RC23522 (W0502-068) February 11, 2005
Computer Science

IBM Research Report

Adaptive Techniques for Scheduling Distributed Data Intensive Applications: Experiments on a Production Grid

Kavitha Ranganathan
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Ian Foster
Department of Computer Science
University of Chicago
Chicago, IL 60637
and
Math and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439

Adaptive Techniques for Scheduling Distributed Data Intensive Applications: Experiments on a Production Grid

Kavitha Ranganathan[1] and Ian Foster [2,3]

[1] *IBM T.J.Watson Research Center, Hawthorne, NY 10523, USA*

[2]*Department of Computer Science, University of Chicago, Chicago, IL 60637, USA*

[3]*Math and Computer Science Division, Argonne National Laboratory, Argonne, IL, 60439, USA*

Abstract

Efficient job and data management in Data Grids is complicated by various factors like unreliable resources, fluctuating load and multi-administrative challenges. We have proposed an architecture for this scenario, where agents distributed across the Grid, cooperate to schedule both jobs and data with the goal of minimizing execution times, maximizing throughput, and/or minimizing data movement. We have deployed our proposed resource management architecture along with a suite of related job and data scheduling algorithms, on a 27-site wide-area Grid laboratory, Grid3. Here, we report the results of detailed experiments in this environment, using a range of scientific application workloads. We find that intelligent data scheduling is essential for certain scenarios. However, when faced with heterogeneous workloads, adaptive scheduling strategies (that can alter between data-centric and compute-centric approaches) are crucial for achieving good performance. We also discuss insights gained (1) while implementing our architecture on Grid3 and (2) by comparing our experimental results on this Grid laboratory to results obtained via simulations.

1 Introduction

Grids allow geographically dispersed computing resources to be harnessed as and when the demand arises. So called "Data-Grids" [9, 11] allow distributed users to share large amounts of data and run data-intensive applications at geographically remote compute resources.

While storage and network costs have improved, the massive amounts of data that need to be shared and harvested (petabytes in some cases) emphasize the need for Grid-wide intelligent data and storage management. Also, given the dynamic and transient nature of the Grid, it seems likely that Data Grids will require a robust and decentralized resource management infrastructure. In previous work [17, 18] we have proposed one such architecture, in which Data Schedulers at each site collect information about the usage of local data and autonomously replicate popular files across the Grid. We also conducted extensive simulation studies on a Grid simulator ChicagoSim [17], to analyze the performance of various job and data scheduling strategies under this architecture. We found that, for the application considered, data-centric scheduling was very effective.

However, since we expect various users to submit tasks to the same Grid, a scheduler will presumably be faced with scheduling a heterogeneous mix of applications. The most effective scheduling strategy may differ from application to application so even from task to task within an application.

In this paper we consider a range of scientific workloads, with the aim of identifying different regimes of applications. The goal is to identify the best scheduling strategies for a particular regime. This classification then enables us to adapt the scheduling strategy to the characteristics of the job in question.

We implemented our proposed resource management architecture on a wide-area production Grid and laboratory: the 30-site, 3000-CPU Grid3 system [23]. Using this testbed, we evaluate a suite of data-replication and job-scheduling strategies, using representative workloads from three scientific applications CMS [21], ATLAS [1] and fMRIDC [22]. The three applications vary widely in their workload characteristics (compute times for jobs, data requirements etc) and hence allow us to evaluate our algorithms under a range of realistic conditions.

We find that intelligent data scheduling coupled with a suitable job placement strategy is very successful in reducing turnaround times for certain workloads. This corroborates earlier simulation results. However, we also find that one scheduling strategy does not work well for all the workloads we considered. Moreover, in the case where a workload contains a mix of job transformations, our adaptive technique is able to dynamically switch from one strategy to another, leading to effective resource utilization.

Finally we compare experimental results obtained on this laboratory to simulation results obtained by closely modeling Grid3 characteristics on ChicagoSim. Based on this comparison, we discuss insights gained for Grid simulation and for wide-area resource management in general.

2 Grid Characteristics

Resource (data, storage, processor) management is a well-researched problem in such areas as cluster computing, web caching, and operating systems. However, a number of assumptions are no longer valid in a geographically dispersed, open collaboration like a Grid. We discuss three distinguishing characteristics of Data Grids, and how they influence requirements for resource management.

Separate administrative domains - The need for decentralized control. The sites that comprise a Grid will typically constitute distinct administrative domains. For example, Table 1 lists some of the sites in Grid3 (names denote affiliations). Each site allocates resources to jobs according to its own local policies. Hence, for sites to be willing to share their resources in a Grid collaboration, local policies should have dominance over any global policies, pointing to a decentralized approach. Thus the resource management framework should enable decentralized control.

Table 1: Location and capacity of selected Grid3 sites.

Site Name	Location (State)	Number of CPUs
BNL_ATLAS	NY	20
CalTech-Grid3	CA	12
UFlorida-PG	FL	80
IU_ATLAS_Tier2	IN	64
UBuffalo-CCR	NY	80
UCSanDiegoPG	CA	42
UFlorida-Grid3	FL	42
UM_ATLAS	MI	11
Vanderbilt	TN	12
ANL_HEP	TS	1
CalTech-PG	CA	66

Variance in resource availability - The need for techniques robust to partial and inaccurate information. A Grid resource may be shared by different users and subgroups and hence may fluctuate in its availability and capability. Figure 2 shows the availability of two Grid3 sites (in terms of number of idle CPUS) over a period of one week. As the figure shows, IU_ATLAS_Tier2 had 60 idle CPUs on one day and only around a quarter of that the next day. Hence resource allocation entities have to work with the assumption that the resource status information used for decisions will be at best an estimate and will quickly be out of date. Approaches that use current resource status or "Grid weather" to map out an overall schedule for hundreds of jobs which might run for weeks, might face a drastically different Grid environment in a matter of hours or even minutes.

Unlike traditional scheduling systems, Grid resource managers cannot assume that they have complete control over all the resources or that resources are always available.

Data transfer latencies - The need for data scheduling. Geographically distributed Grid resources may be connected by oversubscribed WANS as compared to high-speed LAN connections. In addition, many applications are expected to be data intensive [13], with input datasets in the order of Gigabytes. In these conditions, data transfer times and network latency can be significant, compelling resource management decisions to consider the amount of network traffic they generate. Thus, intelligent data management plays an important role in areas (like job scheduling) that were not traditionally concerned with data scheduling.

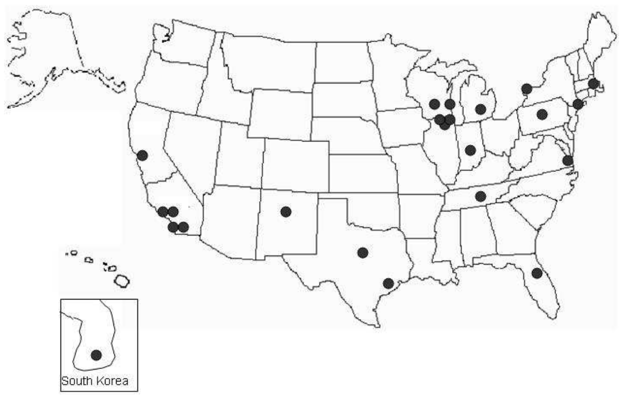


Figure 1: Locations of Grid3 Sites

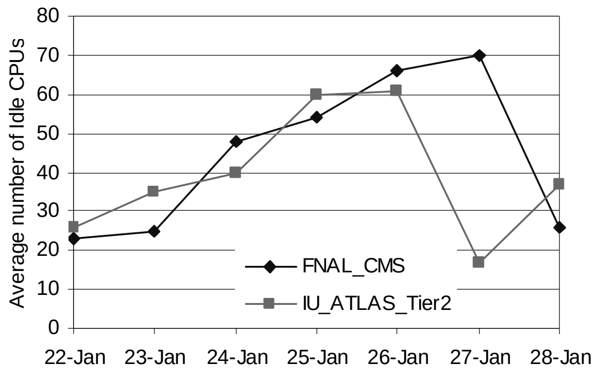


Figure 2: Fluctuations in availability of two Grid3 sites over a week.

3 Resource Management Architecture

We now briefly describe the resource management architecture that we had proposed in [17] and our implementation of this architecture across 27 sites of our testbed.

The Grid is composed of individual sites and users at each site submit execution jobs that could be run on any resource at any site. Individual users submit jobs to *External Schedulers* (ES). An ES then submits the job to the local-scheduler of a particular site in the Grid. There can be multiple ESs, each with its own job-to-site mapping algorithm(s).

One advantage of allowing for multiple ESs is that it provides for decentralized control: any administrative domain can maintain its own ES and thus control the policies used to submit jobs to different resources. A second advantage is that multiple ESs can avoid a central bottleneck and single point of failure.

Figure 3 contrasts the scenarios of multiple ES's versus one Global Scheduler. In our testbed, a Local Scheduler's policy is decided by that particular site's administration.

We evaluate various online scheduling algorithms for an ES, where jobs are dynamically dispatched to remote sites according to some criteria. The aim is to balance load across different available resources while ensuring quick response times for jobs.

We use the Euryale workload management toolkit (developed at the University of Chicago), inside which we implement our different External Scheduler functionalities. An Euryale agent runs at each submit host in the Grid, thus giving rise to multiple independent External Schedulers. Once the job-placement algorithm of the External Scheduler decides where to send a job, the job is dispatched to the appropriate remote site. The job dispatch function is performed by a combination of Condor-G job queuing [12] and Globus Toolkit GRAM secure job submission, as provided by the GriPhyN/PPDG Virtual Data Toolkit.

We have also implemented Data Managers and place one Data Manager at each site. This component manages the storage at each site and decides which files to retain locally and for how long and which files to replicate to other sites. A Data Manager is composed of three entities (Figure 4) :

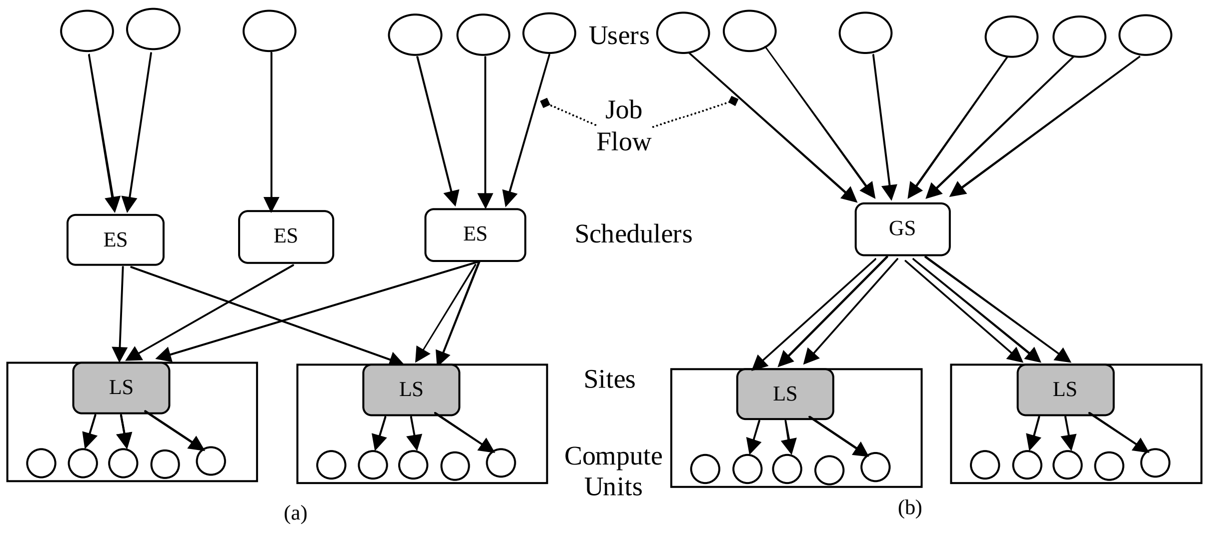


Figure 3: Grid Scheduling Architecture with (a) Multiple External Schedulers (b) One Global Scheduler

Popularity Manager: The popularity manager keeps track of the usage of each file in the data store. When a job requests a locally available file, the popularity manager updates that files usage count. To ensure that recent file accesses are given more weight than old ones, the usage count of each file is periodically decayed.

Data Scheduler (DS): The Data Scheduler uses the usage data maintained by PopMan to guide the replication of popular files to remote sites in the Grid. This strategy can prevent the creation of bottlenecks caused by multiple jobs competing for the same input file.

Cache Manager (CM): The Cache Manager also uses the data maintained by the Popularity Manager to decide which files to delete from the local storage when there is contention for space.

The External Scheduler and Data Scheduler also need Grid-wide status information for their decision making process. For our implementation we use the MonaLisa [15] site status service for load at a site, and the Replica Location Service (RLS) [8] of the Globus Toolkit [10] for file location and tracking.

4 Environment and Workloads

We discuss in turn the experimental configuration and workloads used for our experiments.

4.1 Experimental Configuration

Our wide-area testbed consisted of 11 sites (as shown in Table 1), with an aggregate of 430 CPUs. At any point in time, these resources were also being used by various other users, hence the environment was less controlled than a simulated one. Since this is a characteristic property of Grid-computing, we got the opportunity to test our architecture and algorithms in a more realistic and unpredictable setting. For our initial experiments, we used one External Scheduler, but we also plan to evaluate the effects of multiple External Schedulers submitting jobs to the same resources.

Before each experimental run, we first purge each site's cache/data-storage and then populate the caches by randomly distributing data-files for that workload to the different sites until all the caches are full. (The cache manager ensures that there are no duplicate files at any site.) Then we start the External Scheduler and the Data Managers for each site.

We measure two metrics: the total turnaround times for jobs and network bandwidth consumption. The turnaround time for a job is often a concern for the end user where as effective bandwidth utilization is more of a concern for the system administrator.

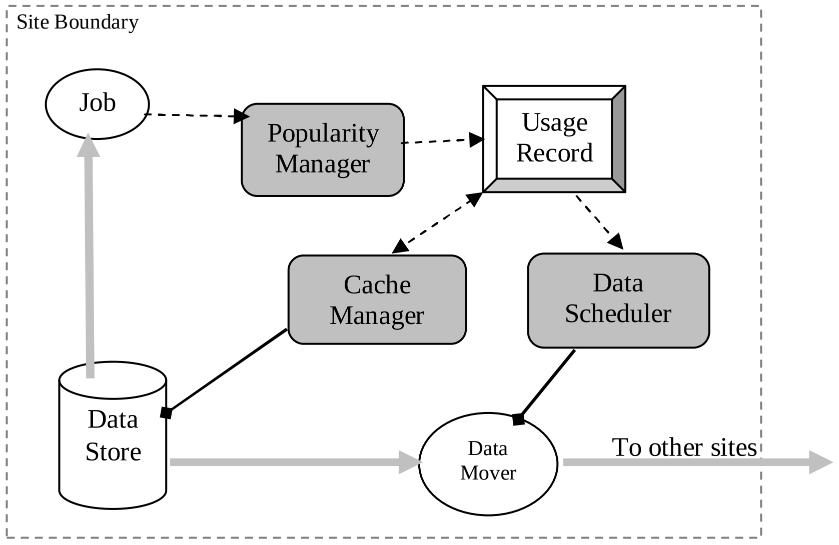


Figure 4: Components of a Data Manager

The total turnaround time is the difference between two timestamps: the first timestamp occurs when the job is submitted at the External Scheduler. The second timestamp occurs, once the job has finished executing and the message reaches the submit host where the External Scheduler runs. Hence, the total turnaround time includes latencies due to file transfers, time spent in job queues at sites, and the actual execution time.

4.2 Application Workloads

We used workloads from three scientific applications that vary significantly in their workload characteristics.

The first workload is modeled on the values expected for the CMS application [13] (i.e. the amount of processing power needed per unit of data, and the size of input datasets). CMS (Compact Muon Solenoid) is a high-energy physics experiment targeted for production by 2006.

We test two possible kinds of file-access patterns within this workload - a uniform pattern where all files are used an equal number of times and a non-uniform pattern where some files are used more often than others. We use a Zipf function [24] to model the non-uniform scenario. Each job in this workload requires one input file of 100 MB to be present at a site, before a job can start at that site. Each job runs for 300*D* seconds, where *D* is the size of the input file in gigabytes. Each workload contains 100 such jobs.

The second workload is based on the Functional Magnetic Resonance Data Centre (fMRIDC). This is a publicly accessible repository that allows members of the neuroscientific community to easily share and analyze neuroimaging data. Each job in our workload processes a unique input file of around 20 MB and runs for around 5 minutes. We consider 100 such independent jobs per workload.

The final application we use is ATLAS [1] which focuses on Monte Carlo simulations of high-energy proton-proton collisions. There are two kinds of ATLAS jobs in a workload : a long running *Simulx* (runs for around 2 hours, requires a few megabytes of data as input and generates around half a gigabyte of data) and *Reconx* which uses the output generated by Simulx and runs for a shorter duration of around 10 minutes. Again each workload contains 100 jobs each : 50 Simulx and 50 Reconx jobs.

5 Algorithms for Job and Data placement

In our architecture, the External Scheduler (ES) decides the job-to-site mapping and the Data Scheduler (DS) is in charge of dynamically identifying and replicating popular files to remote sites. We now briefly describe the various ES and DS strategies we evaluated.

ES-Random: Randomly select a site, such that the expected number of jobs assigned to each site is the same.

ES-LeastLoaded: Select a site that currently has the least load. Here we use the number of idle nodes as a measure of the load at a site.

ES-RandLeastLoaded: Randomly select a site from the top 'n' least-loaded sites. This variation of ES-LeastLoaded aims to minimize herd behavior caused by multiple entities acting independently.

ES-AtData: The job is sent to the site that contains the largest amount of input data needed for that job.

ES-Adaptive: A cost function is used to rank candidate sites according to two different criteria. It might be helpful to trade-off between moving jobs to data, data to jobs or both to a third site, depending on the job characteristics. If an application workload consists of different job types, we want the ES strategy to adapt to the particular job in question.

The following function calculates points for a candidate site S1:

Points(S1) = (c1 * Data-Present(S1) + c2 * IdleNodes(S1))   Where

Data-Present = amount of data for a job already present at the site

IdleNodes = Number of idle nodes at the site

c1 and c2 are application dependant constants. Note that depending on the values of c1 and c2 Es-Adaptive may revert to either ES-AtData or ES-LeastLoaded.

The DS maintains historical information about local file usage. When a sites *job* load exceeds a threshold, it identifies popular files to be replicated. The question then is, where should a certain popular file be replicated ? We now detail some possible placement strategies for a Data Scheduler.

DS-Caching : No purposive replication occurs, but data that is fetched for a particular job is cached for future use.

DS-Random : Randomly select a site to be the new host for a replica.

DS-LeastLoaded : Place the replica at a site that has the least job load.

DS-RandLeastLoaded : Similar to DS-LeastLoaded, except with some randomization to alleviate herd-behavior.

6 Experimental Results from Testbed

As detailed earlier, we use three workloads and we discuss results for each in turn. Note that where no DS algorithm is mentioned DS-Caching was used as the default. Sites were allowed to cache up to 20% of the total data.

6.1 CMS Application

For the CMS workloads, we discuss the results for four strategies ES-Random, ES-RandLeastLoaded, ES-AtData and ES-AtData+DS-RandLeastLoaded. ES-Local performed significantly worse than the others and ES-LeastLoaded is always outperformed by ES-RandLeastLoaded. To avoid clutter, we do not include these two strategies in the charts. Also, since there is only one kind of transformation (job) in the entire workload, we did not test ES-Adaptive which essentially can recognize different transformations and adjust accordingly.

Figure 5 contain results for the experiments with CMS workload. The grey bars are the testbed experiment results (discussed below) and the light grey bars are the simulation results (discussed in Section 7).

Figures 5c and 5d show the results for the uniform access pattern case. ES-AtData (2) helps lower the turnaround time considerably, by reducing the I/O to/from sites. On an average, when ES-RandLeastLoaded plus DS-Caching was used (4), jobs spent 23% of their turnaround time waiting for input data.

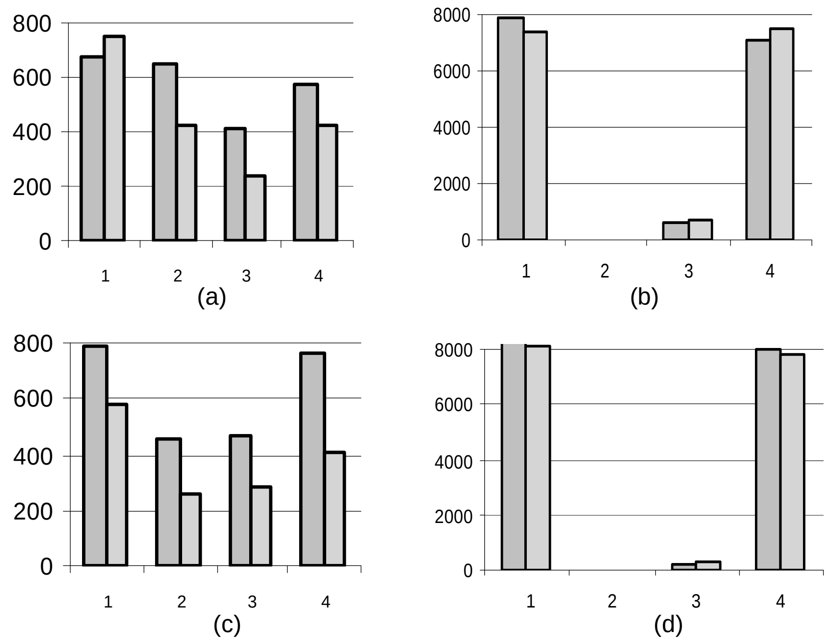


Figure 5: Testbed (dark grey) and simulation (light grey) results for **non-uniform access pattern** (a) turnaround time in seconds (b) bandwidth consumption (MB) and **uniform access pattern** (c) turnaround times (d) bandwidth consumption. (1) ES-Random (2) ES-AtData (3) ES-AtData + DS-RandLeastLoaded (4) ES-RandLeastLoaded

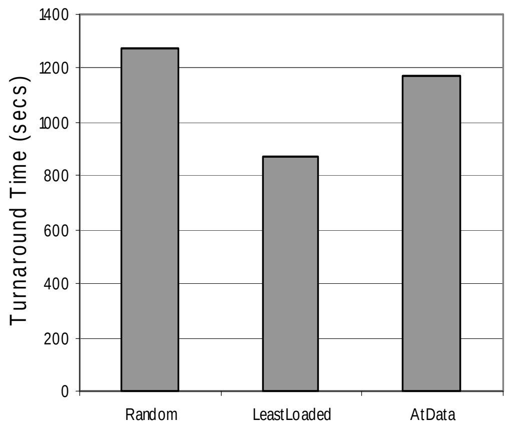
Note that in this particular case dynamic replication does not help improve the performance of ES-AtData (the average turnaround time when using ES-AtData is 459 seconds as opposed to 463 for ES-AtData + DS-RandLeastLoaded). The good performance of ES-AtData is partially because all files are equally popular in this workload. If some files are more popular than others, then ES-AtData causes some sites to get more jobs than others (see the next scenario below).

Uniform file popularity also explains why dynamic replication is not particularly helpful in this situation. The Data Scheduler is unable to identify correctly potential ‘hot’ files that can then be replicated to other sites.

The performance in the non-uniform case (zipf file-access distribution) is depicted in Figures 5a and 5b. When ES-AtData is used along with the Data Scheduler (ES-AtData + DS-RandLeastLoaded), the scenario improves drastically (with around a 35% decrease in turnover time). Not surprisingly, in terms of network bandwidth (Figure 5b) ES-AtData + DS-RandLeastLoaded consumes much less than ES-Random and ES-RandLeastLoaded, since the only files transferred are the “hot” files, triggered by dynamic replication.

6.2 fMRI Application

Next, we ran the fMRI workload repeatedly across the Grid, using different ES strategies for each run. To recall, the fMRI workload consists of a large number of short jobs (an average job spans 5 minutes), that need moderately small amounts of input data (round 20 MB). This data is already spread across the Grid at various locations.



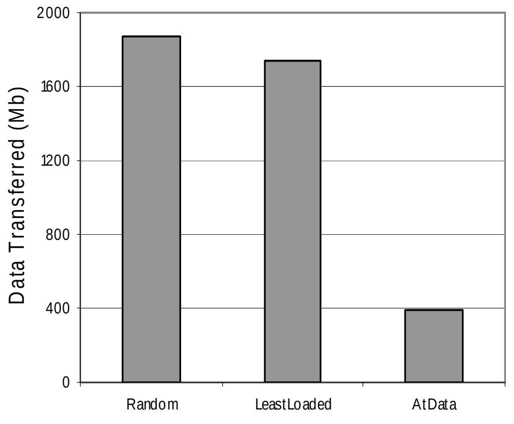


Figure 6: Turnaround times and bandwidth consumption (data transferred) for different ES strategies, when using the fMRI workload.

Since each input data is processed just once, no interesting file distribution patterns are present. Hence, the Data Scheduler does not come into play as no popular files are identified. Again, since the workload in consideration is homogeneous we did not test ES-Adaptive for this case. We tested three strategies: ES-AtData, ES-RandLeastLoaded and ES-Random. Figure 6 reports the turnaround times and network bandwidth consumption respectively.

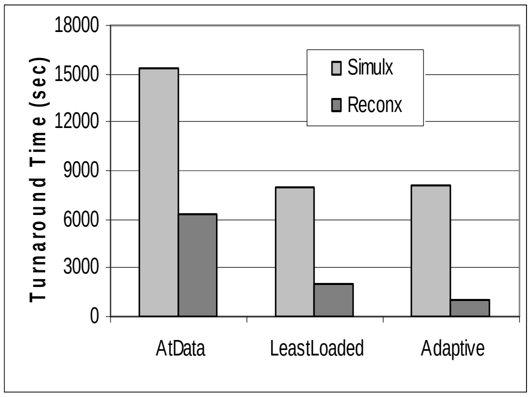
As seen in Figure 6, ES-AtData does not perform substantially better than ES-Random. Note that there is still some data traffic while using ES-AtData because some jobs get re-planned due to site failure or unavailability. Since there is only one copy of each data file on the Grid, the job then has to be moved to a site which does not contain the data, hence triggering a data transfer.

The results can be explained as follows. The input data needed for a particular fMRI job is too little to amount to significant savings by using ES-AtData. In other words the ratio of the job run time to input data size does not warrant data flow conservation. Since the data files are already scattered randomly across the different sites, the only gain in Using ES-AtData is the savings in file-transfer time. Hence ES-AtData is marginally better than ES-Random. Dispatching jobs to where they are likely to get idle CPU cycles works much better than considering the location of the input files and hence our results suggest that ES-RandLeastloaded is the best option in this case.

6.3 ATLAS Application

Next, we present results for experiments with the ATLAS workload. To recall, the ATLAS workload is interesting as it consists of two kinds of jobs, a long running *simulx* and a shorter but data-intensive *reconx*. Note that, again all files in this workload are uniformly popular, so using the Data Scheduler to replicate *hot* files does not trigger any activity. Hence, we only compare the ES strategies. Since we are faced with a non-homogeneous workload, we also tested ES-Adaptive. Figure 7 illustrates the performance of the three strategies, separated for the two kinds of jobs.

As Figure 7 shows, the turnaround time (for both kinds of jobs) is longest for ES-AtData. ES-LeastLoaded and ES-Adaptive are much better for *simulx* jobs but ES-Adaptive has the lowest turnaround time for *reconx* jobs. In terms of bandwidth utilization, ES-LeastLoaded is responsible for moving large amounts of data while the other two strategies minimize data movement.



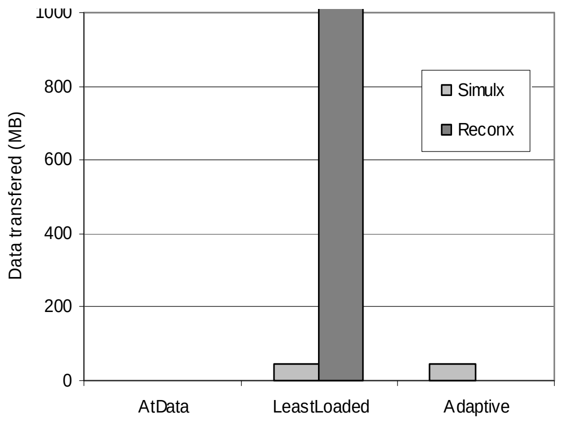


Figure 7: Turnaround times and bandwidth consumption (data transferred) for the ATLAS workload.

The behavior of the strategies can be explained as follows: As all the input data initially resides at a single site, using only ES-AtData results in all jobs being send to that particular site, resulting in very high turnaround times. Naturally, ES-AtData does not generate any network traffic. ES-LeastLoaded is more successful in distributing jobs across the Grid, but incurs high overheads in terms of the amount of data that have to be moved around for the data-intensive *reconx* jobs.

ES-Adaptive is successful in incorporating the best of both worlds. It recognizes the difference in the characteristics of the two kinds of jobs and changes the placement strategy accordingly. When a *simulx* job has to be scheduled (which is compute intensive but not data intensive) ES-Adaptive reverts to ES-LeastLoaded and send the job to a site that is more available than others. However, when it encounters a *reconx*, which is data-intensive, Adaptive reverts to ES-AtData and sends the job to where the input file for that job resides. Thus it manages to decrease both the turnaround time as well as bandwidth utilization, by being able to adapt to the kind of job (transformation) being processed.

Note that if the data had initially been distributed across the Grid, then the improvement in terms of turnaround time by using ES-LeastLoaded or ES-Adpative would still be present but less in scale.

7 Testbed and Simulation Compared

We ran the exact same CMS workloads described above on our simulator ChicagoSim [17], using similar parameters to those found in our test-bed (number of sites, storage allowed at each site, idle CPUs at each site etc). Figure 5 plots these results for two different workloads : uniform and non-uniform access patterns. As seen in the graph, the tesbed and simulator results have a high degree of correspondence (for both the turnaround times and bandwidth consumed). This suggests that ChicagoSim was reasonably successful in recreating a Grid environment using simulations.

One discrepancy that occurred in the two methodologies is the turn-around time. There is an overall consistent lag of around 2-3 minutes between the actual turn-around time and the simulated one (the simulated turn-around times being shorter). The cause turned out to be the (un)responsiveness of the local-schedulers at each site. The local schedulers in our simulations were programmed to immediately pick up a job allocated to their site, as soon as there was some availability at the site. In actuality, each scheduler periodically looks for new jobs to schedule. This creates a lag between when a job is submitted and when it is picked up for execution by the Local Scheduler, even if nodes at that site were available. Depending on the implementation of each Local Scheduler, this lag could be substantial. We then introduced a similar lag in the simulator (Figure 3) and this to an extent bridged this discrepancy.

8 Insights into Resource Management

We now discuss some insights gained in the process of conducting experiments on Grid3.

Access to local policies: Various administrative domains work with different policies regarding who can and cannot use their resources and under what circumstances certain resources may be available. The planning or scheduling entities need access to these policy constraints for effectively dispatching jobs. One problem commonly encountered in our experiments was the perceived starvation of jobs even though the site in question purported to have a large number of available compute cycles. Often, the reason was the local site scheduling policy of preferring jobs from certain categories of users. Without access to the internal policies of a site, adding certain sites to a Grid may actually be detrimental to overall job turnaround times and workload completion. Another solution would be to have an explicit agreement between the resource consumer and provider, stating the nature of resources that will be provided.

Replanning of jobs: Another crucial aspect to workload completion is replanning of jobs, given the uncertain and fluctuating participation of sites. A certain site may be unavailable or unresponsive to certain jobs for a variety of reasons: site maintenance, in which case it is unavailable to any job, network failure, local scheduling policy (mentioned above), congestion, failure of other components etc. Even though each of these occurrences may be rare in themselves, since a Grid is composed of a large number of sites and entities, a significant number of these will be unavailable/unresponsive at any given time. This leads to a large number of jobs being unable to run or complete successfully at the site they are initially sent to. We noticed instances of up to 30% of jobs being replanned in our Grid. Hence, the scheduling architecture should be able to seamlessly replan delayed or failed tasks to other sites.

Late Binding: A related issue is to enable late-binding of jobs to sites, so that the scheduling entity can look at current Grid weather before binding a job to a site. Again, dynamic and erratic resource behavior prevents early-binding approaches (for example: finding an optimal scheduler for the entire workload) from being very successful.

Lag in responsiveness of entities: While modeling distributed entities it is helpful to remember that in real life, every additional component in the chain adds a certain amount of lag time. For example, in our architecture a job travels through the Site selector, to the Condor agent at the submit host till it is picked up by the local scheduler of a site. Each of these components introduces some lag in the job-travel time

9 Related Work

We discuss a few projects that deal with data management issues in the context of Grids.

Various projects have explored job scheduling and data replication research in Grid environments. Most have used simulated environments to evaluate various approaches and solutions. Examples include SimGrid [7], GridSim [6], OptorSim [5], and ChicagoSim [18]. While simulators are helpful in providing insights and enabling experiments in futuristic scenarios, they necessarily must make assumptions about key aspects such as dynamic resource availability that may not match the conditions found in an actual Grid.

Stork [14] uses a scheduler to manage data placement activities on a Grid. The aim is to schedule, monitor and manage data movement activities automatically. The project recognizes the need to make data management a more efficient transaction in a Grid environment, and can be used in conjunction with our efforts on minimizing transfers to increase resource utility.

Thain et al. [20] propose to link jobs and data by defining I/O communities that bind execution and storage sites. Similar applications can join the same community and use the same storage sites, and their corresponding compute units, thus reducing usage of wide-area resources. Our approach is related to this idea of sending jobs to the place where the required input date exists, but additionally, if one site gets overloaded, we dynamically replicate files to another data store.

Shoshani et al. use Storage Resource Managers (SRM) [19] to manage storage at an individual site. We use a similar construct to manage storage at individual sites, except that we facilitate dynamic replication of files across sites.

The Storage Resource Broker (SRB) [16] client-server middleware provides a uniform interface for accessing heterogeneous data-stores across a Grid. Our implementation of the Data Manager (that facilitates cache management, usage tracking and dynamic replication for data-stores of different sites) is an initial prototype, the features of which could be built into a toolkit like the SRB for grid-wide use.

10 Conclusions and Future Work

We have examined the problem of scheduling jobs and data movement in the context of data-intensive workloads in large, distributed collaborative settings.

First we briefly described a decentralized and scalable resource management architecture for this scenario and described our implementation of this architecture across the Grid3 laboratory of 30 sites in the U.S. and Korea. The architecture places intelligent data-management modules at each site, which asynchronously manage and replicate data and facilitates dynamic job-placement. Next we tested a suite of data and job scheduling strategies using various scientific application workloads.

We found that no one particular scheduling strategy worked well across all applications. A data-centric approach (dynamic data scheduling combined with placing jobs at data locations) proved effective in some cases like the CMS application. But in other cases (such as the fMRI workload), there were either no interesting file-access patterns that could be exploited, or the jobs were not sufficiently data-intensive to warrant moving jobs to data. In this case, a compute-intensive approach worked better.

We also tested an adaptive strategy on a heterogeneous workload (ATLAS) which was effective in identifying varying tasks and changing the scheduling strategy from data to compute centric and vise-versa.

Our results strengthen the case for adaptive techniques that can change the scheduling strategy on the fly, depending on what kind of jobs are currently being handled. In essence this entails being able to classify tasks into different regimes and adopting the strategy that is recommended for that regime (identifying the recommended strategy for a particular application was the focus of this paper). In future work, we hope to map many more applications to these different regimes.

Note that, a particular strategy still needs fine-tuning for a particular environment (for example the constant factors in the adaptive strategy).To take this approach one step further, we could have a feedback loop that monitors how well a strategy worked for that particular regime. This would then help the External Scheduler automatically fine-tune the recommended strategy for different regimes, depending on past performance data.

We also compared results obtained on the testbed to simulations with similar parameters. There was a high degree of correspondence in the results. However, for the testbed, we noticed a significant lag between when a job is submitted to a site and when it is recognized and picked up by the Local Scheduler of that site, even if the site was completely idle.

We hope that our experiences in resource management in a realistic Grid environment and with actual applications will both help identify suitable approaches for scheduling and also help various modeling efforts to better understand such environments.

Acknowledgements

We thank Mike Wilde, Jens Voeckler, and Catalin Dumitrescu for assisting us with the use of Grid3 for our experiments. We also thank Asit Dan for his feedback and suggestions. This research was supported by the National Science Foundation's GriPhyN project under contract ITR-0086044.

References

1. Atlas: http://atlas.web.cern.ch/Atlas.
2. iVDGL : International Virtual Data Grid Laboratory www.ivdgl.org.
3. Particle Physics Data Grid www.ppdg.net.

4. Avery, P. and Foster, I. The GriPhyN Project: Towards Petascale Virtual Data Grids, 2001. www.griphyn.org, 2001.
5. Bell, W.H., Cameron, D.G., Capozza, L., Millar, A.P., Stockinger, K. and Zini, F., Simulation of Dynamic Grid Replication Strategies in OptorSim. in *3rd Int'l IEEE Workshop on Grid Computing (Grid'2002)*, (Baltimore, 2002).
6. Buyya, R. and Murshed, M. GridSim: A Toolkit for the Modelling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, *14* (13-15).
7. Casanova, H., Legrand, A. and L.Marchal, Scheduling Distributed Applications: the SImGrid Simulation Framework. in *IEEE International Symposium on Cluster Computing and the Grid (CCGrid' 03)*, (2003).
8. Chervenak, A. and al, e., A Framework for Constructing Scalable Replica Location Services. in *SC'02: High Performance Networking and Computing*, (2002).
9. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets. *Journal of Network and Computer Applications: Special Issue on Network-Based Storage Services*, *23* (3). 187-200.
10. Foster, I. and Kesselman, C. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputing Applications*, *11* (2). 115-128.
11. Foster, I. and Kesselman, C. *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.
12. Frey, J., Tannenbaum, T., Foster, I., Livny, M. and Tuecke, S., Condor-G: A Computation Management Agent for Multi-Institutional Grids. in *Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*, (San Francisco, California, 2001).
13. Holtman, K., CMS Requirements for the Grid. in *International Conference on Computing in High Energy and Nuclear Physics (CHEP2001),*, (Beijing, 2001).
14. Kosar, T. and Livny, M., Stork: Making Data Placement a First Class Citizen in the Grid. in *24th IEEE Int. Conference on Distributed Computing Systems (ICDCS2004)*, (Tokyo, Japan, 2004).
15. Legrand, I.C., Monalisa - Monitoring Agents using a Large Integrated Service Architecture. in *International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, (Tsukuba, Japan, 2003).
16. Rajasekar, A., Wan, M. and Moore, R., MySRB & SRB - Components of a Data Grid. in *The 11th International Symposium on High Performance Distributed Computing (HPDC -11)*, (Edinburgh, 2002).
17. Ranganathan, K. and Foster, I., Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. in *International Symposium of High Performance Distributed Computing*, (Edinburgh, 2002).
18. Ranganathan, K. and Foster, I. Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing*, *1* (1).
19. Shoshani, A., Sim, A. and Gu, J. Storage Resource Managers: Essential Components for the Grid. in J.Nabrzyski, J.Schopf and J.Weglarz eds. *Grid Resource Management : State of the Art and Future Trends*, 2003.
20. 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).
21. Compact Muon Solenoid – http://cmsinfo.cern.ch/welcome.html
22. fMRIDC http://www.fmridc.org
23. Foster, I. and others, The Grid2003 Production Grid: Principles and Practice. *IEEE International Symposium on High Performance Distributed Computing*, 2004, IEEE Computer Science Press.
24. Breslau, L and others. Web caching and zipf-like distributions: Evidence and Implications. In *Proceedings of IEEE Infocom Conference* (New York, NY 1999).