# Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests*

Vijay Subramani Rajkumar Kettimuthu Srividya Srinivasan P. Sadayappan

*Department of Computer and Information Science*
*The Ohio State University*
{*subraman,kettimut,srinivas,saday*}*@cis.ohio-state.edu*

**Abstract**

*Even though middleware support for grid computing has been the subject of extensive research, scheduling policies for the grid context have not been much studied. In addition to processor utilization, it is important to consider the response times of jobs in evaluating the performance of grid scheduling strategies. In this paper we propose distributed scheduling algorithms that use multiple simultaneous requests at different sites. Trace-based simulations show that the use of multiple simultaneous requests provides significant performance benefits. We also show how this scheme can be adapted to provide priority to local jobs, without much loss of performance.*

## 1 Introduction

Grid Computing [1] is emerging as a dominant technology for High-Performance Computing. Although there has been considerable recent progress on building the software infrastructure to enable transparent use of globally distributed computers on the grid, the issue of job scheduling on grids has not received much attention. An analysis of the resource usage pattern at several supercomputer centers (San Diego Supercomputer Center, National Center for Supercomputer Applications, Cornell Theory Center, KTH Royal Institute of Technology), shows an interesting "sine wave" pattern. During evenings, the resource requested reaches and sometimes exceeds the maximum capacity of the system, while usage dips to a minimum in the very early hours of the morning. By integrating centers in a computational grid, in addition to providing more computation power than any single site can provide, the time dependent and bursty nature of resource requests can be better averaged by distributing the requests to different centers. Effective scheduling is important in optimizing resource usage, but the task of scheduling is more complex in a metacomputing environment since many clusters with different local scheduling policies are involved.

In this paper, we consider the issue of meta-scheduling of jobs among a number of geographically distributed centers. We first review previously proposed meta-scheduling schemes, both centralized and distributed, and evaluate them using trace-based simulation. We show that the centralized schemes (which however are not scalable) produce schedules with lower slowdown and turn-around time than a more scalable distributed scheme. We propose new distributed schemes that attempt to improve on previously proposed schemes. The key idea we evaluate is that of redundantly distributing each job to multiple sites instead of only sending it to the most lightly loaded site. We show that the new schemes provide significant reductions in average job slowdown and turn-around time. We also evaluate the impact of user inaccuracy in estimation of job runtimes, and the effect of remote data transfer overhead on the efficacy of the proposed schemes.

The paper is organized as follows. In Section 2, we provide some background and review previously proposed meta-scheduling schemes. In Section 3, we evaluate these existing schemes through trace-based simulation. The results from Section 3 motivate the new schemes, which are described in Section 4. In Section 5, we present results of evaluation of the new schemes with the same job traces used earlier. The impact of user inaccuracy in job runtime estimation, and the modeling of remote data transfer overhead are studied in Section 6 and Section 7. Related work is covered in Section 8 and we conclude in Section 9.

## 2 Background and Review of Metascheduling Schemes

The goal of effectively utilizing the power of geographically distributed computing resources has been the subject of many projects like Globus [5], Condor [6] and Legion [9]. These systems provide a metacomputing framework

and focus on issues of security, fault tolerance, resource location and resource co-allocation issues.

Job scheduling is usually visualized in terms of a 2D chart with time along one axis (say vertical) and the number of processors along the other axis. Each job is represented as a rectangle whose height is the user estimated run time and width is the number of processors required. The surface of a schedule represents this 2D chart, as shown in Figure 1.

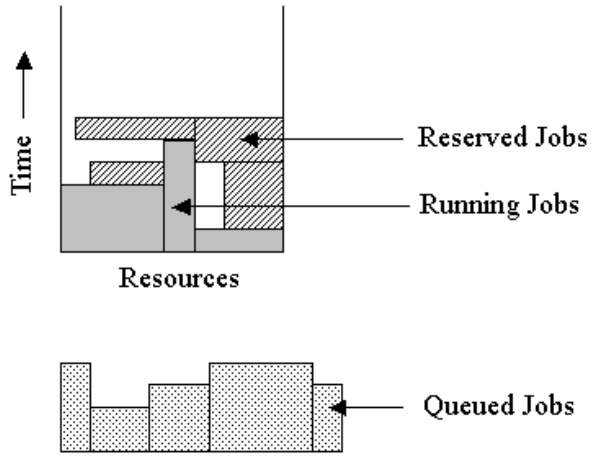


**Figure 1.** Surface/Load Information

## 2.1 Review of Metascheduling Schemes

In this section, we review metascheduling schemes discussed in [10].

### 2.1.1 Centralized Scheme

In the centralized model, the metascheduler maintains surface information about all sites. All jobs are submitted to the metascheduler. Based on the queue of jobs submitted, and the surface information about all the constituent sites, the metascheduler makes scheduling decisions. With this model, the local sites do not perform any scheduling decisions, but are only responsible for dispatching the jobs that are supplied by the metascheduler, and providing information to the metascheduler when jobs complete and free processors.

The centralized scheme is not very scalable because the metascheduler must maintain a lot of detailed information about the constituent sites. This scheme also does not facilitate the use of different priority schemes at the different

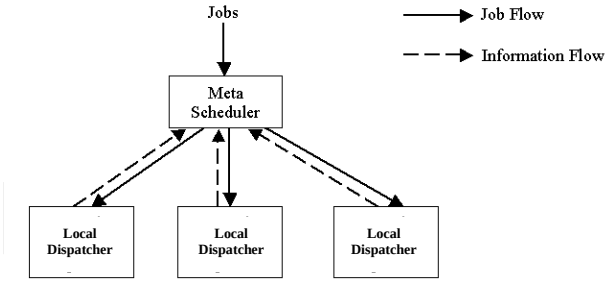


**Figure 2.** Centralized Metascheduler

constituent centers. This can adversely affect the local jobs while benefiting remote jobs.

### 2.1.2 Hierarchical Scheme

With the hierarchical scheme, the scheduling process is shared between the metascheduler and the local sites. All jobs are still submitted to the metascheduler. But unlike the centralized scheme, jobs are not maintained in the metascheduler queue until dispatch time. Instead, at submission time, the metascheduler sends the job to that site at which the earliest start time is expected for it. Each site maintains a local queue from which it schedules jobs for execution. It is possible for different sites to use different scheduling policies. Once submitted to a local scheduler, the metascheduler has no further direct influence on the scheduling on the job, and the job cannot be moved to another site even if the load at the other site becomes lower at some time in the future.

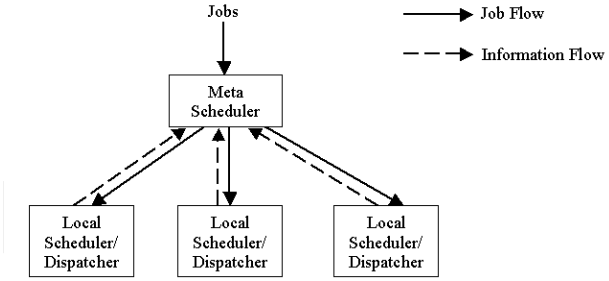


**Figure 3.** Hierarchical Metascheduler

### 2.1.3 Distributed Scheme

This scheme is similar to the hierarchical scheme except that there is a metascheduler at every site and jobs are submitted to the local metascheduler where the job originates.

The metaschedulers query each other periodically to collect instantaneous load information. If any of the other schedulers has a lower load, the job is transferred to the site with the lowest load. Since all jobs are submitted locally, the distributed scheme is more scalable than the hierarchical scheme.

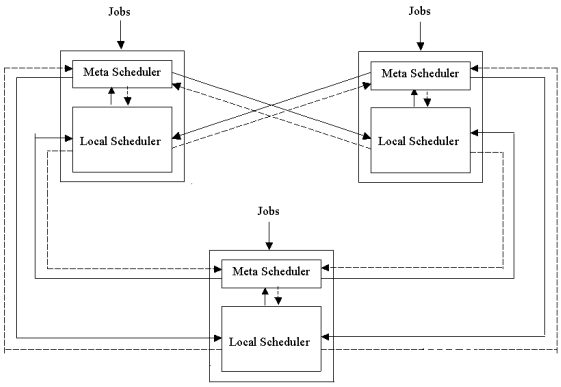


**Figure 4.** Distributed Metascheduler

## 3 Performance of Metascheduling Schemes

We evaluated the metascheduling schemes using a 5000-job contiguous subset of a year-long supercomputer workload trace from the Cornell Theory Center [4]. We considered four sites, with two of them subject to a load corresponding to the actual trace, and two of the sites encountering a heavier load. The heavier load was simulated by stretching the run-time of all jobs in the trace by a factor of 1.7. An analysis of the resource usage pattern at several supercomputer centers (San Diego Supercomputer Center, National Center for Supercomputer Applications, Cornell Theory Center, KTH Royal Institute of Technology), shows an interesting "sine wave" pattern. During evenings, the resource requested reaches and sometimes exceeds the maximum capacity of the system, while usage dips to a minimum in the very early hours of the morning. Figure 5 shows the averaged instantaneous load over the selected CTC trace, grouped by the time of day. We simulated 4 sites that were geographically distributed, with a maximum time difference of 3-hours among them. A FCFS (First-Come First-Served) policy with EASY [12] backfilling was used for all simulations.

We measured the average job turnaround time and average job slowdown, defined as the ratio of the response time of a job to its actual run time.

$$\text{Slowdown} = (\text{Wait Time} + \text{Run Time}) / \text{Run Time}$$

Figure 6 shows the average slowdown for the three metascheduling schemes described in the previous section (Centralized, Hierarchical and Distributed). The slow-

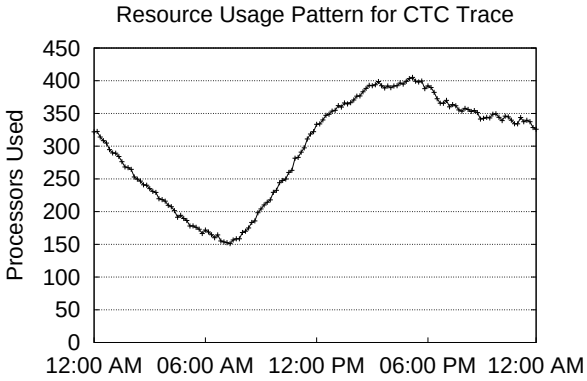


**Figure 5.** The usage pattern in the graph shows a "sinusoidal" pattern for the total number of processors requested, averaged over all 24 hour periods. Similar trends were observed for other supercomputer traces.

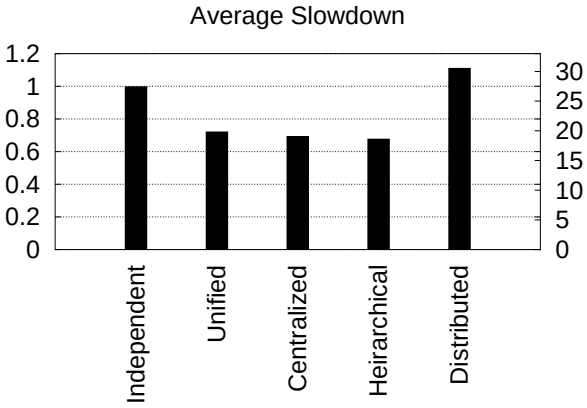


**Figure 6.** The overall slowdown of a greedy distributed scheme, which distributes requests based on instantaneous load, performs even worse than the case when the sites are not interconnected (Independent). The unified scheme represents a case where the entire computation power is assumed to be present at a single site. The values on the left end of the grid line indicate the average slowdowns normalized to the average slowdown when the sites are not interconnected. The values on the right end of grid line indicates the absolute slowdowns.

downs for the different schemes are normalized with respect to the average slowdown over all jobs, obtained by having each of the sites independently scheduled, with only their own local jobs (Independent). For comparison purposes, another simulation was performed where all the jobs streams were merged and scheduled on a single system that had as many processors as all sites combined (Unified).

Our rationale for simulating the Unified case was that its performance might serve as a reference for evaluating the performance of the distributed schemes. We were surprised to find that the Centralized and Hierarchical schemes achieved slightly lower average slowdown than the Unified case. In order to understand the reason for this, we looked at the average slowdowns of jobs in different categories, based on their length (job duration) and width (number of processors). On examining the schedules, we found that the short-wide jobs were able to backfill better on the Unified system (due to larger total number of processors) and in turn inhibited many long-narrow jobs from backfilling. In contrast, with the split systems (Centralized and Hierarchical), short-wide jobs did not easily backfill and create backfill-preventing "blockades" for long-narrow jobs. So long-narrow jobs had better slowdown with the split scenarios. Overall, although short-wide jobs did better with the Unified scheme, the poorer performance of the (larger number of) long-narrow jobs caused a lower average slowdown for the Unified scheme.

The greedy Distributed scheme also performed surprisingly poorly. On looking at the category-wise slowdowns, it was observed that very short (less than 15 minutes runtime) jobs that required very few processors (less than 8) had much higher slowdown with the Distributed scheme than either the Centralized scheme or the Hierarchical scheme. These jobs are the ones that generally get the greatest benefit from backfilling. The results in Figure 6 highlight the fact that backfill dynamics are very complex. Even if each site were to maintain complete global information, the site that seems to offer the earliest anticipated start at the time a job is submitted, may not necessarily be the one that is actually able to start the job earliest. With only summary information on system load, there is greater likelihood that the most lightly loaded system may not always be the best choice - a short-narrow job might possibly find a "hole" in the schedule of a more heavily loaded system and thus start earlier than on a less loaded system. This prompted us to evaluate a distributed scheme with multiple simultaneous requests, that we describe next.

# 4 Proposed Metascheduling Schemes

We propose a distributed scheme which requires minimal information about other sites, makes decisions based on the current global picture and adapts itself to future changes in resource usage.

## 4.1 K-Distributed Model

This scheme is similar to the distributed scheme, except that the local metascheduler distributes each job to the K least loaded sites. Each of these K sites schedules the job locally. When a job is is able to start at any of the sites, the site informs the metascheduler at the job-originating site, which in turn contacts the K-1 other metaschedulers to cancel the jobs from their respective queues. This operation must be atomic to ensure that processor cycles are not wasted by starting a job at more than one site.

By placing each job in the queue at multiple sites, the expectation is that better backfilling will be facilitated, improving system utilization and reducing average job turnaround times. However, a higher degree of "overbooking" results in an increase in the amount of work done at each local scheduler. Also, the amount of inter-site communication and synchronization will increase as K is increased. The parameter K can be varied depending upon the scalability required.

## 4.2 K-Dual Queue Model

In addition to the utilization and turnaround times being the metric used for evaluating the scheduling model, an important consideration might be that locally submitted jobs get priority over remote jobs. We define remote jobs as those jobs that are submitted to a site by the meta scheduler of some other site. One possible way to to give priority to local jobs is to specify that remote jobs are to be executed on the system only when it does not adversely affect the start times of any currently queued local jobs. Thus the remote jobs only utilize the unused processor cycles. This scheme develops on the K-distributed model and incorporates priority for local jobs into the model.

Dual queues are used at each site - one for local jobs and the other for remote jobs. When a job is submitted to the meta scheduler at a site, it distributes it to the K least loaded sites. The job is queued in the remote queue at these other sites in addition to the being queued in the local queue at the site where the job originates. Start time guarantees, if any, are given only to jobs from the local queue. Local jobs are given priority during backfilling.

# 5 Performance Evaluation

In Section 3 we showed the results for the existing metascheduling schemes. In this section we compare the results of the proposed schemes with the existing schemes.

Figure 8 and Figure 9 presents the results for K-Distributed scheme with K as 4. The K-Distributed scheme can be seen to have the best performance, with a 45% improvement in slowdown compared to the simple greedy distributed scheme. The improvement in average turnaround time is less than the improvement in average

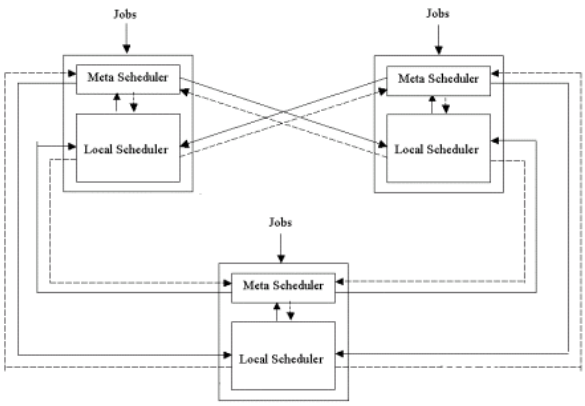


**Figure 7.** Dual Queue Scheduling There is a separate queue for local jobs and remote jobs. Jobs in the remote queue are dispatched only when they backfill without violating the reservations of the jobs in the local queue.

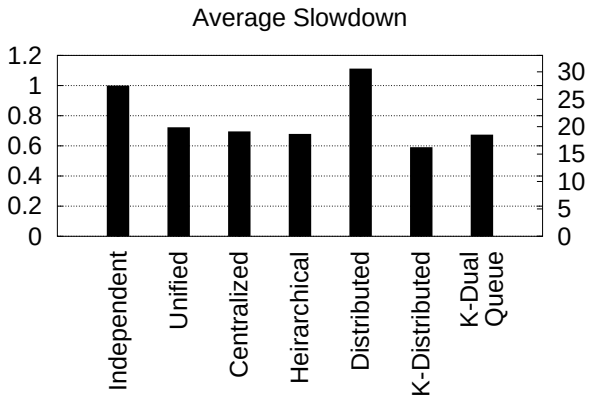


**Figure 8.** The proposed K-Distributed scheme shows a 45% improvement in slowdown compared to a greedy distributed scheme. The K-Dual Queue model gives priority to local jobs while still achieving an overall slowdown better than the greedy distributed scheme.

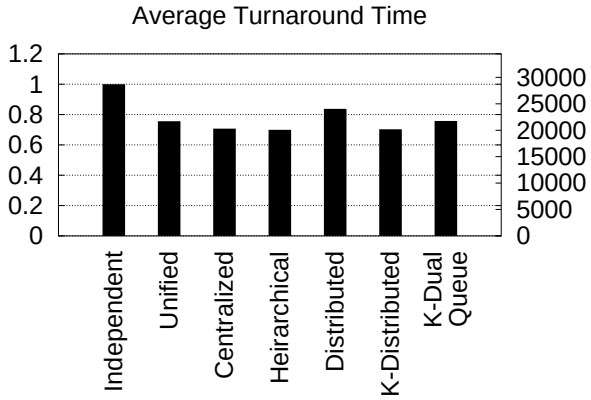


**Figure 9.** The proposed K-Distributed scheme shows a 15% improvement in overall turnaround compared to the greedy distributed scheme.

slowdown. This is because, the primary benefits are to the the shorter jobs, which benefit much more from the enhanced backfilling opportunities with the K-Distributed scheme. The number of short jobs in supercomputer traces generally far exceeds the number of long jobs, so that the average job slowdown is influenced significantly by the short jobs. In contrast, the average job turnaround time tends to be influenced much more by the long jobs, which do not get as much of a benefit from the K-Distributed scheme.

The overall performance of the K-Dual scheme is slightly worse than the K-Distributed scheme, but much better than the greedy Distributed scheme. Since its rationale was to provide priority to local jobs, we examined the performance of jobs, grouped by submission site. Since we used two lightly loaded sites and two heavily loaded sites, we summarize the results by grouping together all jobs submitted at the lightly loaded sites, and similarly grouping all jobs submitted at the heavily loaded sites. Figure 10 shows the effect of the different schemes on the slowdowns of the jobs. When the sites are independent, the jobs submitted at the lightly loaded sites do not experience external load created by load balancing and hence have significantly lower average slowdown than the jobs submitted at the heavily loaded sites. With the Distributed scheme, these lightly loaded sites are subjected to external load and the average slowdown of locally submitted jobs increases significantly. In fact, an interesting inversion happens: the average slowdown of jobs submitted at the heavily loaded sites is lower than the average slowdown of the jobs submitted at the lightly loaded sites! This would be strongly objectionable if it happened in practice. A similar trend, but of a smaller magnitude is observed with the K-Distributed scheme; the inversion still

occurs. With the K-Dual scheme, compared to the independent case, the average slowdown for the jobs submitted at the heavily loaded sites decreases significantly, and the average slowdown of jobs submitted at the lightly loaded sites increases slightly, but there is no inversion: the average slowdown of jobs from the lightly loaded sites is lower than the average slowdown of jobs from the heavily loaded sites. Thus, although the K-Dual scheme has a slightly higher overall slowdown than the K-Distributed scheme, it performs much better for the jobs originating at the lightly loaded sites than the K-Distributed scheme, and the performance of the jobs originating at the heavily loaded sites is also very good.

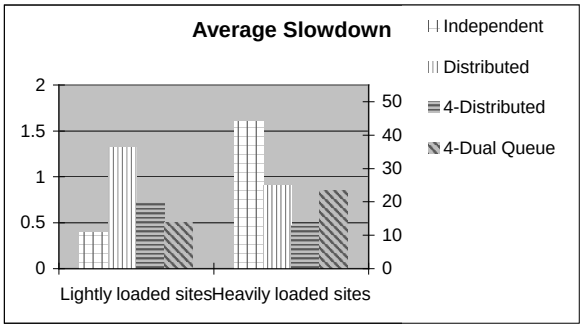


**Figure 10.** The K-Dual Queue scheme which incorporates priority for local jobs, performs the best for the lightly loaded sites.

This parameter K can be varied depending upon the scalability desired. Figure 11 represents the performance variation for different values of K. It may be seen that the greatest benefit accrues in going from K=1 to K=2. There is much less improvement in going from K=2 to K=3, and virtually no change in going from K=3 to K=4. Thus, there are diminishing returns as the value of K is increased. Since the overhead of the distributed scheduling schemes increases with K, a small value for K may be best overall.

## 6 Impact of Inaccuracies in User Estimates of Runtime

User estimates of job runtime (specified as wall clock limits for the job) can often be very inaccurate, especially for short jobs. A number of studies have attempted to evaluate the impact of the inaccuracy of user estimates on metrics such as average job slowdown and average turnaround time [12].

So it is of interest to evaluate the effect of inaccuracy of job runtime estimates on the effectiveness of the proposed distributed scheduling schemes. The simulation results of the previous sections were based on user specified estimates of job runtime. Therefore in this section we report on simulations where the actual runtime of a job was

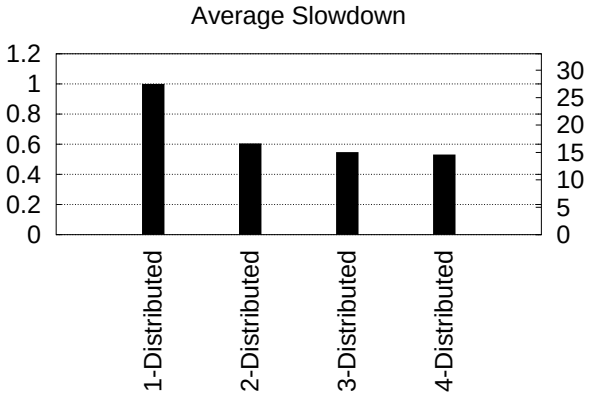


**Figure 11.** The overall slowdown for the K-Distributed scheme improves as K is increased. A similar trend is observed when comparing average turnaround times.

used also as the user-specified wall clock limit for the job, i.e. we model a scenario where the user estimates of job runtime are all perfectly accurate.

Figure 12 shows the average job slowdowns for the various schemes, with the actual slowdown being marked on the vertical axis on the right side of the chart, and the normalized slowdown (relative to the case of independent sites) shown on the left side. It can be observed from Figure 13 that the impact of using multiple requests is even more significant here than was the case in the previous section (with inaccurate estimates). With K=2, the slowdown is less than 65% of that with K=1; and with K=3, it is under 45% of that with K=1. This is a consequence of the fact that when user estimates are perfect, there are fewer perturbations to create "holes" in the schedule. In contrast, when user estimates are inaccurate, many jobs complete earlier than anticipated when they were scheduled, thereby creating gaps in the schedule, that provide opportunities for queued jobs to backfill. Therefore, since there are fewer opportunities for short jobs to backfill in a schedule, the creation of multiple simultaneous requests for a job at different sites provides a greater benefit to short jobs in terms of backfill opportunities.

In order to verify the above conjecture regarding the dynamics of backfilling, we carried out an experiment where the job traces were modified to make all jobs request an identical number of processors. The number of processors was chosen so that the total load (total number of processor-seconds for all jobs combined) was kept about the same. When all jobs have the same width, there will never be any "holes" in the schedule, i.e. there cannot be any back-filling. Figure 14 shows the relative slowdowns for different values of K when the K-distributed

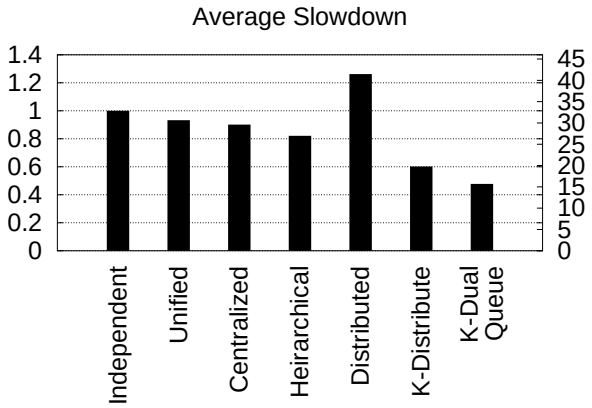


**Figure 12.** Average slowdowns for various schemes with perfect user estimates

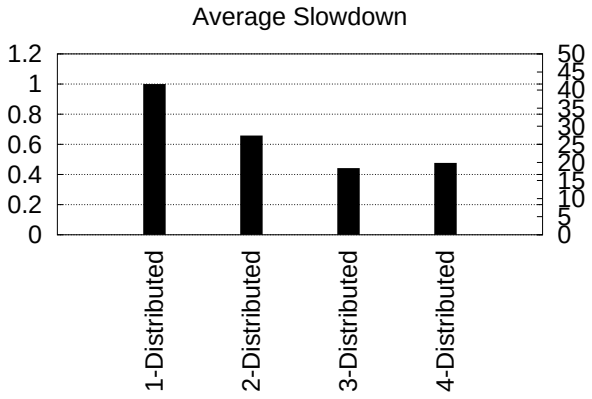


**Figure 13.** Average Slowdowns for the K-Distributed scheme with exact user estimate. The impact of use of multiple requests is even more significant here than for the case with inaccurate user estimates. With K=2, the slowdown is less than 65% of that with K=1; and with K=3, it is under 45% of that with K=1.

scheme was applied to this trace. It can be seen that indeed now the performance is virtually unchanged as K is increased. This strongly supports our belief that the unpredictable and complex backfill dynamics (when scheduling a real job trace) is the reason that the schemes with multiple simultaneous reservations outperform the simple greedy distributed scheduling scheme.

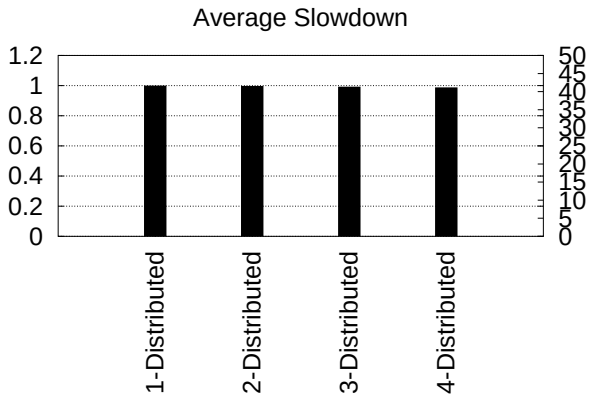


**Figure 14.** The relative slowdowns for different values of K for the equal-width job trace

## 7 Impact of Communication Overhead

We have so far assumed no overhead for job transfer. In this section, we report on simulation results that incorporate overheads for job transfer. Since the job trace did not have information about job memory requirements, we considered the memory requirement of jobs to be random and uniformly distributed between 100MB and 1GB. The results presented are based on a very conservative assumption of 10Mbps bandwidth between the sites.

From Figure 15 it can be observed that even with very conservative assumptions about the communication bandwidth available between the clusters, the K-Distributed and K-Dual schemes perform better than the Unified and Centralized schemes.

To implement these schemes, it would suffice if every cluster maintained an atomic binary semaphore associated with each job that originated at that cluster. When one of the K multiple requests starts job execution at the target cluster, the target cluster contacts the originating cluster and the originating cluster sets the semaphore associated with the job. The originating cluster also cancels the job requests at the other K-1 clusters. Every job requires a total of K messages. Although the use of the proposed schemes will impose some overhead compared to a simple greedy distributed scheme, the cost of a semaphore operation and message communication can be expected to be many orders of magnitude smaller than the average runtime of the jobs that are being scheduled through these schemes.

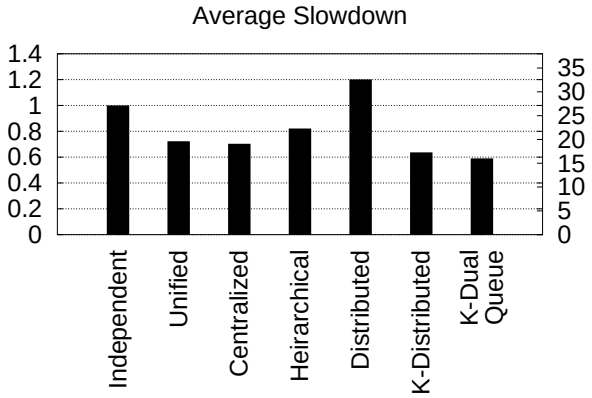


**Figure 15.** The overall average slowdown for various schemes with job transfer overhead. The K-Distributed and K-Dual schemes perform better than the Unified and Centralized schemes even with limited inter-site communication bandwidth.

## 8 Related Work

Many studies have focused on various aspects of metacomputing [1]. Considerable effort has gone into the development of systems like Globus [5, 3], Legion [9], Condor-G [6] and UNICORE [13]. These systems deal with a variety of problems such as of resource specification, discovery, allocation and security issues in a metacomputing environment involving different administrative domains. The MOL-Kernel [8] aims at providing a robust software infrastructure which can be used as a building block for a large computational grid. However there has been little work on the problem that we have addressed in this paper - of developing effective distributed scheduling schemes for the metacomputing environment.

Studies that have focused on developing scheduling algorithms for the metacomputing environment include [2, 7, 10, 11, 14]. Most of these studies present only centralized schemes. In [11] a few centralized schemes for sequential jobs were evaluated. In [7], the performance of a centralized meta scheduler was studied under different levels of information exchange between the meta scheduler and the local resource management systems. In [2], scheduling techniques were developed to efficiently deploy parameter sweep applications over the grid. The impact of advance reservations for meta jobs on the overall system performance was studied in [14]. In [10], some centralized and decentralized scheduling algorithms were proposed for metacomputing; these were reviewed in Section 2.

## 9 Conclusion

Much recent effort has been concentrated on providing middleware and software programming layers to facilitate metacomputing. We have presented a distributed scheduling model which adapts to changes in global resource usage. We also described how our scheme can be adapted to provide priority to jobs submitted locally. Our initial trace-based simulations indicate that the new algorithms might be very effective for adaptive distributed scheduling in computational grids.

## Acknowledgments

We wish to thank the anonymous referees for their helpful suggestions on improving the presentation.

## References

[1] The Grid Forum. http://www.gridforum.org.

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

[3] K. Czajkowski, I. T. Foster, N. T. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proc. JSSPP '98*, pages 62–82, 1998.

[4] D. G. Feitelson. Logs of Real Parallel Workloads from Production Systems. http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[5] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. J. Supercomputer Applications*, 11(2):115–128, Summer 1997.

[6] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S.Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proc. HPDC-10*, 2001.

[7] J. Gehring and T. Preiss. Scheduling a Metacomputer with Uncooperative Sub-schedulers. In *Proc. JSSPP '99*, pages 179–201, 1999.

[8] J. Gehring and A. Streit. Robust Resource Management for Metacomputers. In *Proc. HPDC '00*, pages 105–111, 2000.

[9] A. S. Grimshaw, W. A. Wulf, and the Legion team. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1):39–45, January 1997.

[10] 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.

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

[12] A. W. Mu'alem and D. G. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. In *Proc. 12th Intl. Parallel Processing Symposium*, pages 542–546, 1998.

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

[14] Q. Snell, M. J. Clement, D. Jackson, and C. Gregory. The Performance Impact of Advance Reservation Meta-Scheduling. In *Proc. JSSPP '00*, pages 137–153, 2000.