

Simulation of Dynamic Grid Replication Strategies in OptorSim

William H. Bell¹, David G. Cameron¹, Luigi Capozza²,
A. Paul Millar¹, Kurt Stockinger³, Floriano Zini²

¹ University of Glasgow, Glasgow, G12 8QQ, Scotland

² ITC-irst, Via Sommarive 18, 38050 Povo (Trento), Italy

³ CERN, European Organization for Nuclear Research, 1211 Geneva, Switzerland

Abstract. Computational Grids normally deal with large computation-intensive problems on small data sets. In contrast, Data Grids mostly deal with large computational problems that in turn require evaluating and mining large amounts of data. Replication is regarded as one of the major optimisation techniques for providing fast data access.

Within this paper, several replication algorithms are studied. This is achieved using the Grid simulator: OptorSim. OptorSim provides a modular framework within which optimisation strategies can be studied under different Grid configurations. The goal is to explore the stability and transient behaviour of selected optimisation techniques.

1 Introduction

Within the Grid community much work has been done on providing the basic infrastructure for a typical Grid environment. Globus [4], Condor [1] and recently the EU DataGrid [3] have contributed substantially to core Grid middleware services software that are available as the basis for further application development. However, little effort has been made so far to optimise the use of Grid resources.

To use a Data Grid, users typically submit *jobs*. In order for a job to be executed, three types of resources are required: computing facilities, data access and storage, and network connectivity. The Grid must make scheduling decisions for each job based on the current state of these resources (workload and features of Computing Elements, location of data, network load). Complete optimisation is achieved when the combined resource impact of all jobs is minimised, allowing jobs to run as fast as possible.

File replication (i.e. spread of multiple copies of files across the Grid) is an effective technique for reducing data access overhead. Maintaining an optimal distribution of replicas implies that the Grid optimisation service [7] must be able to modify the geographic location of data files. This is achieved by triggering both replication and deletion of data files. By reflecting the dynamic load on the Grid, such replica management will affect the migration of particular files toward sites that show increased frequency of file-access requests.

In order to study the complex nature of a typical Grid environment and evaluate various replica optimisation algorithms, a Grid simulator (called *OptorSim*) was developed. In this paper the design concepts of *OptorSim* are discussed and preliminary results based on selected replication algorithms are reported.

The paper is structured as follows. Section 2 describes the design of the simulator *OptorSim*. Various replication algorithms are discussed in Section 3. After setting the simulation configuration in Section 4, Section 5 is dedicated to a description of simulation results. Section 6 highlights related work. Finally, Section 7 concludes the paper and reports on future work.

2 Simulation Design

OptorSim [2] is a simulation package written in Java™. It was developed to study the effectiveness of replica optimisation algorithms within a Data Grid environment.

2.1 Architecture

One of the main design considerations for *OptorSim* is to model the interactions of the individual Grid components of a running Data Grid as realistically as possible. Therefore, the simulation is based on the architecture of the EU DataGrid project [14] as illustrated in Figure 1.

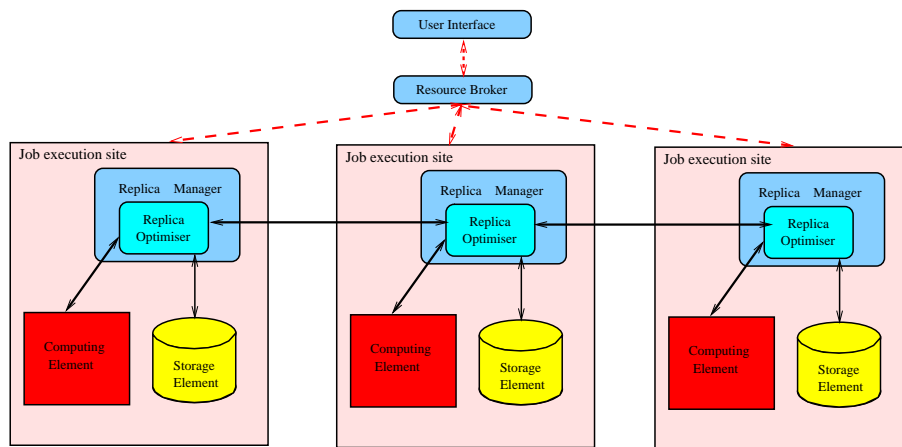


Fig. 1. Simulated DataGrid Architecture.

The simulation was constructed assuming that the Grid consists of several sites, each of which may provide computational and data-storage resources for submitted jobs. Each *site* consists of zero or more *Computing Elements* and zero

or more *Storage Elements*. Computing Elements run jobs, which use the data in files stored on Storage Elements and a *Resource Broker* controls the scheduling of jobs to Computing Elements. Sites without Storage or Computing Elements act as network nodes or routers.

The decision about data movement associated with jobs between sites is performed by a component called the *Replica Manager*. Within the Replica Manager the decision to create or delete replicas is controlled by a *Replica Optimiser* called *Optor*. At the heart of *Optor* is a replica optimisation algorithm, the properties of which are discussed in Section 3.

2.2 Internals

In the simulation each Computing Element is represented by a thread. Job submission to the Computing Elements is managed by another thread: the Resource Broker. The Resource Broker ensures every Computing Element continuously runs jobs by frequently attempting to distribute jobs to all the Computing Elements. When the Resource Broker finds an idle Computing Element, it selects a job to run on it according to the policy of the Computing Element, i.e. which type of jobs it will run and how often it will run each job.

At any time, a Computing Element will be running at most one job. As soon as the job finishes, another is assigned by the Resource Broker. So, although there is no explicit job scheduling algorithm, all Computing Elements process jobs for the duration of the simulation but are never overloaded. Currently, optimisation only occurs after a job has been scheduled to a Computing Element. The more complex scenario of optimising both job scheduling and data access will be part of future work.

Each job has a set of files it may request. Two types of reference may be used for a file: a logical file name (LFN) and a physical file name (PFN). An LFN is an abstract reference to a file that is independent of both where the file is stored and how many replicas exist. A PFN refers to a specific replica of some LFN, located at a definite site. Each LFN will have one PFN for each replica in the Grid.

A job will typically request a set of LFNs for data access. The order in which those files are requested is determined by the access pattern. The following access patterns were considered: *sequential* (the set of LFNs is ordered, forming a list of successive requests), *random* (files are selected randomly from set with a flat probability distribution), *unitary random walk* (set is ordered and successive file requests are exactly one element away from previous file request, direction is random) and *Gaussian random walk* (as with unitary random walk, but files are selected from a Gaussian distribution centred on the previous file request).

When a file is required by a job, the file's LFN is used to locate the best replica via the Replica Optimiser function `getBestFile(LFN, destinationStorageElement)`, where `destinationStorageElement` is the Storage Element to which the replica may be copied. It is assumed the Computing Element on which the job is running and requested Storage Element are located at the same site.

`getBestFile()` checks the *Replica Catalogue* for copies of the file. The Replica Catalogue is a Grid middleware service currently implemented within the simulation as a table of LFNs and all corresponding PFNs. By examining the available bandwidth between `destinationStorageElement` and all sites on which a replica of the file is stored, `getBestFile()` can choose the PFN that will be accessed fastest and hence decrease the job running time.

The simulated version of `getBestFile()` partially fulfils the functionality as described in [7]. It is a blocking call that may cause replication to a Storage Element located in the site where the job is running. After any replication has completed, the PFN of the best available replica is returned to the job. If replication has not occurred, the best replica is located on a remote site and is accessed by the job using remote I/O.

Both the replication time (if replication occurs) and the file access time (if from a remote site) are dependent on the network characteristics over the duration of the connection. At any time, the bandwidth available to a transfer is limited by the lowest bandwidth along the transfer path. For transfers utilising a common network element, the bandwidth of that element is shared so each transfer receives an equal share.

3 Optimisation Algorithms

Replica optimisation algorithms are the core of the Replica Optimiser. Over the duration of a submitted job, PFNs for each LFN are requested by calling `getBestFile()`. Optimisation algorithms implement `getBestFile()` so that it may copy the requested file from the remote site to a Storage Element on the same site as the requesting Computing Element. If all Storage Elements on this site are full then a file must be deleted for the replication to succeed.

The strategy used to decide which file should be deleted differentiates optimisation algorithms. In the following, we briefly present three simple algorithms and a more sophisticated one in greater detail. These algorithms have been implemented into `OptorSim`.

3.1 Simple Algorithms

No replication. This algorithm never replicates a file. The distribution of initial file replicas is decided at the beginning of the simulation and does not change during its execution. This algorithm returns a PFN with the largest expected bandwidth. Since the network load varies during the simulation, the optimal PFN may change.

Unconditional replication, oldest file deleted. This algorithm always replicates a file to the site where the job is executing. If there is no space to accommodate the replication, the oldest file in the Storage Element is deleted.

Unconditional replication, least accessed file deleted. This algorithm behaves as the previous method, except the least accessed file in the past time interval δt is deleted.

3.2 An Economic Approach

This section presents a replication strategy based on an economic model for Grid resource optimisation. A general description of this economic approach can be found in [9].

The economic model we propose includes actors (autonomous goal-seeking entities) and the resources in the Grid. Optimisation is achieved via interaction of the actors in the model, whose goals are maximising the profits and minimising the costs of data resource management. Data files represent the goods in the market. They are purchased by Computing Elements for jobs and by Storage Elements in order to make an investment that will improve their revenues in the future. They are sold by Storage Elements to Computing Elements and to other Storage Elements. Computing Elements try to minimise the file purchase cost, while Storage Elements have the goal of maximising profits.

This economic model is utilised to solve two distinct problems: in deciding if replication should occur and in the selection of the expendable file(s) when creating space for a new replica.

When a job running on a Computing Element requests a file, the optimisation tries to locate the cheapest copy of it in the Grid by starting an auction. Storage Elements that have the file locally may reply, bidding a price that indicates the file transfer cost. A site that does not have a file locally may initiate its own auction to establish if, by replication, it can satisfy the file request. This mechanism realises the global optimisation mentioned above. Currently, the auction protocol has still to be integrated into *OptorSim*. In the following discussion, we used the simpler protocol described in Section 2.2.

The mechanism for deciding if replication should occur is implemented in *OptorSim*. It is described the following section.

Replication Decision. Within our economic model the Replica Optimiser needs to make an informed decision about whether it should replicate a file to a local Storage Element. This decision is based on whether the replication (with associated file transfer and file deletion) will result in reduced expected future file access cost for the local Computing Element.

In order to make this decision, the Replica Optimiser keeps track of the file requests it receives and uses this history as input to an evaluation function $E(f, r, n)$. This function, defined in [9], returns the predicted number of times a file f will be requested in the next n requests based on the past r requests in the history.

After any new file request is received by the Replica Optimiser (say, for file f), the prediction function E is calculated for f and every file in the storage. If there is no file in the Storage Element that has a value less than the value

of f then no replication occurs. Otherwise, the least valuable file is selected for deletion and a new replica of f is created on the Storage Element. If multiple files on the Storage Element share the minimum value, the file having the earliest last access time is deleted.

The evaluation function $E(f, r, n)$ is defined by the equation

$$E(f, r, n) = \sum_{i=1}^n p_i(f), \quad (1)$$

with the following argument.

Assuming that requests for files containing similar data are clustered in spatial and time locality, the request history can be described as a random walk in the space of integer file identifiers¹. In the random walk, the identifier of the next requested file is obtained from the current identifier by the addition of a step, the value of which is given by some probability distribution. Assuming a binomial distribution of the steps, the probability of receiving a request for file f at step i of the random walk is given by the equation

$$p_i(f) = \frac{1}{2^{2iS}} \binom{2iS}{id(f) - \bar{s} + iS}, \quad |id(f) - \bar{s}| \leq iS \quad (2)$$

where \bar{s} is the mean value of the binomial distribution, S is the maximum value for the step, and $id(f)$ is a unique file identifier (for instance, the LFN). Then, the *most probable number of times file f will be requested during the next n requests* is given by (1).

A time interval δt describes how far back the history goes and thus determines the number r of previous requests which are considered in the prediction function. We assume that the mean arrival rate of requests is constant. Once δt has been decided, n is obtained by

$$n = r \frac{\delta t'}{\delta t} \quad (3)$$

where $\delta t'$ is the future interval for which we intend to do the prediction.

The value for S in (2) depends on the value of r . The mean value \bar{s} is obtained from the recent values of the step in the random walk. In particular, \bar{s} is calculated as the weighted average of the last r steps, where weights decrease over past time.

4 Simulation Configuration

4.1 Grid Configuration

The study of optimisation algorithms was carried out using a model of EU DataGrid TestBed 1 sites and their associated network geometry as illustrated

¹ We assume a mapping between file names and identifiers that preserve file content similarity.

in Figure 2. Within this model, each site was allocated storage resources proportional to their actual hardware allocations. Each TestBed site, excluding CERN, was assigned a Computing and Storage Element. CERN was allocated a Storage Element to hold all of the master files but was not assigned a Computing Element. Routers, as previously stated, were described by creating a site without Computing or Storage Elements. The size of the Storage Elements for each TestBed site are given in Table 1.

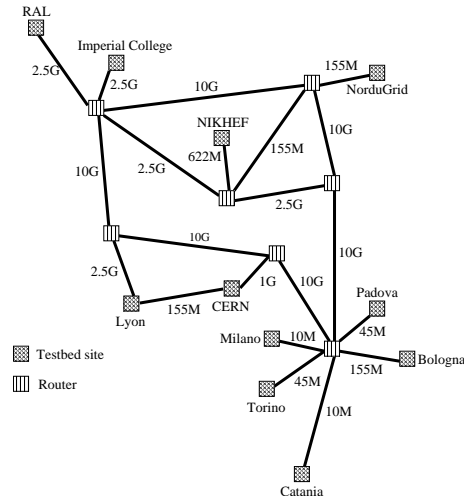


Fig. 2. The EU DataGrid TestBed 1 sites and the approximate network geometry. The numbers indicate the bandwidth between two sites.

Site Name	Bologna	Catania	CERN	Imperial College	Lyon
Storage Element (GBytes)	30	30	10000	80	50

Site Name	Milano	NIKHEF	NorduGrid	Padova	RAL	Torino
Storage Element (GBytes)	50	70	63	50	50	50

Table 1. A list of resources allocated to the TestBed 1 sites, from which the results in this paper were generated.

4.2 Job Configuration

Initially, all files were placed on the CERN Storage Element. Jobs were based on the CDF use-case as described in [12]. There were six job types, with no overlap

between the set of files each job accessed. The total size of the file accessed by any job type were estimated in [12] and are summarised in Table 2. Each set of files was assumed to be composed of 10GByte files.

There will be some distribution of jobs each site performs. In the simulation, we modelled this distribution such that each site ran an equal number of jobs of each type except for a preferred job type, which ran twice as often. This job type was chosen for each site based on storage considerations; for the replication algorithms to be effective, the local storage on each site had to be able to hold all the files for the preferred job type.

Data Sample	Total Size (GBytes)
Central J/ψ	1200
High p_t leptons	200
Inclusive electrons	5000
Inclusive muons	1400
High E_t photons	5800
$Z^0 \rightarrow b\bar{b}$	600

Table 2. Estimated sizes of CDF secondary data sets (from [12]).

5 Results

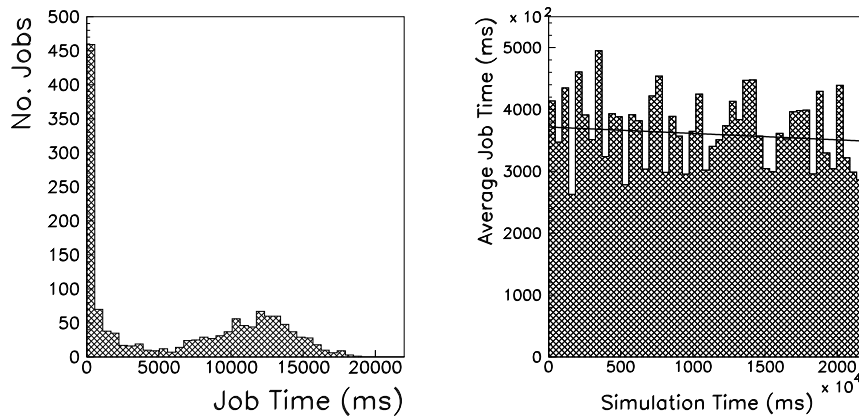


Fig. 3. A histogram of job duration (left) and the progression of job duration over course of the simulation (right).

The left histogram in Figure 3 shows a typical spread of job duration for a single job type at a selected Computing Element over the course of a simulation run. The large spike near zero is due to the job requesting files that are available on the local site, hence no time-consuming file transfers need to take place. The longer durations are due to the job requesting some files not present at the local site. The spread is due to the network load, which can vary over time, affecting the file transfer times.

The variation of job duration over the simulation is shown in the right histogram in Figure 3 for the same job type and Computing Element as above. There is clearly a large variation in the job duration due to the factors already mentioned, but the general trend is for jobs to be executed more quickly over time, indicating the movement toward a more optimal replica configuration.

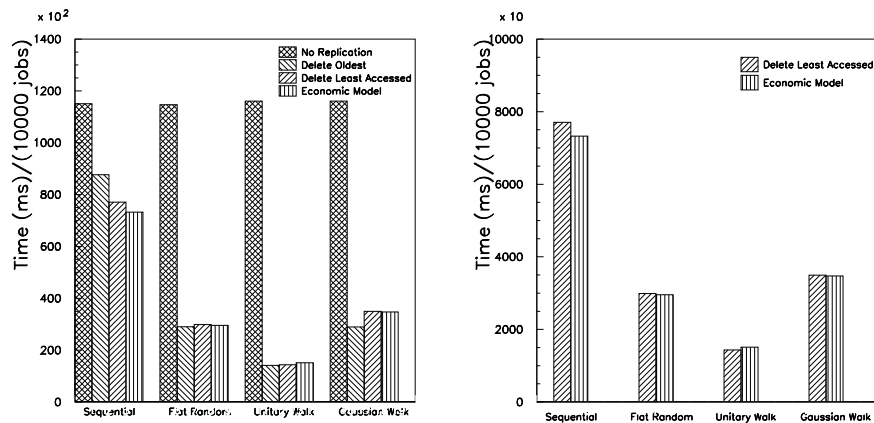


Fig. 4. Integrated running times for 10000 jobs using each access pattern and replica optimisation algorithm.

Further tests were conducted simulating 10000 jobs using each of the four algorithms:

1. *No replication*
2. *Unconditional replication, oldest file deleted*
3. *Unconditional replication, least accessed file deleted*
4. *Economic Model*

For each replication algorithm, each of the following four file access patterns (as defined in Section 2.2) was tested.

1. *Sequential*
2. *Random*

3. *Unitary random walk*
4. *Gaussian random walk*

Figure 4 shows the total time to complete 10000 jobs for each of the four access patterns using the four optimisation algorithms.

With no optimisation, the jobs take much longer than even the simplest optimisation algorithm as all the files for every job have to be transferred from CERN every time a job is run.

The three algorithms where replication is conducted all show a marked reduction in the time to execute 10000 jobs. This is not surprising as with no replication, all file requests from all jobs must come from CERN.

The three optimisation algorithms that implement replication show similar performance for Random, Unitary random walk and Gaussian random walk. For sequential access patterns, the running time is at least 10% faster using the Economic Model optimiser than the other optimisers. These results were expected as the Economic Model assumes a sequential access pattern. However, this can be adjusted to match the observed distribution, if needed.

6 Related Work

Recently there has been great interest in modelling Data Grid environments. A simulator for modelling complex data access patterns of concurrent users in a distributed system is found in [13]. These studies were mainly conducted within the setting of scientific experiments such as the LHC, which finally resulted in the creation of the EU DataGrid project [3].

MicroGrid [18] is a simulation tool for designing and evaluating Grid middleware, applications and network services for the computational Grid. Currently, this simulator does not take data management issues into consideration. Further Grid simulators are presented in [11, 6]

In [15] an approach is proposed for automatically creating replicas in a typical decentralised Peer-to-Peer network. The goal is to create a certain number of replicas on a given site in order to guarantee some minimal availability requirements.

In Nimrod-G [8, 5] an economic model for job scheduling is introduced in where “Grid credits” are assigned to users that are proportional to their level of priority. In this model, optimisation is achieved at the scheduling stage of a job. However, our approach differs by including both optimal replica selection and automated replica creation in addition to scheduling-stage optimisation.

Various replication and caching strategies within a simulated Grid environment are discussed in [16] and their combination with scheduling algorithms is studied in [17]. The replication algorithms proposed are based on the assumption that popular files in one site are also popular in other sites. Replication from one site to another is triggered when the popularity of a file overcomes a threshold and the destination site is chosen either randomly or by selecting the least loaded site. We take a complementary approach. Our replication algorithms are

used by Grid sites when they need data locally and are based on the assumption that in computational Grids there are areas (so called “data hot-spots”) where particular sets of data are highly requested. Our algorithms have been designed to move data files toward “data hot-spots”.

7 Conclusions and Future Work

In this paper we described the design of the Grid simulator *OptorSim*. In particular, *OptorSim* allows the analysis of various replication algorithms. The goal is to evaluate the impact of the choice of an algorithm on the throughput of typical Grid jobs. We have chosen two traditional cache management algorithms (oldest file deletion and least accessed file deletion) and compared them to a novel algorithm based on an economic model.

We based our analysis on several Grid scenarios with various work loads. Results obtained from *OptorSim* suggest that the economic model performs at least as well as traditional methods. In addition, there are specific realistic cases where the economic model shows marked performance improvements.

Our future work will extend the simulator by including the auction protocol proposed in [10]. This is motivated by the additional functionality of automatic replication to third party sites, allowing file migration to accurately match demand.

Acknowledgements

The authors thank Erwin Laure, Heinz Stockinger and Ekow Otoo for valuable discussions during the preparation of this paper.

William Bell and David Cameron thank PPARC for funding as part of the GridPP(EDG) project and as an e-Science student respectively; Paul Millar thanks SHEFC for funding under the ScotGRID project.

References

1. The condor project. <http://www.cs.wisc.edu/condor/>.
2. *OptorSim* - A Replica Optimiser Simulation. <http://grid-data-management.web.cern.ch/grid-data-management/optimisati%on/optor/>.
3. The DataGrid Project. <http://www.eu-datagrid.org>.
4. The Globus Project. <http://www.globus.org>.
5. D. Abramson, R. Buuya, and J. Giddy. A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. In *Future Generation Computer Systems*, to appear.
6. K. Aida, A. Takefusa, H. Nakaka, S. Matsuoka, S. Sekiguchi, and U. Nagashima. Performance Evaluation Model for Scheduling in a Global Computing System. *International Journal of High Performance Applications*, 14(3), 2000.
7. W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. Design of a Query Optimisation Service. Technical report, CERN, 2002. WP2 - Data Management, EU DataGrid Project. <http://edms.cern.ch/document/337977>.

8. R. Buyya, H. Stockinger, J. Giddy, and D. Abramson. Economic Models for Management of Resources in Peer-to-Peer and Grid Computing. In *Commercial Applications for High-Performance Computing, SPIE's International Symposium on the Convergence of Information Technologies and Communications (ITCom 2001)*, Denver, Colorado, USA, August 2001.
9. L. Capozza, K. Stockinger, and F. Zini. Preliminary Evaluation of Revenue Prediction Functions for Economically-Effective File Replication, June 2002.
10. M. Carman, F. Zini, L. Serafini, and K. Stockinger. Towards an Economy-Based Optimisation of File Access and Replication on a Data Grid. In *International Workshop on Agent based Cluster and Grid Computing at International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, Berlin, Germany, May 2002. IEEE Computer Society Press. Also appears as IRST Technical Report 0112-04, Istituto Trentino di Cultura, December 2001.
11. H. Casanova, G. Obertelli an F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proc. of Super Computing 2002*, Dallas, Texas, USA, November 2002.
12. B. T. Huffman et al. The CDF/D0 UK GridPP Project. CDF Internal Note. 5858.
13. I. C. Legrand. Multi-Threaded, Discrete Event Simulation of Distributed Computing Systems. In *Proc. of Computing in High Energy Physics (CHEP 2000)*, Padova, Italy, February 2000.
14. EU DataGrid Project. The DataGrid Architecture, 2001.
15. K. Ranganathan, A. Iamnitchi, and I. Foster. Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities. In *Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop*, Berlin, Germany, May 2002.
16. K. Ranganathana and I. Foster. Identifying Dynamic Replication Strategies for a High Performance Data Grid. In *Proc. of the International Grid Computing Workshop*, Denver, Colorado, USA, November 2001.
17. K. Ranganathana and I. Foster. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In *International Symposium of High Performance Distributed Computing*, Edinburgh, Scotland, July 2002. To appear.
18. H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The MicroGrid: a Scientific Tool for Modeling Computational Grids. *Scientific Programming*, 8(3):127–141, 2000.