Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment*

Gerald Sabin, Rajkumar Kettimuthu, Arun Rajan and P Sadayappan
The Ohio State University
Columbus OH 43201
{sabin, kettimut, rajan, saday}@cis.ohio-state.edu

Abstract

Most previous research on job scheduling for heterogeneous systems considers a scenario where each job or task is mapped to a single processor. On the other hand, research on parallel job scheduling has concentrated primarily on the homogeneous context. In this paper, we address the scheduling of parallel jobs in a heterogeneous multi-site environment, where each site has a homogeneous cluster of processors, but processors at different sites have different speeds. Starting with a simple greedy scheduling strategy, we propose and evaluate several enhancements using trace driven simulations. We consider the use of multiple simultaneous reservations at different sites, use of relative job efficacy as a queuing priority, and compare the use of conservative versus aggressive backfilling. Unlike the single-site case, conservative backfilling is found to be consistently superior to aggressive backfilling for the heterogeneous multi-site environment.

1. Introduction

Considerable research has been conducted over the last decade on the topic of job scheduling for parallel systems, such as those used for batch processing at Supercomputer Centers. Much of this research has been presented at the annual "Workshops on Job Scheduling Strategies for Parallel Processing [39]." With significant recent developments in creating the infrastructure for grid computing, the transparent sharing of resources at multiple geographically distributed sites is being facilitated. An important aspect of significance to multi-site job scheduling is that of heterogeneity – different sites are generally unlikely to have identical configurations of their processors and can be expected to have different performance characteristics. Much of the research to date on job scheduling for heterogeneous systems has only addressed the scheduling of independent sequential jobs or precedence constrained task graphs where each task is sequential [2][20][35]. A direct extension of the heterogeneous scheduling strategies for sequential jobs and coarse-grained task-graphs is not attractive for scheduling thousands of processors, across multiple sites, due to the explosion in computational complexity. Instead we seek to extend the practically effective back-filling based parallel job scheduling strategies [11] used in practice for single-site scheduling. In this paper, we address the problem of heterogeneous multi-site job scheduling, with a homogeneous cluster of processors at each site.

* Supported in part by NSF grant EIA-9986052

The paper is organized as follows. In Section 2, we provide some background information about parallel job scheduling and heterogeneous job scheduling. Section 3 provides information about the simulation environment used for this study. In Section 4, we begin by considering a simple greedy scheduling strategy for scheduling parallel jobs in a multi-site environment, where each site has a homogeneous cluster of processors, but processors at different sites have different speeds. We progressively improve on the simple greedy scheme, starting with the use of multiple simultaneous reservations at different sites. In Section 5, we compare the use of conservative versus aggressive backfilling in the heterogeneous context, and show that the trends are very different from the single-site case. In Section 6, we evaluate a scheduling scheme that uses the relative performance of jobs at different sites as the queue priority criterion for back-filling. In Section 7, we evaluate the implications of restricting the number of sites used for simultaneous reservations. Section 8 discusses related work. We conclude in Section 9.

2. Background

The problem we address in this paper is the following: Given a number of heterogeneous sites, with a homogeneous cluster of processors at each site, and a stream of parallel jobs submitted to a metascheduler, find an effective schedule for the jobs so that the average turnaround time of jobs is optimized. There has been a considerable body of work that has addressed the parallel job scheduling problem in the homogeneous context [5][30][32][26][12]. There has also been work on heterogeneous job scheduling [2][20][34][35], but this has generally been restricted to the case of sequential jobs or coarse-grained precedence constrained task graphs. The fundamental approach used for scheduling in these two contexts has been very different. We provide a very brief overview.

The Min-Min algorithm is representative of the scheduling approaches proposed for scheduling tasks on heterogeneous systems. A set of N tasks is given, with their runtimes on each of a set of P processors. Given a partial schedule of already scheduled jobs, for each unscheduled task, the earliest possible completion time is determined by considering each of the P processors. After the minimum possible completion time for each task is determined, the task that has the lowest "earliest completion time" is identified and is scheduled on the processor that provides its earliest completion time. This process is repeated N times, till all N tasks are scheduled. The problem has primarily been evaluated in a static "off-line" context - where all tasks are known before scheduling begins, and the objective is the minimization of makespan, i.e. the time to finish all tasks. The algorithms can be applied also in the dynamic "on-line" context, by "unscheduling" all non-started jobs at each scheduling event - when either a new job arrives or a job completes.

Scheduling of parallel jobs has been addressed in the homogeneous context. It 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 at a single site is to use a First-Come-First-Served (FCFS) policy. This approach suffers from low system utilization [22]. Backfilling was proposed to improve the system utilization and has been implemented in several production schedulers. Backfilling works by identifying "holes" in the 2D chart and moving forward smaller jobs that fit those holes, without delaying any jobs with future reservations.

There are two common variations to backfilling - conservative and aggressive (EASY)[12][26]. 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.

Thus, prior work on job scheduling algorithms for heterogeneous systems has primarily focused on independent sequential jobs or collections of single-processor tasks with precedence constraints. On the other hand, schemes for parallel job scheduling have not considered heterogeneity of the target systems. Extensions of algorithms like Min-Min are possible, but their computational complexity will be explosively high for realistic systems. Instead, we pursue an extension to an approach that we previously proposed for distributed multi-site scheduling on homogeneous systems [31]. The basic idea is to submit each job to multiple sites, and cancel redundant submissions when one of the sites is able to start the job.

3. Simulation Environment

In this work we employ simulations with a locally developed job-scheduler/simulator, using workload logs from supercomputer centers. The job logs were obtained from the collection of workload logs available form Dror Feitelson's archive [10]. Results for a 5000 job subset of the 430 node Cornell Theory Center (CTC) trace and a 5000 job subset of a trace from the 128 node IBM SP2 system at the San Diego Supercomputer Center (SDSC) are reported. The first 5000 jobs were selected, representing roughly a one month set of jobs. These traces were modified to vary load and to model jobs submitted to a metascheduler from geographically distributed users (by time-shifting two of the traces by three hours, to model two centers each in the Pacific and Eastern U.S. time zones) .

The available job traces do not provide any information about runtimes on multiple heterogeneous systems. To model the workload characteristics of a heterogeneous environment, the NAS Parallel Benchmarks 2.0 [37] were used. Four Class B benchmarks were used to model the execution of jobs from the CTC and SDSC trace logs on a heterogeneous system. Each job was randomly chosen to represent one of the NAS benchmarks. The processing power of each remote site was modeled after one of four parallel computers for which NAS benchmark data was available (cluster 0:SGI Origin 2000, cluster 1:IBM SP(WN/66), cluster 2:Cray T3E 900, cluster 3:IBM SP (P2SC 160 MHz). The run times of the various machines were normalized with respect to IBM SP (P2SC 160 MHz) for each benchmark. The jobs were scaled to represent their relative runtime (for the same number of nodes) on each cluster. These scaled runtimes represent the expected runtime of a job on a particular cluster, assuming the estimate from the original trace corresponded to an estimate on the IBM SP (P2SC 160 MHz). The number of total processors at each remote site was chosen to be the same as the original traces submitted to that node (430 when simulating a trace from CTC and 128 for SDSC). Therefore, in these simulations all jobs can run at any of the simulated sites. In this paper, we do not consider the scheduling of a single job across multiple sites.

The benchmarks used were: LU (an application benchmark, solving a finite difference discretization of the 3-D compressible Navier - Stokes equations[1]), MG (Multi-Grid, a kernel benchmark, implementing a V-cycle multi-grid algorithm to solve the scalar discrete Poisson equation [25]), CG (Conjugate Gradient, that computes an

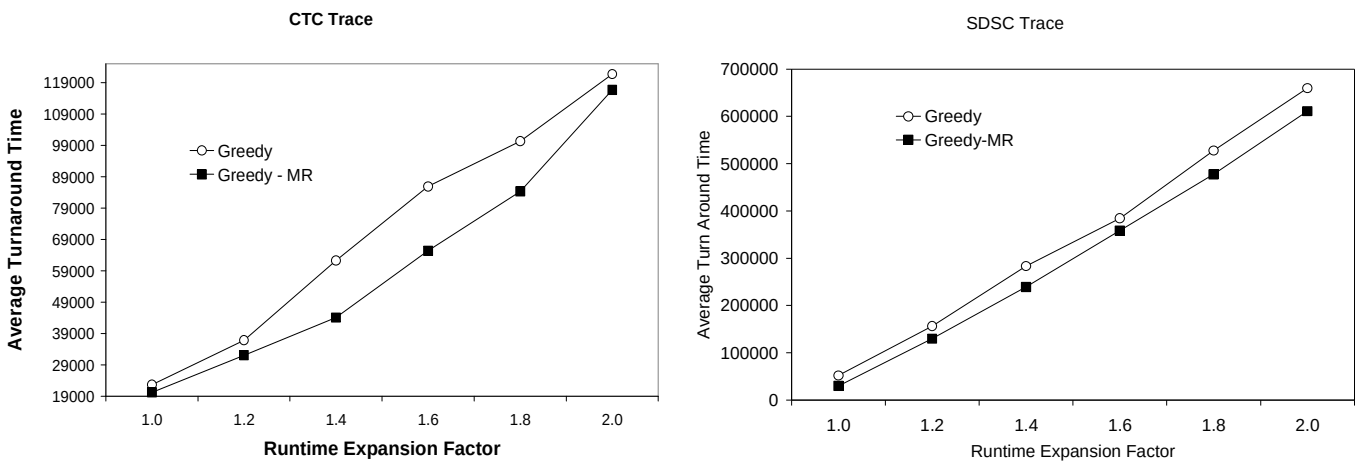

Figure 1: Greedy Metascheduling in a Heterogeneous Environment: Turnaround Time Making multiple redundant requests only improves the turnaround time slightly.

approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix, and IS (Integer Sort, that tests a sorting operation that is important in "particle method" codes).

The performance of scheduling strategies at different loads was simulated by multiplying the runtimes of all jobs by a load factor. The runtime of jobs were expanded to leave the duration of the simulated trace (roughly one month) unchanged. This will generate a schedule equivalent to a trace where the inter-arrival time of the jobs are reduced by a constant factor. However, this model of increasing load results in a linear increase in turnaround time, even if the wait times remain unchanged. However, the increase in wait time (due to the higher load), may cause the turnaround time to increase at a faster rate (i.e. an exponential increase in wait time will cause an exponential increase in turnaround time). Simulations were run with load factors ranging from 1.0 to 2.0, in increments of 0.2.

In these simulations all jobs are submitted to the metascheduler, however, none of the schemes presented require this. These scheduling strategies do not prevent local jobs from being submitted to the local queues.

4. Greedy Metascheduling

We first consider a simple greedy scheduling scheme, where jobs are processed in arrival order by the meta-scheduler, and each job is assigned to the site with the lowest instantaneous load. The instantaneous load at a site is considered to be the ratio of the total remaining processor-runtime product for all jobs (either queued or running at that site) to the number of processors at the site. It thus represents the total amount of time needed to run all jobs assuming no processor cycles are wasted (i.e. jobs can be ideally packed).

Recently, we had evaluated a scheduling strategy that uses multiple simultaneous requests for the homogeneous multi-site context [31] and showed that it provided significant improvement in turnaround time. We first applied the idea of multiple simultaneous

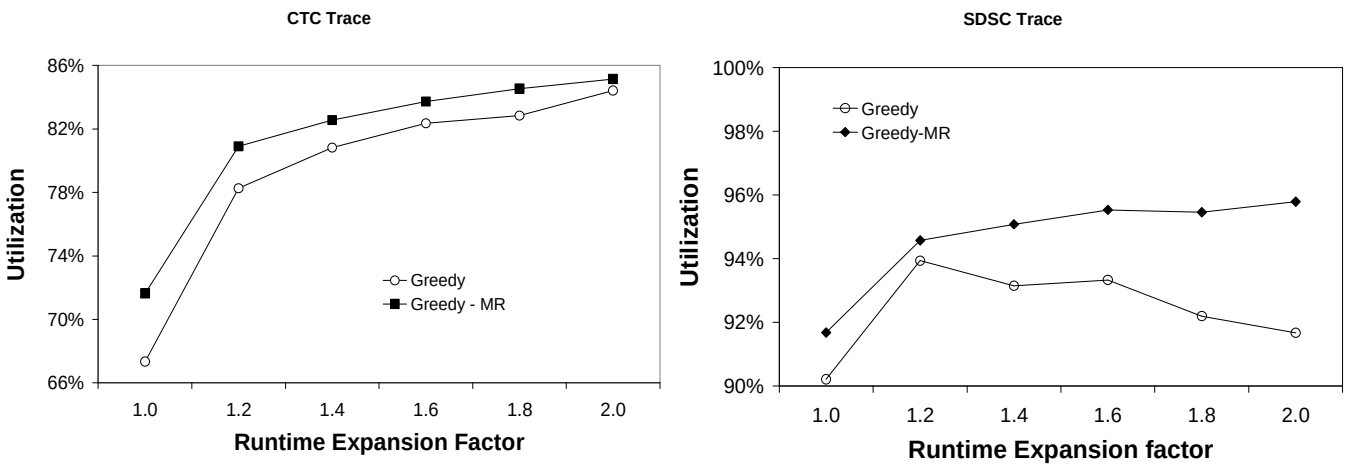

Figure 2: Greedy Metascheduling: Utilization

With respect to utilization, making multiple requests results in a slight improvement over the greedy scheme

requests to the heterogeneous environment and compared its performance with the simple greedy scheme. With the MR (Multiple Requests) scheme, each job is sent to the "K" least loaded sites. Each of these K sites schedules the job locally. The scheduling scheme at the sites is aggressive backfilling, using a FCFS queue priority. When a job is able to start at any of the sites, the site informs the metascheduler, which in turn contacts the K-1 other local schedulers to cancel that redundant request from their respective queues. This operation must be atomic to ensure that the job is only executed at one site. By placing each job in multiple queues, the expectation is that more jobs will be available in all local queues; thereby the jobs will fit into a backfill window. Furthermore, more "holes" will be created in the schedule due to K - 1 reservations being removed when a job starts running, enhancing backfill opportunities for queued jobs.

The MR scheme was simulated with K=4, i.e. each job was submitted to all four sites. It was hoped that these additional backfilling opportunities would lead to improved system utilization and reduced turnaround times. However, as shown in Figure 1, the average turnaround time decreases only very slightly with the MR scheme, when compared to the simple greedy scheme. Figure 2 shows that the average system utilization for the greedy-MR scheme improves only slightly when compared to the simple greedy scheme, and utilization is quite high for both schemes.

In a heterogeneous environment, the same application may perform differently when run on different clusters. Table 1 shows the measured runtime for three applications (NAS benchmarks) on the four parallel systems mentioned earlier, for execution on 8 or 256 nodes. It can be seen that performance differs for each application on the different machines. Further, no machine is the fastest on all applications; the relative performance of different applications on the machines can be very different. With the simple greedy and MR schemes, it is possible that jobs may execute on machines where their performance is not the

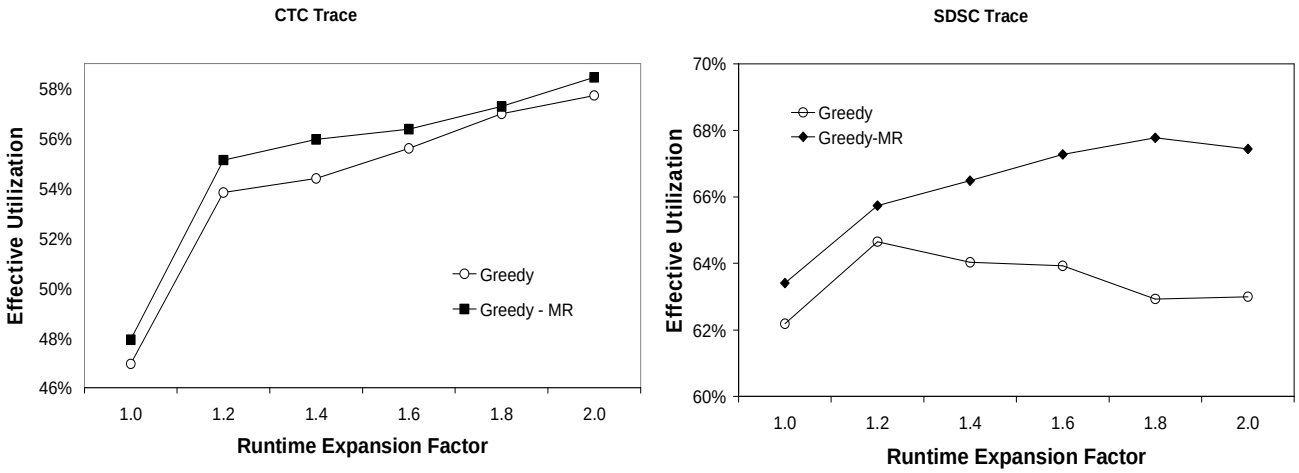

Figure 3: Greedy Metascheduling: Effective Utilization

With respect to effective utilization, making multiple requests only results in a slight improvement over the greedy scheme

TABLE 1 (Heterogeneous Job Runtimes)

	SGI Origin 2000	IBM SP (WN/66)	Cray T3E 900	IBM SP+ (P2SC 160 MHz)
IS Class B (8 Nodes)	23.3	22.6	16.3*	17.7
MG Class B (8 Nodes)	35.5	34.3	25.3	17.2*
MG Class B (256 Nodes)	1.3147	2.2724	1.8	1.1*
LU Class B (256 Nodes)	20.328*	94.893	35.6	24.2

*Best runtime the a job

+Original estimated runtime

best. In order to assess this, we computed an "effective utilization" metric. We first define the efficacy of a job at any site to be the ratio of its best runtime (among all the sites) to its runtime at that site. The effective utilization is a weighted utilization metric, where each job's processor-runtime product is weighted by its efficacy on that site. While utilization is a measure of the fraction of used processor cycles on the system, the effective utilization is a measure of the fraction of used processor cycles with respect to its best possible usage.

$$EffectiveUtilization = \frac{\sum Efficacy_i * Runtime_i * ProcessorsUsed_i}{Makespan * TotalNumberOfProcessors}$$

Where

$$Makespan = MaxCompletetionTime - MinStartTime$$

and

$$Efficacy_i = \frac{OptimalRuntime_i * OptimalProcessorsUsed_i}{ActualRuntime_i * ActualProcessorsUsed_i}$$

5. Aggressive Vs Conservative Scheduling

So far we have used aggressive backfilling, and seen that using multiple requests only improves performance slightly. In this scheme (Greedy-MR) a job runs at the site where it starts the earliest. In a heterogeneous context, the site where the job starts the earliest may not be the best site. The heterogeneity of the sites means that any given job can have different runtimes at the various sites. Thus, the site that gives the earliest start time need not give the earliest completion time. Therefore, it would be beneficial to use completion time when deciding whether a job can start or not. However, in order to be able to estimate the completion of a job at all relevant sites, conservative backfilling has to be employed at each site. When a job is about to start at a site, the scheduler has to check the expected completion time of the same job at all sites where the job is scheduled, and determine if some other site has a better completion time. If the job is found to have a better completion time at another site, the job is not run and is removed from the queue at this site.

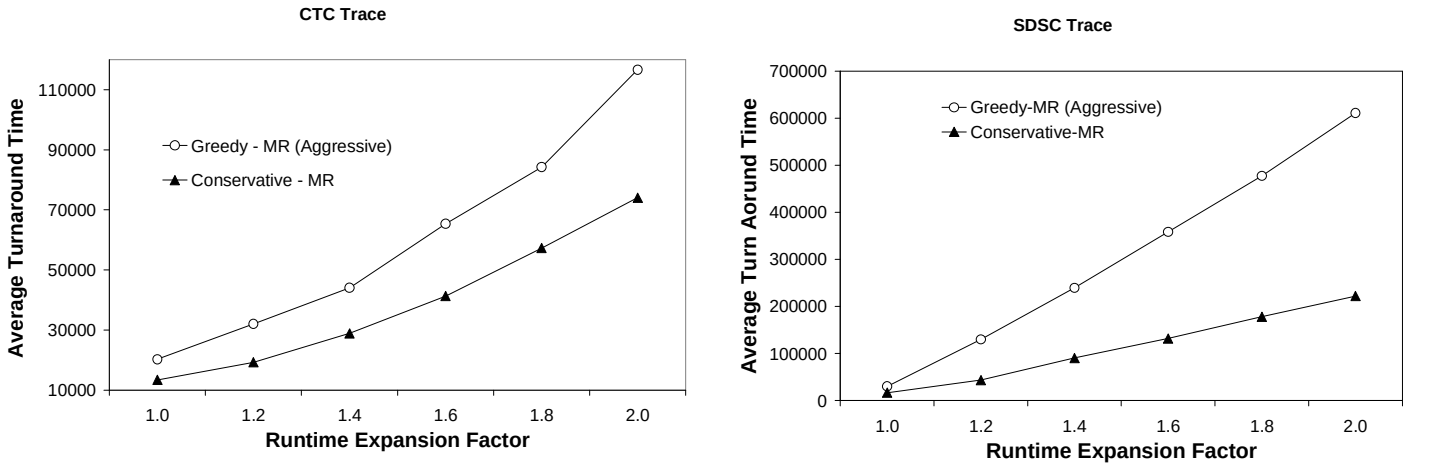

Figure 4: Performance of Conservative Multiple Request Scheme

Using a conservative completion-based scheme has a significant impact on the Greedy –MR scheme.

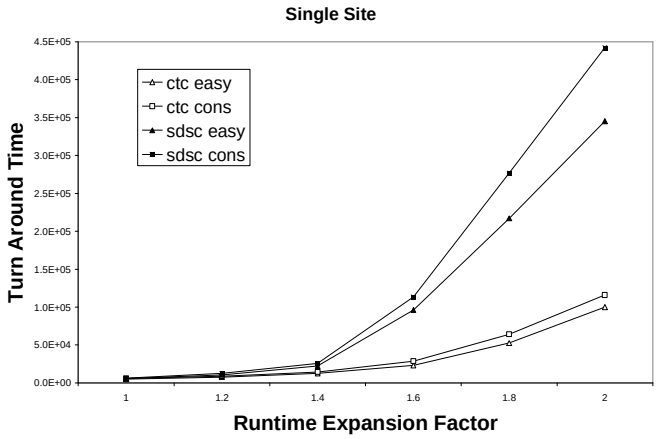

Figure 5: Aggressive vs. Conservative Back-filling: Single Site

For a single site, aggressive backfilling outperforms conservative backfilling.

Figure 4 compares the performance of the completion-based conservative scheme with the previously evaluated start-based aggressive scheme for the CTC and SDSC traces. It can be observed that the conservative scheme performs much better than the aggressive scheme. This is quite the opposite of what generally is observed with single-site scheduling, where aggressive backfilling performs better than conservative backfilling in regards to the turnaround time metric. It has been shown that aggressive backfilling consistently improves the performance of long jobs relative to conservative backfilling [29] and the turnaround time metric is dominated by the long jobs. Indeed, that is what we observe with single site scheduling for these traces (in order to make the overall load comparable to the four-site experiments, the runtime of all jobs was scaled down by a factor of 4). Figures 6 and 7 provide insights into the reason for the superior performance of the completion based conservative scheme for multi-site scheduling. Even though the aggressive scheme has better "raw" utilization, the effective utilization is worse than the conservative scheme. The higher effective utilization with the completion-based conservative scheme suggests that basing the decision on expected job completion time rather than start time improves the chances of a job running on a site where its efficacy is higher, thereby making more effective use of the processor cycles. The figures show similar trends for both CTC and SDSC traces.

The average turnaround time for aggressive backfilling at a single site is better, compared to conservative backfilling, because of improved backfilling chances with aggressive backfilling. The backfilling opportunities in the single-site context are poorer with conservative backfilling because each waiting job has a reservation, and the presence of multiple reservations creates impediments to backfilling. Conservative backfilling has been shown to especially prevent long narrow jobs from backfilling. The improvement in the average turnaround time with conservative backfilling in the heterogeneous context is also attributed to improved backfilling caused by the holes created by the dynamic removal of replicated jobs at each site, and an increased number of jobs to attempt to backfill at each site. Multiple reservation requests causes there to be an increased number of jobs in all local queues at

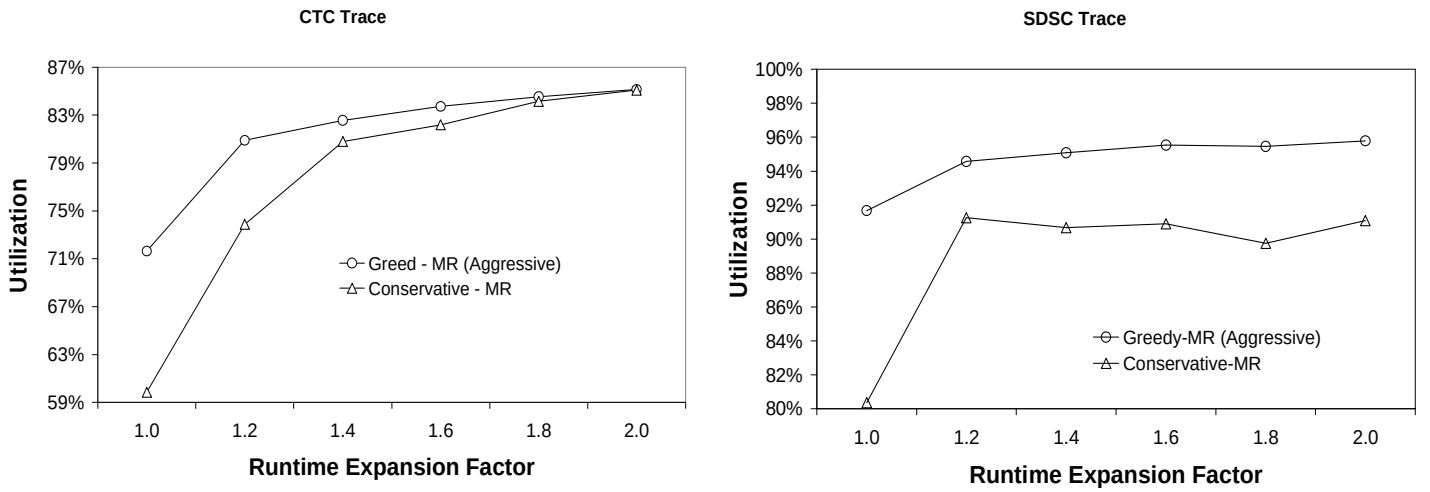

Figure 6: Utilization of Conservative Completion-Based Multiple Requests Scheme

When using the conservative competition-based scheme, the raw utilization actually decreases, even though the turnaround time has improved.

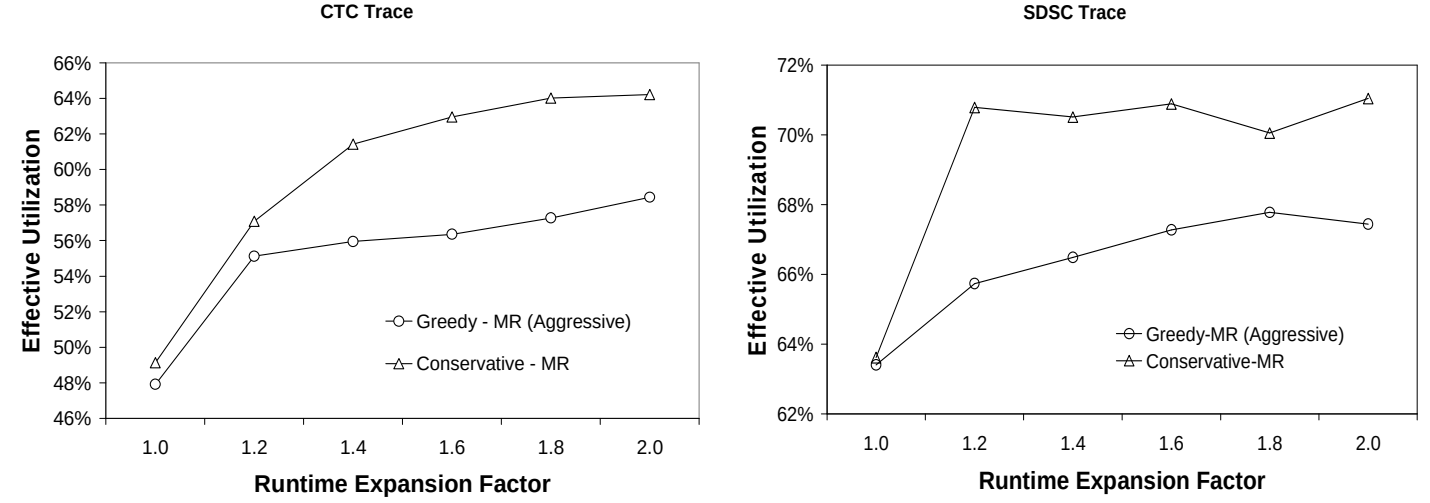

Figure 7: Effective Utilization of Conservative Completion-Based Multiple Requests Scheme

Using a conservative completion-based scheme significantly increases effective utilization, even though raw utilization decreases.

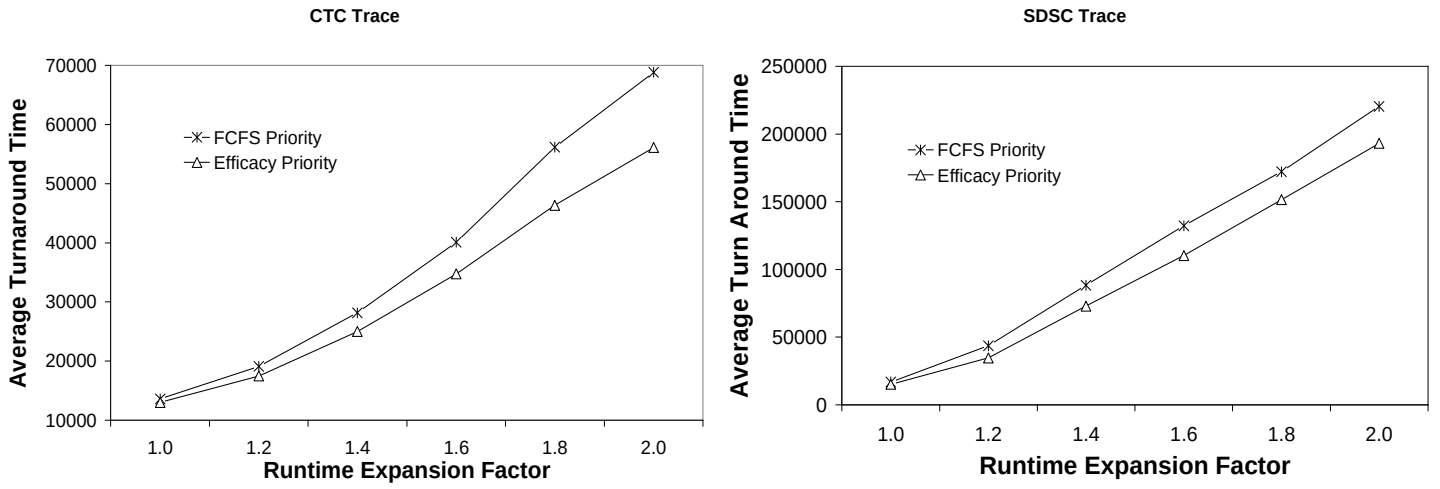

Figure 8: Performance of Efficacy Based Scheduling

Explicitly accounting for efficacy (by using efficacy for the priority policy) reduces turnaround time

any given time. This gives the scheduler more jobs to choose from, when attempting to fill a backfill window. In a heterogeneous environment the presence of reservation replicas brings both backfilling advantages and the advantages due to reservation guarantees.

We then incorporated a refinement to the completion-based multiple-reservation scheduling strategy. In the homogeneous case, when a job is ready to start at one of the sites, all other copies of that job at other sites can be cancelled because none of those could possibly produce an earlier completion time. In the heterogeneous context, when a job is ready to start at a site, and appears to have the earliest completion time when compared to its reservation at other sites, there is still a possibility that future backfilling at a faster site might allow a faster completion at the faster site. In order to take advantage of these possible backfills, it might be worthwhile to keep the jobs in the queues at the faster sites, even though the current remote reservations do not provide a completion time better than at the site where the job is ready to start. We implemented a version of the completion - based conservative backfilling scheme where only jobs at slower sites were cancelled when a job was started at a site. This improved performance, but not to a significant extent.

6. Efficacy Based Scheduling

The previous data has shown that the raw utilization is not a good indicator for how well a scheduling strategy performs in a heterogeneous environment. The turnaround time tracks more closely with the effective utilization. Therefore, a scheme which directly takes into account the efficacy of jobs would be desirable. A strategy which increases the efficacy of the jobs would lead to a higher effective utilization, which we expect will lower the average turn around time.

To include efficacy in our strategies we propose using efficacy as the priority order for the jobs in the queue. Changing the order of the reserved jobs will change the backfilling order. In this case jobs with higher efficacies will attempt to backfill before jobs with lower efficacies, and thus will have more backfilling opportunities. In the case of identical efficacies

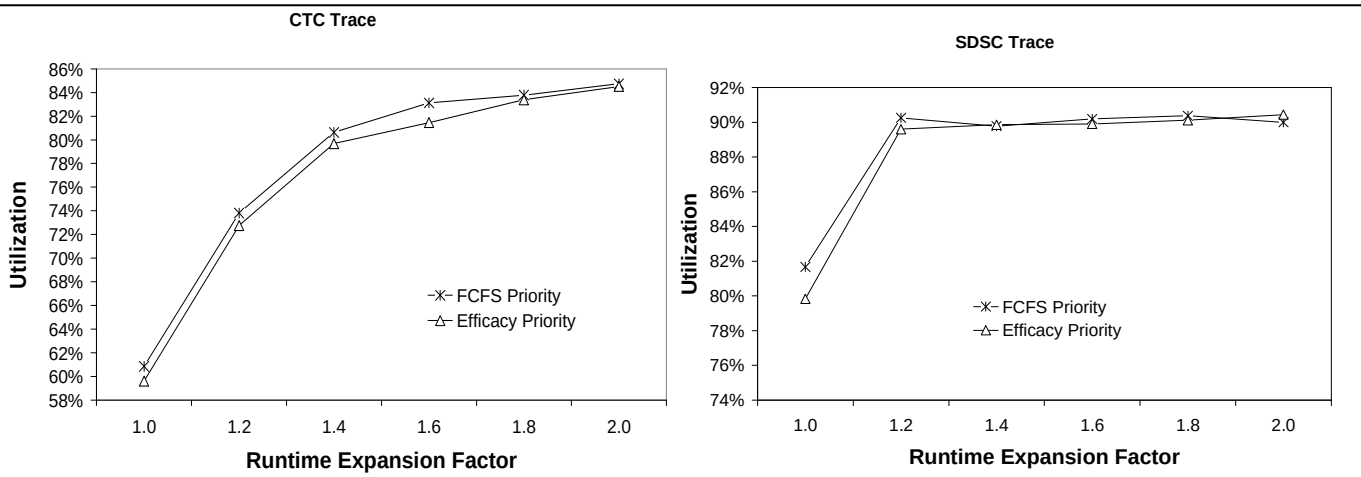

Figure 9: Utilization of efficacy based scheduling

Raw utilization is slightly decreased when using an efficacy priority queue. Even though the turnaround time has improved.

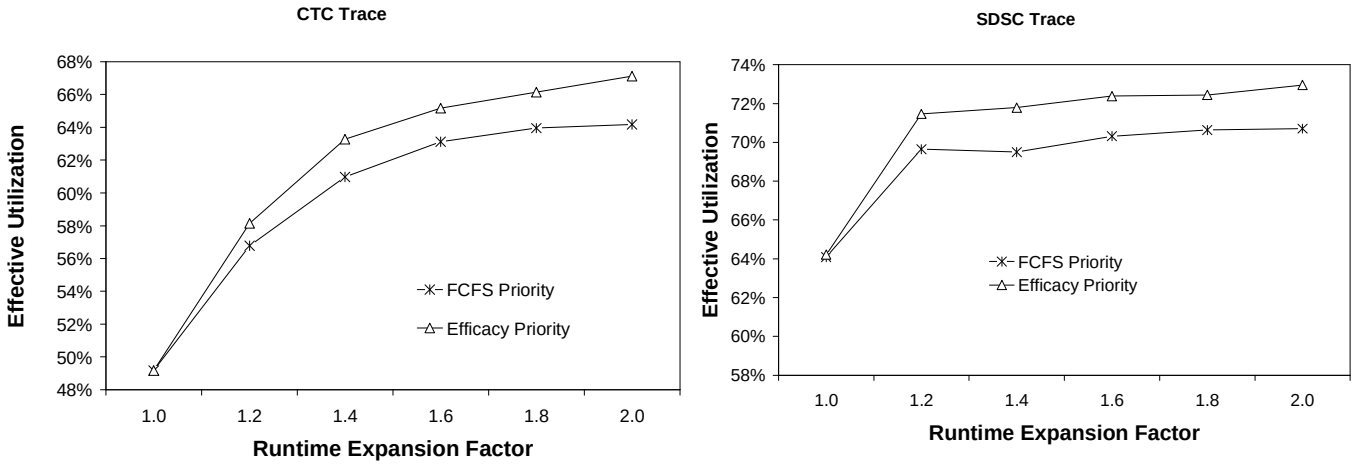

Figure 10: Effective utilization of efficacy based scheduling

Using an efficacy priority queue increases effective utilization, in spite of the decrease in raw utilization. This explains the improvement in turnaround time.

the secondary priority will be FCFS. This priority scheme will guarantee a starvation free system using any of the given strategies. Furthermore, in a conservative backfilling scheduler, a priority queue based on efficacy will result in bounded delays for all jobs. This is due to each job being guaranteed to have an efficacy of 1.0 on at least one site. The job is guaranteed to make progress at this site, leading to a starvation free system. This has resulted in a minimal (<5%) increase in worst case turnaround time, when compared to an FCFS priority.

By changing the priority order to efficacy, a job will have a greater chance to run on its faster machines (because it will have a higher priority on these machines than the jobs with a lower efficacy). This can be expected to increase the average efficacy of the system. This higher average efficacy (as shown by the effective utilization) is expected to lead to a lower turn around time for strategies which use an efficacy priority policy. Figure 10 shows that using an efficacy based priority policy indeed leads to a higher effective utilization. This is in spite of a FCFS priority queue having better raw utilization (Figure 9). The higher effective utilization provides the basis for the improved turn around time seen in Figure 8.

7. Restricted Multi-Site Reservations

So far the strategies implemented in this paper have concentrated on either making one reservation on a single site or reservations at all sites (where the total number of reservations is equal to the total number of sites). We have seen that making multiple reservations shows a substantial improvement in the average turnaround time. However, it is of interest to make fewer reservations, if possible. This is due to the overhead involved in maintaining a larger number of reservations (network latency bound). When a job is ready to start at a site, it must contact all other sites and determine whether it should start, based on the current strategy. When a job has determined it should start (by contacting all other sites where a reservation is held and receiving a reply), it must inform all other sites that the job is starting. This process may happen multiple times for each job (a maximum of once for each site which attempts to start the job). Therefore, a minimum of 3*(K-1) messages must be transferred for each job to start. Further, the job must be transferred to each site where a reservation is made (network bandwidth bound).

This network overhead could be substantially reduced by limiting the number of reservations. When fewer reservations are used per job, each site does not have to contact as many other sites before starting a job, and there is a lower chance that a job will be denied at start (there will be fewer sites to deny the job). These factors can substantially reduce the communication overhead needed.

Figure 11 shows the turnaround time results when each job is submitted to K sites, with K varied from 1 to 4. The graphs show that the greatest degree of improvement is when the number of sites is increased from one to two. There is less of a benefit as the number of sites is further increased. Therefore, when network latencies are high, jobs can be submitted to a smaller number of sites and the multi-reservation scheduler can still realize a substantial fraction of the benefits achievable with a scheduler that schedules each job at all sites. In order to avoid starvation (when using efficacy as the priority) the efficacy is relative to the sites where the job was scheduled. Therefore, each job will still be guaranteed to have an efficacy of one at least on one of the sites. Hence the jobs are still guaranteed to be free from starvation.

In our previous graphs and data, we use the instantaneous load metric (either maintained

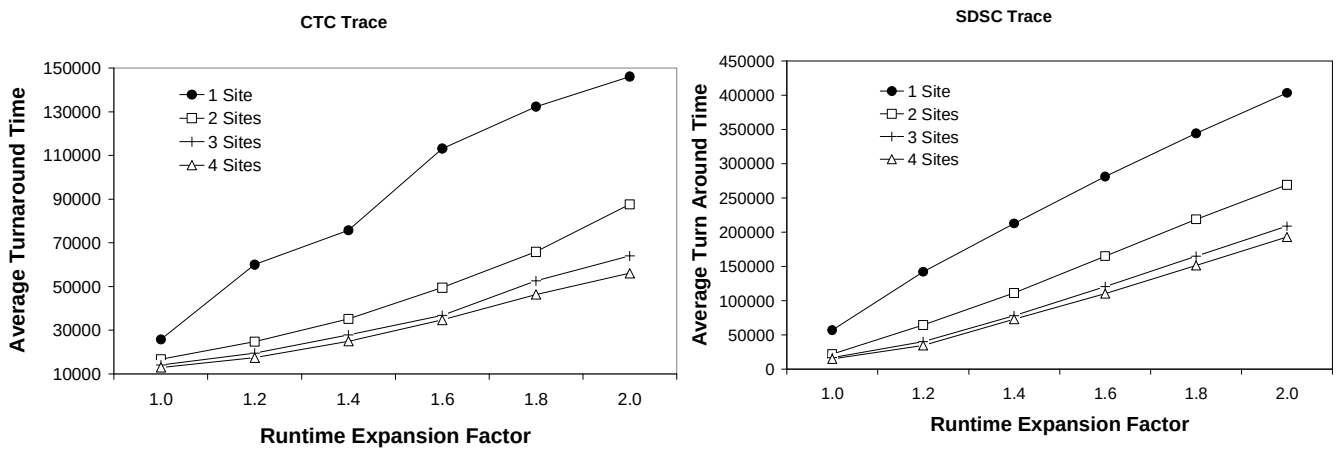

Figure 11: Performance with restricted multi-site reservation, no communication costs

As the number of scheduling sites are increased, the turnaround time monotonically decreases

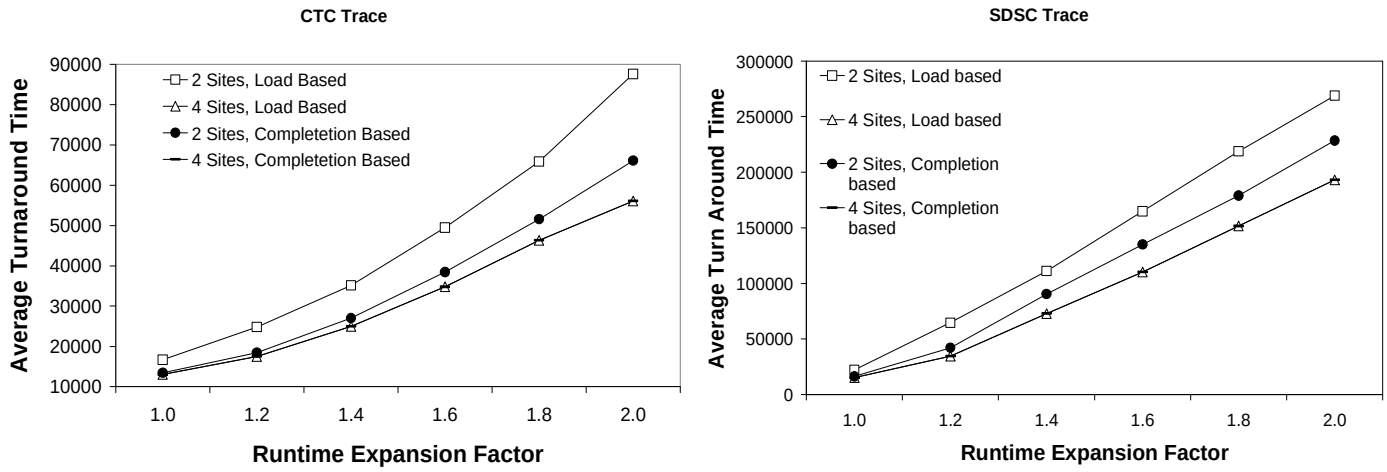

Figure 12: Effect of site selection criteria, no communication costs

Using the completion time of the reservations, as opposed to an instantaneous load, improves the turnaround time.

at the metascheduler or by periodically polling the remote sites) to choose which K sites to schedule each job. We next consider a more accurate approach to selecting the sites. Instead of using the instantaneous load, we query each site to determine the earliest completion time, based on its current schedule. This further takes into account the efficacy of the job at each location and there is a higher probability that the job will run on a site where its efficacy is maximum. Figure 12 shows that changing the mechanism for site selection can have significant impact on the turn around time. There is of course no change when the job is submitted to the maximum number of sites, because all sites are being chosen, regardless of the site selection mechanism.

From Figure 12 it can be observed that when using completion time as the site selection criterion, submitting to fewer sites can be almost as effective as submitting to all sites. However, this more accurate approach does not come free. There is an additional initial overhead that must be incurred to determine a job's K best completion times, which is not incurred when using the instantaneous load. The load of each site can be maintained incrementally by the metascheduler, or the metascheduler can periodically update the load of each site; therefore there are no per-job communication costs incurred in selecting the K least loaded sites. In contrast, to determine the K best completion times , each site must be queried for its expected completion time. For N sites, the querying will require 2*N messages, N messages from the metascheduler (to contact each site with the job specifications) and a response from each site. When using completion time to determine the K sites, there are an additional 2*N messages needed to determine the minimum completion times. Therefore, a minimum of 3*(K-1) messages per job are required when using the instantaneous load and a minimum of 2*N+3*(K-1) messages when using completion time. Thus, for a substantially large N, if the scheduler can generate similar results with a smaller K the dependence on the network can be reduced, even if the K sites are chosen via the best completion time.

Next we assess the impact of communication overhead for data transfer when running a job at a remote site. We assumed a data transfer rate of 10Mbps. Each job was assigned a random size (for executable plus data) between 500MB and 3GB. The data transfer overhead was modeled by simply increasing the length of a job if it is run remotely (local jobs do not involve any data transfer). Figure 13 shows the average turnaround time including the extra overheard. The turnaround times have increased due to the additional overhead, but relative trends remain the same. Figure 14 shows the number of control messages which were actually needed to maintain the schedule. There is a substantial increase in the number of messages when the value of K is increased.

8. Related Work

Recent advances in creating the infrastructure for grid computing (e.g. Globus [13], Legion[18], Condor-G[14] and UNICORE [24]) facilitate the deployment of metaschedulers that schedule jobs onto multiple heterogeneous sites. However there has been little work on developing and evaluating job scheduling schemes for a heterogeneous environment.

Research into scheduling for the grid environment can be broadly classified into two categories: a) projects where the focus is on approaches to optimize the performance of a single job in a grid environment, and b) projects that focus on performance optimization across a collection of independent jobs. Much of the work on scheduling at GRAIL (Grid Research And Innovation Laboratory at UCSD) and the

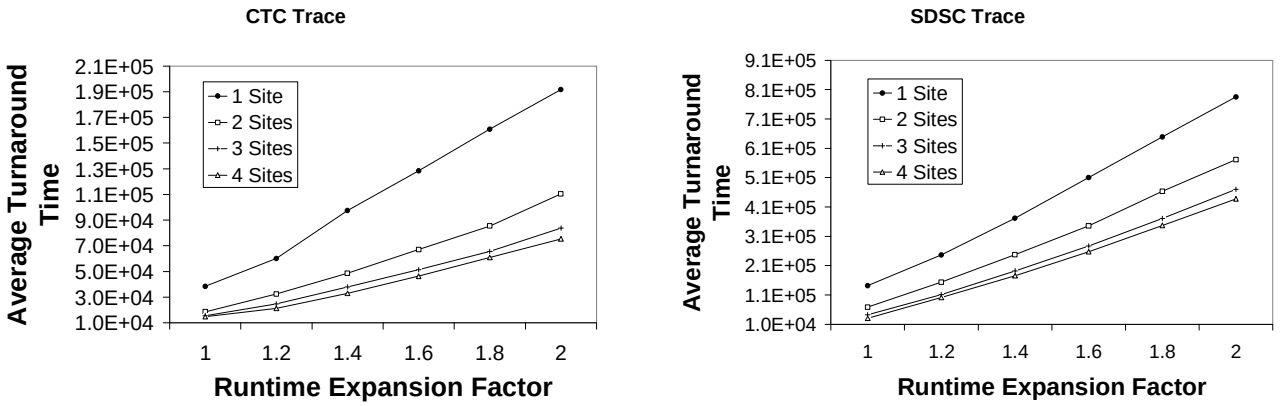

Figure 13: Efficacy based queue and a contention-less network model

Adding data transfer time uniformly increase turnaround time, but does not affect the trends

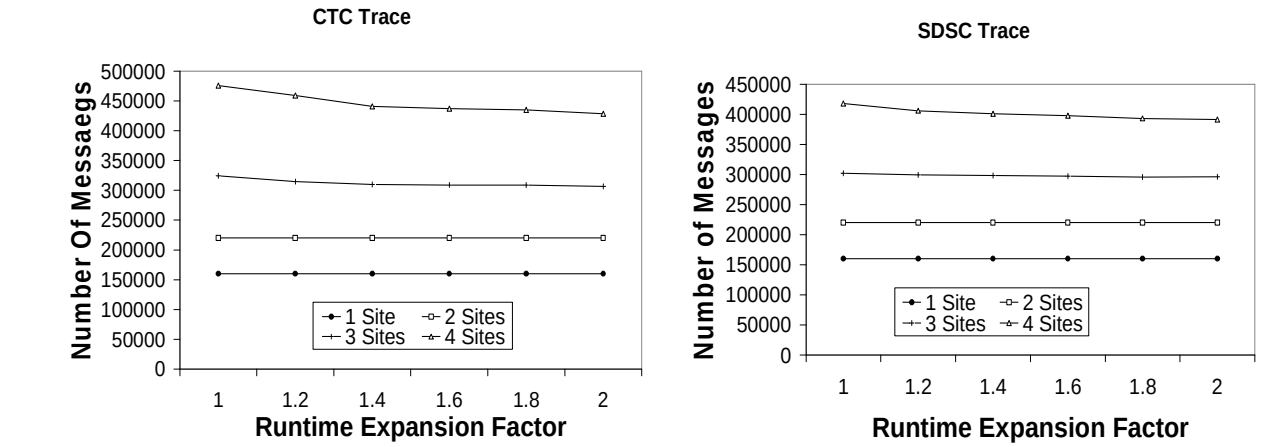

Figure 14: Efficacy based queue and a contention-less network model

Decreasing the number of scheduling sites significantly reduces the number of control messages needed to maintain the schedule.

GrADS (Grid Applications Development Software project headed by Rice U.) project belongs in the former category [1][2][4] [6][7][27][33]. In contrast, the scheduling strategy proposed in this paper falls in the latter.

Application level scheduling techniques [1][4][16] have been developed to efficiently deploy resource intensive applications that require more resources than available at a single site and parameter sweep applications over the grid. There have been some studies on decoupling the scheduler core and application specific components [7] and introducing a Metascheduler [33] to balance the interests of different applications. But none of the above works address the problem of developing effective scheduling strategies for a heterogeneous environment. [9] proposes an economic model for scheduling in the heterogeneous grid environments where the objective is to minimize the cost function associated with each job - an aspect somewhat orthogonal to that addressed in this paper. [36] proposes a load sharing facility with emphasis on distributing the jobs among the various machines, based on the workload on the machines.

Studies that have focused on developing job scheduling algorithms for the grid computing environment include [15][17][19][28][31]. Most of these studies do not address the issue of heterogeneity. In [21] a few centralized schemes for sequential jobs were evaluated. In [15], the performance of a centralized metascheduler was studied under different levels of information exchange between the meta scheduler and the local resource management systems where the individual MPP's are heterogeneous in that the number of processors at different sites differs, but processors at all sites are equally powerful. In [8], the impact of scheduling jobs across multiple homogenous MPP's was studied, where jobs can be run on a collection of homogenous nodes from independent MPP's, where each MPP may have a different number of nodes. The impact of advance reservations for meta-jobs on the overall system performance was studied in [28]. In [19][31], some centralized and decentralized scheduling algorithms were evaluated for metacomputing, but only the homogeneous context is considered.

9. Current Status and Future Work

The simulation results show that the proposed scheduling strategy is promising. We plan next to implement the strategy in the Silver/Maui scheduler and evaluate it on the Cluster Ohio distributed system. The Ohio Supercomputer Center recently initiated the Cluster Ohio project [38] to encourage increased academic usage of cluster computing and to leverage software advances in distributed computing. OSC acquires and puts into production a large new cluster approximately every two years, following the budget cycle. OSC distributes the older machine in chunks to academic laboratories at universities around the state, with the proviso that the machines continue to be controlled centrally and available for general use by the OSC community. Each remote cluster is designed to be fully stand-alone, with its own file system and scheduling daemons. To allow non-trivial access by remote users, currently PBS and Maui/Silver are used with one queue for each remote cluster and require that users explicitly choose the remote destination. Remote users can access any cluster for a PBS job by using the Silver metascheduler. Globus is used to handle the mechanics of authentication among the many distributed clusters. We plan to deploy and evaluate the heterogeneous scheduling approach on the Cluster Ohio systems.

Acknowledgements

We thank the Ohio Supercomputer Center for access to their resources. We also thank the anonymous referees for their suggestions.

References

[1] D. Bailey, T. Harris, W. Saphir, R. Wijngaart, A. Woo, and M. Yarrow, "The NAS Parallel Benchmarks 2.0," Report NAS-95-020, December, 1995 Numerical Aerodynamic Simulation Facility, NASA Ames Research Center http://www.nas.nasa.gov/NAS/NPB/Specs/npb2_report.ps

[2] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra A. Hensgen, Richard F. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", Journal of Parallel and Distributed Computing, Volume 61, 2001, pages 810-837

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

[4] H. Casanova, G. Obertelli, F. Berman, and R. Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid," In *Proceedings of Supercomputing*'00, Nov. 2000.

[5] S-H. Chiang and M. K. Vernon, "Production job scheduling for parallel shared memory systems," *Proc. Intl. Parallel and Distributed Processing Symposium*, San Francisco, CA, Apr. 2001.

[6] G. Cooperman, H. Casanova, J. Hayes, T. Witzel, "Using TOP-C and AMPIC to Port Large Parallel Applications to the Computational Grid," *Proc. CCGrid 02*, May 2002.

[7] H. Dail, H. Casanova, F. Berman, "A Decoupled Scheduling Approach for the GrADS Environment," *Proceedings of Supercomputing'02*, November 2002.

[8] C. Ernemann, V.Hamscher, U. Schwiegelshohn, R. Yahyapour, A. Streit, "On Advantages of Grid Computing for Parallel Job Scheduling"

[9] C. Ernemann, V. Hamscher and R. Yahyapour, "Economic Scheduling in Grid Computing," Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '02), in conjunction with the High Performance Distributed Computing Symposium (HPDC '02), July, 2002.

[10] D. Feitelson, Parallel Workloads Archive, URL: http://www.cs.huji.ac.il/labs/parallel/workload/

[11] D. Feitelson and M. Jette, "Improved Utilization and Responsiveness with Gang Scheduling," *3rd Workshop on Job Scheduling Strategies for Parallel Processing*, LNCS 1291 pp. 238-261, 1997.

[12] D. Feitelson, L. Rudolph, U. Schweigelshohn, K. Sevcik, and P. Wong. "Theory and Practice in Parallel Job Scheduling," *3rd Workshop on Job Scheduling Strategies for Parallel Processing*, Springer-Verlag Lecture Notes in Computer Science, Vol. 1291, pp. 1-34, April 1997.

[13] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *Intl. J. Supercomputer Applications*, Vol. 11, No. 2, pp. 115-128, 1997,

[14] J. Frey, T. Tannenbaum, M. Livny, I. Foster and S.Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," Proc. Intl. Symp. On High Performance Distributed Computing (HPDC 10), 2001.

[15] J. Gehring and T. Preiss, "Scheduling a Metacomputer with Uncooperative Sub-schedulers," In *Proc. JSSPP '9*9, pages 179–201, 1999

[16] J. Gehring and A. Reinefeld, "MARS - A Framework for Minimizing the Job Execution Time in a Metacomputing Environment," Future Generation Computer Systems, FGCS-12, 1. (1996), Elsevier, pp. 87-90

[17] J. Gehring and A. Streit, "Robust Resource Management for Metacomputers," In *Proc. HPDC '0*0, pages 105–111, 2000.

[18] A. S. Grimshaw, W. A. Wulf and the Legion team, "The Legion Vision of a Worldwide Computer," *Communications of the ACM*, Vol. 4, No. 1, pp. 39-45, Jan. 1997.

[19] 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.

[20] P. Holenarsipur, V. Yarmolenko, J. Duato, D. K. Panda and P. Sadayappan, "Characterization and Enhancement of Static Mapping Heuristics for Heterogeneous Systems," *Proc. Intl. Conf. On High-Performance Computing*, December 2000.

[21] H. A. James, K. A. Hawick, and P. D. Coddington, "Scheduling Independent Tasks on Metacomputing Systems," In *Proc. Conf. on Parallel and Distributed System*s, 1999.

[22] J.P. Jones and B. Nitzberg, "Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization," *5th Workshop on Job Scheduling Strategies for Parallel Processing*, 1999

[23] A.W. Mu'alem and D. G. Feitelson, Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling

[24] M. Romberg, "The UNICORE Architecture: Seamless Access to Distributed Resources," In *Proc. HPDC '99*, pages 287–293, 1999.

[25] W. Saphir, A. Woo and M. Yarrow, "The NAS Parallel Benchmarks 2.1 Results," Report NAS-96-010, August 1996 http://www.nas.nasa.gov/NAS/NPB/Reports/NAS-96-010.ps

[26] J. Skovira, W. Chan, H. Zhou and D. Lifka, "The EASY-Loadleveller API Project," *Proc. 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, Honolulu, Apr. 1996, pp. 41-47. Lecture Notes in Comp. Sci. Vol. 1162, Springer-Verlag.

[27] S. Smallen, H. Casanova, F. Berman, "Applying Scheduling and Tuning to On-line Parallel Tomography," *Proc. Supercomputing'01*, Nov. 2001.

[28] Q. Snell, M. Clement, D. Jackson, and C. Gregory, "The Performance Impact of Advance Reservation Meta-Scheduling," D. G. Feitelson and L. Rudolph (Eds.), *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, Lecture Notes in Computer Science vol. 1911, 2000.

[29] S. Srinivasan, R. Kettimuthu, V. Subramani and P. Sadayappan, "Characterization of Backfilling Strategies for Job Scheduling," *Proc. of 2002 Intl. Workshops on Parallel Processing (held in conjunction with the 2002 Intl. Conf. on Parallel Processing, ICPP 2002)*, Aug. 2002.

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

[31] V. Subramani, R. Kettimuthu, S. Srinivasan and P. Sadayappan, "Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests," *Proc. of 11-th IEEE Symposium on High Performance Distributed Computing (HPDC 2002)*, July 2002.

[32] D. Talby and D. G. Feitelson, "Supporting priorities and improving utilization of the IBM SP2 scheduler using slack-based backfilling," In 13th *Intl. Parallel Processing Symp.*, pp. 513-517, Apr 1999

[33] S. S. Vadhiyar and J. J. Dongarra, "A Metascheduler for the Grid," *Proc. of 11-th IEEE Symposium on High Performance Distributed Computing (HPDC 2002)*, July 2002.

[34] J. Weissman and A. Grimshaw, "A Framework for Partitioning Parallel Computations in Heterogeneous Environments," *Concurrency: Practice and Experience*, Vol. 7, No. 5, August 1995.

[35] V. Yarmolenko, J. Duato, D. K. Panda and P. Sadayappan, "Characterization and Enhancement of Dynamic Mapping Heuristics for Heterogeneous Systems," *ICPP 2000 Workshop on Network-Based Computing*, August 2000.

[36] S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia: A load sharing facility for large heterogeneous distributed computer systems," Software - Practice and Experience (SPE), December 1993

[37] http://www.nas.nasa.gov/NAS/NPB/NPB2Results/971117/all.html

[38] http://oscinfo.osc.edu/clusterohio

[39] Workshops on Job Scheduling Strategies for Parallel Processing, www.cs.huji.ac.il/~feit/parsched/