Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications

Kavitha Ranganathan[*] Ian Foster[*#]
[*] *Department of Computer Science, University of Chicago, Chicago, IL 60637, USA*
[#] *Math and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA*
{krangana,foster}@cs.uchicago.edu

Abstract

In high energy physics, bioinformatics, and other disciplines, we encounter applications involving numerous, loosely coupled jobs that both access and generate large data sets. So-called Data Grids seek to harness geographically distributed resources for such large-scale data-intensive problems. Yet effective scheduling in such environments is challenging, due to a need to address a variety of metrics and constraints (e.g., resource utilization, response time, global and local allocation policies) while dealing with multiple, potentially independent sources of jobs and a large number of storage, compute, and network resources.

We describe a scheduling framework that addresses these problems. Within this framework, data movement operations may be either tightly bound to job scheduling decisions or, alternatively, performed by a decoupled, asynchronous process on the basis of observed data access patterns and load. We develop a family of job scheduling and data movement (replication) algorithms and use simulation studies to evaluate various combinations. Our results suggest that while it is necessary to consider the impact of replication on the scheduling strategy, it is not always necessary to couple data movement and computation scheduling. Instead, these two activities can be addressed separately, thus significantly simplifying the design and implementation of the overall Data Grid system.

1. Introduction

A Grid is a distributed collection of computer and storage resources maintained to serve the needs of some community or *virtual organization* (VO) [18, 19]. Any of the potentially large number of authorized users within that VO has access to all or some of these resources, and is able to submit jobs to the Grid and expect responses. The choice of algorithms used to schedule jobs in such environments will depend on the target application. Our focus here is on scheduling algorithms suitable for large-scale data-intensive problems, such as those that arise in the high-energy physics (HEP) experiments currently being developed at CERN [1], which will generate petabytes of scientific data by 2006. In those experiments, a community of hundreds of physicists around the world will submit, individually and collectively, ultimately millions of jobs, each accessing some subset of that data.

Scheduling is a challenging task in this context. The data-intensive nature of individual jobs means it can be important to take data location into account when determining job placement. Replication of data from primary repositories to other locations can be an important optimization step, so as to reduce the frequency of remote data access. And the large number of jobs and resources means that centralized algorithms may be ineffective. Thus, for example, scheduling algorithms that focus only on maximizing processor utilization by mapping jobs to idle processors (disregarding costs associated with fetching remote data) are unlikely to be efficient.

To address this problem, we have defined a general and extensible scheduling framework within which we can instantiate a wide variety of scheduling algorithms, and then used simulation studies to explore the effectiveness of different algorithms within this framework.

We assume a system model in which many users submit requests for job execution from any one of a large number of sites. At each site, we place three components: an External Scheduler (ES), responsible for determining where to send jobs submitted to that site; a Local Scheduler (LS), responsible for determining the order in which jobs are executed at that particular site; and a Dataset Scheduler (DS), responsible for determining if and when to replicate data and/or delete local files. The choice of algorithms for each component defines a particular scheduling system.

Within this framework, we have defined a family of four ES and three DS algorithms, LS algorithms being widely researched in the past [4]. Our ES algorithms dispatch jobs to a random site, the least loaded site, the local site, or a site where required data already exists. Our DS algorithms perform no asynchronous replication, or alternatively, choose a random or the least loaded neighbor for replication of popular datasets (we shall use file and dataset interchangeably for the rest of the paper). In the 'no replication' case, a job

execution is preceded by a fetch of the required data, leading to a strong coupling between job scheduling and data movement. By contrast, the other two replication strategies are loosely coupled to job execution.

To study the effectiveness of these different scheduling algorithms, we have developed a modular and extensible discrete event Data Grid simulation system, ChicSim (Chicago Simulator). Our simulation results show a marked increase in Grid performance when the right combination of loosely coupled replication and scheduling policies are used. Our results also show that evaluating scheduling algorithms on their own, without considering the impact of replication techniques can lead to sub-optimal choices.

The outline of the paper is as follows. Section 2 reviews related work in the arena of grid scheduling and data placement. In Section 3, we provide details of our proposed model and in Section 4 describe the scheduling and replication algorithms that we evaluate. Simulation details and results are discussed in Section 5 and we conclude and point to future directions in Section 6.

2. Related Work

Most previous scheduling work has considered data locality/storage issues as secondary. We discuss work that has recognized the importance of data location in grid scheduling.

Thain et al. [26] describe a system that links jobs and data by binding execution and storage sites into I/O communities that reflect physical reality. As they present building blocks for such communities but do not address policy issues, this work is complementary to our work on scheduler policies. The same comment applies to work on Execution Domains [9], a framework that defines bindings between computing power and data resources in a grid such that applications are scheduled to run at CPUs that have access to required data and storage.

Casanova et al. [14] describe an adaptive scheduling algorithm for parameter sweep applications in Grid environments that takes data storage issues into account. Their approach is to place files strategically for maximum reuse. The basic difference between their work and ours is that our heuristic also actively replicates/pre-stages files. In addition, while [14] makes scheduling decisions centrally, assuming full knowledge of current loads, network conditions and topology, we concentrate on a distributed and presumably more scalable model, in which each site takes informed decisions based on its view of the Grid.

Heuristics like Max-min and Min-Min are used for Level-by-Level scheduling of DAGS by Alhusaini et al in [5]. They consider data location as a parameter but assume that resource performance characteristics are perfectly predictable. What sets our work apart from other grid scheduling research is that we consider dynamic data replication as a fundamental part of the scheduling problem.

3. System Model and Scheduling Problem

We model a Data Grid as a set of sites, each comprising a number of processors and a limited amount of storage; a set of users, each associated with a site; and a set of files, each of a specified size, initially mapped to sites according to some distribution. We assume that all processors have the same performance and that all processors at a site can access any storage at that site. Each user generates jobs according to some distribution. Each job requires that a specified set of files be available before it can execute. It then executes for a specified amount of time on a single processor, and finally generates a specified set of files.

A particular *Data Grid execution* (DGE) is defined by a sequence of job submissions, allocations, and executions along with data movements. A DGE can be characterized according to various metrics, such as elapsed time, average response time, processor utilization, network utilization, and storage utilization. The scheduling problem in a Data Grid is thus to define algorithms that will produce DGEs that are both correct and good with respect to one or more metrics.

In order to allow for a systematic description and analysis of a range of algorithms, we define a scheduling framework in which scheduling logic is encapsulated in three modules (see Figure 1):

- *External Scheduler (ES):* Users submit jobs to the External Scheduler they are associated with. The ES then decides which remote site to send the job to depending on some scheduling algorithm. It may need external information like load at a remote site or the location of a dataset, to make its decisions.
- *Local Scheduler (LS):* Once a job is assigned to run at a particular site (and sent to an incoming job queue) it is then managed by the Local Scheduler. The LS of a site decides how to schedule all jobs allocated to it, on its local resources.
- *Dataset Scheduler (DS)*: The DS at each site keeps track of the popularity of each dataset locally available. It then replicates popular datasets to remote sites depending on some algorithm. The DS may need external information like whether the data already exists at a site, load at a remote site etc. before taking a decision.

Different mappings between users and External Schedulers lead to different scenarios. For example, a one-to-one mapping between External Schedulers and users would mean each user takes scheduling decisions on their own, while a single ES in the system would mean a central scheduler to which all users submit their jobs. For our experiments we assume one ES per site. We will study other mappings in the future.

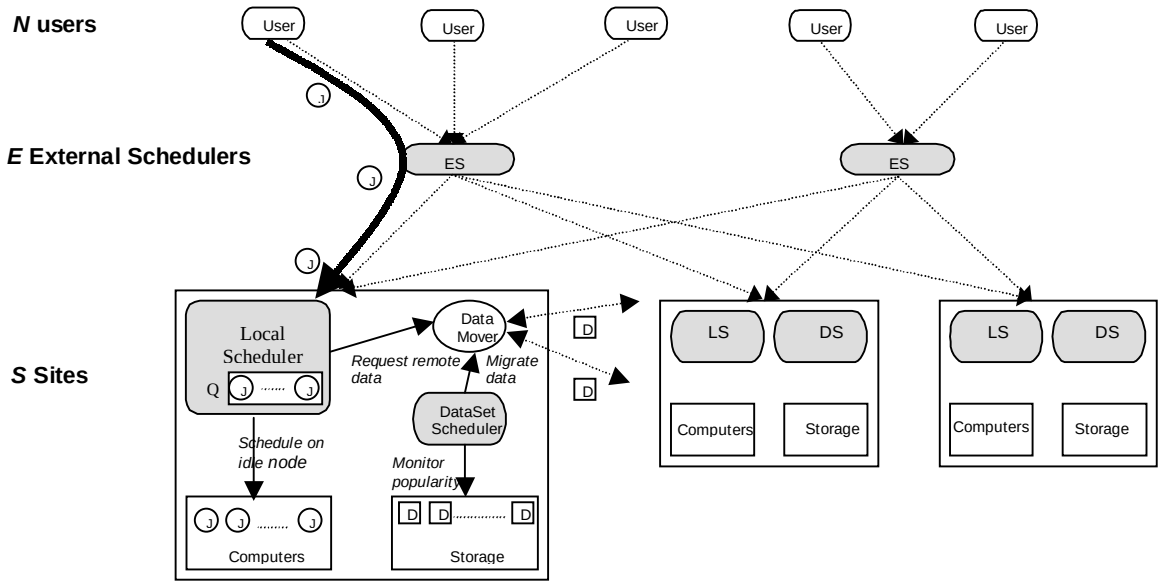


Figure 1: Interactions among Data Grid components

The external information a module needs can be obtained either from an information service (e.g., the Globus Toolkit's Monitoring and Discovery Service [15], Network Weather Service [28]) or directly from sites.

4. Scheduling and Replication Algorithms

There are two distinct functionalities we are interested in, *external scheduling* and *data replication*. For each, we define and evaluate a range of different algorithms.

Each site may have its own local scheduling policy that is implemented by the LS. Management of internal resources is a problem widely researched in the past [4] and we use FIFO (first in first out) as a simplification.

An External Scheduler selects a remote site to which to send a job, based on one of four algorithms:

- *JobRandom*: A randomly selected site.
- *JobLeastLoaded*: The site that currently has the least load. (A variety of definitions for load are possible; here we define it simply as the least number of jobs waiting to run.)
- *JobDataPresent*: A site that already has the required data. If more than one site qualifies choose the least loaded one.
- *JobLocal*: Always run jobs locally.

In each case, any data required to run a job is fetched locally before the task is run if it is not already present at the site.

For the Dataset Scheduler, we define three alternative algorithms:

- *DataDoNothing*: No active replication takes place. Datasets are pre-assigned to different sites and no dynamic replication policy is in place. Data may be fetched from a remote site for a particular job, in which case it is cached and managed using LRU. A cached dataset is then available to the grid as a replica.
- *DataRandom*: The DSM keeps track of the popularity of the datasets it contains, and when the popularity exceeds a threshold those datasets are replicated to a random site on the grid.
- *DataLeastLoaded:* The DSM chooses the least loaded site from its list of known sites (we define this as neighbors) as a new host for a popular dataset.

We thus have a total of 4x3=12 algorithms to evaluate.

5. Simulation Studies and Results

We have constructed a discrete event simulator ChicSim to evaluate the performance of different combinations of job and task scheduling algorithms. ChicSim is built on top of Parsec [3], a C-based simulation language. We describe in turn the simulation framework, experiments performed, and results.

5.1. Simulation Framework

In the absence of real traces from real data grids, we model the amount of processing power needed per unit of data, and the size of input and output datasets, on the expected values of CMS experiments [21], but otherwise generate synthetic data distributions and workloads, as we now describe.

As described in Section 3, we assume a certain number of users, sites, and datasets. (Table 1 specifies the simulation parameters used for our study.) We assume a hierarchical network topology much like that envisioned by the GriPhyN project [6]. Datasets sizes are selected randomly with a uniform distribution between 500 MB to 2 GB and with initially only one replica per dataset in the system. Users are mapped evenly across sites and submit a number of jobs in strict sequence, with each job being submitted only after the previous job submitted by that user has completed.

Each job requires a single input file and runs for $300D$ seconds, where D is the size of the input file in GB. The transfer of input files from one site to another incurs a cost corresponding to the size of the file divided by the nominal speed of the link. As job output is of negligible size as compared to input, we ignore output costs. We model network contention by keeping track of the number of simultaneous data transfers across a link and decreasing the bandwidth available for each transfer accordingly.

Table 1: Simulation parameters used in study

Total number of Users	120
Number of Sites	30
Compute Elements/Site	2-5
Total number of Datasets	200
Connectivity Bandwidth	10 MB/sec (scenario 1) 100 MB/sec (scenario 2)
Size of Workload	6000 jobs

The jobs (i.e., input file names) needed by a particular user are generated randomly according to a geometric distribution (Figure 2), with the goal of modeling situations in which a community focuses on some datasets more than others. Note that we do not attempt to model changes in dataset popularity over time.

5.2. Experiments

We ran a total of 72 simulation experiments. For each of our 4x3=12 pairs of scheduling algorithms, we ran six experiments: three with data grid parameters as above and three with network bandwidth increased by a factor of ten. Within each set of three, we ran with different random seeds in order to evaluate variance; in practice, we found no significance variation. For each experiment, we measured:

- Average amount of data transferred (bandwidth consumed) per job
- Average job completion time (max (queue time, data transfer time) + compute time)
- Average idle time for a processor

The amount of data transferred is important from the perspective of overall resource utilization, while system response time is of more concern to users. Since the data transfer needed for a job starts while the job is still in the processor queue of a site, the average job completion time includes the maximum of the queue time and transfer time.

The idle time of processors helps measure the total utilization of the system under the different algorithms. When a processor is idle, either the job queue of that site is empty or the datasets needed for the jobs in the queue are not yet available at that site.

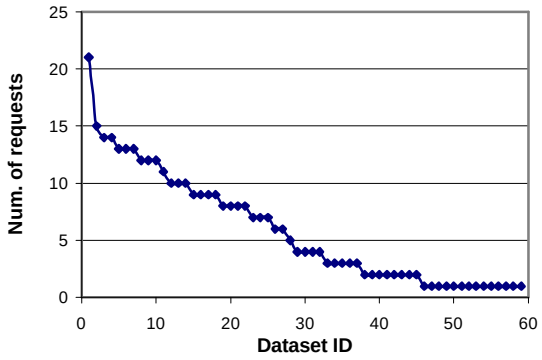


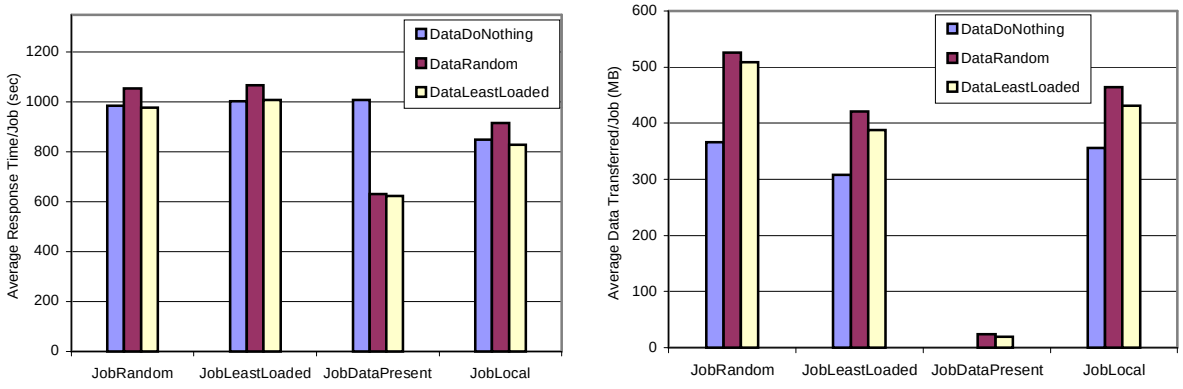
Figure 2: Dataset popularity follows a geometric distribution. Here we show the popularity of 60 datasets.

5.3. Results and Discussion

Figures 3 and 4 show the average response time, data transported and average idle time for the system parameters of Table 1 for the different combinations of the data migration and job scheduling algorithms. The results are the average over the three experiments performed for each algorithm pair.

When no replication (DataDoNothing) is used, algorithm JobLocal ("compute where the job originates") performs the best in terms of response time and algorithm JobDataPresent ("compute where the data is") performs the worst. Although data is uniformly distributed across the gird, the geometric distribution of dataset popularity causes certain sites to be overloaded when algorithm JobDataPresent is used, thus degrading its performance.

When we introduce a replication policy, algorithm JobDataPresent performs remarkably better than the other three alternatives with respect to all three metrics. It also performs much better than the best algorithm in



(a) (b)

Figure 3: Average response time (a) and average data transferred (b) for the various algorithms

the absence of replication. Clearly, dynamic replication helps to reduce hotspots created by popular data and enables load sharing. Notice that strategy JobDataPresent performs little data transfer (Figure 3b), as datasets are moved only as a result of explicit replication. Moreover, job completion times are significantly shorter for JobDataPresent + Replication (Figure 3a), as jobs are not held up waiting for required input data. Similarly, the idle time of processors is significantly smaller (Figure 4) for JobDataPresent with replication. However, replication does not have a positive effect on the other three algorithms; the response times remain the same or worsen.

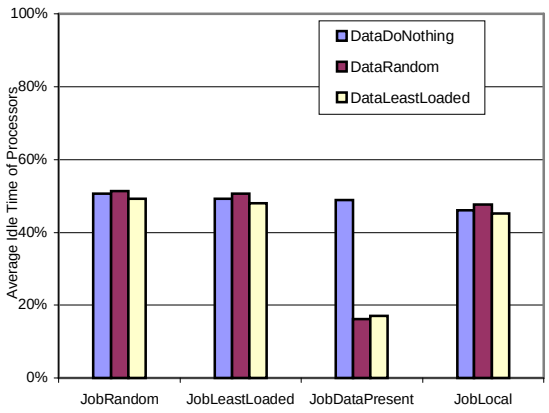


Figure 4: Percentage of time when processors are idle (not in use or waiting for data)

As Figure 3b illustrates, the difference in the average amount of data transferred between algorithm JobDataPresent and the others is very large (> 400 MB/job). Clearly, if data locality issues are not considered, even the best scheduling algorithms fall prey to data transfer bottlenecks. The point we want to bring forth via these results is the following: if the scheduling algorithms were studied by themselves (using DataDoNothing), we would conclude that algorithm JobLocal of running jobs locally was the best choice. However algorithm JobDataPresent teamed with replication works much better than any other scenario. Similarly, a replication policy that might work well by itself may not guarantee the best overall performance of the grid. Only by studying the effects of the combination of different replication and scheduling policies were we able to come up with a solution that works better than each isolated study.

Perhaps surprisingly, we found no significant performance differences between the two replication algorithms that we evaluated, DataLeastLoaded and DataRandom.

Strategy JobDataPresent along with DataRandom or DataLeastLoaded, "scheduling jobs at data sources and actively replicated popular data" turns out to be the winner as it ensures load sharing in the Grid with minimal data transfer.

5.4. Impact of Network Performance

The experiments performed with faster networks allow us to evaluate the sensitivity of our algorithms to this factor. We find, not surprisingly, that if network bandwidth is increased by a factor of ten, the performance of all algorithms that involve extensive data transfer (JobRandom, JobLeastLoaded, and JobLocal) improve dramatically (Figure 5), while strategy JobDataPresent performs more or less consistently as it does not involve a large amount of data movement. The point to be noted here is that under these new conditions, algorithm JobLocal of fetching data and executing jobs locally performs almost as well as algorithm JobDataPresent and there is no clear winner in terms of response time. Thus, while we believe that the system parameters of Table 1 are realistic for a global scientific Grid, we must be careful to evaluate the impact of future technological changes on our results.

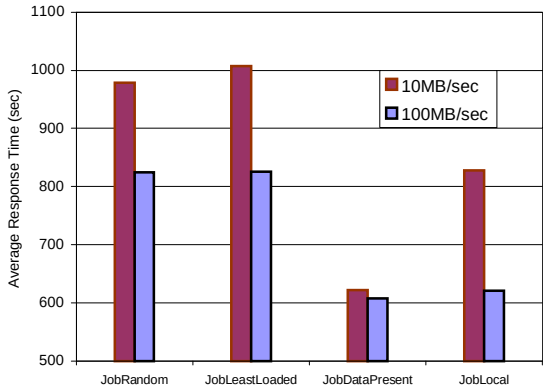


Figure 5: Response times for different bandwidth scenarios (replication algorithm DataLeastLoaded)

6. Conclusions and Future Work

We have addressed the problem of scheduling job executions and data movement operations in a distributed "Data Grid" environment, with the goal of identifying both general principles and specific algorithms that can be used to achieve good system utilization and/or response times. In support of this investigation, we have developed a modular and extensible Data Grid scheduling framework within which can be instantiated a variety of different job scheduling and data movement scheduling algorithms. We have also instantiated this framework with four different job scheduling algorithms and three different replication algorithms, and then used a Data Grid simulation system, ChicSim, to evaluate the performance of different algorithm combinations.

Our results show, first of all, that the choice of scheduling algorithm has a significant impact on system performance. Second, we find that it is important to address both job and data scheduling explicitly: for example, simply scheduling jobs to idle processors, and then moving data if required, performs significantly less well than algorithms that also consider data location when scheduling. Most interestingly, we find that we can achieve particularly good performance with an approach in which jobs are always scheduled where data is located, and a separate replication process at each site periodically generates new replicas of popular datasets. We note that this approach has significant implementation advantages when compared to (say) approaches that attempt to generate a globally optimal schedule: first, it effectively decouples job scheduling and data replication, so that these two functions can be implemented and optimized separately, and second it permits highly decentralized implementations.

These results are promising, but in interpreting their significance we have to bear in mind that they are based on synthetic workloads and simplified Grid scenarios. In future work, we will investigate more realistic scenarios (e.g., multiple input files) and real user access patterns (we are currently working on using workloads from Fermi Laboratory [2]).

We also plan to validate our simulation results on real grids, such as those being developed within the GriPhyN project [6] and by participants in the International Virtual Data Grid Laboratory [7].

Finally, we plan to explore adaptive algorithms that select algorithms dynamically depending on current Grid conditions. For example, slow links and large datasets might imply scheduling the jobs at the data source and using a replication policy similar to the ones we used for our studies. On the other hand, if the data is small and networks links are not congested, moving the data to the job source, or to a third, mutually unrelated site might be viable alternatives.

Acknowledgements

We thank Jenny Schopf for her valuable discussions and feedback, and Koen Holtman for his input on physics workloads. This research was supported by the National Science Foundation's GriPhyN project under contract ITR-0086044.

References

1. CMS: Compact Muon Solenoid: http://cmsinfo.cern.ch/Welcome.html/
2. Fermi National Accelerator Laboratory: http://www.fnal.gov
3. Parsec : Parallel Simulation Environment for Complex Systems: http://pcl.cs.ucla.edu/projects/parsec
4. Proceedings of Job Scheduling Strategies for Parallel Processing Workshop: http://www.link.springer.de/link/service/series/0558/tocs/t2221.htm
5. Alhusaini, A.H., Prasanna, V.K. and Raghavendra, C.S., A Unified Resource Scheduling Framework for Heterogeneous Computing Environments. in *8th Heterogeneous Computing Workshop*, (1999).
6. Avery, P. and Foster, I. The GriPhyN Project: Towards Petascale Virtual Data Grids, 2001.
7. Avery, P., Foster, I., Gardner, R., Newman, H. and Szalay, A. An International Virtual-Data Grid Laboratory for Data Intensive Science, 2001.
8. Basney, J., Livny, M. and Mazzanti, P., Harnessing the Capacity of Computational Grids for High Energy Physics. in *Computing in High Energy and Nuclear Physics*, (2000).
9. Basney, J., Livny, M. and Mazzanti, P. Utilizing Widely Distributed Computational Resources Efficiently with Execution Domains. *Computer Physics Communications*.
10. Berman, F., Wolski, R., Figueira, S., Schopf, J. and Shao, G., Application-Level Scheduling on Distributed Heterogeneous Networks. in *Supercomputing'96*, (Pittsburg, 1996).

11. Bestavros, A., Demand-based document dissemination to reduce traffic and balance load in distributed information systems. in *IEEE symposium on Parallel and Distributed Processing*, (San Antonio, TX, 1995), 338--345.
12. Bester, J., Foster, I., Kesselman, C., Tedesco, J. and Tuecke, S., GASS: A data movement and access service for wide area computing systems. in *Sixth Workshop on Input/Output in Parallel and Distributed Systems*, (1999).
13. Braun, T., A Taxonomy of scheduling in general-purpose distributed computing systems. in *Workshop on Advances in Parallel and Distributed Systems (APADS)*, (West Lafayette, IN, 1998).
14. Casanova, H., Obertelli, G., Berman, F. and Wolski, R., The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. in *Super Computing*, (Denver, 2000).
15. Czajkowski, K., Fitzzgerald, S., Foster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing. in *Tenth IEEE International Symposium on High Performance Distributed Computing(HPDC-10)*, (2001).
16. Fan, L., Cao, P., Almeida, J. and Broder, A., Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. in *Proceedings of ACM SIGCOMM'98*, (Vancouver, Canada, 1998).
17. Foster, I. and Kesselman, C. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputing Applications*, *11* (2). 115-128.
18. Foster, I. and Kesselman, C. (eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
19. Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, *15* (3). 200-222.
20. Hamscher, V., Schwiegelshohn, U., Streit, A. and Yahyapour, R., Evaluation of Job-Scheduling Strategies for Grid Computing. in *7th International Conference of High Performance Computing*, (Bangalore, India, 2000).
21. Holtman, K., CMS Requirements for the Grid. in *CHEP*, (Beijing, 2001).
22. M.Maheswaran, S.Ali, H.J.Siegel and D.Hensgen, Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. in *8th Heterogeneous Computing Workshop*, (1999).
23. Ranganathan, K. and Foster, I., Identifying Dynamic Replication Strategies for a High Performance Data Grid. in *International Workshop on Grid Computing*, (Denver, CO, 2001), Springer-Verlag.
24. Sih, G.C. and Lee, E.A., Dynamic-level scheduling for heterogeneous processor networks. in *Second IEEE Symposium on Parallel and Distributed Systems*, (1990).
25. Thain, D., Basney, J., Son, S.-C. and Livny, M., The Kangaroo approach to data movement on the grid. in *Tenth IEEE Symposium on High Performance Distributed Computing*, (San Francisco, 2001).
26. Thain, D., Bent, J., Arpaci-Dusseau, A., Arpaci-Dusseau, R. and Livny, M., Gathering at the Well: Creating Communities for Grid I/O. in *Supercomputing*, (Denver, CO, 2001).
27. Wolman, A., Voelker, G.M., Sharma, N., Cardwell, N., Karlin, A. and Levy, H.M., On the scale and performance of cooperative Web proxy caching. in *Proceedings of 17th ACM Symposium on Operating Systems Principles (SOPS'99)*, (Kiawah Island Resort, SC, USA, 1999), 16-31.
28. Wolski, R. Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service. in *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, Portland, Oregon, 1997.
29. Yu, P.S. and MacNair, E.A., Performance study of a collaborative method for hierarchical caching in proxy servers. in *Proceedings of 7th International World Wide Web Conference (WWW7)*, (1998).