Costs and Benefits of Load Sharing in the Computational Grid

Darin England
and Jon B. Weissman
Department of Computer Science and Engineering
University of Minnesota, Twin Cities
Email: {england,jon}@cs.umn.edu

***Abstract*— We present an analysis of the costs and benefits of load sharing of parallel jobs in the computational grid. We begin with a workload generation model that captures the essential properties of parallel jobs and use it as input to a grid simulation model. Our experiments are performed for both homogeneous and heterogeneous grids. We measured average job slowdown with respect to both local and remote jobs and we show that, with some reasonable assumptions concerning the migration policy, load sharing proves to be beneficial when the grid is homogeneous, and that load sharing can adversely affect job slowdown for lightly-loaded machines in a heterogeneous grid. With respect to the number of sites in a grid, we find that the benefits obtained by load sharing do not scale well. Small to modest-size grids can employ load sharing as effectively as large-scale grids. We also present and evaluate an effective scheduling heuristic for migrating a job within the grid.**

I. Introduction

An emerging trend in high-performance computing is to build interconnected networks of supercomputing centers known as computational grids. Individually, these centers house computing resources and instruments needed for large-scale collaborative applications. As these applications place increasing demands on existing resources, increased efficiency in scheduling jobs onto the grid is becoming more important. Already proven in the LAN environment, load sharing is becoming feasible in WANs and grids. The emergence of test-beds like TeraGrid[1] promises remarkable network bandwidth between distant sites, enabling load sharing with minimal network penalties.

In this work, we investigated the costs and benefits of load sharing of parallel jobs in a simulated computational grid. First, we present a detailed model of a supercomputer workload. The nature of our workload model makes it easy to use as input to the grid simulation experiments. We performed experiments for both homogeneous and heterogeneous grids. The results indicate that load sharing among sites is indeed worthwhile. We find that in a homogeneous grid, any amount of load sharing results in decreased wait times for users. In a heterogeneous grid with differing workloads and machine capacities, we find that the processing of remote jobs on a (previously) lightly-loaded machine can cause delay to local jobs. We also investigate how well the benefits of load sharing scale as the number of sites in a grid increases. The benefits are limited in that most of the opportunities for load sharing are exploited in small and medium-sized grids. Finally, we present a simple heuristic for determining the target machine of a migrated job. This heuristic, which we call *Weighted Queue*, is easy to compute and does not require estimates of job run time. The paper is organized as follows: In Section 2 we discuss related research. We present our workload model in Section 3. Section 4 represents the bulk of the paper, which includes a description of the simulation model, the homogeneous and heterogeneous grid results, the scaling results, and the evaluation of the scheduling heuristic. Section 5 concludes the work.

II. Related Work

The quality of the input data is paramount to any simulation model. Cirne and Berman [2] developed a comprehensive model of workloads for space-shared parallel supercomputers. They modeled the variation in job arrival rates throughout the work day and they examined the differences between estimated and actual job run times. Our workload model differs from theirs in that we provide an alternative method for generating the job arrivals and for modeling the job run times and job run lengths. We describe our workload model in the next section. In considering the importance of workload traces in simulation experiments, Lo et. al. [3] investigated the effects on job scheduling algorithms due to the use of real workload traces vs. synthetic workload models. They found that the use of either real or synthetic workloads did not affect the overall performance of job scheduling algorithms. However, we note that the use of a real workload trace necessarily limits the simulation to a single run. By using a workload model and its generated job traces, a large number of simulation runs may be conducted, thereby producing enough data to make statistically significant comparisons among alternative scenarios. Lo et. al. did find that other workload characteristics such as the proportion of *power-of-two* job sizes and the correlation between job size and job run time did affect scheduler performance. In the next section, we discuss these two characteristics as they relate to our experiments.

Hollingsworth and Maneewongvatana [4] propose a novel approach to scheduling parallel jobs in a computational grid. They present the idea of an imprecise calendar where jobs are scheduled into time slots by a hierarchical system of manager nodes. Time slots that are further into the future are scheduled at a coarse level. As the time for a slot nears, it is scheduled at a finer level. Like the imprecise calendar approach, we wish to efficiently distribute parallel jobs in a grid. However, we employed simple scheduling methods that do not require job information such as run length[1]. Eager et. al. [5] examined the relative benefits of simple vs. complex load sharing policies. Using an analytical model for a homogeneous network, they concluded that simple policies that require only a small amount of state information perform as well as complex policies. We also examined simple policies and we extended their work by comparing relative amounts of load sharing in both homogeneous and in heterogeneous networks. More recently, Subramani et. al. [6] used simulation to evaluate distributed scheduling algorithms in a grid environment. They use a scheme in which jobs are placed in queues at multiple sites. Then, the system tracks which copy of the job is first to begin execution and all other copies are canceled. Our work differs from theirs in that we employ a scheduling algorithm that is easy to implement and we have a more complete workload model with which we are able to make multiple simulation runs and hence reduce the variance in our performance measures.

III. Modeling the Workload

Our workload generation model takes an actual job trace as input. It produces a synthetic workload in standard workload format[7] that captures the following job characteristics:

1) Job inter-arrival times
2) Job sizes
3) Job run times

In this section, we discuss how we model each of these characteristics. Our model was created from careful examination of the traces shown in Table I. Since the SDSC (San Diego Supercomputing Center) and the CTC (Cornell Theory Center) traces are more recent and come from a machine that is more widely used, these two traces were used as the basis for our simulation experiments.

Center	*Machine*	*Nodes*	*Time Period*
LANL	CM5	1024	Oct 1994 to Sep 1996
SDSC	IBM SP2	128	Apr 1998 to Apr 2000
CTC	IBM SP2	512	Jun 1996 to May 1997

TABLE I
Actual Workloads Examined

In production systems, it is likely that some jobs will be unable to begin execution until other jobs have finished due to precedence constraints. The standard workload format allows for the specification of precedence constraints; however, none of the workloads that we examined contained this information. For this reason and for simplicity, we assume that all jobs arrive to the system independently. Figure 1 shows the average

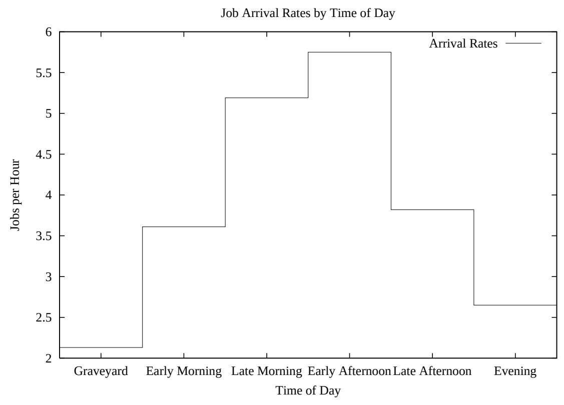

Fig. 1. SDSC Job Arrival Rates by Time of Day

arrival rates of jobs to the IBM SP2 supercomputer at the San Diego Supercomputing Center. From the figure, it is clear that more jobs arrive to the system during the working hours than during the night. This phenomenon, which was also observed by Cirne and Berman, occurs in all of the workload traces that we examined. We capture the variation in job arrival rates by using a Nonstationary Poisson Process to model the job arrivals. In a Poisson Process, the inter-arrival times (the times between job arrivals) follow an Exponential probability distribution. Thus, in our model, the job inter-arrival times are generated from six Exponential distributions, one for each period of the day as shown in Figure 1. Modeling the job arrivals in this manner helps to produce a realistic workload which is more intense during the middle of the day.

For our workload model, the size of a job is characterized by the number of CPUs it requests. The workload traces that we examined are dominated by *power-of-two* sizes, i.e. 2, 4, 8, 16, 32, and 64. All other job sizes occur infrequently. Cirne and Berman [2] model the size of a job with a uniform-log distribution. In order to capture the prevalence of the power-of-two job sizes, they added a direct probability for turning a job size into its nearest power-of-two neighbor. In contrast, we chose to use a discrete probability distribution that reflects the frequency with which the job sizes appear in an actual workload. The discrete probabilities are computed directly from the ratios in the real workload. Figure 2 shows the job size probabilities for the SDSC data up to 64 CPUs.

We made several attempts at modeling job run times as a function of job size. However, we found no correlation between these two characteristics. At each center, we assume an independent work model in which there is no correlation between job size and job run time. After experimenting with several probability distributions, we found that the job run times fit the Weibull distribution quite well. The quality of the fit is not surprising since Weibull random variables are commonly used to model task completion times. Figure 3 shows the actual job run times from the SDSC workload plotted against our estimation of the Weibull distribution. The plot is a histogram with a bin size of 500 seconds. It is clear that most jobs run for a short period of time, while a few jobs

[1]That is, no estimate of run length is required for job migration to another machine in the grid. However, we employ backfilling at the local machine level which requires run time estimates.

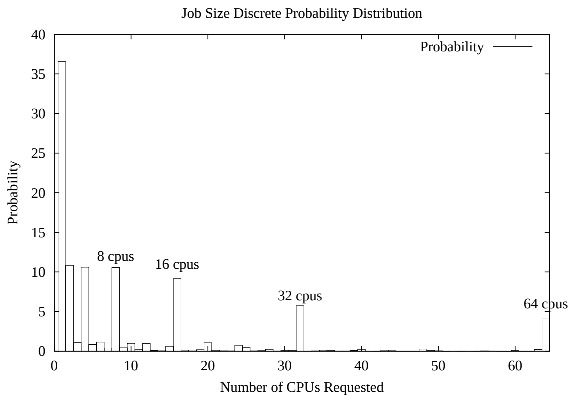

Fig. 2. Discrete Probability Distribution for SDSC Job Sizes

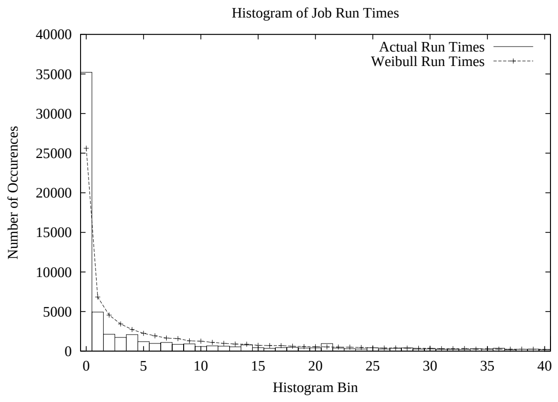

Fig. 3. SDSC Actual Job Run Times vs. Estimated Weibull Distribution

run for very long periods of time. Again, the size of the job is not a good predictor of job run time. We only included jobs that ran to completion in order to avoid jobs that were killed or that died.

IV. Simulation

A. Model Description

Our simulation model of a computational grid consists of four supercomputer centers. Traditionally, each center would operate in an autonomous fashion with no job migration to other sites. The model shown in Figure 4 illustrates the idea of cooperation among the centers by allowing some jobs to be migrated to remote sites. Each center has a local workload that is representative of the actual workload for its machine type. Some percentage of the job arrivals are flagged as migratable. We envision this occurring as users indicating via a job submission script that they are willing to allow a particular job to be migrated. Of course not all jobs are migratable due to various reasons: locality of data, parallel architecture-specific code, security concerns, etc. Therefore, our experiments were conducted with varying percentages of the workload being migratable. The choice of which jobs are flagged as such is completely random. In this way, we are certain to simulate the migration of both large jobs and small jobs.

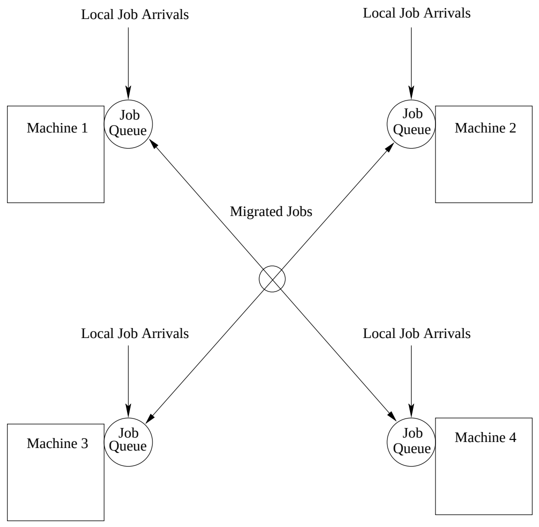

Fig. 4. Supercomputing Grid Simulation Model

We say that a *local* job is a job that executes on the machine at which the job originally arrived. A *remote* job is one that has been migrated and executes on a remote machine. We make this distinction because a job that is flagged as migratable might actually execute on its local machine if it appears more favorable. There are two requirements for a job to be transferred to a remote machine:

1) The originating machine's job queue must be nonempty.
2) There must be a remote machine with a more favorable queue status.

In other words, if a machine is currently lightly loaded, i.e. its queue is empty, then it will not attempt to migrate an arriving job (even though the job may be flagged as migratable.) In addition, when a machine's queue is nonempty and it attempts to migrate a job that just arrived, then it will transfer the job to the machine whose queue size is smallest, thus we employ the Shortest Queue scheduling policy for the experiments in this section. Later, we will introduce a new scheduling policy called Weighted Queue. These requirements are common sense attempts to create a reasonable migration policy. This means that before a machine attempts to migrate a job, it must poll the other machines in the grid to obtain their load information.

The cost of job migration includes estimates for network bandwidth and the amount of data to be transferred. As part of their work in predicting data transfer costs, Vazhkudai et. al. [8] measured the end-to-end bandwidth between two remote supercomputing centers. Their measurements were made using GridFTP, the file transfer service of the Globus Toolkit[9]. They found the network bandwidth to vary from 1.5 to 10.2 MB/sec (megabits). For our experiments, we use a constant network bandwidth of 5 MB/sec. Based on the work of Vazhkudai et. al., this represents an achievable bandwidth for current systems. In the future, advances in network infrastructure will help to reduce the cost of job migration. For example, the TeraGrid project [1] will have the ability to transfer data at the rate of 40GB/sec. The actual workload traces that we examined did not contain information about data sizes. In the absence of this information, we used a Triangular distribution as an approximation. The range of the distribution is from 1MB to 1GB, with a mode of 100MB (megabytes)[2].

For scheduling jobs at each local machine, we employ backfilling, a technique by which a job is allowed to move ahead of other jobs in the queue and begin execution as long it does not cause the first job in the queue to be delayed. The version of backfilling that we use is known as aggressive backfilling. It is employed in the EASY scheduler on the IBM SP2. Our implementation is exactly the one described in Mu'alem and Feitelson [10]. For an excellent description of backfilling and its sensitivity to user run time estimates, we refer the reader to their work.

B. Experimental Design

	Homogenous Network	*Heterogeneous Network*
Machine 1	SDSC1	SDSC1
Machine 2	SDSC2	SDSC2
Machine 3	SDSC3	CTC1
Machine 4	SDSC4	CTC2

TABLE II

NETWORK CONFIGURATIONS FOR SIMULATION RUNS

Table II shows the homogeneous and the heterogeneous grid makeup for our simulation experiments. We made 20 independent replications of the simulation for each type of network and for each level of load sharing, 0, 25, 50, 75, and 100%. A level of load sharing indicates the percentage of jobs flagged as migratable. Each replication of an experiment was performed with a different (but statistically similar) workload that was generated in accordance with our workload model. Our performance measure of interest is job slowdown, which is defined as follows.

$$\text{Slowdown} = \begin{cases} \frac{\text{Queue Time}+\text{Run Time}}{\text{Run Time}} & \text{local job,} \\ \frac{\text{Migration Time}+\text{Queue Time}+\text{Run Time}}{\text{Run Time}} & \text{remote job.} \end{cases}$$

Job slowdown captures the notion that users are more willing to accept long queue times for long-running jobs than for short-running jobs. For each measurement shown in the *Results* section, we present an average of the 20 replications for an experiment. In order to be certain that performance differences among the different levels of load sharing are not due to randomness in the synthetic workloads, we used the same sets of synthetic workloads as input to each experiment.

C. Results

1) Homogeneous Grid Simulation: In the homogeneous grid simulation, all machines have statistically identical workloads. Therefore, all machines get roughly the same intensity of workload regardless of the amount of load sharing performed. We present the average job slowdown for each level of load sharing in Figure 5. Since the results for all machines in the homogeneous network are similar, only the results for one SDSC-type machine are presented. Each level of load sharing corresponds to an experiment and the average job slowdown is presented. The results are broken out by *local* and *remote* jobs. From the figure, we see that as the amount of load sharing is increased, the average slowdown for local jobs decreases. This is because as more jobs are allowed to be migrated, there is more opportunity to exploit the benefits of load sharing, i.e. machines are able to offload more work to less heavily-loaded machines. Also, by the nature of our migration policy, a machine will not attempt to migrate a job if its own job queue is empty. Therefore, local jobs arriving to an empty queue (which is common when backfilling is employed) are guaranteed to execute on the lightly-loaded local machine. At the 25% load sharing level, remote jobs have shorter average slowdown than local jobs. This is because migrated jobs get sent to machines with more favorable queue statuses. At 25% load sharing, the majority of jobs (75%) are not allowed to be migrated and so they must execute locally, regardless of the load on the local machine. Compared to the slowdown for local jobs, the slowdown for remote jobs remains relatively unchanged as the amount of load sharing is increased, although there is a slight increase at the 100% level. We note that this increase is possible because the Shortest Queue scheduling policy is not optimal. Although not presented, we also collected average and median job queue times and average queue sizes for each experiment. These statistics exhibit the same general trends as job slowdown. We conclude that for a homogeneous grid, even a small amount of load sharing produces benefits. In addition, by the use of a reasonable migration policy, local jobs can greatly benefit from large amounts of load sharing, while remote jobs still experience lower slowdown than when there is no load sharing.

2) A Confidence Interval for Improvement in Average Job Slowdown: Here, we statistically compare the improvement in average job slowdown for local jobs when the amount of load sharing is increased from zero to 25%. We present a paired-t confidence interval. Since different sets of workloads were used for each replication of an experiment, our observations

[2]Data sizes were estimated based on a survey by Cirne.

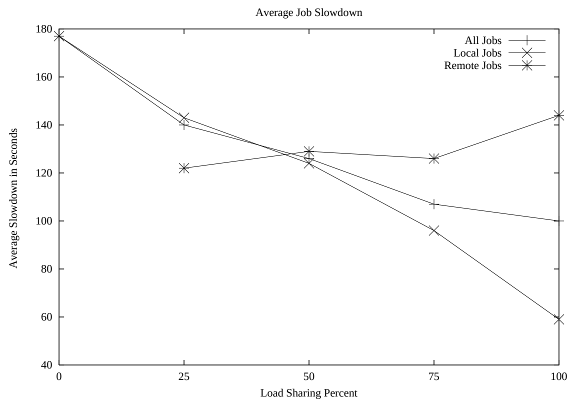

Fig. 5. Machine SDSC1 Homogeneous Grid Simulation

of average slowdown are IID (Independent and Identically Distributed.) Let our observations of slowdown be labeled as X_{ij} for $i = 1, 2$ (for no load sharing and for 25% load sharing respectively), and for $j = 1, \ldots, n$ (where n is 20 because there are 20 replications.) Let $Z_j = X_{1j} - X_{2j}$. We construct a 90% confidence interval for $E(Z_j)$, i.e. for the expected value of the difference in average job slowdown. If this confidence interval does not contain zero, then we can state with approximately 90% confidence that a small amount of load sharing (25%) decreases the average job slowdown (assuming the accurateness of our workload and simulation models.) The confidence interval is constructed as follows. We first compute the average and an estimate of the variance of the Z_j's.

$$\bar{Z}(n) = \frac{\sum_{j=1}^{n} Z_j}{n}$$

and

$$\widehat{var}[\bar{Z}(n)] = \frac{\sum_{j=1}^{n} [Z_j - \bar{Z}(n)]^2}{n(n-1)}$$

The 90% confidence interval is

$$\bar{Z}(n) \pm t_{n-1,0.95} \sqrt{\widehat{var}[\bar{Z}(n)]}$$

We computed $\bar{Z}(20) = 34.2$ and $\widehat{var}[\bar{Z}(20)] = 169.8$, which leads to a 90% confidence interval of $[11.7, 56.7]$. Therefore, we can state (with approximately 90% confidence) that under our workload and simulation assumptions, allowing 25% load sharing results in a decrease in slowdown for local jobs of between 11.7 and 56.7.

3) Heterogeneous Grid Simulation: Grids consist of machines that have different capacities, speeds, and workload characteristics. Our simulation of a heterogeneous grid captures those differences in capacities and workload characteristics. We did make the simplification that remote jobs, although generated from different distributions for different machine types, will execute at the same speed on any machine in the network, given the same number of processors. This is a reasonable assumption for our simulations since all of the workloads in the model are based on job traces from IBM SP2 supercomputers.

The model for the heterogeneous grid consists of two SDSC machines and two CTC machines. Each CTC machine has 512 processors and each SDSC machine has 128 processors. Although the CTC machines have more computing capacity, their workloads are more intense than those at the SDSC machines. In fact, the CTC machines handle more than twice the number of jobs; and the average and the median run times for CTC jobs are more than twice those for SDSC jobs[7].

Figures 6 and 7 show the average slowdown for SDSC-type and CTC-type machines respectively. Again, we only present the results for one machine of each type since the results for other two machines are similar. We can see from the ordinate scale in the two figures that the CTC machines have lower job slowdown. The computing capacity of these machines is able to handle their heavy workloads. An interesting result is that the slowdown for SDSC local jobs increases at the 25, 50, and 75% load sharing levels when compared to no load sharing. This is because the remote jobs that get processed by the SDSC machines are in general of a longer duration than the normal local SDSC jobs. Hence, the long-running remote jobs tend to interfere with the processing of local jobs. Not until we have 100% load sharing do the SDSC local jobs actually experience lower average slowdown than under no load sharing (0%.) The slowdown for remote jobs processed at the SDSC machines increases with the amount of load sharing since the queue time increases as more long-running CTC jobs are processed.

It is easy to see that load sharing has greater benefits for users of machines that are more heavily loaded. The slowdown measurements for CTC local jobs become more favorable as the amount of load sharing is increased. The slowdown for remote jobs processed at the CTC machines remains relatively unchanged, although there is a slight increase with the increase in the amount of load sharing. We conclude that load sharing in a heterogeneous grid can adversely affect local jobs on (previously) lightly-loaded machines. Machines that were previously heavily-loaded receive the most benefit. In this type of environment, our results indicate that as much load sharing as possible should be permitted so that the workload can be evenly distributed.

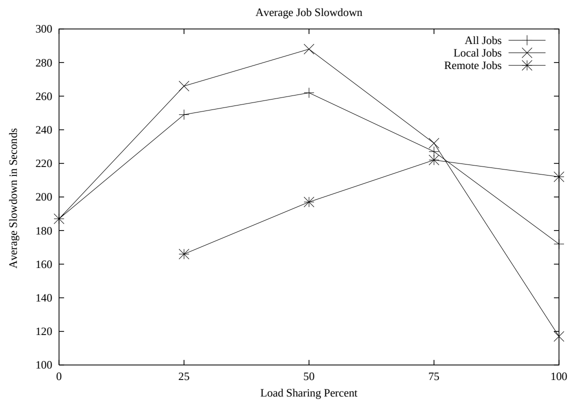

Fig. 6. Machine SDSC1 Heterogeneous Grid Simulation

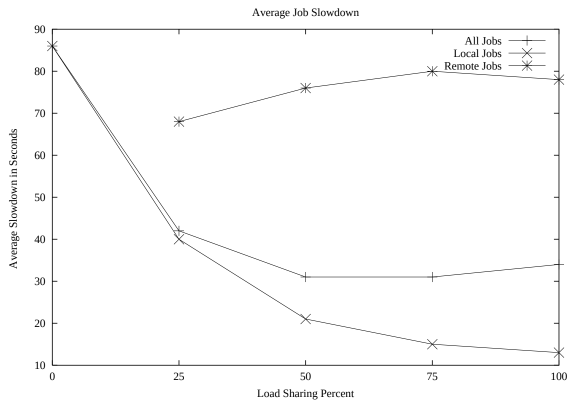

Fig. 7. Machine CTC1 Heterogeneous Grid Simulation

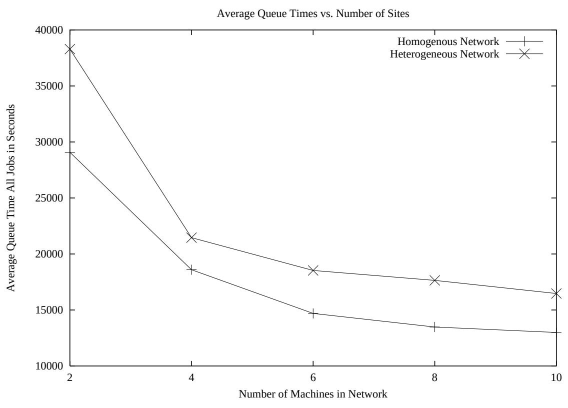

Fig. 8. Scaling the Number of Sites in a Grid

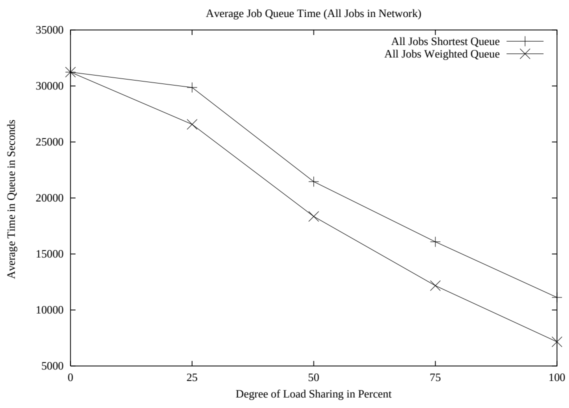

Fig. 9. Weighted Queue vs. Shortest Queue

4) Scaling the Number of Sites: Large-scale projects that include the administration of a computational grid may need to consider expansion of the grid to new sites. An example is the addition of the Pittsburgh Supercomputer Center (PSC) to the TeraGrid project in October 2002. If load sharing is employed, then the effect of the new site will be an important consideration. In this section, we test the performance of load sharing with respect to the number of sites in a grid. In addition to the runs with 4 sites as described in the previous sections, we made runs with 2, 6, 8, and 10 sites for both homogeneous and heterogeneous grids. The homogeneous grid consists entirely of SDSC-type machines. For the heterogeneous grid, we split the number of machines evenly between SDSC-type machines and CTC-type machines. For example, in the run with with 10 total machines, the grid consists of 5 SDSC machines and 5 CTC machines. All runs for this experiment were performed at the 50% load sharing level. We present the average job queue times in Figure 8. In this figure, the average queue times for the heterogeneous networks are higher due to the heavy workloads at the CTC machines. The results for both types of grids are presented in the same figure in order to save space. We are not implying that all homogeneous grids perform better than heterogeneous grids. The figure shows that the average job queue time decreases as the number of sites increases; however, the improvements come at a decreasing rate. In moving from a small number of sites (2 or 4) to a larger number of sites, the benefits of load sharing are readily apparent. As the number of sites increases, the benefits of load sharing still exist, but there seems to be a saturation point where all of the opportunities for load sharing have been exploited. This suggests that small to modest-sized grids can be as effective as large-scale grids with respect to load sharing.

5) A Proposed Scheduling Heuristic: In this section, we present a new heuristic for choosing the target machine for job migration. In the absence of detailed job information, or when low scheduling overhead is desired, one simple measure is the number of jobs in the remote machine's job queue. By itself, this criterion does not always yield the best migration decisions because it does not take into account the job runtime. Nevertheless, schedulers only have estimates of job run time a priori to job execution and these estimates are notoriously inaccurate. Also, backfilling has a significant effect on a machine's queue size. A simple and natural extension to using shortest queue size is to compute the ratio of the total number of CPUs being requested by jobs currently in the queue to the number of CPUs in the machine. We call this criterion *Weighted Queue*. It measures the percentage of a machine's capacity that has already been requested, which could be greater than 100%. The appeal of this heuristic is that it is easy to compute and it does not require estimates of job run time. In a homogeneous grid, the Weighted Queue heuristic performs exactly the same as Shortest Queue because all machines have the same workload characteristics and the same capacity. However, in a heterogeneous grid, this heuristic can exploit the differences in workloads and machine capacities. We compare the performance of Weighted Queue vs. Shortest Queue in a simulation experiment for a heterogeneous grid of two SDSC-type machines and two CTC-type machines. The average job queue times are presented in Figure 9. In this figure, we are directly comparing the two measures. For both heuristics, the average queue time decreases as the amount of load sharing increases. Depending on the level of load sharing, the reductions in queue times range between 4% and 64% for Shortest Queue, and between 15% and 77% for Weighted Queue. Thus, Weighted Queue performs better in the heterogeneous environment.

V. Conclusions

In this work we investigated the benefits of load sharing of parallel jobs among supercomputer centers in a computational grid. By closely examining actual job traces, we were able to create a model that generates accurate synthetic workloads. Using these workloads as input, we employed a discrete-event simulation model to explore the effects of load sharing in both homogeneous and heterogeneous grids. For homogeneous grids, our results demonstrate that cooperation among sites in the form of load sharing leads to overall reduced job slowdown. By the use of a migration policy that only allows migration from a nonempty queue to a queue that is more favorable, local jobs receive the most benefit from load sharing.

For heterogeneous grids, where there are large differences in workload characteristics among the sites, a small amount of load sharing results in increased job slowdown for local jobs on lightly-loaded machines. Local jobs in the heavily-loaded machines receive the most benefit. In this case, the migration policy should be carefully considered and simulation is one tool that can help in this evaluation. We also see that the benefits of load sharing do not scale particularly well. There is a point of diminishing returns as the number of sites in a grid increases. Thus, we conclude that modest-sized grids can provide as much benefit with respect to load sharing as large-scale grids. Finally, we presented a simple heuristic for selecting the target machine of migrated job. The Weighted Queue measure, which considers the number of CPUs being requested relative to a machine's capacity, is effective, easy to compute, and does not require estimates of job run time.

ACKNOWLEDGMENT

The authors would like to acknowledge support from the Department of Energy through contract DE-FG02-03ER25554 and the National Science Foundation 0305641.

REFERENCES

[1] The TeraGrid Project, "A distributed computing infrastructure for scientific research," `www.teragrid.org`.

[2] W. Cirne and F. Berman, "A comprehensive model of the supercomputer workload," in *4th Workshop on Workload Characterization*, Dec 2001.

[3] V. Lo, J. Mache, and K. Windisch, "A comparative study of real workload traces and synthetic workload models for parallel job scheduling," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, Eds. Springer Verlag, 1998, vol. 1459, pp. 25–46, Lecture Notes in Computer Science.

[4] J. K. Hollingsworth and S. Maneewongvatana, "Imprecise calendars: an approach to scheduling computational grids," in *19th IEEE International Conference on Distributed Computing Systems*, 1999.

[5] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogenous distributed systems," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 5, may 1986.

[6] V. Subramani *et al.*, "Distributed job scheduling on computational grids using multiple simultaneous requests," in *11th IEEE International Symposium on High Performance Distributed Computing*, 2002.

[7] Parallel Workload Archive, "The hebrew university of jerusalem, school of computer science and engineering," `www.cs.huji.ac.il/labs/parallel/workload`.

[8] S. Vazhkudai *et al.*, "Predicting the performance of wide area data transfers," in *Proceedings of the International Parallel and Distributed Processing Symposium*, 2002.

[9] The Globus Alliance, `http://www.globus.org`.

[10] A. Mu'alem and D. Feitelson, "Utilization, predictability, workloads, and user run time estimates in scheduling the ibm sp2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, jun 2001.

[11] D. G. Feitelson, "Packing schemes for gang scheduling," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, Eds. Springer-Verlag, 1996, vol. 1162, pp. 89–110, Lecture Notes in Computer Science.

[12] I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a New Computing Infrastructure*. Margan Kaufmann, 1998.

[13] D. M. Gross and C. M. Harris, *Fundamentals of Queueing Theory*, 2nd ed. John Wiley and Sons, 1985.

[14] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, 2nd ed. McGraw Hill, 1991.

[15] W. Smith, I. Foster, and V. Taylor, "Predicting application run times using historical information," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, Eds. Springer Verlag, 1998, vol. 1459, pp. 122–142, Lecture Notes in Computer Science.

[16] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing and Computer Science Applications*, 2nd ed. John Wiley and Sons, Inc., 2002.