Planning Spatial Workflows to Optimize Grid Performance

Luiz Meyer
Federal University of Rio de Janeiro
COPPE - Computer Science
vivacqua@cos.ufrj.br

James Annis
Fermilab
Experimental Astrophysics
annis@fnal.gov

Mike Wilde
Argonne National Laboratory
MCS Division
wilde@mcs.anl.gov

Marta Mattoso
Federal University of Rio de Janeiro
COPPE - Computer Science
marta@cos.ufrj.br

Ian Foster
Argonne National Laboratory
MCS Division
foster@mcs.anl.gov

ABSTRACT

In many scientific workflows, particularly those that operate on spatially oriented data, jobs that process adjacent regions of space often reference large numbers of files in common. Such workflows, when processed using workflow planning algorithms that are unaware of the application's file reference pattern, result in a huge number of redundant file transfers between grid sites and consequently perform poorly. This work presents a generalized approach to planning spatial workflow schedules for Grid execution based on the spatial proximity of files and the spatial range of jobs. We evaluate our solution to this problem using the file access pattern of an astronomy application that performs co-addition of images from the Sloan Digital Sky Survey. We show that, in initial tests on Grids of 5 to 25 sites, our spatial clustering approach eliminates 50% to 90% of the file transfers between Grid sites relative to the next-best planning algorithms we tested that were not "spatially aware". At moderate levels of concurrent file transfer, this reduction of redundant network I/O improves the application execution time by 30% to 70%, reduces Grid network and storage overhead and is broadly applicable to a wide range of spatially-oriented problems.

Categories and Subject Descriptors

J.2 [**Computer Applications**]: Physical Sciences and Engineering – *astronomy*

General Terms

Algorithms, Performance, Design, Experimentation.

Keywords

Grid, Spatial Workflow, File Transfer, Performance.

1. INTRODUCTION

In many scientific domains, typified by astronomy, earth systems science, and geographic information systems, datasets represent the characteristics of some region of the sky, the atmosphere, or the earth. Analogous spatial dataset representation is often found in domains such as material science, computational chemistry, and fluid dynamics, where the entities being modeled have spatial characteristics. The datasets of these domains are indexed (and often named and cataloged) by the spatial coordinates of the region that the data pertains to. Spatial datasets are often processed by "gridding" the problem space into uniformly-sized partitions. Applications that process spatial data are frequently partitioned into jobs that perform processing in manageable units of resources such as file storage and CPU or RAM. Jobs are typically sized to make them small enough (in memory and duration) to be practical to run, yet large enough to reduce scheduling overhead to acceptable levels. Frequently, these jobs are chained, based on data or resource dependencies, into flow graphs, to form complex scientific workflows. Running such workflows in a Grid imposes many challenges due to the large number of jobs, file transfers, and amount of disk space needed to process them at distributed sites, and to the various tradeoffs in the scheduling of Grid resources.

The thesis of this work is that spatial workflows can be best planned for Grid execution using spatially-aware scheduling algorithms; that such algorithms are simple to develop and deploy; and that such scheduling algorithm has very general applicability to this popular and important class of application workflows. The opportunity to exploit knowledge of spatial proximity to improve the performance of a Grid workflow is manifest in many application codes used by the Sloan Digital Sky Survey (SDSS) [11], an astronomical survey project, creating a publicly accessible database of about one quarter of the sky, and measuring the characteristics of over 100M celestial objects and the distances to over one million galaxies and quasars. In such spatial problems, we are typically constrained by the following objectives. First, we want to leave files processed at sites at the end of jobs, so that adjacent jobs can use these files. But, at the same time, we want to remove files from sites as soon as they are no longer needed, as the amount of space needed to hold the input dataset poses a significant space pressure on the sites.

We describe here an algorithm to cluster jobs by their spatial proximity and to schedule these clusters on Grid sites such that the problem space is processed in an order that attempts to take maximum advantage of files already present at each site. Our results show that this algorithm, which we name SPCL (for

SAC'06, April, 23-27, 2006, Dijon, France.

"spatial clustering"), reduces the number of file transfers between grid sites and improve overall workflow execution time. Our SPCL strategy takes advantage of data locality through the use of dynamic replication and schedule jobs in a manner that reduces the number of replicas created and the number of file transfers performed when executing a workflow. We evaluated the effectiveness of SPCL via simulation studies measuring the file reference trace and performance of a complete run of the survey's southern hemisphere data, comprising 44,400 jobs with approximately 5.4 million file references. The results show that the SPCL planning algorithm, in initial simulations on Grids of 5 to 25 sites, and live tests on Grids of 4 and 10 sites, eliminates 50% to 90% of the file transfers between Grid sites relative to the next-best planning algorithms we tested that were not "spatially aware". At moderate levels of concurrent file transfer, this reduction of redundant network I/O improves the application execution time by 30% to 70%, reduces Grid network and storage overhead and is broadly applicable to a wide range of spatially-oriented problems.

The rest of this work is organized as follows. Section 2 discusses related work that deals with grid scheduling and data placement. Section 3 describes the characteristics of the SDSSS/Coadd workflow that we consider in our experiments, and the design of our clustering by access affinity algorithm. Section 4 describes the simulation framework adopted in the experiments, while in Section 5 the experimental results are analyzed. Finally, Section 6 concludes this work and points to future directions.

2. RELATED WORK

Several works in the literature address the benefits of data replication in data grid scenarios. Casanova et al. [1] propose an adaptive scheduling algorithm for parameter sweep applications where shared files are pre-staged strategically to improve reuse. Ranganathan and Foster [9, 10] evaluate a set of scheduling and replication algorithms and the impact of network bandwidth, user access patterns and data placement in the performance of job executions. The evaluation was done in a simulation environment and the results showed that scheduling jobs to locations where the input data already exists, and asynchronously replicating popular data files across the Grid, provide good results. Cameron et al. [5, 6] also measure the effects of various job scheduling and data replication strategies in a simulation environment. Their results show benefits of scheduling taking into account the workload in the computing resources. However, this work assumes that each job requires a single input file. Also, the workload is generated from different users spread over different sites. An integrated replication and scheduling strategy is proposed and evaluated by Chakrabarti et al. [2]. Like in the previous mentioned works, the workload scenario consists of a set of independent jobs, submitted from different sites at different times. Data replication is performed asynchronously but jobs can access multiple files. The effect of transferring input/output data on overall grid performance is studied by Shan et al. [12]. However, the work assumes that all input files are present at the host where the job is submitted and files are not reused by different jobs. Mohamed and Epema [8] propose an algorithm to place jobs on clusters close to the site where the input files reside. The work assumes that the jobs require the same single input file which can be replicated. Singh et al. analyze how clustering jobs can improve workflow execution [13]. In their work, the authors discuss different configurations of the workflow engine and their impact on performance results. Our work differs by implementing a clustering approach with the goal to improve performance through the reduction of redundant file transfers while scheduling work in a manner that matches the sites storage capacity. Files are accessed by jobs belonging to a single workflow. Thus, when deciding to replicate files we do not take into account accesses by other users. Files are temporarily replicated because they will be used by other jobs in the context of the same workflow. Although some files have more accesses than others, we do not proactively replicate them. The selection of a site to run a job is done based on an expectation that subsets of the required files will be in place. These replicas may be deleted once they are no longer required by a group of jobs. This deletion is done because it may not be feasible to retain these replicas during the execution of the whole workflow. We assume that we have advance knowledge of the entire workflow for a large-scale operation such as the co-addition of an entire sky region, and that we know the spatial coordinates of every job as an n-tuple in the coordinate space of the application. Further, unlike the approach taken in [10] we do not assume that specific files will grow in average popularity over the lifetime of the run, but rather that such bursts in file popularity will be highly localized, transient, and due to file reference overlaps.

3. CLUSTERING BY SPATIAL PROXIMITY

Clustering of jobs and files can be computed in several ways. Given a workflow definition, its set of jobs, and its files, dependencies can be extracted and passed to a clustering algorithm that can apply heuristics to produce the clustering definition for jobs and files. In cases where the workflow is formed by several pipelines this definition is simple and intuitive, i.e. clustering together all jobs that are pipelined in the workflow. If a job that reads a file is placed at the same site of the job that produced the file, no file transfer is needed. In spatially-oriented applications, the ability for a data-flow-driven algorithm to automatically detect file-reference clusters is more challenging, because it must discover how jobs are related to each other based on their file access pattern. This process can be computationally intensive, and does not take advantage of the easier-to-process information present in the correlated coordinate systems of the jobs and the members of the dataset. The main goal of the SPCL clustering algorithm is to reduce the number of file transfers between grid sites. The idea is to create clusters of jobs by spatial proximity, assign each job to a cluster, each cluster to a grid site and during the execution of the workflow, schedule all jobs belonging to a cluster to the same site. The SDSS co-addition workflow consists of 44,400 jobs that operate on 2.5 Terabytes of input data stored in 544,500 distinct files, performing 5,419,370 file references. On average, each file is referenced by approximately 10.1 different jobs. The portions of the SDSS dataset processed by co-add consists of image data, and are cataloged at a coordinate grid consisting of three dimensions: right ascension (ra), declination (dec), and spectral filter. For simplicity, we treat ra, dec, and filter as if they were coordinates (X, Y, Z) in a three-dimensional space. Each job operates on a region centered at a specific (ra,dec,filter) value, and processes files related to adjacent (ra,dec) grid points. The SPCL algorithm for clustering jobs and files together was trivial, performed by a

simple division of the overall three-space into approximately uniformly sized partitions. The z or filter dimension was treated as a dependent variable of the ra,dec: within each cluster, all the filter values for jobs and data files at a given ra,dec coordinate were placed in the same cluster. Most files are referenced by jobs in three declination grid points and eight ra grid points. Thus we generated clusters with these dimensions and assigned each job to a cluster based on the value of both attributes. As result, 372 clusters were created: 368 cluster with 120 jobs each, and 4 with 60 jobs each. Since the number of generated clusters is greater than the number of grid sites, jobs from different clusters have to be assigned to run at the same site. To guarantee that files that are shared by jobs belonging to the same cluster will be in cache, the SPCL algorithm does not schedule jobs from different clusters to run at same time at the same site. This is accomplished by creating dependencies among jobs from different clusters. Figure 1 illustrates SPCL scheduling approach. Suppose that clusters i and j are assigned to run at site k. In order to maximize file reuse by jobs belonging to the same cluster, jobs from cluster i must be scheduled before jobs from cluster j. This is achieved by creating a "parent-child" relationship between the jobs of the workflow.

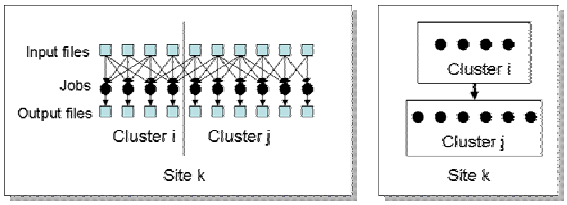

Figure 1. The SPCL scheduling approach.

4. THE SIMULATION FRAMEWORK

We model the Grid environment as a set of sites, each having a fixed number of processors and a fixed amount of total disk storage space. We assume that at each site all processors have the same performance and have access to all disk files stored at that site. At a separate submit host, external to the Grid, a workflow manager controls the scheduling of the jobs to the sites. A replica location service (RLS) keeps track of the location of file replicas spread over the Grid.

Each file has a unique filename and is characterized by its size. Each job is characterized by a unique job identifier and the list of files it needs to process. In order to execute, a job needs the set of input files to be available at the site that was selected for its execution. In our studies up to this point, we disregard the transfer back of output results, since in the specific application that motivated this work the transfer of output data was a negligible portion of the overall execution time and load on the Grid. As in Condor/DAGman [4], a “prescript” and a “postscript” step is associated with each workflow job. The prescript and postscript steps are responsible for transferring input files and deleting output files, respectively. The number of prescript that can be started concurrently (across all jobs) by the submit host is controlled by the DAGman parameter MAXPRE, which serves as a convenient workflow-wide throttle on the data transfer load that the workflow manager can impose on the Grid from the submit host. Figure 2 depicts the Grid environment modeled by our simulator.

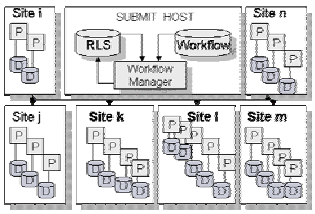

Figure 2. Grid environment.

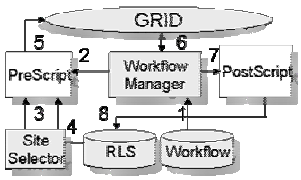

Figure 3. Submit host.

We modeled the behavior of this Grid using discrete event simulation coded in Perl and tailored its algorithms and data structures to the needs of the Grid elements in our simulation domain. Figure 3 details the architecture of the submit host in the simulation environment. The execution of a job is simulated in the following stages, while a simulation clock keeps track of elapsed time and drives the event loop: First, the workflow manager takes the identification of the job and its list of required files and starts the input file transfer processing for that job. The prescript processing is responsible for two tasks: deciding where to run the job and transferring needed input files to that site. The decision on where to run a job is based on a selectable scheduling algorithm while the transfer task is performed based on the information from a replica location service (RLS). The job is then sent to a site, waits in a local scheduler queue for a free CPU, and then runs during a specified amount of time. After job execution, the workflow manager starts the postscript processing that is responsible for registering or deleting the new replicas in the RLS, depending on the replica policy being enforced.

5. EXPERIMENTS

We modeled our entire test Grid on a large subset (25) of the 30+ sites of Grid3 [7], using the site parameters listed on table 1. The column S indicates the identification of the site, and the columns P and D show the number of processors and disk available at each site. We assume that we have a target of obtaining 10% of the resources in each site to define our Grid simulation environment. For CPU allocation, we assume an approach where fixed-size pools of CPUs are held dedicated at each site for the duration of the workflow, in a manner similar to that implemented by the Condor “Glide-In” [3] facility. Each simulated test Grid of size N (N=5, 10, 15, 20, 25) consisted of the first N sites listed on this table. Each input file is approximately 5 Megabytes in size, is initially located at one site (as in practice) and takes five seconds to be transferred. Each job runs for 15 minutes. The workflow manager starts the prescript process for each job with an interval of one second. When the number of prescripts running reaches the maximum number of prescripts allowed, the workflow manager has to wait for the end of the first prescript already running in order to start a new prescript. In the same way, when a job is submitted, if there are no available processors at the execution site then the job has to wait in the remote queue. The total time to run a job is computed by summing the time in the queue at the submit host, the time to transfer the necessary input files, the time in a queue at the execution site, and the time to run the job. We analyze, in our experiments, both (i) the total number of file transfers and (ii) the total workflow execution time. The number of file transfers is important from the perspective of overall resource use and also for its impact in performance. The overall completion time is important for users that want to run their workloads in a short period of time.

Table 1. Simulation parameters for the Grid sites

S	Grid		Sim		S	Grid		Sim	
-	P	D	P	D	-	P	D	P	D
1	720	958	72	95	14	76	1190	8	119
2	516	865	51	86	15	64	1723	7	172
3	400	200	40	20	16	60	569	6	57
4	350	2170	35	217	17	55	1190	6	119
5	312	1627	31	162	18	42	243	4	24
6	296	1130	29	113	19	42	1712	4	172
7	272	647	27	65	20	40	109	4	11
8	158	1365	16	139	21	40	53	4	6
9	136	591	13	59	22	32	976	3	98
10	100	976	10	97	23	20	1311	2	131
11	82	1712	8	171	24	19	55	2	5
12	80	2172	8	217	25	12	1294	2	130
13	80	879	8	88					

5.1 Execution Strategies

We compare the results according to two external scheduling algorithms combined with two replication policies. We assume that at each site there is a local scheduling policy which in our study is set to FIFO (first in first out), and a file deletion policy which is set to LRU. Thus, the workflow can be executed based on one of the four execution/replication strategies:

Random. The execution site is selected randomly. Input files are replicated at the execution site but do not become available to other jobs. Once a job finishes, the transferred input files are deleted.

Clustered. The execution site is selected based on the job identification. Each job is assigned to a cluster as a consequence of the cluster design and each cluster is assigned to a site. The number of clusters per site is acquired by dividing the number of clusters by the number of sites. Input files are replicated at the execution site.

RandomCaching. The execution site is selected randomly. Input files are replicated at the execution site and become available for other jobs.

DataPresent. Jobs are sent to a site with the most files that they need. If more than one site qualifies then a random one is chosen.

5.2 Results

As can be observed in figure 4, executing the Coadd workflow according to strategies 1 and 3 is extremely expensive in terms of file transfers, requiring almost 5.5 million transfers. The reuse of input files with *RandomCaching* results in only a minor reduction in the number of transfers. The *clustering* approach can reduce the number of file transfers to approximately 550,000 when running with the first five and ten sites.

The assignment of clusters to sites did not take into account the resources available per site. Thus, when we ran the workflow with 15, 20 and 25 sites, sites with minimal storage availability were added to the pool of running sites causing the number of file deletions to increase and consequently increasing the number of transfers. The *DataPresent* strategy also caches data and takes advantage of the total amount of storage available, showing significant reduction on the number of file transfers when running with more sites in the pool.

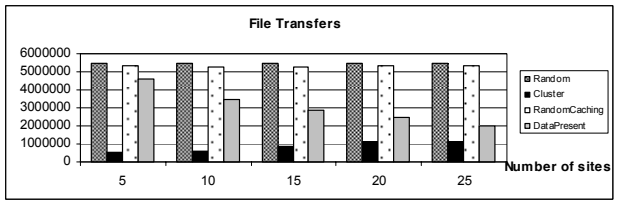

Figure 4. Total number of file transfers by strategy

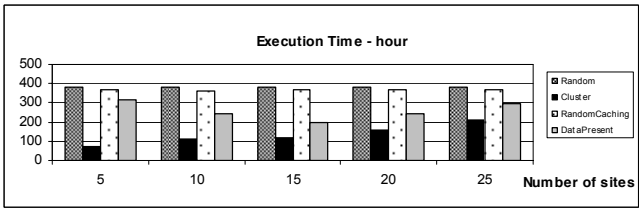

Figure 5. Execution time for MAXPRE=20

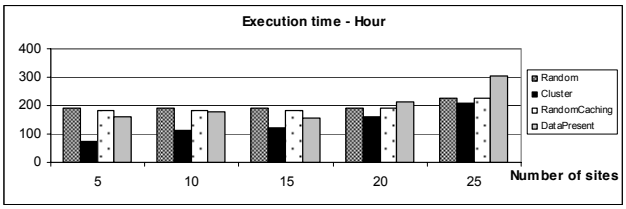

Figure 6. Execution time for MAXPRE=40

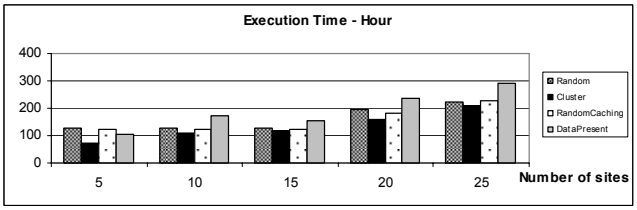

Figure 7. Execution time for MAXPRE=60

Figures 5, 6 and 7 show the performance of the four strategies when running the simulator with 20, 40 and 60 prescript jobs respectively. The figures show similar results for the *Clustering* strategy with 20, 40 and 60 prescript jobs. *Clustering* allows file reuse, reduces the number of file transfers and consequently the time to run the prescript job. Transferring fewer files for a job can bring performance benefits if, by the time the transfers end, there are available processors to run the job. Increasing the number of prescripts in this strategy causes more jobs to be ready to run, but they have to wait for free processors because the time to run the job is higher than the time to perform the few file transfers. Also, it can be observed that the performance decreases for the *Clustering* strategy when the number of sites grows. This result is a consequence of adding sites with less number of processors in the execution pool. Since clusters are assigned to sites not considering the resources available, sites with few processors receive the same load as sites with more resources causing jobs to wait for free processors. Increasing the number of prescript jobs

enhances the performance for *Random* and *RandomCaching*. For these two strategies, the cost to transfer more files is reflected in the performance when the number of prescript is 20. When more file transfers are performed in parallel, the execution time for both strategies improves. However, having a high number of prescript jobs running may not be feasible if jobs perform large number of file transfers. The simultaneous number of hits over the gridftp server can cause problems to this service. Increasing the number of sites in the pool does not show better execution times for neither strategy. With MAXPRE set to 20, the *Datapresent* strategy presents improvements until the number of sites reaches 15. After this point, the inclusion of sites does not bring benefits for any number prescripts. Increasing the number of prescripts to 60 only achieves improvement when the number of sites is five. Finally, we have just begun the validation of these simulation results on live Grid3 resources, modeling a run-time simulation of glide-in execution by simply running a "mock" job on the submit host, with accurately simulated queuing. Thus, the live-Grid runs represent actual data transfer, and a closely-simulated run time for glide-in CPU allocation, but without consuming actual CPU resources from the production Grid. Two live comparison runs have been completed. In the first, a 5% sample of the full 44K job coadd workflow was run on 4 Grid sites, using full file sizes and job durations. This run was performed for two of the four algorithms, SPCL and Random. Simulation results show run completion time of 22.83 hours for the Random algorithm and 18.80 hours for SPCL; measured results were about 15% longer. In a second test, we compared two smaller runs across a larger Grid of 10 sites. This scaled-down test used 1000 jobs of 3 minutes each, and a data file size of 1MB. Simulation results yielded 78 minutes for the *Random* algorithm and 37 minutes for SPCL; measured results were 1.3 times longer. While highly preliminary, these initial live Grid tests offer encouraging indications that SPCL holds promising potential for speedup of spatial data processing.

6. CONCLUSION AND FUTURE WORK

We presented a job and file clustering approach to execute workflows in Grid environments. We evaluated the proposed solution with a real file access pattern in a simulation environment. The results show that clustering jobs by affinity of files reduces total number of file transfers. This reduction benefits the Grid as a whole by reducing traffic between the sites. It also benefits the application by improving its performance. We believe that the same approach can be adopted by other scientific applications that present the same characteristics of spatial processing. The good results encourage us to plan the validation of the clustering approach in a real Grid. We also intend to integrate this approach with other site selections algorithms that take into account the resources availabilities in the sites in order to schedule jobs. Furthermore, we plan to apply job clustering techniques in other workflows patterns. For example, a pattern that is frequently observed in scientific workflows is the presence of several independent pipelines where jobs communicate with the preceding and succeeding jobs via data files. We intend to exploit the idea of clustering using DAG transformation by grouping together the jobs that are part of the pipeline portions of the workflow. Such transformation can reduce the number of file transfers and also the total number of job submissions.

7. ACKNOWLEDGEMENTS

Our thanks to CAPES and CNPq Brazilian funding agencies.

8. REFERENCES

[1] Casanova, H., Obertelli, G., Berman, F., Wolski, R., The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid, in SC'2000, Denver, USA, 2000.

[2] Chakrabarti, A., Dheepak, R.A., Sengupta, S., Integration of Scheduling and Replication in Data Grids, Proceedings of 11th International Conference on High Performance Computing, Bangalore, India, December 2004.

[3] Condor Project, http://www.cs.wisc.edu/condor

[4] DagMan, http://www.cs.wisc.edu/condor/dagman/

[5] Cameron, R. Carvajal-Schiaffino, A.P.Millar, Nicholson C., Stockinger K., Zini, F., Evaluating Scheduling and Replica Optimization Strategies in OptorSim, in Proc. of 4th International Workshop on Grid Computing (Grid2003). Phoenix, USA, November 2003.

[6] Cameron, R. Carvajal-Schiaffino, A.P.Millar, Nicholson C., Stockinger K., Zini, F., Evaluation of an Economic-Based File Replication Strategy for a Data Grid, in Int. Workshop on Agent Based Cluster and Grid Computing, CCGrid2003, Tokyo, Japan, May 2003.

[7] Foster, I. et. al., The Grid2003 Production Grid: Principles and Practice, HPDC 2004, Honolulu, USA, June 2004.

[8] Mohamed, H.H., Epema, D.H.J., An Evaluation of the Close-to-Files Processor and Data Co-Allocation Policy in Multiclusters, IEEE International Conference on Cluster Computing, San Diego, USA, September 2004.

[9] Ranganathan, K., Foster, I., Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids, in Journal of Grid Computing, V1(1) 2003.

[10] Ranganathan,K., Foster, I., Computation Scheduling and Data Replication Algorithms for Data Grids, 'Grid Resource Management: State of the Art and Future Trends', J. Nabrzyski, J. Schopf, and J. Weglarz, eds. Kluwer Academic Publishers, 2003.

[11] SDSS Project, https://www.darkenergysurvey.org.

[12] Shan, H., Oliker, L., Smith, W., Biswas, R., Scheduling in Heterogeneous Grid Environments: The Effects of Data Migration, International Conference on Advanced Computing and Communication, Gujarat, India, 2004.

[13] Singh, G., Kesselman, C., Deelman, E., Optimizing Grid-Based Workflow Execution, work submitted to 14th IEEE International Symposium on High Performance Distributing Computing, July 2005.