Priority-enabled Scheduling for Resizable Parallel Applications

Rajesh Sudarsan, *Student Member, IEEE,* Calvin J. Ribbens,and Srinidhi Varadarajan, *Member, IEEE*

Abstract—In this paper, we illustrate the impact of dynamic resizability on parallel scheduling. Our ReSHAPE framework includes an application scheduler that supports dynamic resizing of parallel applications. We propose and evaluate new scheduling policies made possible by our ReSHAPE framework. The framework also provides a platform to experiment with more interesting and sophisticated scheduling policies and scenarios for resizable parallel applications. The proposed policies support scheduling of parallel applications with and without user assigned priorities. Experimental results show that these scheduling policies significantly improve individual application turn around time as well as overall cluster utilization.

Index Terms—Dynamic resizing, parallel job scheduling, scheduling policies, resizable applications

1 Introduction

In the last few years, low cost commodity clusters have emerged as a viable alternative to mainstream supercomputer platforms. A typical size of a cluster may range from a few hundred to thousands of processor cores. Although the increased use of multi-core nodes means that node-counts may rise more slowly, it is still the case that cluster schedulers and cluster applications have more and more processor cores to manage and exploit. As the sheer computational capacity of these high-end machines grows, the challenge of providing effective resource management grows as well—in both importance and difficulty. A fundamental problem with existing cluster job schedulers is that they are static, i.e., once a job is allocated a set of processors, it continues to use those processors until it finishes execution. Under static scheduling, jobs will not be scheduled for execution until the number of processors requested by that job are available. Even though techniques such as backfilling [1] and gang scheduling [2], [3] try to reduce the time spent by an job in the queue, it is common for jobs to be stuck in the queue because they require just a few more processors than are currently available. A more flexible and effective approach would support dynamic resource management and scheduling, where the set of processors allocated to jobs can be expanded or contracted at runtime. This is the focus of our research—dynamically reconfiguring or *resizing*) of parallel applications.

Dynamic resizing can improve the utilization of clusters as well as an individual job's turn around time. A scheduler that supports dynamic resizing can squeeze a job that is stuck in the queue onto the processors that are available and possibly add more processors later. Alternatively, the scheduler can add unused processors to a job so that the job finishes earlier, thereby freeing up processors earlier for waiting jobs. Schedulers can also expand or contract the processor allocation for an already running application in order to accommodate higher priority jobs, or to meet a quality of service or advance reservation deadline. More ambitious scenarios are possible as well, where, for example, the scheduler gathers data about the performance of running applications in order to inform decisions about who should get extra processors or from whom processors should be harvested.

We have developed a software framework, *ReSHAPE*, to explore the potential benefits and challenges of dynamic resizing. In our previous paper [4] we described the design and implementations of ReSHAPE and illustrated its potential for individual jobs and work loads. In this paper, we explore the potential for interesting and effective parallel scheduling techniques, given resizable applications and a framework such as ReSHAPE. We describe two typical scheduling policies and explore a set of related scheduling scenarios and strategies. Depending upon the policy, the ReSHAPE scheduler decides which jobs to expand and which to contract. We evaluate the scenarios using a realistic job trace and show that these policies significantly improve overall system utilization and application turn-around time.

Static scheduling is a classic and much studied topic. Feitelson et al. [5], [6] give a comprehensive overview of the recent work in this area. Though most of the recent research on dynamic scheduling has focused on grid environments, a few researchers have focused on cluster scheduling. Weissman et al. [7] describe an application-aware job scheduler that dynamically controls resource allocation among concurrently executing jobs. The scheduler implements policies for adding or removing resources from jobs based on performance predictions from the Prophet system [8]. The authors present simulated results based on supercomputer workload traces. Cirne and Berman [9] describe the adaptive selection of partition size for an application using their AppLeS application level scheduler. In their work, the application scheduler AppLeS selects the job with the least estimated turn-around time out of

- *R. Sudarsan is a student with the Department of Computer Science, Virginia Tech, Blacksburg, VA 24060.*
  E-mail: sudarsar@vt.edu
- *C. J. Ribbens is an Associate professor in the Department of Computer Science, Virginia Tech.*
- *S. Varadarajan is an Associate professor in the Department of Computer Science, Virginia Tech.*

a set of moldable jobs, based on the current state of the parallel computer. Possible processor configurations are specified by the user, and the number of processors assigned to a job does not change after job-initiation time.

The rest of this paper is organized as follows. Section 2 briefly discusses the ReSHAPE framework, highlighting its components and capabilities. Section 3 describes scheduling policies in ReSHAPE for improving application execution turn-around time and overall system utilization, possible scenarios associated with these policies, and the strategies used to build these scenarios. Section 4 describes the experimental setup used to evaluate these scheduling policies and scenarios and their performance.

2 RESHAPE FRAMEWORK

The architecture of the ReSHAPE framework, shown in Figure 1, consists of two main components. The first component is the application scheduling and monitoring module which schedules and monitors jobs and gathers performance data in order to make resizing decisions based on application performance, available system resources, resources allocated to other jobs in the system, and jobs waiting in the queue.The second component of the framework consists of a programming model for resizing applications. This includes a resizing library and an API for applications to communicate with the scheduler to send performance data and actuate resizing decisions. The resizing library includes algorithms for mapping processor topologies and redistributing data from one processor topology to another. ReSHAPE targets applications that are *homogeneous* in two important ways. First, our approach is best suited to applications where data and computations are relatively uniformly distributed across processors. Second, we assume that the application is iterative, with the amount of computation done in each iteration being roughly the same. While these assumptions do not hold for all large-scale applications, they do hold for a significant number of large-scale scientific simulations. The application scheduling and monitoring module includes five components, each executed by a separate thread. The different components are System Monitor, Application Scheduler, Job Startup, Remap Scheduler, and Performance Profiler. We describe each component in turn, detailing their capabilities.

System Monitor. An application monitor is instantiated on every compute node to monitor the status of an application executing on the node and report the status back to the System Monitor. If an application fails due to an internal error or finishes its execution successfully, the application monitor sends a job error or a job end signal to the System Monitor. The System Monitor then deletes the job and recovers the application's resources. For each application, only the monitor running on the first node of its processor set communicates with the System Monitor.

Application Scheduler. An application is submitted to the scheduler for execution using a command line submission process. The scheduler enqueues the job and waits for the requested number of processors to become available. As soon as the resources become available, the scheduler selects the compute nodes, marks them as unavailable in the resource pool, and sends a signal to the job startup thread to begin execution. Different scheduling policies implemented in ReSHAPE are described in detail in Section 3.

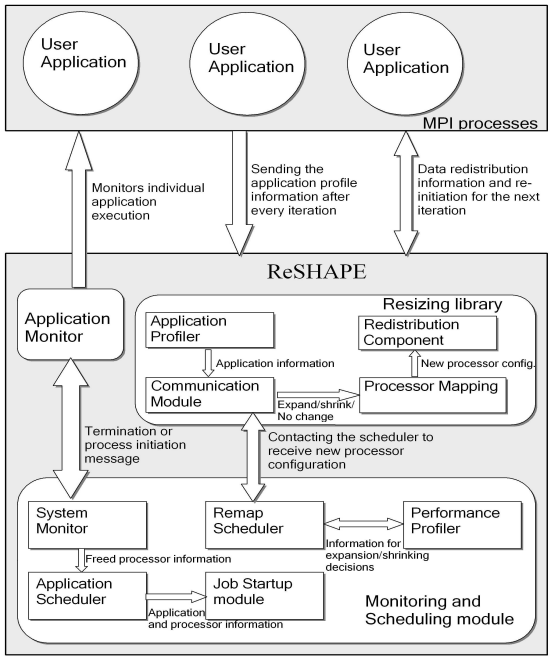

Fig. 1. Architecture of ReSHAPE

Job Startup. Once the Application Scheduler allocates the requested number of processors to a job, the job startup thread initiates an application startup process on the set of processors assigned to the job. The startup thread sends job start information to the application monitor executing on the first node of the allocated set. The application monitor sends job error or job completion messages back to the System Monitor.

Performance Profiler. At every resize point, the Remap Scheduler receives performance data from a resizable application. The performance data includes the number of processors used, time taken to complete the previous iteration, and the redistribution time for mapping the distributed data from one processor set to another, if any. The Performance Profiler maintains lists of the various processor sizes each application has run on and the performance of the application at each of those sizes. Note that applications can only contract to processor configurations on which they have previously run.

Remap Scheduler. The point between two subsequent iterations in an application is called a resize point. After each iteration, a resizable application contacts the Remap Scheduler with its latest iteration time. The Remap Scheduler contacts the Performance Profiler to retrieve information about the application's past performance and decides to expand or shrink the number of processors assigned to an application according to the scheduling policies enforced by the particular scheduler. The size and topology of the expanded processor set can be problem and application dependent. In our current implementation we require that the global data be equally distributable

across the new processor set. Furthermore, at job submission time applications can indicate (in a simple configuration file) if they prefer a particular processor topology, e.g., a rectangular processor grid. In the case where applications prefer "nearly-square" topologies, additional processors are added to the smallest row or column of the existing topology.

The ReSHAPE architecture is described in detail in Sudarsan and Ribbens [4]. The resizing library, API and the redistribution algorithms implemented in ReSHAPE are described in detail in Sudarsan and Ribbens [10].

3 SCHEDULING WITH RESHAPE

We use the term *scheduling policy* to refer to an abstract high-level objective that a scheduler strives to achieve when scheduling arriving jobs. For example, one scheduling policy might be to minimize individual application turn-around time while keeping overall system utilization as high as possible. Clearly, such a policy may not be achievable in a mathematically optimal sense. Rather, a policy simply gives an indication of the approach to scheduling used on a particular parallel resource. Given such high-level policy, a *scheduling scenario* defines a specific, concrete attempt at achieving the scheduling policy. A scenario defines a procedure that the scheduler is configured to follow in order to achieve the objectives dictated by the policy. In the context of the ReSHAPE framework for dynamically resizable applications, a scheduling scenario must answer three fundamental questions: when to resize a job, which jobs to resize, and which direction to resize (expand or contract). We use the term *scheduling strategies* to refer to specific underlying methods or algorithms, implemented to realize resizing decisions. These methods or strategies define whether a job should be expanded or contracted and by how much. For example, the ReSHAPE scheduler could use a strategy which selects those jobs for expansion that are predicted to have maximum benefit from an expansion. Similarly, a strategy for harvesting processors might be to choose those jobs that are expected to suffer the least impact from contraction. In summary, a scheduling policy can be implemented in multiple scenarios, each realized using a particular collection of strategies. In the following subsections we briefly describe some simple strategies and scenarios that are implemented in ReSHAPE and which we use to illustrate the power of ReSHAPE for parallel scheduling research and development.

3.1 Calculating priority for user applications

Most commercial job schedulers (like Moab [11], PBS [12], Maui [13]) assign an internal priority to parallel applications submitted through them. Based on these priority values, the scheduler determines how quickly should a job waiting in the queue be scheduled. The priority typically depends on the number of processors requested by the job, its walltime and the time it has been waiting in the queue. This priority value, generally referred to as *aging* priority, is used to benefit small jobs (both in terms of processor count and walltime), keeping them from waiting too long in the queue. ReSHAPE derives its relation to calculate aging priority from the Moab scheduler. The relation is as follows:

$$\begin{aligned} Job\_priority = Qfactor\_weight * Qfactor + \\ queue\_time\_weight * queue\_time + \\ nnodes\_weight * nnodes \end{aligned}$$

where, $queue\_time$ indicates the total time the job has been waiting in the queue and $nnodes$ indicates the number of nodes requested by the job. $Qfactor\_weight$, $queue\_time\_weight$ and $node\_weight$ are set as configuration parameters for the scheduler. The weights determine the impact of each variable in the final priority value. $Qfactor$ determines the urgency of a job to get scheduled. It mainly benefits short running jobs. It is computed as:

$$Qfactor = 1 + \frac{(queue\_time)}{max(Qfactorlimit, walltime)}$$

The value of $Qfactorlimit$ is generally set to 1 to compute $Qfactor$ based on walltime. In addition to aging priority, some schedulers also allow user priority for jobs, i.e., jobs submitted by high priority users must be given preference in scheduling compared to others.

If a user priority is assigned to jobs, then the absolute priority for a job is calculated as follows.

$$\begin{aligned} Job\_priority = Qfactor\_weight * Qfactor + \\ queue\_time\_weight * queue\_time + \\ nnodes\_weight * nnodes + user\_assigned\_priority \end{aligned}$$

For an application that is already executing, its priority value is updated based on how much walltime is remaining for that application to finish its execution. An application that is closer to finishing its execution will be assigned a higher priority than an application that has just started its execution. Priority value for running application is computed as follows.

$$\begin{aligned} Job\_priority = (100 - pct\_time\_left) + \\ user\_assigned\_priority \end{aligned}$$

where,

$$pct\_time\_left = \frac{(walltime - time\_elapsed)}{walltime} * 100$$

3.2 Priority-based scheduling strategies

Scheduling strategies can be categorized into processor allocation and processor harvesting strategies. A processor allocation strategy decides which applications to expand and by how much whereas a processor harvesting strategy decides which applications to contract and by how much. In our current implementation, all allocation strategies use a simple model to predict the performance of a given job on a candidate processor size where that job has not yet run. Data from previous iterations on smaller processor allocations is used to inform contraction decisions. This combination of predictive

model and historical data is also used to predict the time an application will take to reach its next resize point. An application must be expanded a minimum number of times before it is considered as a candidate for a resizing strategy. The minimum number is indicated by *remap_window_size* and its value is set by the system administrator. The *expand potential* for an application at a resize point is a quantified value of performance of an application at that resize point. It indicates an application's ability to benefit from more processors beyond its current resize point. The performance measure can be any reasonable metric such as speedup, computational rate, etc. The *expand potential* is calculated only after the application has been resized *remap_window_size* times. In the current implementation of ReSHAPE, the *expand potential* is calculated by fitting a polynomial curve to a speedup curve of that application at each of its last *remap_window_size* resize points and computing the slope of the curve at its current resize point. The larger the value of the expand potential, the greater the chances that the job will benefit from a further expansion. The scheduling policy includes a minimum threshold which expand potential must exceed in order to warrant additional processors for a given job. A job that has reached its *sweet spot* is not eligible for additional processor allocation. (The *sweet spot* is an estimate of the processor count beyond which no performance improvement is realized for a given job.) However, an application can be contracted below its sweet spot. The ReSHAPE scheduler supports both aging and user assigned priorities for scheduling parallel applications. Once an application starts execution, the ReSHAPE scheduler updates its priority based on the amount of walltime left for its completion. Thus an accurate user runtime estimate will help in better scheduling and resizing decisions.

For scheduling of priority-based applications, ReSHAPE supports two processor harvesting strategies — Least-Impact-Priority and FCFS-contract-priority — and one processor allocation strategy — Max-Benefit-priority. We have also implemented other scheduling strategies without priorities. These scheduling strategies with their corresponding scenarios and policies are described in detail in Sudarsan and Ribbens [14]. We believe that the scheduling strategies and scenarios presented in this paper depict a more realistic view in which jobs are scheduled in a typical cluster. One of main motivations of ReSHAPE is to provide a platform to experiment with more interesting and sophisticated scenarios and policies for resizable parallel applications. For the remaining part of the discussion in this section, we assume all jobs to be resizable.

Least-Impact-Priority: In this processor harvesting strategy, jobs are contracted in the ascending order of their expected performance impact suffered due to contraction. At every resize point, a list is created to indicate all the low and high priority jobs that are running and their possible performance impact at the next resize point. In the list all the low priority jobs are listed above the high priority jobs. Within each set of high and low priority jobs in the list, the jobs are sorted in ascending order of the expected performance impact. If there are jobs waiting to be scheduled, the list is traversed till the required number of processors can be freed. A high priority running job will not be contracted to schedule a lower priority queued job. The current job is contracted to one of its previous processor allocations if it is one of the possible candidates in the traversed list, i.e., if it is encountered on the list before the total number of desired processors has been identified and has a lower priority than the first queued job. The procedure is continued till the required number of processors are available or till all jobs have reached their starting processor configuration.

FCFS-Contract-Priority: In this processor harvesting strategy, jobs are contracted in the order they arrive at their resize point. A high priority job that arrives at its resize point will be contracted only if the queued job has a higher priority than the current job. A low priority job will be contracted to it previous resize point irrespective of the priority of queued jobs.

Max-Benefit-Priority: In this processor allocation strategy, idle processors are allocated to jobs arranged in the descending order of their expand potential at their last resize point. In other words, we allow a job to grow that is predicted to benefit most when expanded to its next possible processor size. A list, sorted in the descending order of job priority, is created at every resize point for an application. Within each set of high and low priority jobs in the list, they are again sorted in the descending order of their expand potential at their last resize point. A job is allowed to expand if it is the first job in the list. If a job does not find itself at the top of the list, then follow the steps listed below:

* For every job that has a higher priority than the current job, count the number of processors required to expand that job to its next possible larger size.
* If two or more jobs have the same priority, then for every job that has a higher expand potential than the current job and is expected to reach its resize point before the current job's next resize point, count the number of processors required to expand that job to its next possible larger size.
* If there are still sufficient idle processors remaining after the above "pre-allocation" steps, then assign them to the current job, and expand.

3.3 Priority-based Scheduling policies and scenarios

To illustrate the potential of ReSHAPE for priority-based applications, we define two typical scheduling policies as follows. The first policy aims at improving a priority application's turn around time and the overall system utilization by favoring queued applications over running applications. In this policy, a low priority job in the queue will not be scheduled before a a high priority job. Also, a low priority running application will not be allowed to expand unless all the queued jobs have been scheduled. On the other hand, a high priority application will be allowed to expand at its resize point if all the queued jobs have a lower priority than this job. The second policy aims at improving a priority application's turn around time and the overall system utilization by favoring running applications over queued applications. In this policy, running (high and low priority) applications are favored over queued applications and are allowed to expand to their next valid processor size, irrespective of the number of queued jobs.

Among running jobs, a high priority job will be favored for expansion compared to a low priority job. To realize these policies, we describe different scheduling scenarios. These scenarios are implemented by combining different scheduling strategies that support priority. Two of the these scenarios aim to achieve the objective of the policy that favors queued jobs. They are Performance-based-allocation with priority (PBA-PR) and FCFS-Max-Benefit with priority (FCFS-PR). The third scenario which aims to achieve the policy that favors running applications is referred to as Max-benefit with priority (Max-B-PR). All the scenarios (described in detail below) support priority-based backfilling as part of their queuing strategy. ReSHAPE uses a aggressive backfilling (EASY) [1], [15] technique to move smaller jobs to the front of the queue. In an aggressive backfilling technique, only the job at the head of the queue has a reservation. A small job with a higher or lower priority is allowed to move to the top of the queue as long as it does not delay the first queued job. All jobs arriving at the queue may or may not have equal priority. As jobs wait in the queue, their priority changes depending on their job size and how long they have been waiting in the queue. All jobs are queued in the descending order of their priority and are scheduled if the requested number of processors becomes available.

Policy 1: Improve application turn-around time and system utilization for applications with priority, favoring queued applications.

Scenario 1: Performance-based allocation with priority (PBA-PR). In this scenario, jobs are expanded using the *Max-Benefit-Priority* processor allocation strategy and contracted using the *Least-Impact-Priority* harvesting strategy. The procedure followed to determine whether to expand or contract a job or to maintain its current processor set size is detailed below.

* When a job at its resize point contacts the scheduler for remapping, check whether there are any queued jobs with a higher priority than the current job. If there are higher priority jobs waiting in the queue, then contract the current job if it is selected based on the *Least-Impact-Priority* processor harvesting strategy.
* If there are queued jobs but the current job is not selected for contraction, check whether the application has already reached its sweet spot processor configuration. If it has, then maintain the current processor size for the job.
* If the application has not yet reached its sweet spot configuration, check whether the current job benefited from its previous expansion. If it did not then contract the job to its immediate previous configuration and record the application sweet spot.
* If the first queued job cannot yet be scheduled using the processors harvested from the current job, then schedule as many waiting jobs as possible using priority-based backfill.
* If there are no queued jobs and if the application benefited due to expansion at its last resize point, then expand the job using the *Max-Benefit-Priority* processor allocation strategy.
* If the job cannot be expanded due to insufficient processors, then maintain the job's current processor size.
* If there are idle processors after backfill and if there are still queued jobs, then expand the current job using the *Max-Benefit-Priority* strategy.

Scenario 2: FCFS-Max-Benefit with priority (FCFS-PR): The FCFS scenario uses the *Max-Benefit-Priority* processor allocation strategy to expand jobs and the *FCFS-Contract-Priority* processor harvesting strategy to contract jobs. The procedure followed in this scenario is identical to the steps in Scenario 1. The only difference is that the jobs are contracted using the FCFS-contract harvesting strategy.

Policy 2: Improve application turn-around time and system utilization for applications with priority, favor running applications. The scenarios implemented in this policy do not consider the number of queued jobs in their resizing decision and expand jobs if enough processors are available. Schedulers implementing this policy will not contract running jobs to schedule queued jobs.

Scenario 1: Max-benefit with priority (Max-B-PR).

* When a job contacts the scheduler at its resize point, check whether the job benefited from expansion at its last resize point. If it did not, then contract the job to its previous processor size.
* If it benefited from previous expansion, then expand the job to its next possible processor size using the *Max-Benefit-Priority* processor allocation strategy.
* If the job cannot be expanded due to insufficient processors, then maintain the job's current processor size.
* Schedule queued jobs using priority-based backfill to use the idle processors available in the cluster.

4 Experiments and Results

This section presents experimental results to demonstrate the potential of dynamic resizing for parallel job scheduling As mentioned above, we consider two broad categories of scheduling policies, one that favors queued applications and other that favors running applications. All the applications used in the following experiments are assigned aging priorities by the scheduler. We consider two scenarios (PBA-PR and FCFS-PR) for the policy that favors queued applications assigned with priorities and one scenario (Max-B-PR) for the policy that favors running applications. All policies seek to reduce job turn-around time while maintaining high system utilization.

The experiments were conducted on 400 processors of a large homogeneous cluster (System G at Virginia Tech). Each node is a dual socket quad-core 2.8 GHz Intel Xeon processors with 8GB of main memory. Message passing uses Open-MPI [16], [17] over an Infiniband interconnection network. We perform three experiments to evaluate the above described scheduling policies. The first experiment using this setup compares the performance of three priority-based scenarios

(PBA-PR, FCFS-PR, Max-B-PR) against static scheduling with backfill. Although the jobs are assigned any user assigned priority, the scheduler assigns a aging priority to all the queued and running jobs. In the second experiment, we vary the percentage of resizable jobs in a workload to see whether even a small percentage of these jobs can have an impact on the overall system utilization and on individual application's turnaround time. Even in this case the jobs are not assigned any user priority but are assigned aging priority by the scheduler. In the third experiment, we assign categorize jobs as high and low priority user applications and evaluate the impact of user priorities on a performance of these scheduling scenarios. The job mix used in these experiments consists of synthetic applications which do not perform any computation. The execution time for these synthetic applications at different processor sizes are computed using the speedup model described in Section 4.1. No data was redistributed at resize points as the focus of this paper is on evaluating job scheduling policies in ReSHAPE. Other characteristics of the experimental setup are as follows:

1) Each workload consists of 120 jobs. The percentage of resizable jobs in the workload varies with each experiment.
2) Each job in the workload can be either a small, medium or large job. The fraction of small, medium and large jobs in the job mix is set to $\frac{1}{3}$.
3) The initial number of processors, expected walltime and the execution time for one iteration on the initial processor allocation for different job sizes are as follows:
   - Small jobs: 35 processors (32 for power-of-2 processor topology), 156 seconds, 8 seconds/iteration
   - Medium jobs: 81 processors (64 for power-of-2 processor topology), 240 seconds, 20 seconds/iteration
   - Large jobs: 136 processors (128 for power-of-2 processor topology), 324 seconds, 32 seconds/iteration
4) Each job in the workload is assigned one of the following processor topologies — arbitrary, nearly-square or power-of-2. The percentages of jobs with specific processor topologies in the workload are as follows: 60% arbitrary, 30% nearly-square, 10% power-of-2.
5) The arrival time of jobs is calculated using a Poisson distribution with a mean arrival time set at 32 seconds.
6) Each job runs for 7 iterations and tries to resize after every iteration.

The number of processors for small, medium and large jobs (arbitrary and nearly square topology) were randomly chosen so that following conditions are met — sum of processor set size of two small jobs is not equal to a medium or large job, sum of processor set sizes of a medium and small job is not equal to a large job and finally, sum of processor set sizes of two medium jobs is not equal to the processor set size of a large job.

4.1 Cluster workload and parallel application speedup model

Various workload models are available to model rigid and moldable jobs. Calzarossa and Serazzi [18] propose a model for the arrival process for interactive jobs in a multi-cluster environment. It gives arrival rate as a function of the time of day. Leland and Ott [19] propose a model for calculating the runtime of processes for interactive jobs in a Unix environment. Sevcik [20] proposes a model for application speedup which is useful for jobs scheduled on varying partition sizes. The model proposed by Feitelson [3] uses six job traces to characterize parallel applications. This model provides a distribution of jobs based on processor count, correlation of runtime with parallelism and number of runs in a single job. Downey [21], Jann et al. [22] and Lublin [23] provide workload models for rigid jobs. For the model proposed by Jann et al., new parameters were introduced later to evaluate job scheduling on the ASCI Blue-Pacific machine. Cirne and Berman [24] propose a comprehensive model to generate a stream of rigid jobs and a model to transform them to moldable jobs. The model proposed by Tsafrir [25], [26] generates realistic user runtime estimates that helps schedulers to make better backfilling decisions.

The above workload models do not support resizability or malleability in jobs. We have developed a simple model for application speedup for resizable applications based on four parameters — current processor size P1, resized processor size P2, runtime at current processor size R(P1) and node efficiency α. The factor by which the runtime for a given fixed computation will change at P2 is given by

$$factor = (\frac{P2}{P1})^{(\alpha * \frac{P2-P1}{P1})}, 0 < \alpha \leq 1,$$

and the new runtime at processor set size P2 is given by

$$R(P2) = \frac{R(P1)}{factor}.$$

Intuitively, the node efficiency measures how much useful work we get out of the $(P2 - P1)$ new processors. For the experiments in this section, we use $\alpha = 0.8$

We have also developed a synthetic applications framework which uses this speedup model to generate a workload of jobs. The framework uses a set configurable parameters such as job size (both in terms of processor size and runtime), processor topology, application resizability and priority.

The jobs in the workload for the experiments in this experimental setup were generated using our synthetic applications framework. A synthetic job in the workload has one of following job size — small, medium or large. Similarly, these jobs support one of the following processor topologies — *arbitrary*, *nearly-square* or *power-of-2*. An application with *arbitrary* processor topology does not have any restrictions on the possible processor expansion sizes. A job with *nearly-square* processor topology will expand to a processor size so that the nearest square processor topology is maintained. A job with *power-of-2* processor topology will expand only to processor sizes that are powers of 2.

Figure 2 shows the speedup curve for a job scaled from a initial value of $P\_0 = 400$. P varies from 400 to 2000 processors with a step size of 20 processors. The initial runtime (T_0) was set at 200 seconds and $\alpha = 0.75$. The new

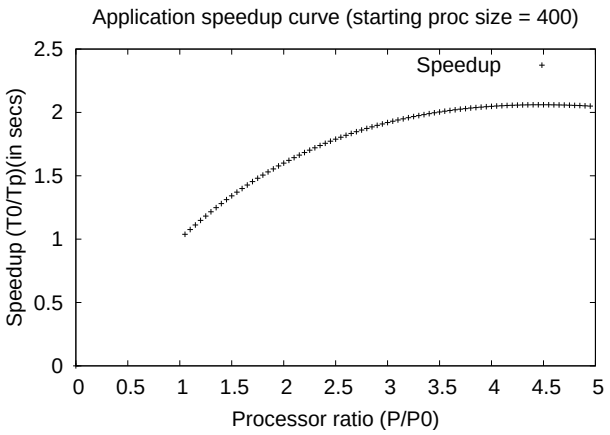

Fig. 2. Speedup curve.

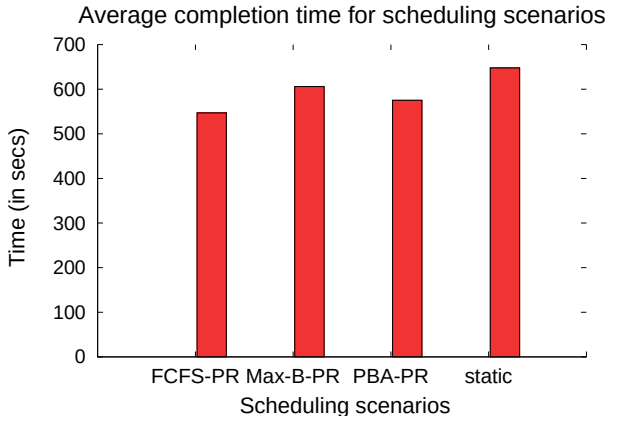

Fig. 3. Average completion time for various scheduling scenarios.

runtimes are calculated using the above described model. This model provides a realistic view of the speedup characteristics of a typical parallel application running on a cluster. A simpler model with a linear speedup for applications can significantly improve the overall utilization of a cluster but such a model is scientifically uninteresting and does not depict a realistic view of a typical workload in a cluster. In our model, depending on the initial processor set size an application may or may not be able to detect its execution sweet spot.

4.2 Workloads with 100% resizable jobs

The goal of this experiment is to demonstrate the potential of different scheduling scenarios in ReSHAPE using workloads where all jobs are resizable. We compare the performance of FCFS-PR, Max-B-PR and PBA-PR scenarios with a baseline scheduling scenario — static scheduling with backfill. The results are averaged across seven different runs. Each run or job mix represents a random arrival order for jobs with random inter-job arrival times. In this experiment we make the following claims.

We claim that scheduling a workload of resizable jobs using ReSHAPE improves the overall completion time for all jobs. Figure 3 shows the average completion time for the four scheduling scenarios. From the graph, we see that FCFS-PR improves the average job completion time by 15.5% compared to static. The jobs scheduled using PBA-PR and Max-B-PR scenario show an improvement of 11.1% and 6.4% respectively. The relatively small improvement in average completion time for the Max-B-PR scenario is because most of the jobs experience a high queue wait time. FCFS-PR has the highest standard deviation compared to PBA-PR and Max-B-PR for improvement in average job completion time across 7 runs at 57.40. The standard deviation for Max-B-PR and PBA-PR are 43.31 and 48.80 respectively. The reason for high standard deviation for average completion time is due to the large variation in queue wait times across different runs.

Since the scenarios are designed to achieve a specific objective, ReSHAPE provides the flexibility of choosing a scenario based on the scheduling policy set for a particular queue or for the entire system. For example, if the objective is to only improve the execution time of all the jobs without being concerned about their queue wait time, then Max-B-PR scenario emerges as the best scenario. Figure 4 compares

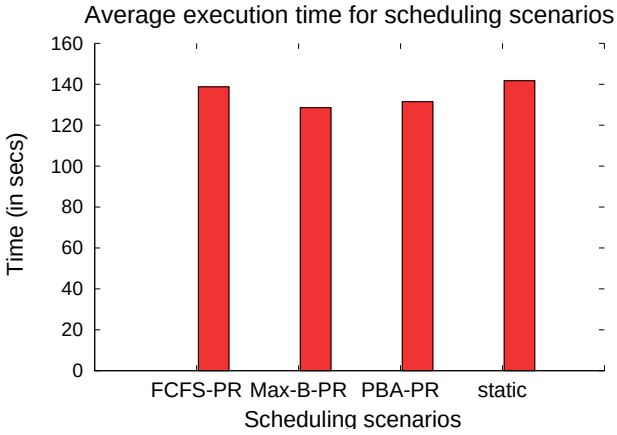

Fig. 4. Average execution time for various scheduling scenarios.

the average execution time for jobs scheduled using static scheduling and with ReSHAPE. From the figure, we see that Max-B-PR improves average execution time by 9.2% compared to static scheduling. This is because it favors expansion of running applications over queued applications. PBA-PR improves the average execution time by 7.2%. Jobs scheduled using FCFS-PR show a small improvement of just 2.08% over static scheduling. This is because the running jobs are always contracted (if they were expanded earlier) to schedule queued jobs. Analogously, for a policy set to improve the overall queue time, FCFS-PR performs better compared to PBA-PR and Max-B-PR scenarios. Figure 5 shows the average queue wait time for different ReSHAPE scenarios. FCFS-PR reduces the average queue wait time for jobs by 20.3% whereas PBA-PR reduces it by 13.13%. Average queue wait time is reduced by 8.6% with Max-B-PR. The average queue wait time depends on the order in which the large jobs arrive in the trace. Job arrival order in the workload results in large variations in the queue wait times across multiple runs, resulting in high

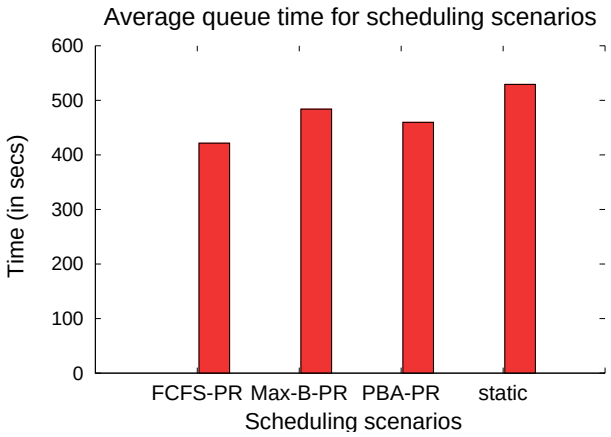

Fig. 5. Average queue time for various scheduling scenarios.

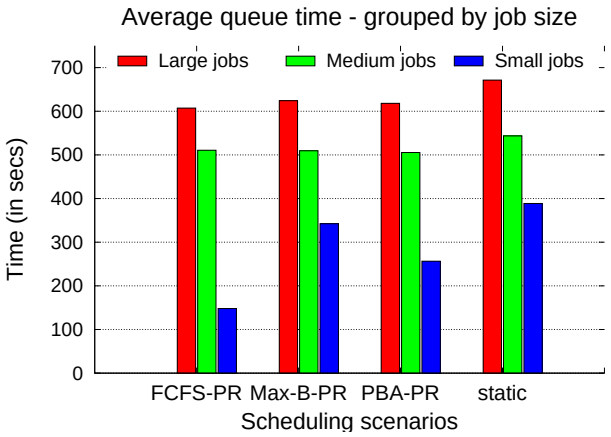

Fig. 7. Average queue wait time by job size for various scheduling scenarios.

standard deviation.

If a policy is set to improve the overall utilization of the system, then all three ReSHAPE scenarios significantly improve the utilization compared to static scheduling. Figure 6

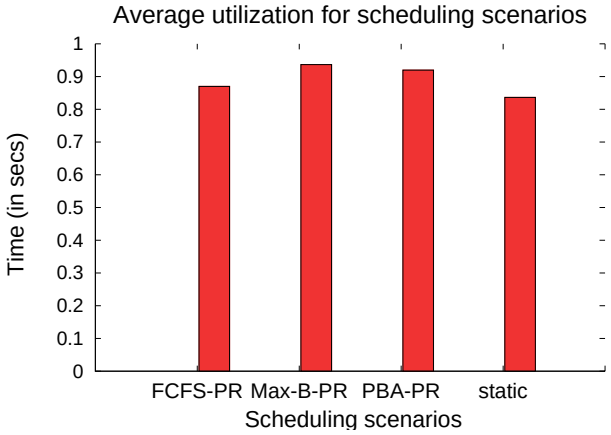

Fig. 6. Average system utilization for various scheduling scenarios.

compares the overall system utilization between ReSHAPE scheduling scenarios and static scheduling. Max-B-PR has the highest average system utilization of 93.6% compared to 83.6% for static scheduling. PBA-PR and FCFS-PR improve the system utilization to 92% and 87% respectively.

If a scheduling policy dictates favoring of small jobs in the system, then FCFS-PR emerges as the best scenario for improving the queue time for small jobs and Max-B-PR as the best scenario for improving their execution time. Figure 7 and Figure 8 show the average queue wait time and execution time for large, medium, and small sized jobs for various ReSHAPE scheduling scenarios. FCFS-PR has the biggest impact in reducing the queue wait time for small jobs. It reduces the average queue wait time for small jobs by 61.9% compared to static scheduling. The reason for large improvement in queue time is because all previously expanded running jobs will be contracted to one of their

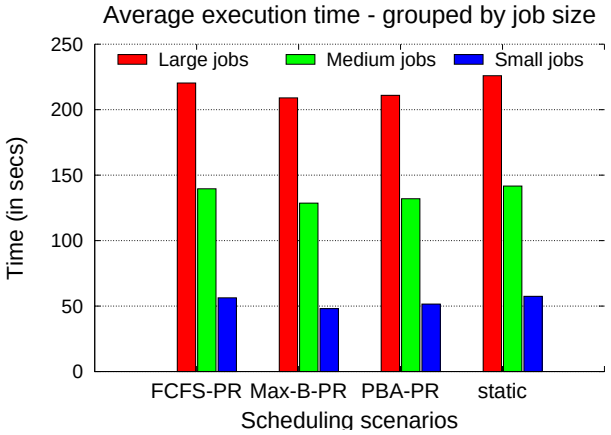

Fig. 8. Average execution time by job size for various scheduling scenarios.

previous resize points in the order they arrive at their resize point to schedule queued jobs. As a result contracting medium and large jobs releases idle processors giving more room to the scheduler to backfill small jobs. PBA-PR and Max-B-PR reduce the average queue wait time for small jobs by 34.01% and 11.9% respectively. For improving the average execution time, Max-B-PR shows a maximum improvement of 16.45% for over static. This is because small jobs have relatively smaller processor requirements for expansion and arrive at their resize points more frequently compared to medium and large jobs. As a result they are able to expand more often thereby improving their execution time. PBA-PR and FCFS-PR improve the performance of small jobs by 10.4% and 1.9% respectively.

Even in the case of large jobs, FCFS-PR and Max-B-PR perform better than other scenarios for improving an application's queue wait and execution time. FCFS-PR, PBA-PR and Max-B-PR reduce the average queue wait time for large jobs by 9.5%, 7.9% and 7% respectively. For medium size jobs, the scheduling scenarios show almost identical improvements in queue wait time, which is 6.0%, 6.3% and 7.0%. Similarly, Max-B-PR improves the execution time for

large jobs by 7.5% compared to 6.7% improvement with PBA-PR and 2.5% improvement with FCFS-PR. For medium size jobs, Max-B-PR improves execution time by 9.2% compare to 6.8% improvement with PBA-PR and 2% improvement with FCFS-PR.

Table 1 summarizes our results for job completion time and job execution time. Since Max-B-PR favors running applications, it does best in terms of reducing execution time and maximizing utilization, but at the expense of higher average queue times and hence higher total job completion times. FCFS-PR on the other hand focuses mostly on getting queued jobs scheduled, which results in a significant improvement in queue waiting time, but not much gain in execution time or utilization. PBA-PR is a compromise, in some sense, as it improves execution time for individual jobs and utilization to almost the same extent that Max-B-PR does, but it also achieves greater than 10% improvement in job completion time.

Another important point is that job arrival order and inter-arrival times can influence queue wait times substantially. Table 2 below shows that ReSHAPE scheduling policies reduce this variability substantially. While this variability due to different job orders is substantial, it is important to note that the improvements due to ReSHAPE scheduling are substantial when comparing one job mix with static scheduling to that same job mix with one of the ReSHAPE policies.

TABLE 3
Summary of job completion time for individual runs. Time in seconds

Case	Static	FCFS-PR	Max-B-PR	PBA-PR
1	783.2	657.8	731.0	675.4
2	475.0	423.9	500.1	469.2
3	752.2	600.8	710.3	632.0
4	604.7	529.8	571.2	527.4
5	774.1	631.0	660.4	645.4
6	381.8	375.2	370.5	356.0
7	762.4	608.6	697.7	721.0
Mean	647.6	546.7	605.9	575.2
St.Dev.	163.7	108.8	132.8	129.8

Table 3 shows the full data set for job completion time, for all seven job mixes tested. We see that the average job completion time for both FCFS-PR and PBA-PR (546.7 and 575.2 seconds, respectively) is significantly smaller compared to static and is nearly five times the standard deviation for those cases (108.8 and 129.8, respectively).

4.3 Workloads with varying percentages of resizability

The goal of this experiment is to demonstrate the potential of ReSHAPE scheduling scenarios even when the percentage of resizable jobs in the workload is less than 100%. In this experiment, we selected PBA-PR because it performs better than FCFS-PR and Max-B with respect to both system utilization and overall completion time. We compared the performance of PBA-PR with static scheduling for the cases when 25%, 50% and 75% of the workloads are resizable. The results are averaged across five individual runs. Job order varied from run to run, but for a given job order we randomly select the desired percentage of jobs that would be resizable. The percentage of resizable jobs was enforced withing job types as well, e.g., for the 50% resizable case, we ensured that 50% of the "(large, nearly-square)" jobs were resizable, 50% of the "(small, power-of-2)" jobs were resizable, etc. All workload jobs have the same user priority.

ReSHAPE provides substantial improvements in overall completion and average utilization even for workloads that have small percentage of resizable jobs. Figure 9 shows average completion time for workloads with varying percentages of resizable jobs. PBA-PR improves the average job completion

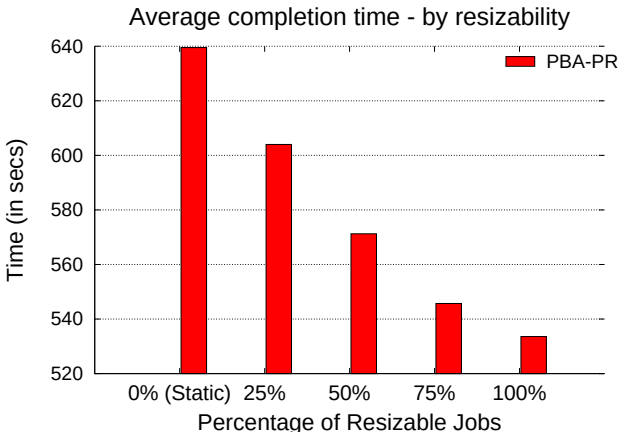

Fig. 9. Average completion time for workloads with varying percentage of resizable jobs.

time by about 5.5% for a workload with only 25% resizable jobs. The improvement increases as the percentage of resizable jobs increase in the workload. For workloads with 50%, 75% and 100% resizability, PBA-PR improves average completion time by 10.6%, 14.6%, and 16.56% respectively. As we have seen, variations in job arrival order can result in large variations in average completion time.

Figure 10 shows overall system utilization for different workloads. The improvement in utilization is the least with a workload that has 25% resizable jobs and increases as the percentage of resizable jobs increase in the workload. PBA-PR improves the system utilization by 2.3%, 4%, 6.5% and 8.4% for workloads with 25%, 50%, 75% and 100% resizable jobs respectively. The absolute values of system utilization for different workloads scheduled using PBA-PR are 85.4%, 87.1%, 89.5% and 91.4%. The system utilization using static scheduling is 83%.

Figure 11 shows the improvement in average completion grouped by job size. With ReSHAPE, the small jobs benefit the most with a maximum improvement of 37% for a 100% resizable workload. The performance improvement for small jobs is much greater than the average improvement in completion for all the jobs. The performance improvements for large, medium and small jobs increase as the percentage of

TABLE 1
Summary of job completion time and job execution time for various scenarios.

Scenarios	Cluster utilization	Job completion time		Execution time	
		Average	% Improv.	Average	% Improv.
Static	83.6	647.6	—	141.7	—
FCFS-PR	87.0	546.7	15.6	138.8	2.1
Max-B-PR	93.7	605.9	6.4	128.6	9.3
PBA-PR	92.0	572.2	11.2	131.5	7.2

TABLE 2
Queue wait time for various scenarios.

Scenario	Average	Std. Dev	Max	Min	Range
Static	529.3	177.7	675.27	240.1	435.2
FCFS-PR	421.5	114.0	548.17	235.0	313.2
Max-B-PR	483.9	133.0	600.26	243.4	356.9
PBA-PR	459.8	133.0	603.94	233.2	370.8

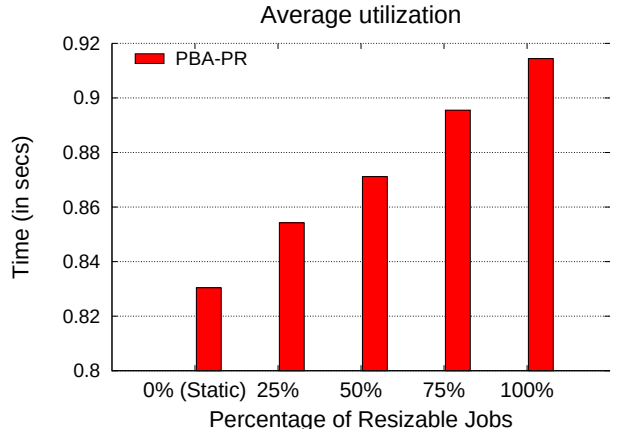

Fig. 10. Average system utilization for workload with varying percentages of resizable jobs.

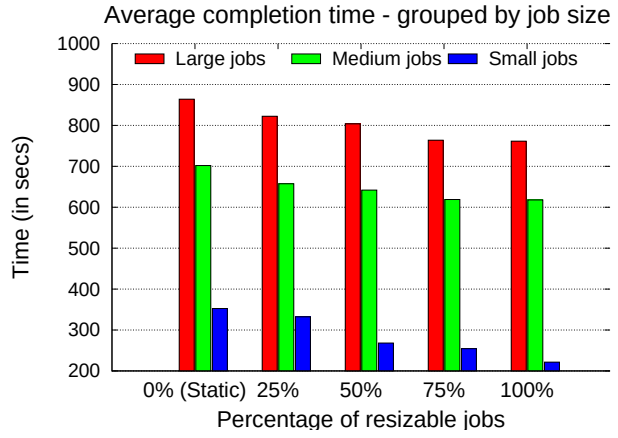

Fig. 11. Average completion time for small, medium and large jobs.

resizable jobs increase in the workload. This is because with more number of resizable jobs the workload is more flexible to expand and contract, giving more opportunities to the schedule queued jobs at an earlier time. as the number of resizable jobs increase, more small jobs are backfilled thereby reducing their queue wait time. The improvement in average completion time for large and medium jobs does not vary significantly when the resizability in the workload is varied from 25% to 100%. In addition to this, the large and medium jobs are rarely allowed to expand, since PBA-PR favors scheduling of queued application over resizing of running applications. As a result, increasing the percentage of resizable jobs from 25% to 100% yields only a 5.2% improvement in average completion time for large jobs. The corresponding improvement for medium jobs is 3.7%. An interesting point to note here is that even with 50% resizable workload, the ReSHAPE scenarios can provide close to 2/3 of the maximum improvement that you would get with a 100% resizable workload.

We know that ReSHAPE scenarios can reduce the sensitivity of job arrival order and inter-arrival times on queue wait times for a 100% resizable workload. Table 4 shows that even with a smaller percentage of resizable jobs in the workload, ReSHAPE scenarios can reduce the influence of these factors on the overall jobs queue wait time.

4.4 Workloads with user assigned priority

In this experiment we aim to show that ReSHAPE can provide quality of service to jobs based on their priority. A high priority job will be serviced better with ReSHAPE compared to static scheduling. Also low priority jobs benefit with ReSHAPE in that they can experience better individual turn-around time. The jobs in the workloads used in this experiment are assigned either gold or platinum user priority. ReSHAPE gives platinum user jobs a higher priority compared to gold user jobs. We compare the performance of PBA-PR and static scheduling scenarios. The results are averaged over five individual runs. To reduce variability in the results due to job order, we select the workload with 50% resizable jobs and reuse the five workloads from the previous experiment. The rationale behind selecting a workload with 50% resizable

TABLE 4
Queue wait time for various percentages of resizable workload.

Resizable workload	Average	Std. Dev	Max	Min	Range
Static	521.06	81.5	601.93	410.29	191.64
25%	478.43	103.6	570.53	320.33	250.2
50%	451.7	82.88	551.39	325.32	226.07
75%	421.71	74.06	513.45	313.04	200.41
100%	412.53	77.83	479.83	300.7	179.13

jobs is that ReSHAPE provides substantial improvements in overall queue wait times and completion times even when only half of the jobs are resizable. We randomly assign platinum (high) priority to 50% of the jobs and gold (low) priority to the remaining 50%. Again, we make sure that the corresponding percentage of each job type (size and topology) is given platinum and gold priority.

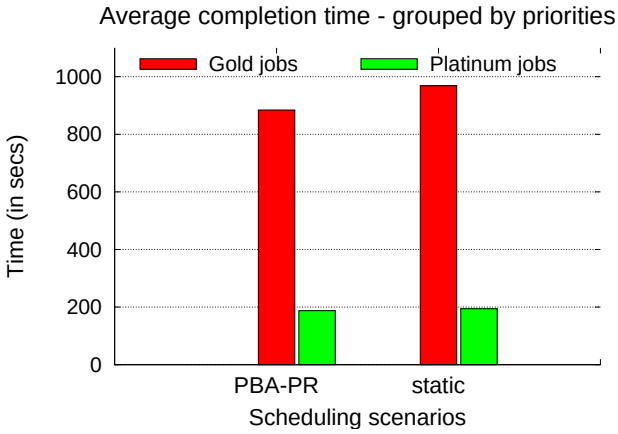

Fig. 12. Average completion time by priority.

With priority-based scheduling, ReSHAPE benefits not only the high priority jobs but also the low priority jobs. Figure 12 shows improvement in average completion time grouped by job priority. From the graph we see that although static scheduling provides a good service to high priority jobs, ReSHAPE still improves it by a small percentage. In addition to this, even the low priority (gold) jobs benefit significantly from ReSHAPE. PBA-PR improves the overall average job completion time for gold jobs by 8% compared to static. The platinum jobs benefit by 3.5% in their average completion time. The high improvement for low priority jobs with ReSHAPE is due to the reason that when a high priority job cannot be scheduled, a low priority jobs are allowed to backfill thereby improving their queue wait time. The small benefit for high priority jobs is due to the reason that high priority running jobs are expanded before any low priority queued or running jobs and contracted after all the low priority jobs have been contracted to their starting processor set size.

Within each group of high and low priority jobs, ReSHAPE significantly improves the average queue wait time for low priority small jobs. This is because low priority small jobs are backfilled when a high priority job at the top of the queue

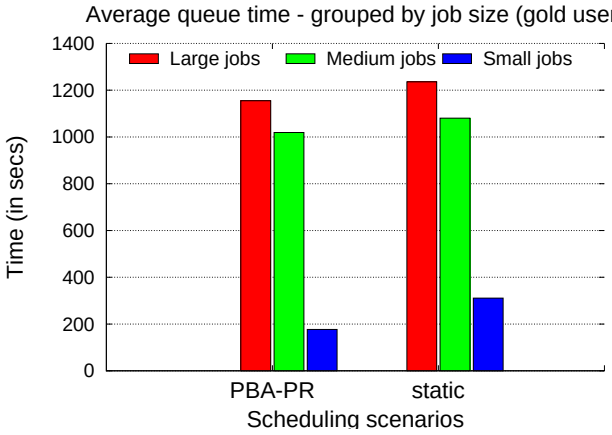

Fig. 13. Average queue wait time for large, medium and small sized jobs for low priority (gold) users.

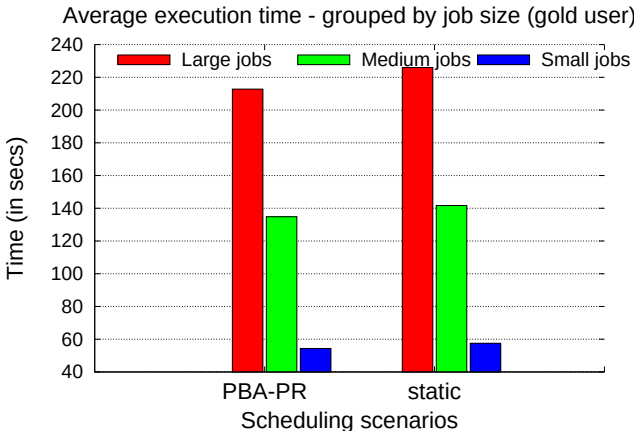

Fig. 14. Average execution wait time for large, medium and small sized jobs for low priority (gold) users.

cannot be scheduled. The low priority jobs are backfilled after backfilling other high priority jobs. ReSHAPE also improves the average execution time for low priority jobs by expanding them at their resize points when no other high priority jobs can scheduled or expanded. Figure 13 and Figure 14 show the average queue wait time and execution time for low priority jobs grouped by job size. The small jobs show an improvement of 43.1% in average queue wait time over static. The medium and large jobs show an improvement of 5.6% and 6.5% over static. Similarly, the average execution time improved by 6%,

TABLE 5
Average completion time for jobs with and without priority (50% resizable jobs, 50% high priority jobs)

Scenarios	Overall ave. completion time	Gold jobs	Platinum jobs
Static w/o priority	639.5	645.7	633.2
Static w/ priority	581.4	968.6	194.2
PBA-PR w/o priority	571.2	565.1	577.3
PBA-PR w priority	535.8	884.2	187.5

5% and 5.5% for large, medium and small jobs respectively.

Table 5 summarizes the improvement in average completion time for jobs with and without priority using PBA-PR and static scheduling scenarios. The overall average completion time for gold and platinum jobs for the static scheduling without priority scenario is computed by selecting the exact same set of jobs from the workload which are assigned priority for the static scheduling with priority scenario. A similar procedure is followed for computing average completion time for gold and platinum jobs for PBA-PR without priority scenario. From the table, we see that not only do the platinum jobs benefit due to resizing, but the low priority jobs also improve their completion time by 8.7% compared to static. The platinum jobs improve their average completion time by 3.5%. When a static job with no priority is made resizable and assigned platinum priority, then PBA-PR improves its average completion time by 70.4% compared to 69.3% by static. Similarly, when a static job with no priority is ReSHAPE-enabled and assigned gold priority, then PBA-PR with priority scenario improves the average completion time by 13% more than static scheduling.

5 Conclusions and Future Work

In this paper we explore the potential of dynamically resizable applications for scheduling on large parallel clusters. We introduce new policies and strategies for scheduling resizable applications using the ReSHAPE framework. A scheduling policy is viewed as an abstract high-level objective that the scheduler strives to achieve. The scheduling scenarios provide a more concrete and implementable representation of the policy. Scheduling scenarios, in turn, are implemented using scheduling strategies which are methods responsible for actuating the resizing decisions. The scheduling policies discussed in this paper improve overall cluster utilization as well an individual application turn around time. The policies were evaluated using the five scenarios.

The current implementation of the ReSHAPE framework uses a common performance model for all resizable applications. The scheduling strategies use this predicted performance value in their resizing decisions for an application. More sophisticated prediction models and policies are certainly possible. Indeed, a primary motivation for ReSHAPE is to serve as a platform for research into more sophisticated resizing and scheduling strategies. Experimental results show that the proposed scheduling policies and scenarios outperform conventional scheduling policies with respect to average execution time, average job completion time and overall system utilization. Results also show that a job mix which includes only 25%resizable job can still realize good performance improvements with ReSHAPE. ReSHAPE also supports scheduling of priority-based jobs. Instead of preempting low priority jobs, ReSHAPE contracts them to a smaller processor allocation to schedule high priority jobs.

We are currently working on including more sophisticated scheduling policies in the ReSHAPE framework. Some of these policies include improving application turn-around time and system utilization using an adaptive partitioning strategy, and supporting advanced reservation services. We also plan to add more accurate and sophisticated performance models to better predict application performance. We plan to make ReSHAPE more extensible so that job-specific performance models and new scheduling policies or strategies can be added to the framework.

References

[1] A. Mu'alem and D. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Transactions on Parallel and Distributed Systems*, pp. 529–543, 2001.

[2] D. Feitelson and M. Jette, "Improved Utilization and Responsiveness with Gang Scheduling," *Lecture Notes in Computer Science*, vol. 1291, pp. 238–261, 1997.

[3] D. Feitelson, "Packing Schemes for Gang Scheduling," *Lecture Notes in Computer Science*, vol. 1162, pp. 89–111, 1996.

[4] R. Sudarsan and C. Ribbens, "ReSHAPE: A Framework for Dynamic Resizing and Scheduling of Homogeneous Applications in a Parallel Environment," in *Proceedings of the 2007 International Conference on Parallel Processing (ICPP)*, XiAn, China, September 10-14, 2007, p. 44.

[5] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *IPPS '97: Proceedings of the Job Scheduling Strategies for Parallel Processing*. London, UK: Springer-Verlag, 1997, pp. 1–34.

[6] D. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel job scheduling–a status report," *Lecture Notes in Computer Science*, vol. 3277, pp. 1–16, 2005.

[7] J. B. Weissman, L. Rao, and D. England, "Integrated Scheduling: The Best of Both Worlds," *Journal of Parallel and Distributed Computing*, vol. 63, no. 6, pp. 649–668, 2003.

[8] J. B. Weissman, "Prophet: Automated scheduling of SPMD programs in workstation networks," *Concurrency: Practice and Experience*, vol. 11, no. 6, pp. 301–321, 1999.

[9] W. Cirne and F. Berman, "Adaptive Selection of Partition Size for Supercomputer Requests," *Lecture Notes in Computer Science*, pp. 187–208, 2000.

[10] R. Sudarsan and C. Ribbens, "Efficient Multidimensional Data Redistribution for Resizable Parallel Computations," in *Proceedings of the International Symposium of Parallel and Distributed Processing and Applications (ISPA '07)*, Niagara falls, ON, Canada, August 29-31, 2007, pp. 182–194.

[11] "Moab scheduler," URL: http://www.clusterresources.com/products-/moab-cluster-suite.php.

[12] URL: http://www.pbsgridworks.com.

[13] "Maui cluster scheduler," URL: http://www.clusterresources.com/products-/maui-cluster-scheduler.php.

[14] R. Sudarsan and C. J. Ribbens, "Scheduling resizable parallel applications," *Parallel and Distributed Processing Symposium, International*, vol. 0, pp. 1–10, 2009.

[15] J. Skovira, W. Chan, H. Zhou, and D. Lifka, "The easy-loadleveler api project," *Lecture Notes in Computer Science*, pp. 41–47, 1996.

[16] "Open MPI v1.3.3," 2008, URL: http://www.open-mpi.org.

[17] R. Graham, T. Woodall, and J. Squyres, "Open MPI: A Flexible High Performance MPI," *Lecture Notes in Computer Science*, vol. 3911, p. 228, 2006.

[18] M. Calzarossa and G. Serazzi, "A characterization of the variation in time of workload arrival patterns," *IEEE Transactions on Computers*, vol. 100, no. 34, pp. 156–162, 1985.

[19] W. Leland and T. Ott, "Load-balancing heuristics and process behavior," *ACM SIGMETRICS Performance Evaluation Review*, vol. 14, no. 1, pp. 54–69, 1986.

[20] K. C. Sevcik, "Application scheduling and processor allocation in multiprogrammed parallel processing systems," *Performance Evaluation*, vol. 19, no. 2-3, pp. 107–140, Mar 1994.

[21] A. B. Downey, "A parallel workload model and its implications for processor allocation," *Cluster Computing*, vol. 1, no. 1, pp. 133–145, 1998.

[22] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riodan, "Modeling of workload in MPPs," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, Eds. Springer Verlag, 1997, pp. 95–116, lect. Notes Comput. Sci. vol. 1291.

[23] U. Lublin and D. G. Feitelson, "The workload on parallel supercomputers: Modeling the characteristics of rigid jobs," *J. Parallel & Distributed Comput.*, vol. 63, no. 11, pp. 1105–1122, Nov 2003.

[24] W. Cirne and F. Berman, "A Model for Moldable Supercomputer Jobs," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, April 2001.

[25] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Modeling user runtime estimates," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, Eds. Springer Verlag, 2005, pp. 1–35, lect. Notes Comput. Sci. vol. 3834.

[26] "A model/utility to generate user runtime estimates and append them to a standard workload file," 2005, URL: http://www.cs.huji.ac.il/labs/parallel/workload/m_tsafrir05.