Opportune Job Shredding: An Effective Approach for Scheduling Parameter Sweep Applications[*]

Rohan Kurian Pavan Balaji P. Sadayappan

Computer and Information Science,
The Ohio State University,
2015 Neil Avenue,
Columbus, OH 43210
{kurian, balaji, saday}@cis.ohio-state.edu

Abstract

A number of applications comprise of several small independent and homogeneous tasks that need to be executed. However, most supercomputer centers enforce a restriction on the number of jobs a single user can submit to the cluster at any time in order to ensure fairness to the other submitted jobs. This forces the users to combine these independent homogeneous tasks into a single parallel Parameter Sweep Application (PSA). In this paper we propose a new and efficient scheme, termed as ***Opportune Job Shredding****, which allows Supercomputer Centers to take advantage of the independence of the component tasks of Parameter Sweep Applications, without affecting the other submitted jobs significantly. We also propose an extension of the previously proposed "Multiple Simultaneous Requests" scheme combining it with the Opportune Job Shredding scheme allowing Parameter Sweep Applications to be executed in parts on remote clusters.*

Keywords: *Parameter Sweep Applications, Multiple Simultaneous Requests, Meta Scheduling*

1 Introduction

Parameter Sweep Applications (PSAs) represent an important class of computationally intense applications that require significant compute resources. Many users of shared computational facilities like supercomputer centers generate PSAs. PSAs generally consist of a large number of independent tasks that may share some common files for input and output. All tasks of a PSA must complete before the job is complete, but the various tasks can be executed independently and in any order. Parallel Tomography [33] and MCell [21] are examples of PSAs. The Parallel Tomography Application (also known as GTOMO) is being used in production at the National Center For Microscopy and Imaging Research. The MCell application is used as a simulator for Cellular Micro-Physiology. Other such applications include those used for modeling photochemical pollution, fluid flows, etc. We will go over some applications in more detail in section 3.

Due to the importance of PSAs, scheduling/resource-management systems have been built to facilitate their execution. These include APST (AppLeS Parameter Sweep Template) [5, 28] and Nimrod/G [2]. APST, Nimrod/G and SETIHome [31] are primarily targeted at utilizing distributed computational resources over the web/grid for PSAs. Without such systems, users of PSAs would have to manually create and submit a large number of independent jobs on different available machines. Middleware like APST greatly eases the burden of PSA users, by automatically spawning multiple tasks for a PSA job at suitable sites. It uses dynamic load information on processors and network links to determine the "best" sites to submit the tasks of a PSA, i.e. the sites that will result in the fastest completion.

It is of great interest to look at effective ways of supporting the use of parallel systems at supercomputer centers in executing PSAs. Besides the tedium of manually creating hundreds of separate jobs to submit to a supercomputer center, users of PSAs have to face fair-share constraints imposed by most centers on the maximum number of simultaneously executable jobs from any user. These fair-share limits are imposed to ensure that the resources of the center are not monopolized by a single user.

While PSA scheduling systems like APST could be set up to work in conjunction with schedulers at supercomputer centers, there is a problem that has to be addressed in this

[*]This research is supported in part by NSF grants #CCR-0204429 and #EIA-9986052

regard. Clearly, the usual fair-share limits of a supercomputer center's policy need to be relaxed if the individual jobs created by APST for a PSA are to execute without delay. But if this were done, it could result in significant delays of other non-PSA jobs in the system - the avoidance of this problem is the reason why fair-share limits are imposed at supercomputer centers.

In this paper we first use trace-driven simulations to characterize the impact of introducing PSA job "fragments" (i.e. the large number of individual independent small jobs spawned for a single PSA job submitted to a system like APST) into a mix of non-PSA jobs in a supercomputer center environment. We show that non-PSA jobs are indeed adversely affected. We then propose a new approach, termed as *Opportune Job Shredding*, to significantly overcome the degradation of non-PSA jobs, while still allowing for considerable improvement of PSA jobs. We demonstrate that the proposed scheme improves the slowdown of PSAs by up to 70% and the overall loss of capacity of the system by up to 21%. Not only does the scheme avoid significant degradation of the average turnaround time of non-PSA jobs in all cases, but it even improves their performance in some cases.

Meta Schedulers and Grid Computing have recently become an area of increasing interest. A number of researchers have been studying various schemes to harness the capabilities of multiple clusters. Different schemes have been proposed to efficiently allow jobs to be executed on remote and potentially heterogeneous clusters. Most of these schemes rely on a centralized administrative authority which handles the scheduling on all clusters. Recently, a "Multiple Simultaneous Requests" scheme had been proposed in order to efficiently schedule jobs on clusters which do not belong to a single administrative domain. However, both the existing variations of this scheme [35, 29] make certain inherent assumptions which do not allow them to be used in a generic environment consisting of clusters of heterogeneous processing speeds and using different local scheduling strategies. In this paper, we combine our scheme with the existing "Multiple Simultaneous Requests" scheme to allow Parameter Sweep Applications to be executed in parts on remote clusters, without restrictive assumptions about the local scheduling policies at the sites.

The remaining part of the paper is organized as follows: In Section 2 we provide some background and discuss related work. In Section 3 we discuss the nature of the Parameter Sweep Applications in more detail. In Section 4 we describe the new scheme developed in this paper. We discuss the simulations we carried out and the results comparing the various schemes in Section 5 and present some concluding remarks and possible future work in Section 6.

2 Background and Related Work

Parallel job scheduling strategies have been widely studied [14, 24, 11, 26, 22, 3, 25, 8, 27, 35, 1]. 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 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 [20]. Backfilling [24, 10] was proposed to improve system utilization and has been implemented is most production schedulers [9]. 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 (EASY) [24, 32]. In conservative backfill, every job is given a reservation when it enters the system. A smaller job is moved forward in the queue as long as it does not delay any previously queued job. In aggressive backfilling, only the job at the head of the queue has a reservation. A small job is allowed to leap forward as long as it does not delay the job at the head of the queue. Fig. 1 shows an example of a schedule with EASY backfilling.

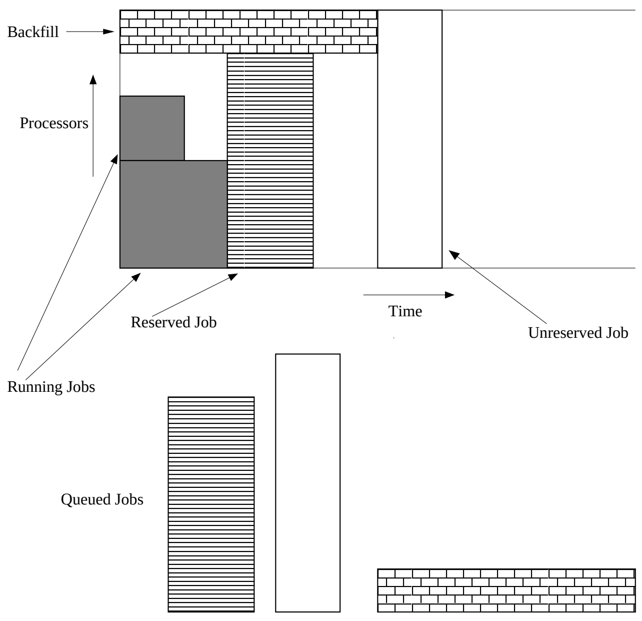


Figure 1. A job schedule with EASY backfilling

2.1 Multi-Site Scheduling

While much of the research on job scheduling has focused on the homogeneous single-site case, there has been considerable recent interest in distributed and multi-site job scheduling [4, 6, 12, 15, 16, 17, 18, 19, 34, 36]. Cooperative multi-site scheduling enables jobs from heavily loaded sites to be executed at remote sites that have a lighter load. Analysis of job log traces [13] shows that the load at supercomputer centers follows a "sinusoidal" pattern of variation, with the load increasing during the day, peaking somewhere in the evening and then decreasing to a low in the very early hours of the morning. By performing multi-site scheduling over geographically distributed sites, better load balancing can be facilitated.

We recently evaluated the impact of using multiple simultaneous reservations in a multi-site environment [35]. Each job was simultaneously placed in the queue at more than one site, and when a job was ready to actually start at any of the sites, the other requests were cancelled. It was found that this resulted in improved average job response times compared to a scheme where each job is placed on the site that had the least load. The primary reason for the effectiveness of the multiple-simultaneous-request scheme is that backfilling dynamics are very complex and unpredictable, especially when user estimates of job runtime are inaccurate. Therefore, it is very possible that a job may be able to backfill and start earlier at a more heavily loaded site than a lightly loaded site. Further, allowing more jobs in each site's queue enhances the chances of backfilling, thereby reducing the loss of capacity of the system.

When the multiple-simultaneous-requests scheme was evaluated in a heterogeneous environment [29], it was found that it needed some adaptation in order to be effective. Primarily, it was important to choose the execution site for a job on the basis of expected completion time rather than expected start time. In order to accurately estimate a job's completion time at multiple sites, it was necessary to use conservative backfilling instead of aggressive backfilling. Thus, some constraints must be imposed on the local scheduling strategy at the sites in order for the multiple-simultaneous-requests strategy to be effective. We show later that in the context of PSA jobs, we can apply the multiple-simultaneous-requests strategy effectively without imposing any constraints on the local job scheduling strategy at the different sites.

3 Parameter Sweep Applications

Parameter Sweep Applications (PSA) are naturally parallel applications that can be structured as a large set of independent tasks that may share common files. The name derives from applications that may be structured as sets of experiments, each of which is executed with a distinct set of parameters. Each experiment (task) may be independently carried out, but the entire set must be completed before results can be generated. There are no task precedence constraints and therefore the total work may be packaged into a number of independent chunks.

Due to the independent nature of the chunks of a PSA, the user of a supercomputer center has many options in creating jobs for submission to the system. At one extreme, each chunk could be submitted as an independent job. At the other extreme, all chunks could be combined and submitted as a very wide parallel job, requesting as many processors as the number of chunks. In between, a range of possibilities exist - the total number of jobs submitted can vary from one to the number of chunks; further, each jobs can be shaped to be short and very wide (requesting many processors) or narrow and long (requesting few processors). A significant consideration is that most supercomputer centers impose "fair-share" constraints, that limit the total number of simultaneous jobs or the number of processors requested by the collection of simultaneously queued/active jobs of a user. If the center imposes limits on the number of simultaneously active jobs of a user, users submit one or a few parallel jobs that include the chunks of the PSA. Since the scheduling system does not identify such jobs as PSA jobs, they can only be started when the requested number of processors is available - opportunities to start off independent chunks of a PSA job on available idle processors cannot be utilized.

Systems like Nimrod/G and APST, that support the execution of PSAs, have primarily targeted a grid environment. Nimrod/G proposed the use of an economic model in determining where the chunks of a PSA job are to be executed - a cost is associated with execution at each of the potential execution sites, and chunks are dispatched to sites based on cost considerations. APST too primarily targets a grid environment, and uses a load monitoring service such as Network Weather Service (NWS) to determine dynamic processor and network load information. The scheduling of the chunks of a PSA job is done using estimates of response time at various sites. Thus the scheduling strategy may be categorized as a greedy approach that attempts to minimize the response time of the PSA job, without any global considerations regarding other jobs. Although a system like APST could be configured to interact with job schedulers at supercomputer centers, its greedy strategy of distributing chunks of a PSA job can be expected to have an adverse effect on the regular non-PSA jobs submitted to the supercomputer centers. The problem we address in this paper is the development of a scheduling strategy that can exploit the independence characteristic of the chunks of a PSA job to improve its performance and that of the system, without much degradation of non-PSA jobs.

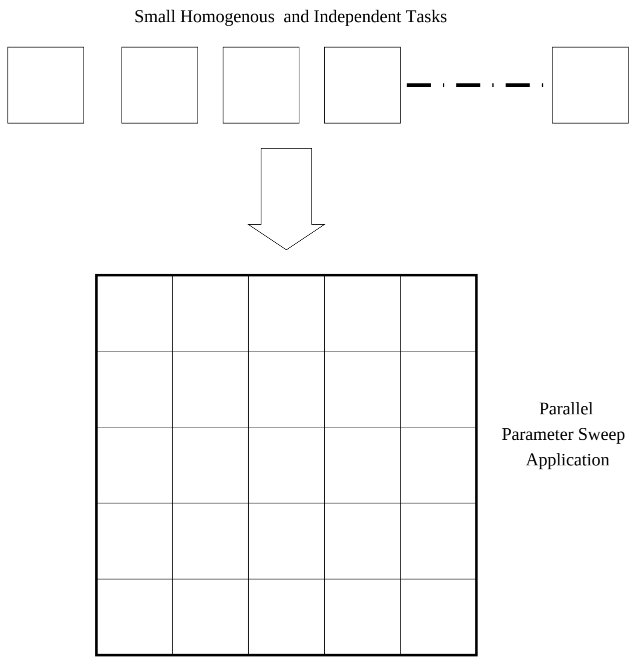

Small Homogenous and Independent Tasks
Parallel Parameter Sweep Application

Figure 2. Parameter Sweep Application Formation

4 Proposed Scheme

A simple approach to take advantage of the characteristics of the PSA jobs is a flooding based scheme where the PSA jobs are shred into a number of small tasks. These tasks are then submitted to the supercomputer center as independent sequential jobs. We'll refer to this approach as the **"Flooding-based approach"**. As we'll see in the later sections, this approach not only improves the average turnaround time and slow down, for the PSA jobs, but also benefits the overall system metrics such as the Loss of Capacity (LOC) due to better back-filling. However, this scheme gives an unfair advantage to the PSA jobs by allowing them to flood the schedule and adversely affects teh performance of the other Non-PSA jobs.

To remedy this, we propose a simple and novel scheme. The basic idea of the scheme is to allow the PSA jobs to shred and back-fill as long as they don't hamper the back-fill opportunities of the Non-PSA jobs. The focus of the scheme is to utilize of the inherent characteristics of the PSA jobs without affecting the Non-PSA jobs significantly. We term this new scheme as **"Opportune Job Shredding"**. In this model, we use an application-level scheduler similar to APST or NIMROD/G which continuously monitors the current state of the schedule, looking for opportunities for the tasks in the PSA jobs to back-fill. If at the current time, the application-level scheduler finds a hole to fit in one or more of the PSA tasks, these tasks are shred from their parent PSA job and allowed to back-fill and fill up the hole.

To demonstrate the functionality of the application-level scheduler, let us consider a PSA job using 'p' processors and running for 'nT' time units. Let us assume that this can be broken into 'pn' number of independent sequential tasks, each running for 'T' time units. When this PSA job is submitted, the supercomputer center follows its local scheduling policy to schedule this "parallel" job together with the other submitted jobs. During the scheduling event, a copy of the PSA job is also given to the application-level scheduler which continuously monitors the state of the schedule. Whenever a hole big enough to contain one or more of the tasks in the PSA job is available at the current execution time, the tasks are shred from the PSA job and allowed to back-fill to be executed. When the PSA job reaches its reserved execution time, it coordinates with the application-level scheduler and executes only the tasks which have not already been executed by the application-level scheduler. This might result in the PSA job terminating before its estimated execution time, depending on the number of tasks that were able to back-fill.

For the multi-cluster scenario, we extend this model to have a two-stage hierarchy of application-level schedulers (Figure 3). Each supercomputer center is associated with a local application-level scheduler. Together with this, there's also a higher level meta-application-level scheduler. When a PSA job is submitted to one cluster, multiple copies of the job are created and submitted simultaneously to all the participating clusters. The local application-level schedulers try to schedule this PSA job on the cluster as described in the single-site scenario. However, the distribution of the tasks of a single PSA job over different clusters is handled by the meta-scheduler.

Consider a PSA job using 'p' processors and running for 'nT' time units. Let us again assume that this can be broken into 'pn' number of independent sequential tasks, each running for 'T' time units. As explained earlier, when the job is submitted to the supercomputer center, and is scheduled with the other jobs, a copy of the job is also given to the local application-level scheduler. However, now the local application-level scheduler forwards a copy of the job to the local application-level schedulers on the other participating clusters and informs the meta-scheduler about this. Each local application-level scheduler tries to schedule the PSA job in it's cluster and informs the meta-scheduler every time it completes a task in the PSA job. The meta-scheduler explicitly exchanges information with each local application-level scheduler to make sure that the tasks executed on one cluster are not re-executed on the others. Once all the tasks of the PSA job are executed (potentially in parts on the different clusters), all the local application-level schedulers are

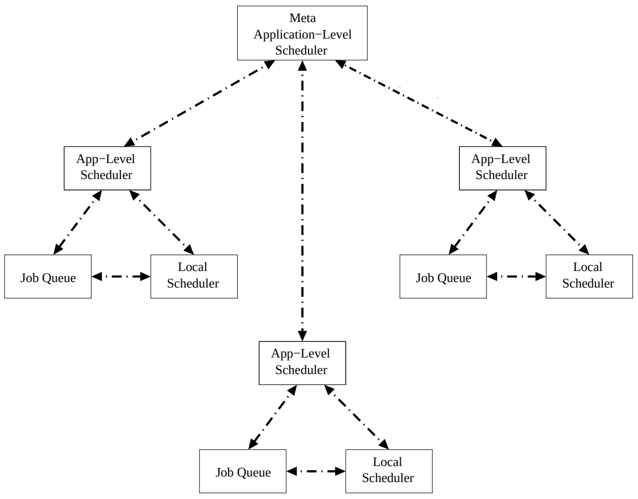


Figure 3. Hierarchical application-level schedulers for Multi-Site Scheduling

notified.

It is to be noted that this scheme does not modify the existing scheduling mechanisms at the clusters. It relies on an external application-level scheduler which keeps track of the current schedule of the cluster to support the "Opportune Job Shredding" model for PSA jobs. Further, in the multi-cluster scenario, each of the clusters can have different processing power, both in terms of the number of nodes available and also the processing speed of the nodes.

An interesting scenario is when a PSA job is submitted to a cluster and one of the participating clusters does not have enough processors to support this job. For example, let us consider that the LANL cluster, the San Diego Supercomputing Center (SDSC) and the Cornell Theory Center (CTC) are the participating clusters. The LANL cluster has 1024 processors. It is possible that a PSA job requesting 800 processors is submitted to this cluster. However, the Cornell Theory Center only has 512 processors and would not be able to support this job as it is. In such a scenario, we take advantage of the moldability of the PSA jobs to reassemble the tasks in the PSA job to form a 512 processor parallel job and submit a copy to CTC.

5 Experimental Results

In this section we use a simulation based approach to evaluate the impacts of the two schemes, "Flooding-based Job Shredding" and "Opportune Job Shredding", on both the PSA jobs and the Non-PSA jobs. We also study the impact of the schemes on the different categories of the jobs within the PSA and Non-PSA jobs, such as short-narrow, short-wide, long-narrow and long-wide. The simulations for Sections 5.1 and 5.2 were carried out on a 5000-job subset of the CTC trace from Feitelson's archive. For Section 5.3, we used three 1-month subsets of the CTC trace. Throughout this section, a number of jobs using more than 8 processors are seelcted randomly and treated as PSA jobs. In general, the breakdown for the time taken by each task in the PSA job is specified by the PSA job. For example, a breakdown factor of 'b' means that the total runtime of the PSA jobs is 'b' times the runtime of each task. In other words, the PSA job can be broken down into 'b' tasks along the time axis. We chose a value of 10 for 'b' in all our experiments.

We have concentrated mainly on three metrics for evaluating the two schemes. The first two are user-metrics: the turn-around time and the slowdown of the job. The turn-around time of the job is the difference between the time when the job is submitted to the cluster and the time when it completes its execution. The slowdown of the job is defined as the ratio of the turn-around time and the runtime.

The third metric is the Loss of Capacity (LOC) of the system. This metric is defined as zero if the number of queued jobs is zero. Otherwise, it's calculated as follows: If at the current time, there are a number of jobs queued which totally require 'p' processors and there are 'q' idle processors in the cluster, then

ΔLOC = min{p, q} x ΔT, where ΔT is the time for which this state lasts.

The Loss of Capacity of the system is the summation of ΔLOC over the time of interest. The reason we have chosen this metric instead of the Utilization metric is the nature of the submissions of the jobs to the cluster. As described earlier, studies have shown that the submission pattern of jobs by users to the cluster tends to be bursty. In particular, the number of jobs submitted tends to be high during the day, peak somewhere in the evenings and decrease to a low during the night [13]. Assuming that the cluster is not in a state of super-saturation (where the number of jobs submitted is far greater than the resources available, resulting in a continuous and non-terminating increase in the job queue length), the jobs piled during the day time can be expected to be completed during the nights. This results in roughly a 50% Utilization for a number of clusters and is relatively invariant with most alternate schemes.

The results we show here correspond to exact user estimates. We have also conducted experiments based on inaccurate user-estimates. However, the trends seem similar to the exact estimate cases and hence have been omitted from this paper. They can be found in [23].

The values for the bars in the graphs are computed by finding the difference between old metric value and new metric value (old value- new value) and then dividing by old value to get the relative difference. For turnaround time as well as slowdown ,if the metric improves then the new value will be less than old value.So a positive bar indicates that the metric improves and vice-versa.

5.1 Flooding-based Approach

This section shows the impact of the “flooding-based scheme” on the user-metrics for both the PSA jobs as well as the non-PSA jobs and the overall system metrics.

Figure 4 shows the average turn-around time and the average slow-down of the PSA and the Non-PSA jobs when 5% of the jobs are PSA jobs. Although this scheme significantly improves both the user-metrics for the PSA jobs as well as the overall system metrics such as Loss of Capacity (Figure 7), it significantly hampers the performance of the Non-PSA jobs.

Figures 5 and 6 show similar trends for the cases when 10% and 20% of the jobs are PSA jobs respectively. Further, it can also be seen that as the percentage of the PSA jobs increases, the benefit the PSA jobs are able to achieve also tends to decrease. We can also notice a drop in the improvement of the turn-around time for the PSA jobs as load increases.

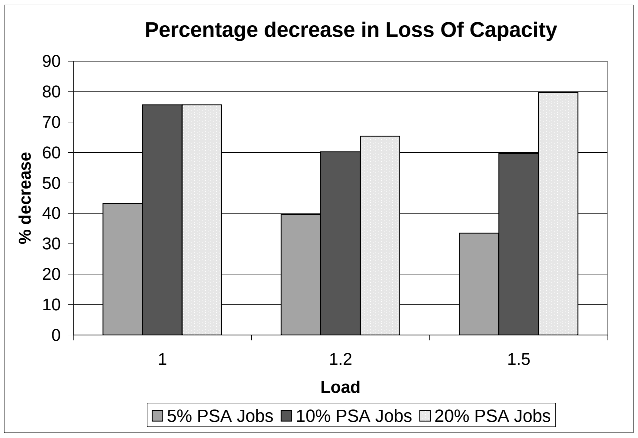


Figure 7. Flooding-based Job Shredding: Loss of Capacity

Though both the turn-around time and the slow-down metrics suffer for the Non-PSA jobs, it is to be noted that the degradation in the slow-down is considerably higher compared to that of the turn-around time. For a clearer understanding of the reason for this, we look at the effect of the flooding-based approach on the various categories of the Non-PSA jobs such as the short-wide, short-narrow, long-wide and the long-narrow jobs.

The jobs are categorized along two dimensions: the number of processors they use and their execution time. Jobs using fewer than 32 processors are categorized as “narrow” and the others as “wide”. Similarly, jobs having execution times less than one hour are categorized as “short” and the others as “long”. Figure 8 shows the average turn-around time and the average slow-down for the different categories of jobs, with 5% PSA jobs. It can be seen that the “short-narrow” jobs suffer much more than the other categories. The reason for this is the reduction in the number of back-fill opportunities available for the “short-narrow” jobs. This category of jobs typically have the most opportunities to back-fill due to their small structure. However, due to the PSA jobs, the multiple segments forming the PSA job flood the network. These segments use up the holes present in the schedule, denying the “short-narrow” Non-PSA jobs opportunities to back-fill.

Figures 9 and 10 show the category wise breakup for the Non-PSA jobs for the cases with 10% and 20% PSA jobs, respectively. These results again point to the conclusion that though an approach such as this can improve the performance of the PSA jobs and that of the overall system (for system metrics such as the Loss of Capacity), it adversely affects the performance of the Non-PSA jobs, especially when the percentage of the PSA jobs is high.

5.2 Opportune Job Shredding

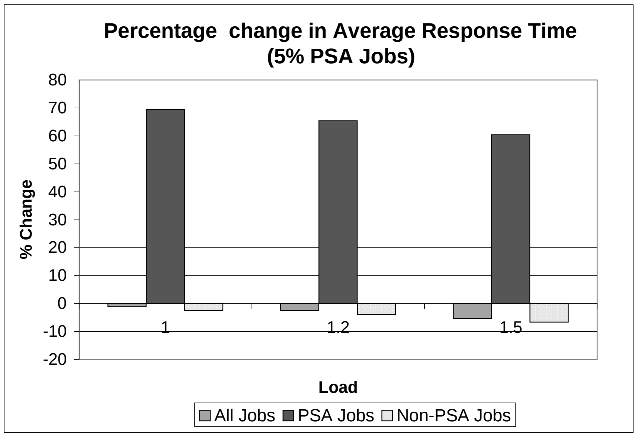
As mentioned earlier, the basic idea of the Opportune Job Shredding scheme is to allow the PSA jobs to shred and back-fill as long as they don’t hamper the back-fill opportunities for the Non-PSA jobs. This section shows the impact of the “Opportune Job-Shredding scheme” on the user-metrics for both the PSA jobs and the non-PSA jobs and the overall Loss of Capacity of the system.

Figure 11 shows the average turn-around time and the average slow-down of the PSA and the Non-PSA jobs when 5% of the jobs are PSA jobs. It can be seen that while this scheme continues to improve the performance of the PSA jobs significantly, it does not adversely affect that of the Non-PSA jobs. The degradation of the performance of the Non-PSA jobs is less than 4% in all cases. In fact, in some cases the better back-filling the scheme allows, results in an improvement in performance for the Non-PSA jobs as well.

Figures 12 and 13 show the impact of the Opportune Job Shredding scheme for the cases when 10% and 20% of the jobs are PSA jobs respectively. It can be seen that the trend continues even for these cases.

To fully understand the impact of the Opportune Job Shredding scheme on the Non-PSA jobs, we’ll now look at the impact of the scheme on the various categories of the Non-PSA jobs such as “short-narrow”, “short-wide”, etc. Figure 15 shows the impact of the scheme on the various categories of the Non-PSA jobs in the case when there are 5% PSA jobs. It can be seen that neither of the categories suffer more than 2.5% for the turn-around time and more than 10% for the slowdown.

Figures 16 and 17 show similar results for the cases when



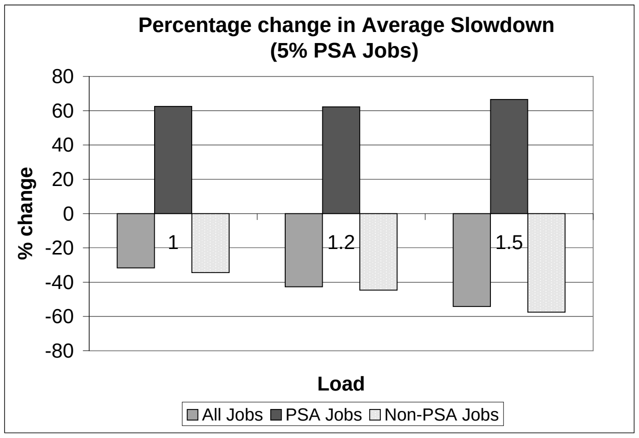
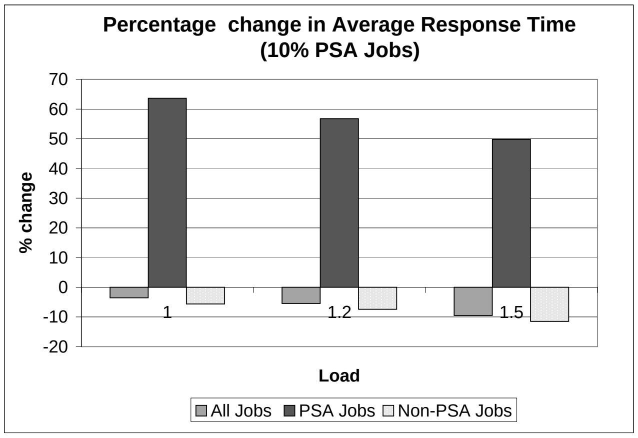


Figure 4. Flooding-based Job Shredding (5% PSA Jobs): (a) Average Response Time; (b) Average Slow Down



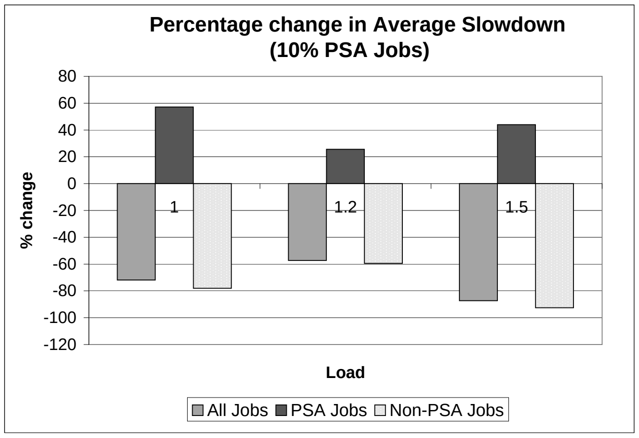
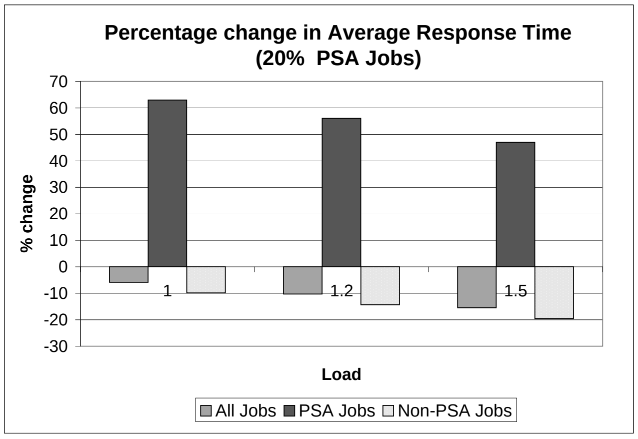


Figure 5. Flooding-based Job Shredding (10% PSA Jobs): (a) Average Response Time; (b) Average Slow Down



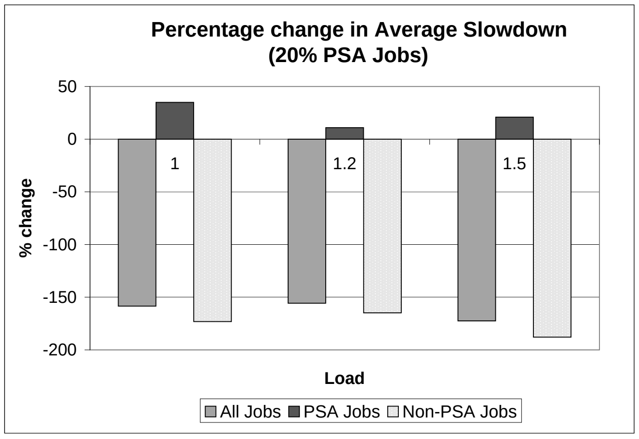
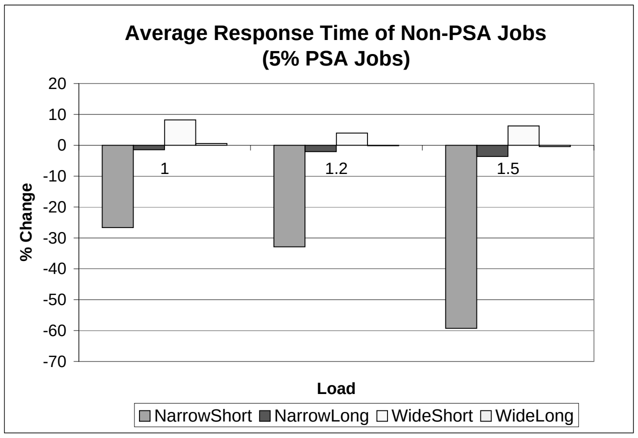


Figure 6. Flooding-based Job Shredding (20% PSA Jobs): (a) Average Response Time; (b) Average Slow Down



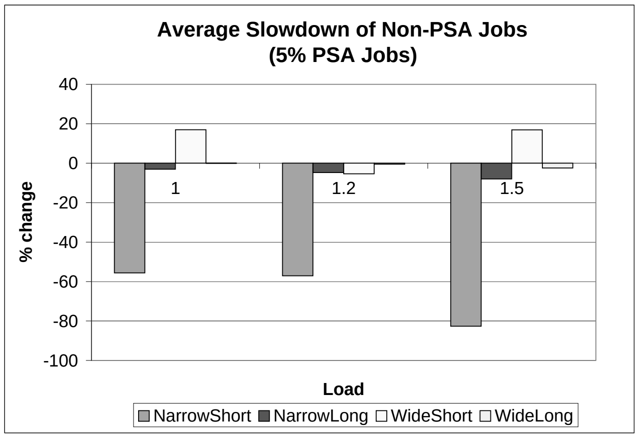
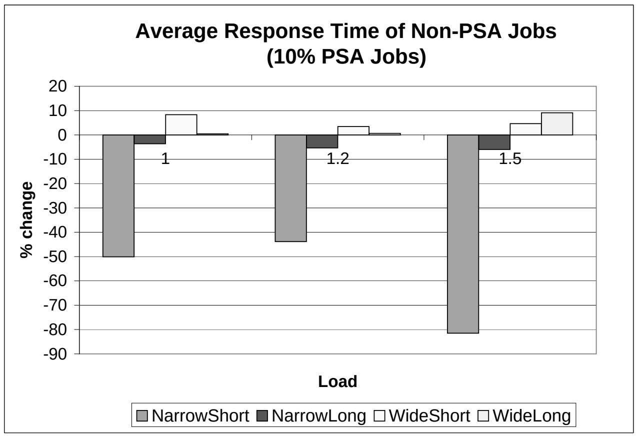


Figure 8. Flooding-based Job Shredding (5% PSA Jobs), Category-wise breakup: (a) Average Response Time; (b) Average Slow Down



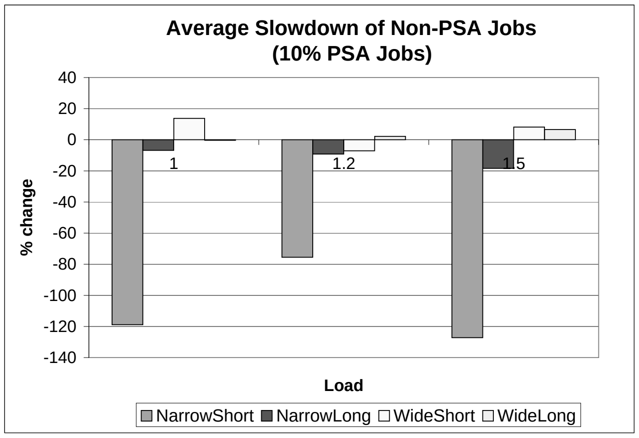
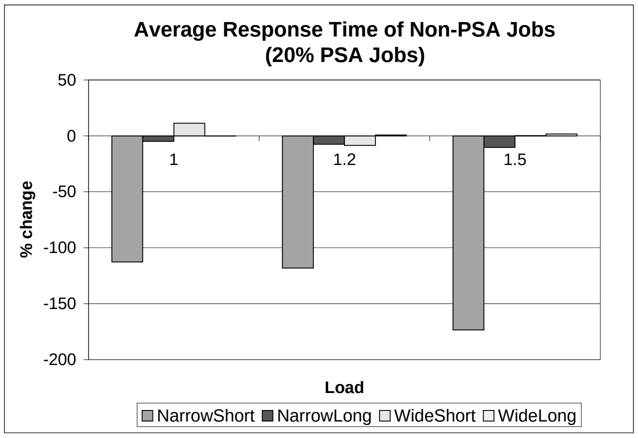


Figure 9. Flooding-based Job Shredding (10% PSA Jobs), Category-wise breakup: (a) Average Response Time; (b) Average Slow Down



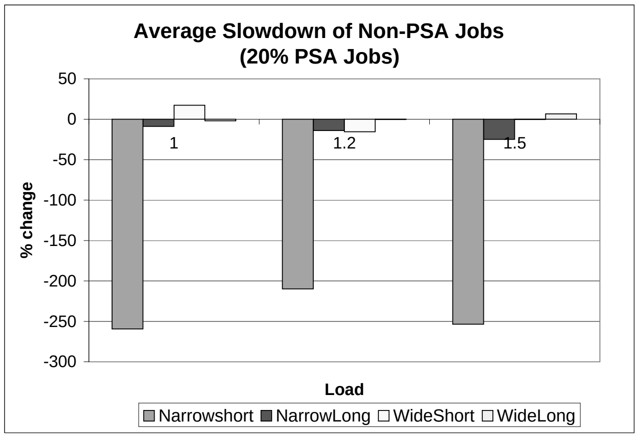
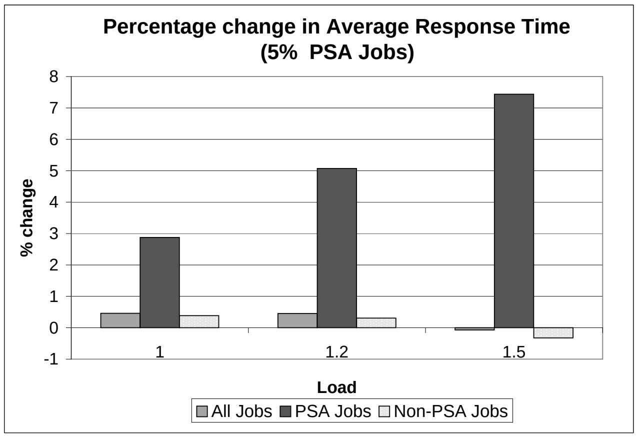


Figure 10. Flooding-based Job Shredding (20% PSA Jobs), Category-wise breakup: (a) Average Response Time; (b) Average Slow Down



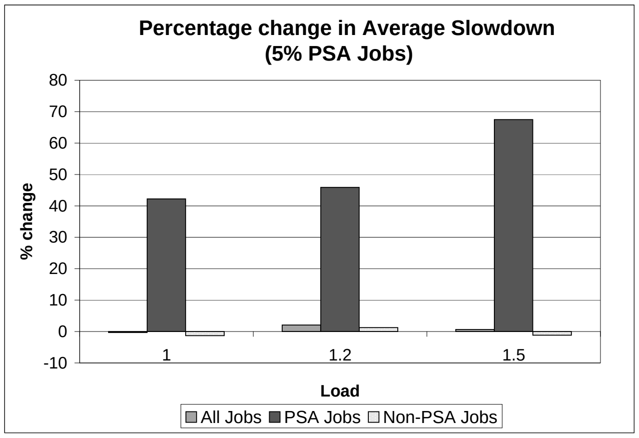
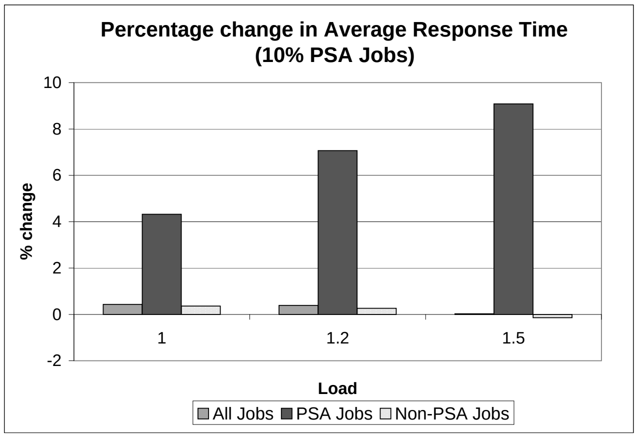


Figure 11. Opportune Job Shredding (5% PSA Jobs): (a) Average Response Time; (b) Average Slow Down



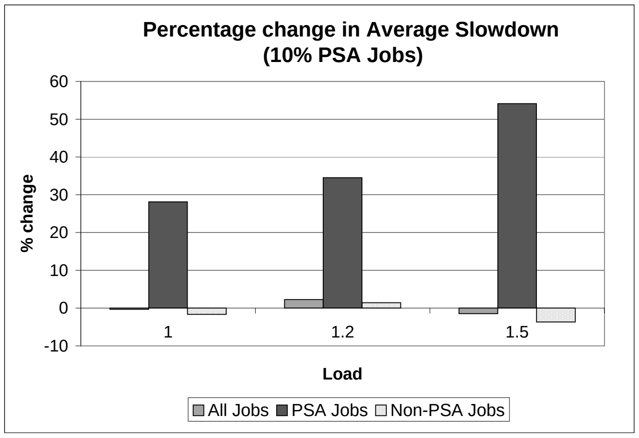
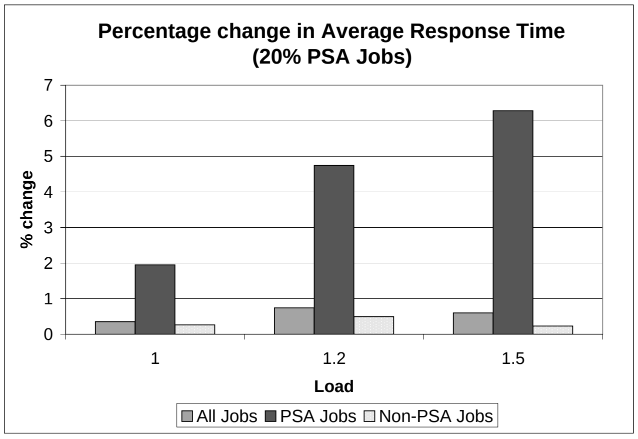


Figure 12. Opportune Job Shredding (10% PSA Jobs): (a) Average Response Time; (b) Average Slow Down



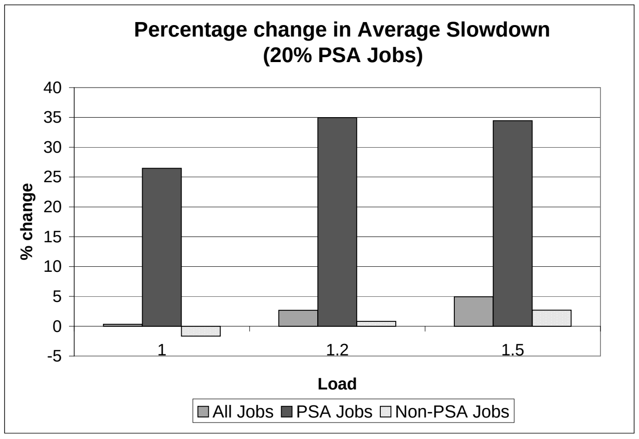


Figure 13. Opportune Job Shredding (20% PSA Jobs): (a) Average Response Time; (b) Average Slow Down

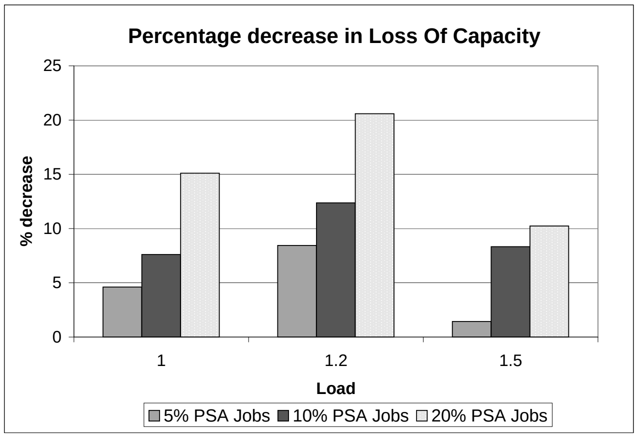


Figure 14. Opportune Job Shredding: Loss of Capacity

there are 10% and 20% PSA jobs respectively. Even as the percentage of the PSA jobs increases, the Non-PSA jobs do not suffer greatly.

5.3 Multi-Site Evaluation

This section evaluates the Opportune Job Shredding scheme in a multi-cluster (or multi-site) environment. As described earlier, in a multi-cluster environment, whenever a PSA job is submitted, a copy of it is sent to the local Application-Level Scheduler of each of the clusters. These Application-Level Schedulers coordinate with the meta-scheduler to execute the different tasks of the application over the clusters. It is to be noted that this scheme is independent of the scheduling mechanism of the individual clusters and takes into consideration the different speeds at which the clusters might operate. Throughout this section, the simulated results correspond to that of three clusters formed by taking three 1-month subsets from the CTC trace. Also, to demonstrate the adaptability of the scheme to heterogeneity in processing speeds for the different clusters, we have considered the processing speeds of the three clusters to be in the ratio 2:1:3.

Figure 18 shows the Average Response Time and the Average Slow-Down of the PSA and Non-PSA jobs in the three clusters for the case when each cluster has 5% PSA jobs. It can be seen that while the performance of the PSA jobs has improved significantly, that of the Non-PSA jobs has hardly been affected. Figures 19 and 20 show similar results for the cases when each cluster has 10% and 20% PSA jobs respectively. The Loss of Capacity of the system is given in Figures 21 through 23.

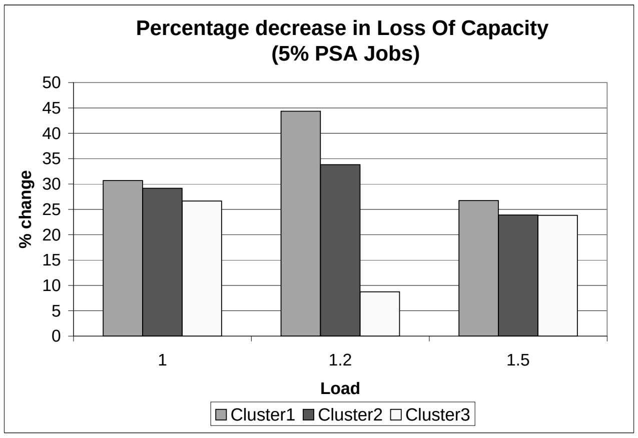


Figure 21. Multi-Site Evaluation (5% PSA Jobs): Loss of Capacity

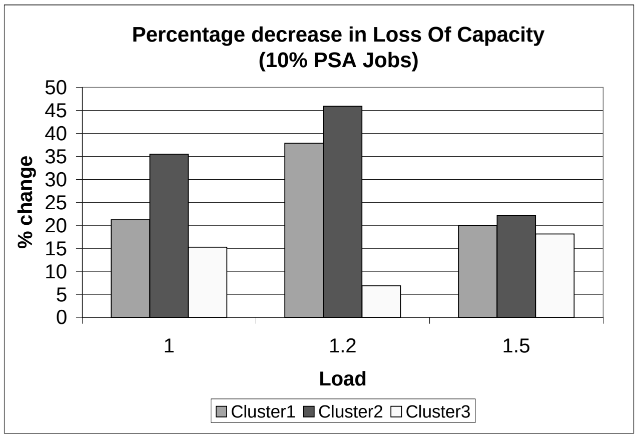


Figure 22. Multi-Site Evaluation (10% PSA Jobs): Loss of Capacity

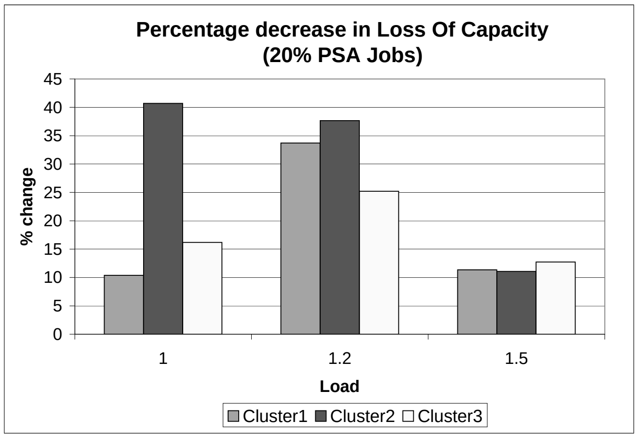
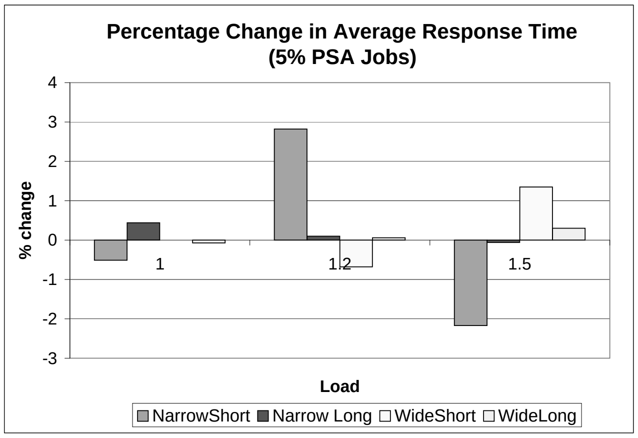


Figure 23. Multi-Site Evaluation (20% PSA Jobs): Loss of Capacity



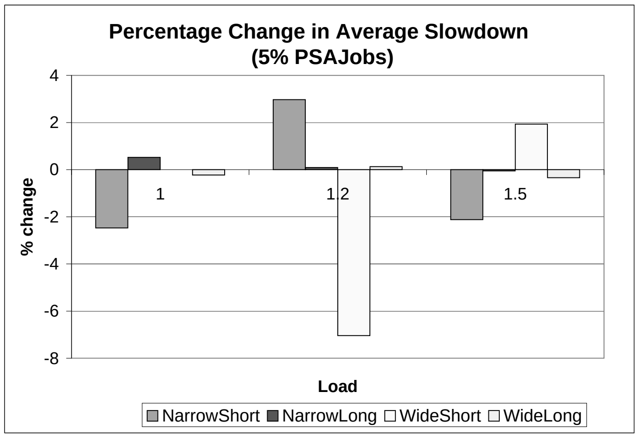
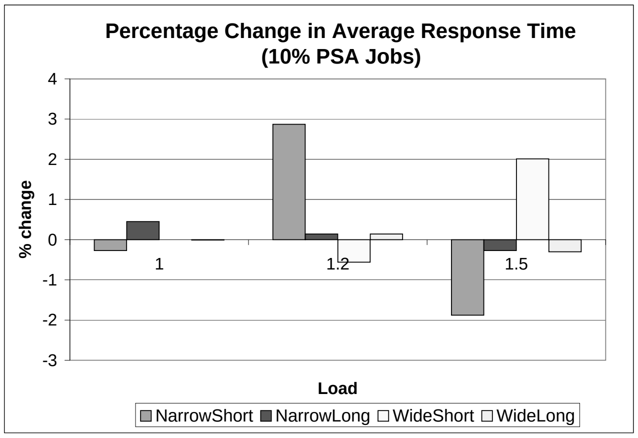


Figure 15. Opportune Job Shredding (5% PSA Jobs), Category-wise breakup: (a) Average Response Time; (b) Average Slow Down



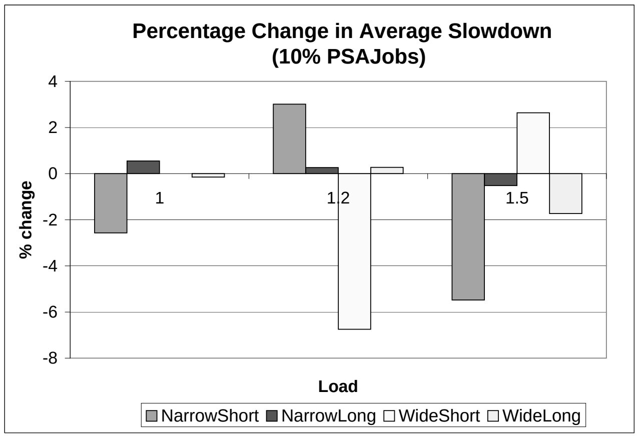
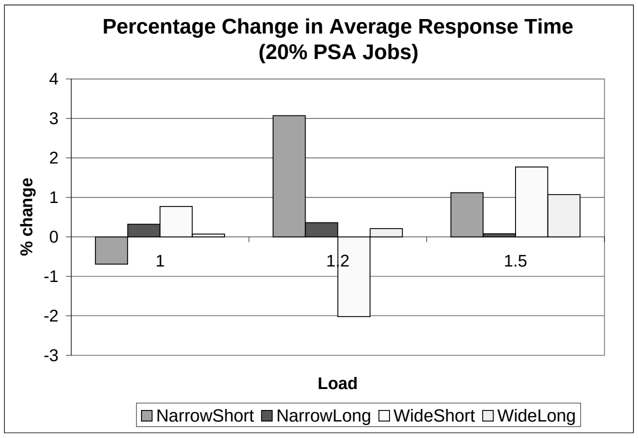


Figure 16. Opportune Job Shredding (10% PSA Jobs), Category-wise breakup: (a) Average Response Time; (b) Average Slow Down



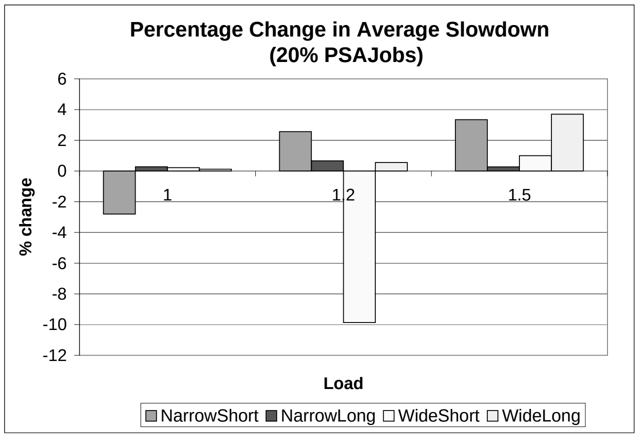
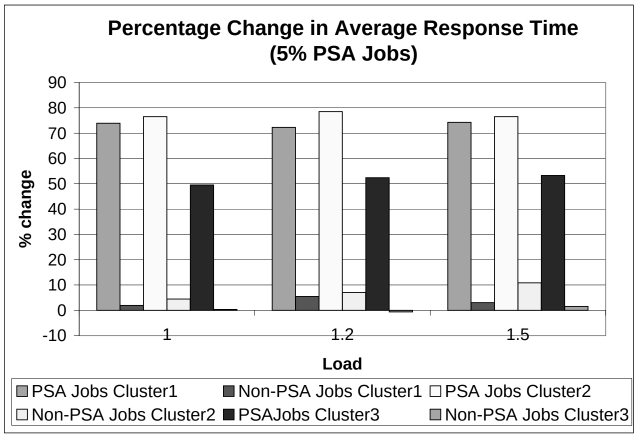


Figure 17. Opportune Job Shredding (20% PSA Jobs), Category-wise breakup: (a) Average Response Time; (b) Average Slow Down



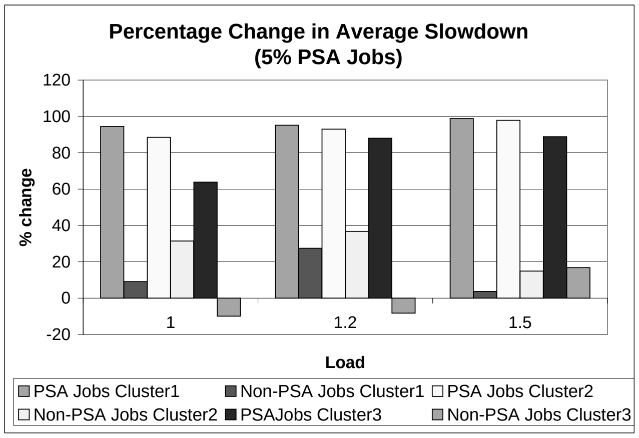
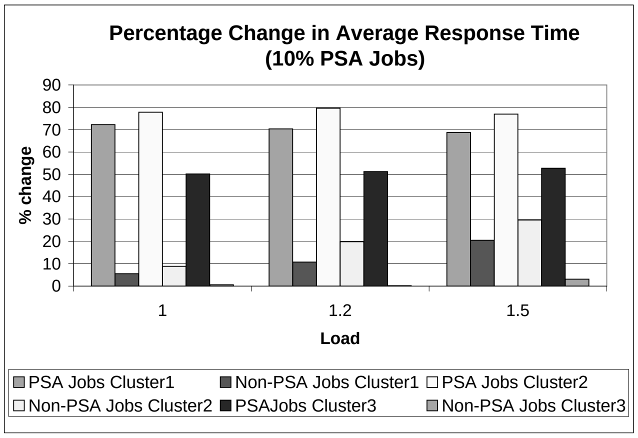


Figure 18. Multi-Site Evaluation (5% PSA Jobs): (a) Average Response Time; (b) Average Slow Down



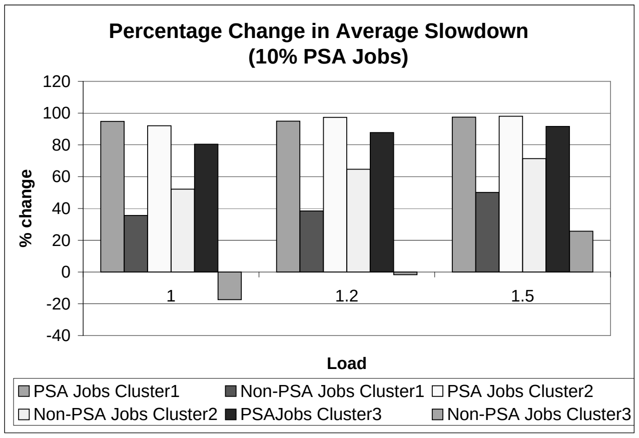
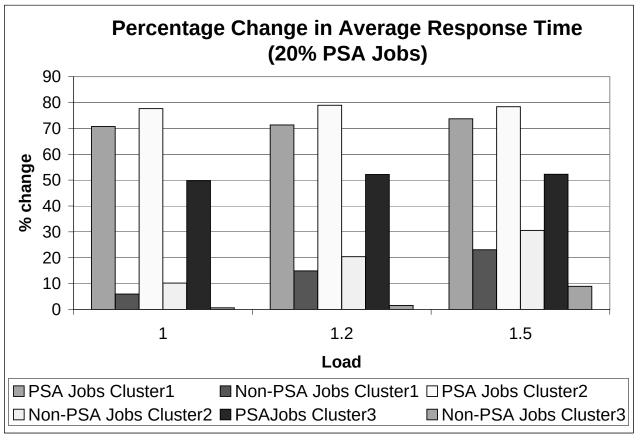


Figure 19. Multi-Site Evaluation (10% PSA Jobs): (a) Average Response Time; (b) Average Slow Down



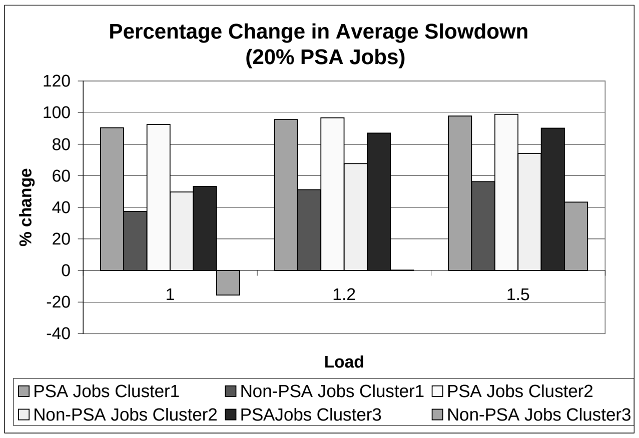


Figure 20. Multi-Site Evaluation (20% PSA Jobs): (a) Average Response Time; (b) Average Slow Down

6 Conclusions and Future Work

A number of applications comprise of several small independent and homogeneous tasks that need to be executed. However, most supercomputer centers enforce a restriction on the number of jobs a single user can submit to the cluster at any time in order to ensure that the resources of the center are not monopolized by a single user.Even if this constraint were relaxed,while PSA scheduling systems like APST could be set up to work in conjunction with schedulers at supercomputer centers, it could result in significant delays of other non-PSA jobs in the system. The avoidance of this problem is the reason why fair-share limits are imposed at supercomputer centers. In this paper we first use trace-driven simulations to characterize the impact of introducing PSA job "fragments" into a mix of non-PSA jobs in a supercomputer center environment. We show that non-PSA jobs are indeed adversely affected. We then propose a new approach, termed as *Opportune Job Shredding*, to significantly overcome the degradation of non-PSA jobs, while still allowing for considerable improvement of PSA jobs. We demonstrate that the proposed scheme improves the slowdown time of PSA's by up to 70% and the overall loss of capacity of the system by up to 21%. Not only does the scheme avoid significant degradation of the performance of the non-PSA jobs in all cases, but it even improves their performance in some cases. We also propose an extension of the previously proposed "Multiple Simultaneous Requests" scheme combining it with the Opportune Job Shredding scheme allowing Parameter Sweep Applications to be executed in parts on remote clusters. We plan next to incorporate and test the Opportune Job Shredding strategy using the Maui scheduler [30] . After testing, we plan to evaluate it on one of the cluster systems at the Ohio Supercomputer Center [7], where PSA jobs represent a significant part of the system load.

References

[1] A. Streit. On Job Scheduling for HPC-Clusters and the dynP Scheduler. In *HiPC*, pages 58–67, 2001.

[2] D. Abramson, J. Giddy, and L. Kotler. High performance parametric modeling with nimrod/G: Killer application for the global grid. pages 520–528.

[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] H. Casanova, T. Bartol, J. Stiles, and F. Berman. Distributing MCell Simulations on the Grid. *The International Journal of High Performance Computing and Supercomputing Applications*, 15(3), Fall 2001.

[5] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: User-level middleware for the grid. pages 75–76, 2000.

[6] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Supercomputing*, November 2000.

[7] Ohio Supercomputer Center. http://www.osc.edu/.

[8] 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.

[9] D. Jackson and Q. Snell and M. J. Clement. Core Algorithms of the Maui Scheduler. In *JSSPP*, pages 87–102, 2001.

[10] D. Lifka. The ANL/IBM SP Scheduling System. In *JSSPP*, pages 295–303, 1995.

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

[12] C. Ernemann, V.Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On advantages of grid computing for parallel job scheduling.

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

[14] 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*. Springer Verlag, 1997.

[15] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S.Tuecke. Condor-g: A computation management agent for multi-institutional grids. In *Proc. Intl. Symp. On High Performance Distributed Computing*, 2001.

[16] J. Gehring and T. Preiss. Scheduling a metacomputer with uncooperative subschedulers. In *In Proc. JSSPP*, pages 179–201, 1999.

[17] J. Gehring and A. Streit. Robust resource management for metacomputers. In *High Performance Distributed Computing*, pages 105–111, 2000.

[18] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *Proc. Grid '00*, pages 191–202, 2000.

[19] H. A. James, K. A. Hawick, , and P. D. Coddington. Scheduling independent tasks on metacomputing systems. In *Parallel and Distributed Systems*, 1999.

[20] J.P. Jones and B. Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In *5th Workshop on Job Scheduling Strategies for Parallel Processing*, 1999.

[21] J.R.Stiles, T.M. Bartol, E.E. Salpeter, and M.M. Salpeter. Monte carlo simulation of neuromuscular transmitter release using mcell,a general simulator of cellular physiological processes. In *Computational NeuroScience*, 1998.

[22] K. Aida. Effect of Job Size Characteristics on Job Scheduling Performance. In *JSSPP*, pages 1–17, 2000.

[23] R. Kurian, P. Balaji, and P. Sadayappan. Opportune Job Shredding: An Efficient for Scheduling Parameter Sweep Applications. Technical report, The Ohio State University, July 2003.

[24] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. In *IEEE Transactions on Parallel and Distributed Systems*, volume 12, pages 529–543, 2001.

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

[26] D. Perkovic and P. J. Keleher. Randomization, speculation, and adaptation in batch schedulers. *Cluster Computing*, 3(4):245–254, 2000.

[27] R. Kettimuthu and V. Subramani and S. Srinivasan and T. B. Gopalsamy and P. Sadayappan. Selective Preemption Strategies for Parallel Job Scheduling. Proceedings of the International Conference on Parallel Processing, 2002.

[28] Grid Research and UCSD Innovation Laboratory. http:// grail.sdsc.edu.

[29] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan. Scheduling of parallel jobs in a heterogeneous multi-site environment. In *In Proc. JSSPP*, 2003.

[30] Maui Scheduler. http://supercluster.org/maui/.

[31] SETI@home. http:// setiathome.ssl.berkeley.edu.

[32] J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY - LoadLeveler API Project. In *JSSPP*, pages 41–47, 1996.

[33] S. Smallen, W.Cirne, J.Frey, F.Berman, R.Wolski, M.Su, C.Kesselman, S.Young, and M.Ellisman. Combining workstations and supercomputers to support grid applications.

[34] Q. Snell, M. Clement, D. Jackson, , and C. Gregory. The performance impact of advance reservation meta-scheduling. In D. G. Feitelson and L. Rudolph, editors, *Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1911 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[35] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan. Distributed job scheduling on computational grids using multiple simultaneous requests. In *Proceedings of the 11th High Performance Distributed Computing Conference*, 2002.

[36] S. S. Vadhiyar and J. J. Dongarra. A metascheduler for the grid. In *11-th IEEE Symposium on High Performance Distributed Computing*, July 2002.