Chapter 1

COMPUTATION SCHEDULING AND DATA REPLICATION ALGORITHMS FOR DATA GRIDS

Kavitha Ranganathan[1] and Ian Foster[1,2]

[1] *Department of Computer Science, The University of Chicago*
[2] *Mathematics and Computer Science Division, Argonne National Laboratory*

Abstract Data Grids seek to harness geographically distributed resources for large-scale data-intensive problems such as those encountered in high energy physics, bioinformatics, and other disciplines. These problems typically involve numerous, loosely coupled jobs that both access and generate large data sets. Effective scheduling in such environments is challenging, because of 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 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) [FK99, FKT01]. 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 depends 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 experiments currently being developed at CERN [CMS] that will generate petabytes of scientific data by 2006. In these experiments, a community of hundreds of physicists around the world will ultimately submit millions of jobs, with each job 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 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 define a general and extensible scheduling framework within which we can instantiate a wide variety of scheduling algorithms. We then use 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 five ES and four DS algorithms, LS algorithms being widely researched in the past [SHK95]. Our ES algorithms dispatch jobs to a random site, the least loaded site, a randomly selected less loaded site, the local site, or a site where required data already exists. Our DS algorithms perform no explicit replication (only caching), 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 chapter). In the case of no replication, 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 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, ChicagoSim (the Chicago Grid Simulator) [Chi]. In this chapter, we synthesize and extend simulation results presented in other articles [RF03, RF02].

Our simulation results show a marked improvement 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 suboptimal choices.

The outline of the chapter is as follows. Section 2 reviews relevant work in the arenas of Web caching and replication, distributed file systems, and Grid resource management. In Section 3, we briefly touch upon alternative Grid scheduling frameworks and provide details of our proposed model. Section 4 describes the scheduling and replication algorithms that we evaluate. Simulation details are discussed in Section 5 while Section 6 contains results. We conclude and point to future directions in Section 7.

2. RELATED WORK

Our work is related to two distinct areas: replication/caching on the Web and data management in distributed file systems. We discuss related work in these areas and their applicability to data management in Grids. We also talk about related research in the arena of Grid computing.

2.1 Data Management on the Web

Replication/caching of files has proved beneficial to reduce access latency, server load, and network bandwidth consumption on the Internet. Dynamic replication strategies are useful when user behavior changes over time, thus facilitating automatic creation and deletion of replicas.

Push caching, proposed by [GS95], uses access history to replicate files on remote servers. A server knows how popular its files are and so it decides when to push one of its popular files to a remote *friend* server. The network is divided into subnets, and a server has a record of how many requests for a file were generated by each subnet. Depending on the number of requests, a server calculates the optimal subnet to replicate a file. A similar strategy can be used for Grids. An entity (such as the proposed dataset scheduler in our framework) can keep track of file popularity at storage centers in the Grid and replicates popular files to less loaded data centers or to data centers near potential users, thus reducing hotspots and access latency.

Both [BC96] and [RA99] propose techniques to exploit geographical and temporal locality exhibited in Web client access patterns, by moving popular data closer to potential users. Bestavros et al. use the information available in the TCP/IP record route option is used to put together a tree of nodes. The server (host) is the root, the various other proxies (replication sites) form the nodes of the tree, and the clients form the leaves. The server keeps track of the popularity of each document and where the requests come from in the tree. It then periodically creates replicas of the files down the tree depending on a popularity function. RaDar [RA99] (Replicator and Distributor and Redirector) uses the information available in routers to move data closer to clients. When a request for data is generated from a client to a server, the route taken by the request is noted and used to generate what is known as a *preference path*. The host computes preference paths periodically from information from the system's router. If one particular node appears frequently on a file's preference path, it is considered a good choice to replicate the file at that node.

Of particular importance is the issue of whether Grid user access patterns exhibit some degree of temporal, geographical, or spatial locality. Such information can then be exploited for intelligent replication strategies, similar to those described above. Initial studies of potential Grid user logs [IR03] suggest that interesting patterns of locality do exist. Earlier [RF01] we studied different file replication strategies for Grid scenarios, with the aim of exploiting locality properties in user behavior, but we did not consider job scheduling. Here we consider the effect of using such replication strategies in combination with certain job scheduling strategies.

2.2 Distributed File Systems

Projects such as OceanStore [KBC$^+$00], Pangaea [SK01] and Freenet [CSWH00] aim to span the globe and facilitate large-scale data sharing, with an emphasis on security, reliability, availability, or anonymity. Others such as MojoNation [MOJ], Gnutella [Gnu], and Napster [Nap] focus on peer-to-peer file-sharing without much concern for security or locality. We discuss here some of the data management techniques employed by two such systems and their applicability to Grid data management.

OceanStore uses two approaches, cluster recognition and replica management, to increase the efficiency of the system. Cluster recognition involves periodically running a clustering algorithm that attempts to identify closely related files. This helps to pre-stage clusters of files that are likely to be used together. Replica management involves tracking the load created by a particular file and creating more replicas for that file on nearby nodes when access requests overwhelm it. Both methods are applicable to data management in Grid computing.

Pangaea focuses on achieving a high availability of files by massive, decentralized and optimistic replication [SK01]. Since the developers of Pangaea believe that predicting access patterns for wide-area transfers is difficult, Pangaea aggressively creates a replica of a file, whenever and wherever it is accessed. Pangaea maintains a sparely connected graph of all replicas for each file that aids in updating or deleting replicas. Replicas are deleted when the disk is out of space or a replica is inactive. To ensure a minimum number of replicas for a file, Pangaea classifies replicas as either gold or bronze and tries to maintain gold replicas on the disk for as long as possible.

Aggressive replication in Grids may be a viable option, especially if access patterns cannot be predicted accurately. Since files are replicated only when they are accessed, unneeded data transfers are avoided. Moreover, disk space is occupied only as long as there is no better use for it. Widespread replication in a Grid could, however, cause thrashing of disks. If a job was sent to a site because of the data it contained, and that data was quickly expunged (to make room for new replicas) before the job could run, the job would have to be transferred elsewhere, or the files would have to be fetched again.

2.3 Resource Management for Grids

A number of projects deal with Grid resource management. Thain et al. [TBAD$^+$01], for example, describe a system that links jobs and data by binding execution and storage sites into I/O communities that reflect physical reality. An I/O community consists of a storage device and several compute devices associated with that storage device. Jobs that run on those compute elements are encouraged to use their own community's storage device for accessing and storing data. Thus, similar applications could form a community and reduce usage of wide-area resources. Since the work of Thain et al. presents building blocks for such communities but does not address policy issues, our work on scheduling policies complements that effort.

Execution Domains [BLM00] is 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. Again, since this work is concerned with building such a system as opposed to defining scheduling policies, our work can be put to use here.

Another recent project, the DaP (Data Placement) Scheduler [DAP], aims to intelligently manage and schedule data to decrease I/O wait time and response time and increase disk usage and throughput in Grid communities. AppLeS (Application Level Scheduling) [BWF$^+$96], involves scheduling from the perspective of an application. Information such as the computation/communication ratio, memory required, and nature of application data structures is all taken into account while generating schedules. Casanova et al. [COBW00] describe an adaptive scheduling algorithm XSufferage, for parameter sweep applications in Grid environments, under the AppLeS guidelines. 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 [COBW00] makes scheduling decisions centrally, we concentrate on a decentralized and presumably more scalable model, where each site decides where and when to place its job and data.

3. ARCHITECTURES FOR SCHEDULING ON DATA GRIDS

One can envision three basic categories of scheduling architectures [HSSY00] for a distributed wide-area community: centralized, hierarchical, and decentralized. In *centralized* scheduling, all jobs, no matter where they originate, are submitted to a central scheduler for the whole Grid. The central scheduler then decides which machine to run each job on, depending on the state of different remote machines. The advantage of a centralized scheme is potentially very efficient schedules, since global state knowledge is used to generate workloads. A drawback of the centralized scheme is the bottleneck caused by all the scheduling functionality being present at one entity. As the size of the Grid grows, the central scheduler must manage more and more computing elements and thus does not scale well. Another disadvantage of a centralized scheme is the conflict in administrative domains. Local administration must give all machine handling rights to the central scheduler that decides what job to run on any machine in the Grid.

A solution to the administrative problem is to use a *hierarchical* scheme. In a hierarchical scheme, both local and global policies can be in place. Jobs are submitted to a central global scheduler, which then sends the jobs to a local scheduler present at a site. The local scheduler allocates those jobs to its machines depending on its local policy.

In the *decentralized* framework, jobs are submitted to more than one global or external schedulers. These external schedulers then select a local scheduler for each job. An advantage is that the failure of a single component does not adversely affect the whole system. Also, this scheme scales well, an important feature when we have thousands of users simultaneously submitting jobs. The disadvantage is that schedulers may have to reply on partial information or a restricted view of the Grid to make their decisions. Coordination among the external schedulers may help minimize the so-called herd behavior [Dah99] of multiple entities submitting simultaneously to a desirable site, causing it to overload.

A typical Data Grid may be composed of many different administrative domains and may ultimately serve hundreds of sites with thousands of users. Intelligent data management techniques can play a crucial role in maximizing the efficiency of Grid resources for such a Data Grid. Keeping the above factors in mind, we define a Grid scheduling framework that facilitates efficient data management, allows priority to local policies, and is decentralized with no single point of failure. We adopt a decentralized architecture, but the proposed framework could also handle the other two architectures described earlier. The scheduling logic is encapsulated in three modules (see Figure 1.1).

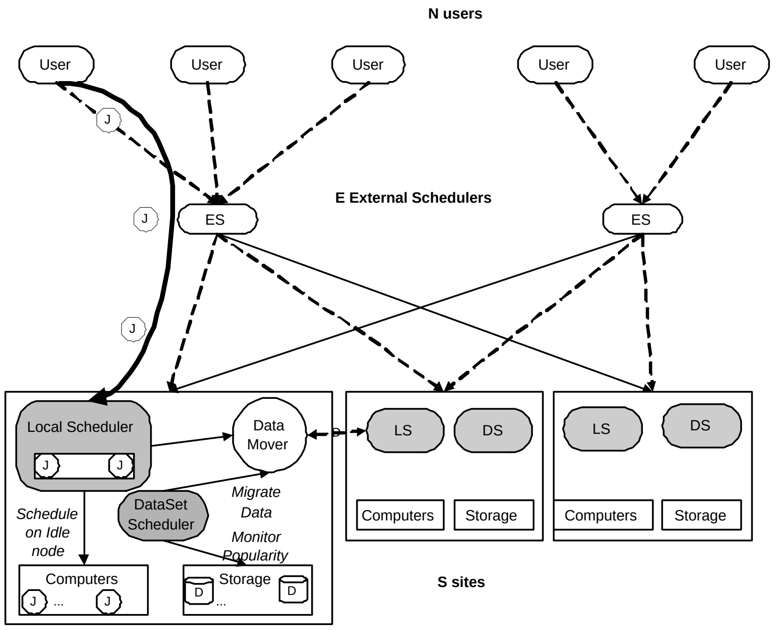

Figure 1.1. Interactions among Data Grid components.

- *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 such as the load at a remote site or the location of a dataset in order 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 its using 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 such as whether the data already exists at a site or the load at a remote site before making a decision.

Different mappings between users and external schedulers lead to different scenarios. For example, in a one-to-one mapping between external schedulers and users, the users make scheduling decisions on their own, whereas having a single ES in the system involves a central scheduler to which all users submit their jobs. For our experiments we assumed one ES per site. We plan to study other mappings in the future. The external information a module needs can be obtained either from an information service (e.g., the Globus Toolkit's Monitoring and Discovery Service [CFFK01] or the Network Weather Service [Wol97]) or directly from the sites.

4. SCHEDULING AND REPLICATION ALGORITHMS

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

Our framework allows each site to have its own local scheduling policy that is implemented by the LS. Management of internal resources is a problem that has been widely researched [FR01, SHK95]. 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 five algorithms:

- *Random*: A randomly selected site. Each site has an equal probability of getting selected, and the expected number of jobs assigned to each site is the same.

- *LeastLoaded*: The site that currently has the least load. Various definitions for load are possible; here we define it simply as the least number of jobs waiting to run, that is, the shortest job queue.

- *RandLeastLoaded*: A site randomly selected from the n least-loaded sites. This is a variation of LeastLoaded with the aim of minimizing any hotspots caused by the symmetry of actions of various sites.

- *DataPresent*: A site that already has the required data. If more than one site qualifies, the least loaded one is chosen.

- *Local*: The site where the job originated. That is, a job is always run 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 four algorithms:

- *DataCaching*: 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 a least recently used (LRU) algorithm. A cached dataset is then available to the Grid as a replica.

- *DataRandom*: The DS keeps track of the popularity of the datasets it contains (by tracking the number of jobs that have needed a particular dataset); and when the site's load exceeds a threshold, those popular datasets are replicated to a random site on the Grid.

- *DataLeastLoaded*: The DS keeps track of dataset popularity and chooses the least loaded site as a new host for a popular dataset. Again it replicates only when the load at its site exceeds a threshold.

- *DataRandLeastLoaded*: This is a variation of DataLeastLoaded where a random site is picked from the top n least loaded sites to avoid symmetry of behavior among sites.

In all cases, data is also cached, and the finite storage at each site is managed by using LRU.

We thus have a total of 5x4 = 20 algorithms to evaluate.

5. SIMULATION STUDIES

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

5.1 Simulation Framework

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.

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 [Hol01], but otherwise generate synthetic data distributions and workloads, as we now describe.

Table 1.1 specifies the simulation parameters used for our study. Dataset sizes are selected randomly with a uniform distribution between 500 MB and 2 GB and with initially only one replica per dataset in the system. Users are mapped evenly across sites and submit jobs according to a Poisson distribution with an inter-arrival time of 5 seconds. Each job requires a single input file and runs for 300D seconds (estimated job characteristics for CMS experiments), where D is the size of the input file in gigabytes. 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. Since job output is often much smaller than input we disregard output optimization for now.

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

Table 1.1. Simulation parameters used in study.

Number of sites/users	30/120
Compute elements per site	2-5
Total number of datasets	1000
Connectivity bandwidth	10 MB/sec
Size of workload	6000 jobs

5.2 Experiments

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 for a Data Grid is to define algorithms that will produce DGEs that are both correct and good with respect to one or more metrics.

We use the following metrics for our experiments:

- 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, in addition to the execution time.

The idle time of processors helps measure the total utilization of the system under the different algorithms. A processor is idle because 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.

We ran each of our 5x4 = 20 pairs of scheduling algorithms five times, with different random seeds in order to evaluate variance. In practice, we found no significant variation.

6. RESULTS AND DISCUSSION

Figures 1.2, 1.3, and 1.4 show the average response time, data transferred, and average idle time of processors for the system parameters of Table 1.1 for the different combinations of the data replication and job scheduling algorithms. The results are the average over five experiments performed for each algorithm pair. The access patterns follow a geometric distribution with α = 0.98.

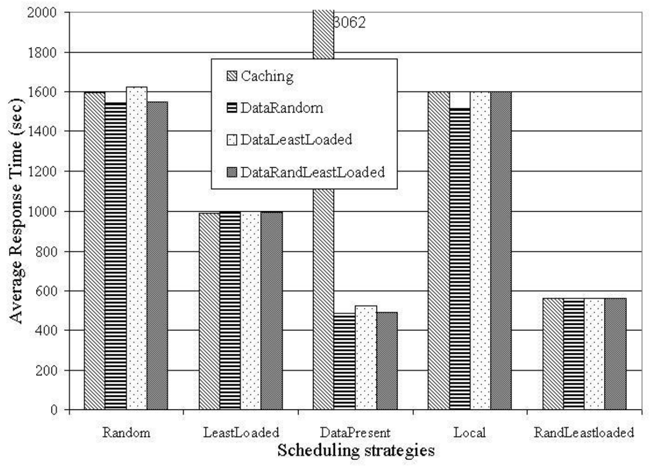

Figure 1.2. Average response time for different combinations of scheduling and replication strategies.

When plain caching (DataCaching) is used, algorithm RandLeastLoaded (send job to a randomly selected least loaded site) performs the best in terms of response time, and algorithm DataPresent (compute where the data is) performs the worst. Random and Local perform worse than the least loaded algorithms but significantly better than DataPresent. This result can be explained as follows. Although data is uniformly distributed across the Grid, the geometric distribution of dataset popularity causes certain sites to contain often-used datasets. When the algorithm DataPresent is used, these sites get more jobs than others and hence tend to get overloaded. This overloading leads to long queues at those particular sites and hence a degradation in performance. Since there is no active replication, the amount of data transferred is zero in this particular case (Figure 1.3). Also, RandLeastLoaded effectively minimizes the

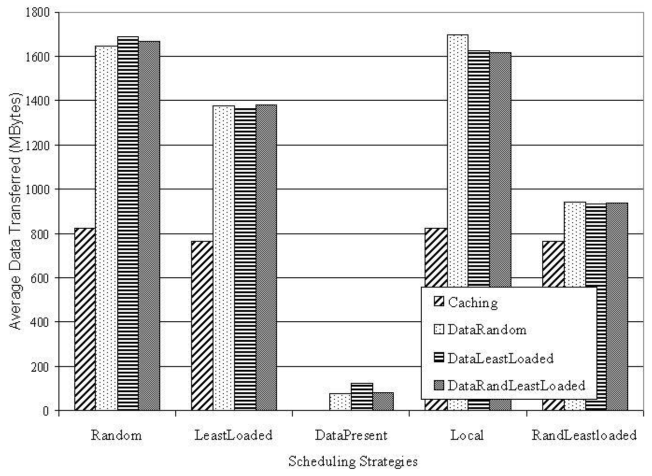

Figure 1.3. Average data transferred per job for the different scheduling and replication combinations.

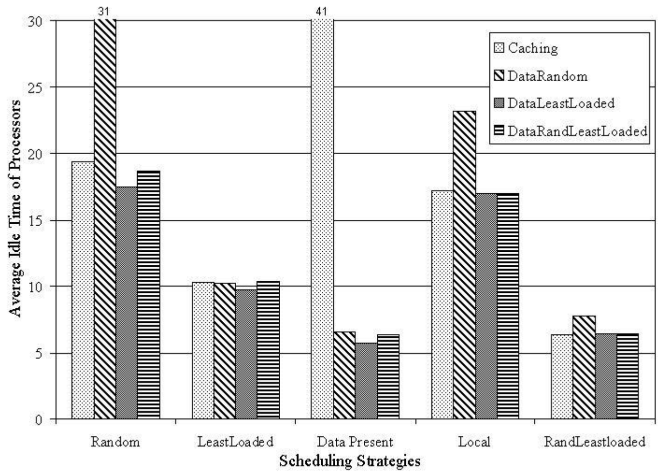

Figure 1.4. Average idle time of processors for the different strategies.

negative impact of symmetry of decisions made by different sites. This fact explains the significant improvement in response times of RandLeastLoaded as compared with LeastLoaded.

Once we introduce a replication policy, however, DataPresent performs better than all the other alternatives (Figure 1.2). It does not seem to matter which particular replication strategy is chosen, as long as one of them is employed. In terms of data transfer (Figure 1.3), the best combination is again DataPresent and DataRandom since the amount of bandwidth used in this case is almost ten times less than the other job allocation choices. Similarly, the idle time of processors is significantly smaller (Figure 1.4) for DataPresent with replication.

Clearly, in this case, dynamic replication helps reduce hotspots created by popular data and enables load sharing. The significantly better performance of strategy DataPresent when combined with any replication policy can be explained as follows.

The scheduling strategy by itself generates minimal data transfer, since jobs are scheduled to the site where the data they need is already present. Datasets are moved only as a result of explicit replication. This strategy ensures that the network does not get overloaded. Moreover, since the input data is already present at the site, jobs are not held up waiting for the required data to arrive; hence, response times are shorter. We note that once we extend our work to consider jobs that require more than one dataset, these statements may not always be true, since a job may be sent to a node that has only some of the datasets required by that job. The replication policy ensures that jobs do not accumulate at a few sites that contain popular data (by replicating popular data to other sites). Thus the computing power at a site does not cause a bottleneck. However, the replication algorithms studied do not have any significant effects on the other three scheduling algorithms.

As Figure 1.3 illustrates, the difference in the average amount of data transferred between algorithm DataPresent and the others is large. Clearly, if data locality issues are not considered, even the best scheduling algorithms fall prey to data transfer bottlenecks. These results point to the following. If the scheduling algorithms are studied by themselves (using plain caching), RandLeastLoaded appears to be the best choice. When the algorithms are studied along with the effect of replication strategies, however, we see that DataPresent works much better than any other choice. 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.

In terms of idle time, DataRandom performs much worse than the other two replication policies DataLeastLoaded, and DataRandLeastLoaded.

6.1 Effect of Bandwidth

The large amounts of data transfers that take place seem to imply that the bandwidth available to processes has a direct impact on performance. Indeed, we find that if network bandwidth is decreased from 10 MB/sec to 1 MB/sec (Figure 1.5), the performance of all algorithms that involve extensive

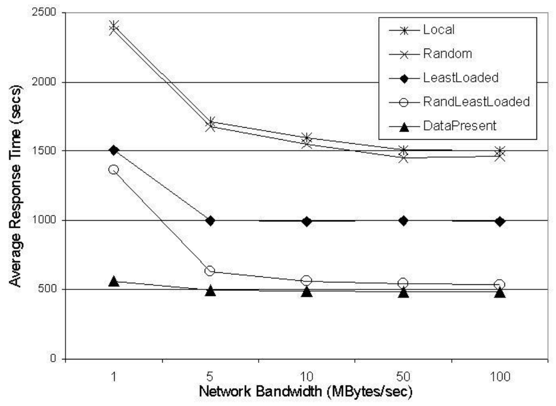

Figure 1.5. Response times of job scheduling algorithms for different bandwidth scenarios (replication algorithm used is DataRandLeastLoaded).

data transfer (Random, LeastLoaded, RandLeastLoaded, and Local) degrade sharply. DataPresent performs more or less consistently, since it does not involve a large amount of data movement. Similarly, when the network bandwidth is increased, the four algorithms with large data transfers perform much better. The point to be noted here is that under these new conditions of ample bandwidth (100 MB/sec), RandLeastLoaded performs almost as well as DataPresent. Thus, while we believe that the system parameters of Table 1.1 are realistic for a global scientific Grid, we must be careful to evaluate the impact of future technological changes on our results.

7. 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. We have instantiated this framework with five different job scheduling algorithms and four different replication algorithms and then used a Data Grid simulation system, ChicagoSim, to evaluate the performance of different algorithm combinations.

Our results are as follows. First, the choice of scheduling algorithm has a significant impact on system performance. Second, 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.

Third, and most interesting, 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 with (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 plan to investigate more realistic scenarios (e.g., multiple input files) and real user access patterns. We are currently working on using workloads from Fermi Laboratory [FNA].

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

Furthermore, 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 those 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.

Acknowledgments

We thank Jennifer Schopf and Mike Wilde for their 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

[BC96] Azer Bestavros and Carlos Cunha. Server-initiated document dissemination for the WWW. *IEEE Data Engineering Bulletin*, 19, 1996.

[BLM00] J. Basney, M. Livny, and P. Mazzanti. Harnessing the capacity of computational Grids for high energy physics. In *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP 2000)*, 2000.

[BWF$^+$96] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of SuperComputing (SC'96)*, 1996.

[CFFK01] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, August 2001.

[Chi] ChicSim: The Chicago Grid Simulator. `http://people.cs.uchicago.edu/~krangana/ChicSim.html`.

[CMS] Compact Muon Solenoid (CMS). `http://cmsinfo.cern.ch/Welcome.html/`.

[COBW00] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: User-level middleware for the Grid. In *Proceedings of SuperComputing (SC'00)*, 2000.

[CSWH00] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[Dah99] M. Dahlin. Interpreting stale load information. In *Proceedings of the Ninteenth International Conference on Distributed Computing Systems*, 1999.

[DAP] Dap scheduler. `http://www.cs.wisc.edu/condor/dap`.

[FK99] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffmann, 1999.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001. Also available from `http://www.globus.org/research/papers/anatomy.pdf`.

[FNA] Fermi National Accelerator Laboratory. `http://www.fnal.gov`.

[FR01] D.G. Feitelson and L. Rudolph, editors. *Job Scheduling Strategies for Parallel Processing (Proceedings of the Seventh International JSSPP Workshop; LNCS #2221)*. Springer Verlag, 2001.

[Gnu] Gnutella. `www.gnutellanews.com/information`.

[GRI] GriPhyN: The Grid Physics Network. `http://www.griphyn.org`.

[GS95] J. Gwertzman and M. Seltzer. The case for geographical push caching. In *Proceedings of the Fifth IEEE Workshop on Hot Topics Operating Systems (HotOS'95)*, 1995.

[Hol01] K. Holtman. CMS requirements for the Grid. In *Proceedings of International Conference on Computing in High Energy and Nuclear Physics (CHEP 2001)*, 2001.

[HSSY00] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for Grid computing. In *Proceedings of the Seventh International Conference of High Performance Computing*, 2000.

[IR03] A. Iamnitchi and M. Ripeanu. Myth and reality: Usage patterns in a large data-intensive physics project. Technical Report TR2003-4, GriPhyN, 2003.

[iVD] iVDGL: International Virtual-Data Grid Laboratory. `http://www.ivdgl.org`.

[KBC$^+$00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer,

C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, 2000.

[MOJ] MojoNation. `http://www.mojonation.net`.

[Nap] Napster. `http://www.napster.com`.

[PAR] Parsec: Parallel simulation environment for complex systems. `http://pcl.cs.ucla.edu/projects/parsec`.

[RA99] M. Rabinovich and A. Aggarwal. RaDaR: A scalable architecture for a global Web hosting service. In *Proceedings of the Eighth International World Wide Web Conference*, 1999.

[RF01] K. Ranganathan and I. Foster. Identifying dynamic replication strategies for a high-performance Data Grid. In *Proceedings of the Second International Workshop on Grid Computing (Grid2001)*, 2001.

[RF02] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data intensive applications. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, 2002.

[RF03] K. Ranganathan and I. Foster. Simulation studies of computation and data scheduling algorithms for DataGrids. *Journal of Grid Computing, to appear,* 2003.

[SHK95] B. A. Shirazi, A. R. Husson, and K. M. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, 1995.

[SK01] Y. Saito and C. Karamanolis. Autonomous and decentralized replication in the Pangaea planetary-scale file service. Technical report, HP, HPL-TR-2001-323, 2001.

[TBAD$^+$01] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Gathering at the well: Creating communities for Grid I/O. In *Proceedings of SuperComputing (SC'01)*, November 2001.

[Wol97] R. Wolski. Forecasting network performance to support dynamic scheduling using the Network Weather Service. In *Proceedings of the Sixth IEEE International Symposium on High-Performance Distributed Computing (HPDC-6)*, 1997.