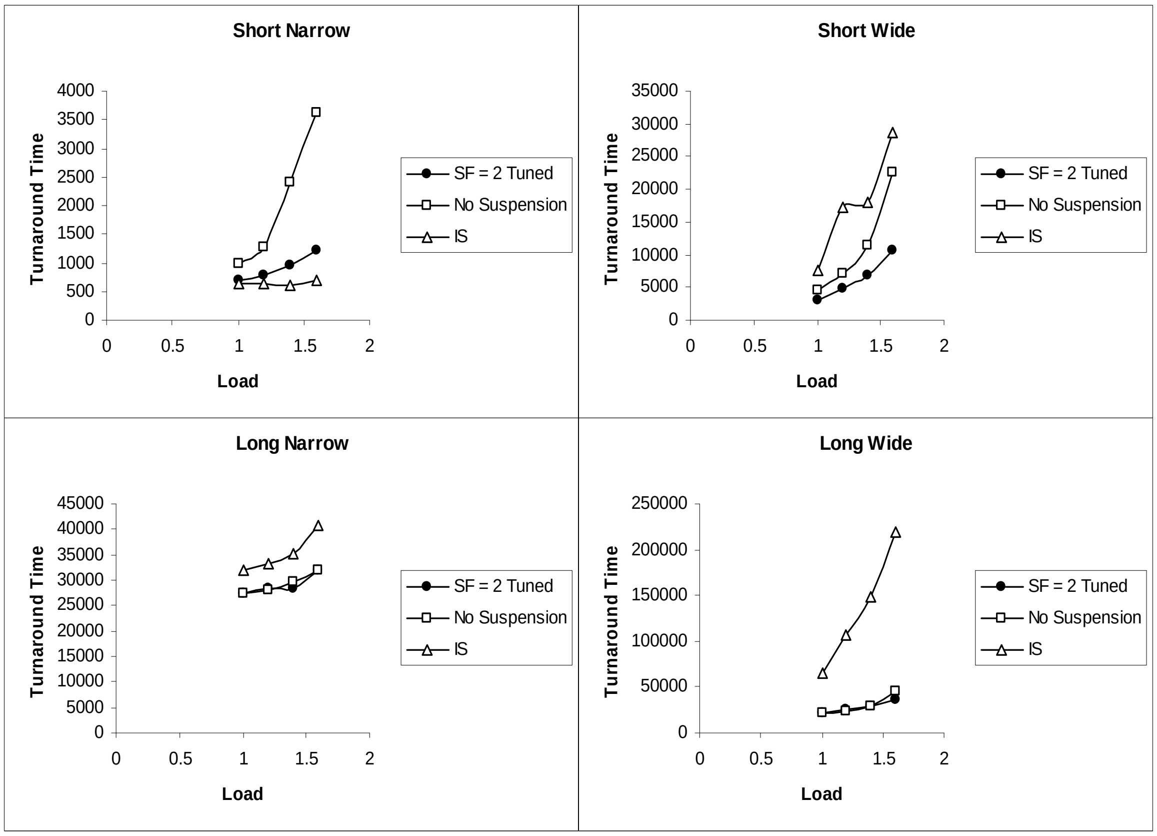

Figure 4.11: Comparison of the average turnaround times of the SS scheme with the IS and NS schemes under different load conditions. The improvements achieved by the SS scheme are more pronounced under high load.

Figure 4.10 and Figure 4.11 compare the performance of the SS scheme with the NS and IS schemes for different job categories under different load conditions. It can be observed that the improvements obtained by the SS scheme are more pronounced under high load. The trends with respect to different categories under higher loads is similar to that observed under the normal load. It provides significant benefit to the short jobs without affecting the performance of long jobs. The IS scheme is better than the SS scheme only for the SN jobs in terms of average turnaround time, whereas it is better than SS for both SN and SW jobs in terms of average slowdown. It implies that the IS scheme improves the performance of only the relatively shorter jobs in SW category by adversely affecting the performance of the relatively longer jobs. Also, the performance of the IS scheme is much worse for the long jobs, which is very undesirable.

From Figure 4.12 and Figure 4.13 compare respectively the average slowdowns and the average turnaround times against the overall system utilization for various schemes. It is evident that the SS scheme is much better than both the IS and NS schemes. Even when the system is highly utilized, the SS scheme is able to provide much better response times for all categories of jobs. The IS scheme is not able to achieve high system utilization.

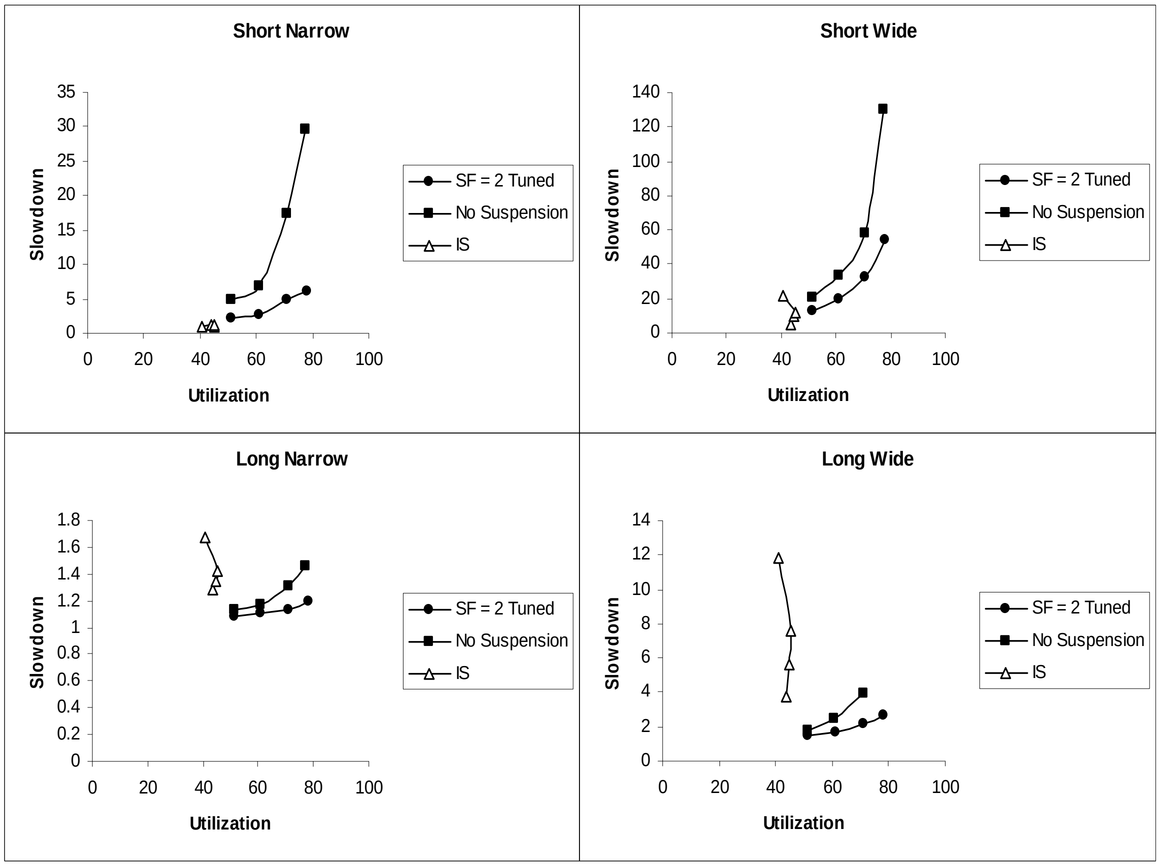

Figure 4.12: Comparison of the average slowdowns against the overall system utilization for the SS, NS and IS schemes.

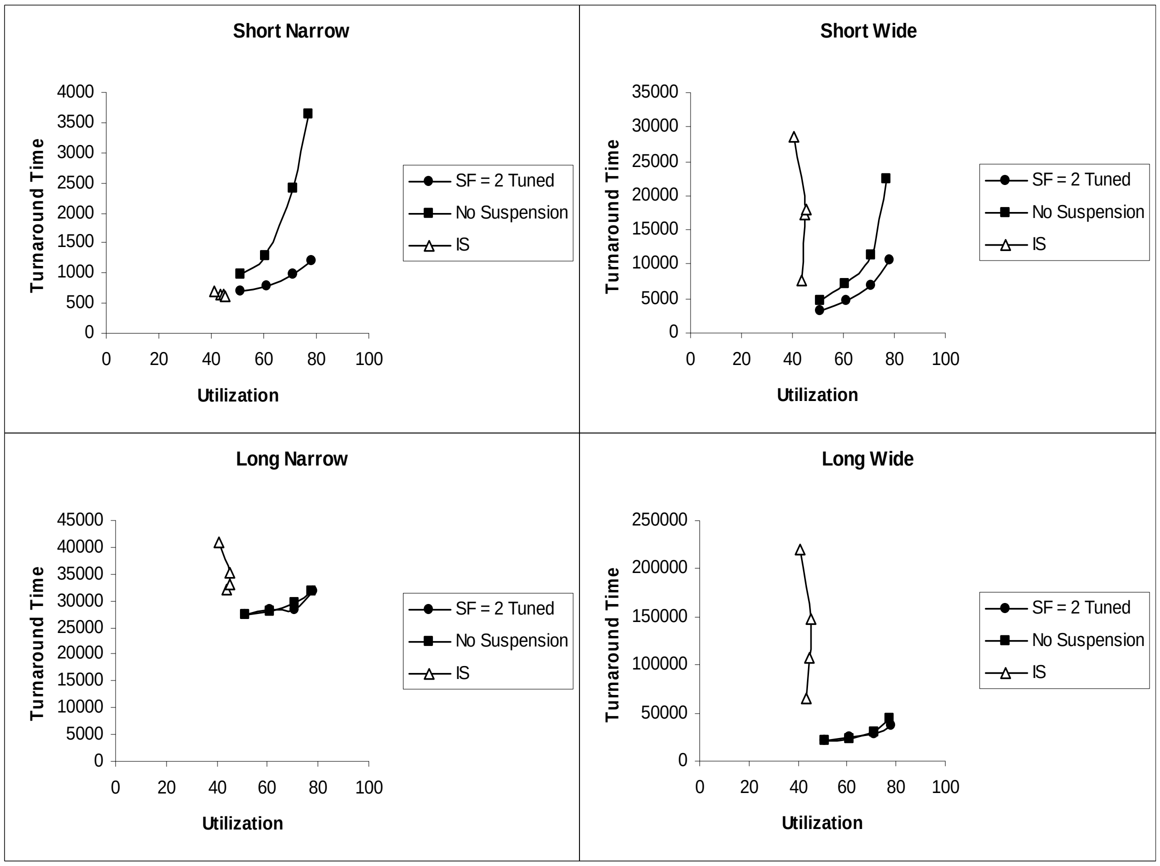

Figure 4.13: Comparison of the average turnaround times against the overall system utilization for the SS, NS and IS schemes.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

In this thesis, 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, selective suspension scheme and demonstrated that it provides significant improvement in the average slowdown and the worst case slowdowns 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. We also evaluated the proposed schemes in the presence of over estimations and showed that it produced good results.

5.1 Future Work

- In this thesis, we compared the performance of the selective preemption schemes with non-premptive and Immediate Service scheme. It is of interest to compare the performance of the selective preemption scheme with gang scheduling [15], [17], [36].

- The system utilization is an important consideration especially under high load. The selective preemption scheme, though it improves the overall system utilization, is not aimed directly at improving the system utilization. It would be useful to devise a scheme that takes utilization also into consideration in performing preemption decisions.

- The next step is to evaluate the selective preemption schemes in a realistic environment by incorporating it in a production job scheduler.

BIBLIOGRAPHY

[1] K. Aida. Effect of Job Size Characteristics on Job Scheduling Performance. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–17, 2000.

[2] S. V. Anastasiadis and K. C. Sevcik. Parallel Application Scheduling on Networks of Workstations. *Journal of Parallel and Distributed Computing*, 43(2):109–124, 1997.

[3] O. Arndt, B. Freisleben, T. Kielmann, and F. Thilo. A Comparative Study of Online Scheduling Algorithms for Networks of Workstations. *Cluster Computing*, 3(2):95–112, 2000.

[4] S. H. Chiang, R. K. Mansharamani, and M. K. Vernon. Use of Application Characteristics and Limited Preemption for Run-to-Completion Parallel Processor Scheduling Policies. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 33–44, 1994.

[5] S. H. Chiang and M. K. Vernon. Production Job Scheduling for Parallel Shared Memory Systems. In *Proceedings of International Parallel and Distributed Processing Symposium*, 2002.

[6] W. Cirne. When the Herd is Smart: The Emergent Behavior of SA. In *IEEE Trans. Par. Distr. Systems*, 2002.

[7] W. Cirne and F. Berman. Adaptive Selection of Partition Size for Supercomputer Requests. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 187–208, 2000.

[8] P. J. Keleher D. Perkovic. Randomization, Speculation, and Adaptation in Batch Schedulers. In *Proceedings of Supercomputing 2000.*

[9] Bhaskar DasGupta and Michael A. Palis. Online real-time preemptive scheduling of jobs with deadlines. In *APPROX*, pages 96–107, 2000.

[10] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive Scheduling of Parallel Jobs on Multiprocessors. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1996.

[11] Xiaotie Deng and Patrick Dymond. On multiprocessor system scheduling. In *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, pages 82–88. ACM Press, 1996.

[12] Leah Epstein. Optimal preemptive scheduling on uniform processors with non-decreasing speed ratios. *Lecture Notes in Computer Science*, 2010:230–248, 2001.

[13] D. G. Feitelson. Logs of real parallel workloads from production systems. http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[14] D. G. Feitelson. Analyzing the Root Causes of Performance Evaluation Results. Technical report, Leibniz Center, Hebrew University, 2002.

[15] D. G. Feitelson and M. A. Jette. Improved utilization and responsiveness with gang scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 238–261. 1997.

[16] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and Practice in Parallel Job Scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing* , pages 1–34. 1997.

[17] Dror G. Feitelson. Packing Schemes for Gang Scheduling. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing – Proceedings of the IPPS'96 Workshop*, volume 1162, pages 89–110. Springer, 1996.

[18] D. Jackson, Q. Snell, and M. J. Clement. Core Algorithms of the Maui Scheduler. In *Wkshp. on Job Sched. Strategies for Parallel Processing*, pages 87–102, 2001.

[19] J. P. Jones and B. Nitzberg. Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–16, 1999.

[20] W. A. Ward Jr., C. L. Mahood, and J. E. West. Scheduling Jobs on Parallel Systems Using a Relaxed Backfill Strategy. In *Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, 2002.

[21] P. J. Keleher, D. Zotkin, and D. Perkovic. Attacking the Bottlenecks of Backfilling Schedulers. *Cluster Computing*, 3(4):245–254, 2000.

[22] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour. On the Design and Evaluation of Job Scheduling Algorithms. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 17–42, 1999.

[23] B. G. Lawson and E. Smirni. Multiple-queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems. In *Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, 2002.

[24] B. G. Lawson, E. Smirni, and D. Puiu. Self-adapting Backfilling Scheduling for Parallel Systems. In *Proceedings of the International Conference on Parallel Processing*, 2002.

[25] L. T. Leutenneger and M. K. Vernon. The Performance of Multiprogrammed Multiprocessor Scheduling Policies. In *ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, pages 226–236, May 1990.

[26] D. Lifka. The ANL/IBM SP Scheduling System. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 295–303, 1995.

[27] C. McCann, R. Vaswani, and J. Zahorjan. A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors. In *ACM Trans. Computer Systems*, volume 11, pages 146–178, 1993.

[28] A. W. Mu'alem and D. G. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. In *IEEE Trans. Par. Distr. Systems*, volume 12, pages 529–543, 2001.

[29] Eric W. Parsons and Kenneth C. Sevcik. Implementing Multiprocessor Scheduling Disciplines. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 166–192. Springer Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.

[30] Uwe Schwiegelshohn and Ramin Yahyapour. Fairness in parallel job scheduling.

[31] K. C. Sevcik. Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems. *Performance Evaluation*, 19(2-3):107–140, 1994.

[32] J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY - LoadLeveler API Project. In *Wkshp. on Job Sched. Strategies for Parallel Processing*, pages 41–47, 1996.

[33] A. Streit. On Job Scheduling for HPC-Clusters and the dynP Scheduler. In *Proc. Intl. Conf. High Perf. Comp.*, pages 58–67, 2001.

[34] D. Talby and D. G. Feitelson. Supporting Priorities and Improving Utilization of the IBM SP Scheduler Using Slack-Based Backfilling. In *Proceedings of the 13th International Parallel Processing Symposium*, pages 513–517, 1999.

[35] J. Zahorjan and C. McCann. Processor Scheduling in Shared Memory Multiprocessors. In *ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, pages 214–225, May 1990.

[36] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam. An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration. *Lecture Notes in Computer Science*, 2221:133–150, 2001.

[37] D. Zotkin and P. Keleher. Job-length Estimation and Performance in Backfilling Schedulers. In *Proceedings of the 8th High Performance Distributed Computing Conference*, pages 236–243, 1999.