UNIVERSITY
of
GLASGOW

Department of Physics & Astronomy
Experimental Particle Physics Group
Kelvin Building, University of Glasgow,
Glasgow, G12 8QQ, Scotland
Telephone: +44 (0)141 339 8855 Fax: +44 (0)141 330 5881

GLAS-PPE/2003-18
24th November 2003

OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies

William H. Bell[1], David G. Cameron[1], Luigi Capozza[2]
A. Paul Millar[1], Kurt Stockinger[3], Floriano Zini[4]

[1] University of Glasgow, Glasgow, G12 8QQ, Scotland
[2] Institut fuer Kernphysik - A4, Becherweg 45,
55099 Mainz, Germany
[3] CERN, European Organization for Nuclear Research,
1211 Geneva, Switzerland
[4] ITC-irst, Via Sommarive 18, 38050 Povo (Trento), Italy

Abstract

Computational Grids process large, computationally intensive problems on small data sets. In contrast, Data Grids process large computational problems that in turn require evaluating, mining and producing large amounts of data. Replication, creating geographically disparate identical copies of data, is regarded as one of the major optimisation techniques for reducing data access costs.

In this paper, several replication algorithms are discussed. These algorithms were studied 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. We detail the design and implementation of OptorSim and analyse various replication algorithms based on different Grid workloads.

International Journal of High Performance Computing Applications, 17(4), 2003

1 Introduction

Within the Grid community much work has been done on providing the basic infrastructure for a typical Grid environment. Globus [3], Condor [1] and recently the EU DataGrid [2] have contributed substantially to core Grid middleware services 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.

A typical Grid *job* requires three types of resources: computing facilities, data access and storage, and network connectivity. The Grid must make scheduling decisions (i.e., decisions about which site is used for job execution) for each job based on the current state of these resources (workload and features of computing facilities, location of data, and 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. spreading multiple copies of files across the Grid) is an effective technique for reducing data access overhead. Since the Grid is a highly dynamic environment, maintaining an optimal distribution of replicas implies that the Grid optimisation service [6] 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 is a simulation package written in Java™. It was developed to mimic the structure of a real Data Grid and study the effectiveness of replica optimisation algorithms within such an 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 [15] as illustrated in Figure 1.

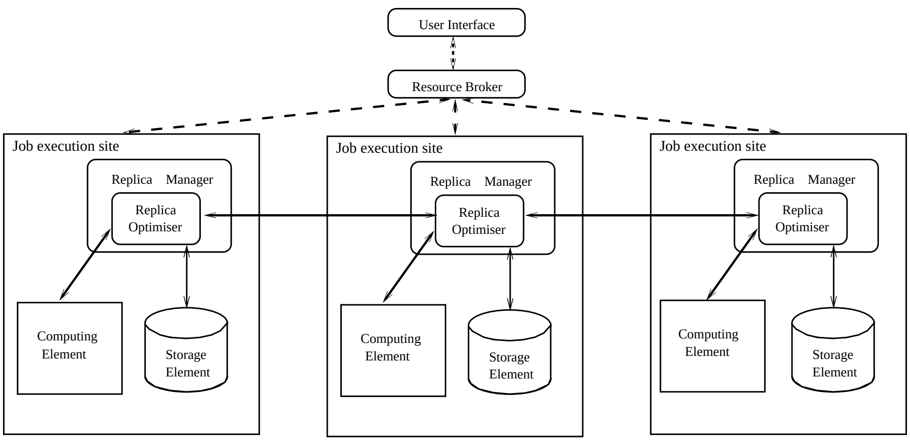

Figure 1: Simulated DataGrid Architecture.

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. A *Resource Broker* controls the scheduling of jobs to Computing Elements. Sites without Storage or Computing Elements act as network nodes or routers.

Since the Data Grid is a highly dynamic environment, the middleware should be able to cope with the changes in the status of resources when performing resource allocation. In particular, the Grid optimisation service should avoid making a priori irrevocable decisions at job scheduling time about which file replicas will be used to access data for a particular job.

Therefore, we see optimisation as an ongoing activity, which is performed at two points in time during the life-time of a job [6]:

1. The first optimisation phase occurs when the Computing Element where the job should run is chosen.

2. In the second phase optimal *Dynamic Replica Selection* is achieved during the run time of a job; in this phase creation of replicas can be triggered by the optimisation algorithm.

In this paper we consider only optimisation that occurs after a job has been scheduled to a Computing Element. This allows us to focus on the performance of replication algorithms under simple conditions. The more complex scenario of optimising both job scheduling and data access will be part of future work.

In the EU DataGrid project, the grid optimisation service is called the *Replica Optimiser* and it is embedded into a component called the *Replica Manager* [12]. The Replica Optimiser makes decisions about data movement associated with jobs between sites and creation or deletion of replicas. The properties of the Replica Optimiser 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 execution flow of these threads is shown in Figure 2. The Resource Broker ensures every Computing Element is continuously running a job 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. what 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.

2.2.1 Simulation Input

As input OptorSim uses two configuration files. One file describes the network topology (network links between Grid sites and available bandwidth for each defined link) and the components of each site (number of Computing and Storage Elements, as well as their sizes). The second configuration file contains information on the simulated jobs, in particular the logical file names of the files they need to access while executing. Two types of references 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 has related PFNs corresponding to each replica in the Grid.

2.2.2 Access Patterns

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 (examples of which

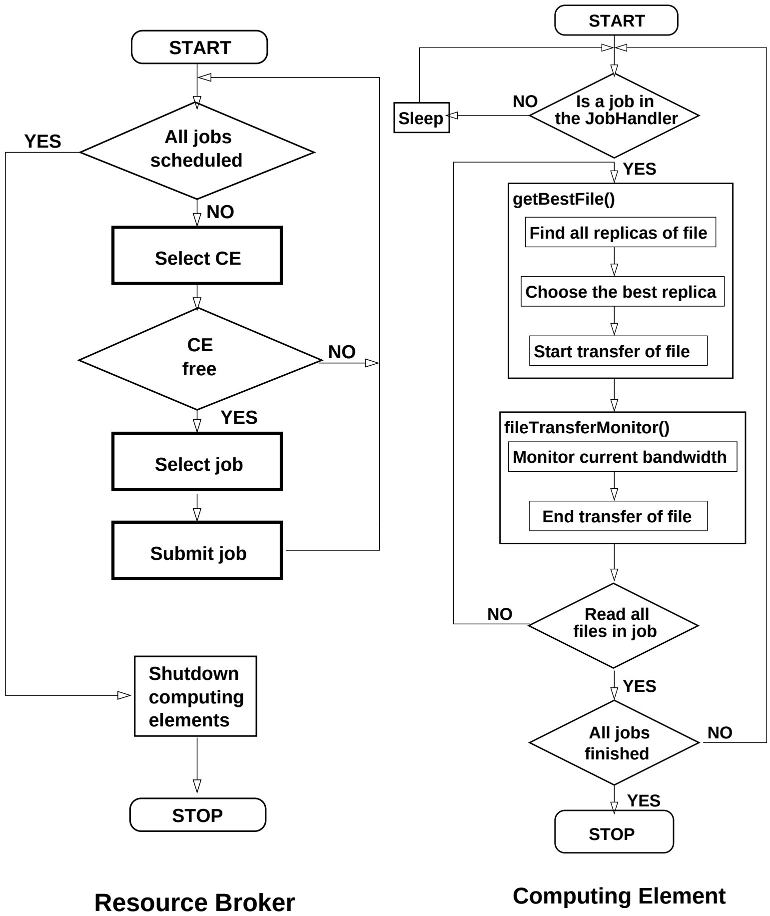

Figure 2: Execution flows of the Resource Broker and Computing Element threads.

are shown in Figure 3): *sequential* (the set of LFNs is ordered, forming a list of successive requests), *random* (files are selected randomly from a set with a flat probability distribution), *unitary random walk* (set is ordered and successive file requests are exactly one element away from the 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).

Using the sequential access pattern every file in the job will be accessed in the order stated in the job definition. For all other access patterns any file in the job can be accessed zero or many times. However, the number of file requests always corresponds to the number of files requests in the job description. For example, Figure 3(c) shows a unitary random walk access pattern. In this case, the job requests 10 files so 10 requests were made in total. The first file was randomly selected as the sixth in the list of possible files. The next file had an equal probability of being file 5 or file 7. As 5 was chosen, the next file had an equal probability of being file 4 or file 6 etc. In this example, the job requested file 3 four times whereas files 1, 7, 8, 9, and 10 were never requested.

2.2.3 Optimisation Function

When a file is requested by a job, the 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, i.e. the destination Storage Element, are located at the same site.

The function *getBestFile()* checks the *Replica Catalogue* for copies of the file. The Replica Catalogue is a Grid middleware service [11] currently implemented within the simulation as a table of

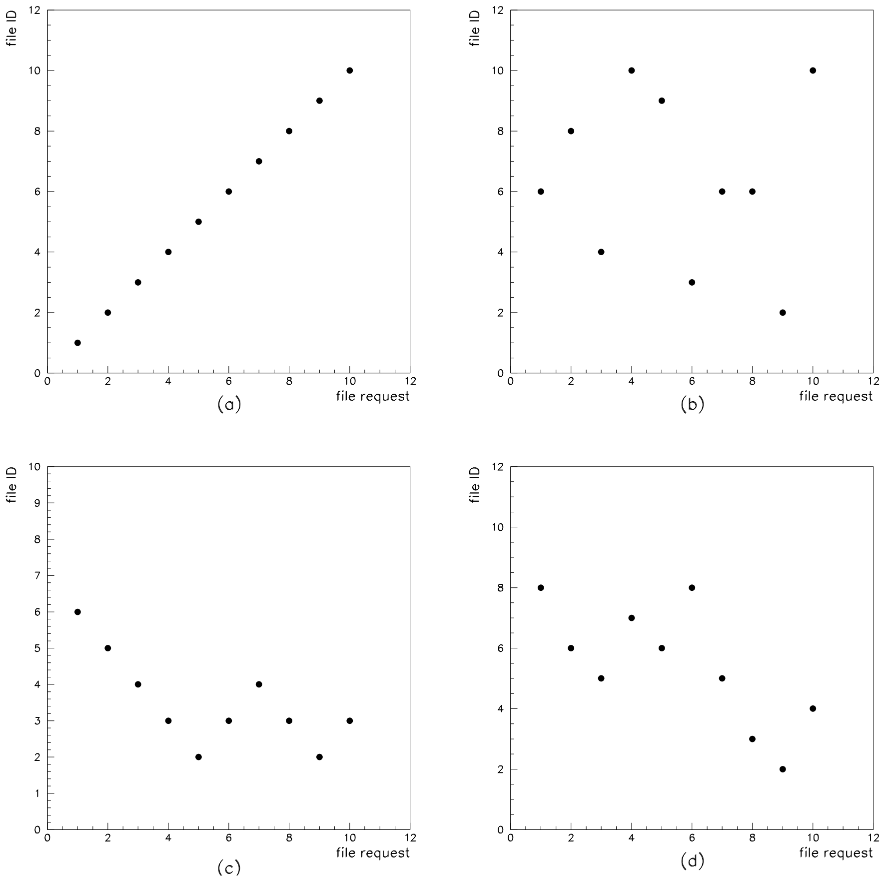

Figure 3: Example access patterns for a job containing 10 files: (a) sequential, (b) random, (c) unitary random walk, (d) Gaussian random walk.

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()* chooses the PFN that will be accessed fastest and hence decrease the job running time.

The simulated version of *getBestFile()* partially fulfills the functionality as described in [6]. 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.

Currently OptorSim does not model the job execution, i.e. the processing of the files by the Computing Element, but only the file transfers required by the job. This is because the processing of the files does not directly affect the replica optimisation process. However, as part of our future work, we will extend OptorSim to simulate both Computing Elements and additional network traffic.

2.2.4 Statistics

OptorSim provides various execution statistics such as job duration and the progression of job duration over the course of the simulation. OptorSim also provides a histogram management package and a

GUI that is dynamically updated during job execution (Figure 4).

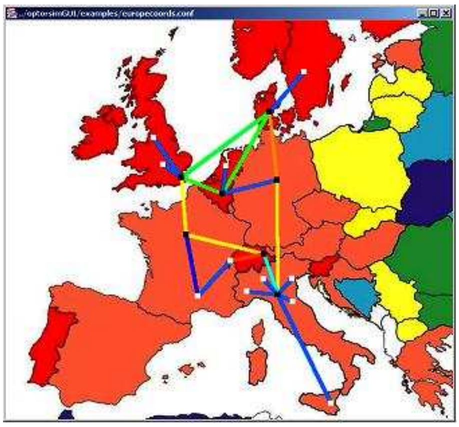

Figure 4: Screen-shot of the OptorSim GUI.

2.2.5 Modular Design Approach

OptorSim is designed in a modular way which makes it easy to plug in various access pattern generators for workload simulation (see Figure 5) and various optimisation algorithms (see Figure 6). Currently OptorSim provides the access patterns as discussed in this section. The optimisation algorithms of OptorSim are detailed in Section 3.

Figure 5: Access patterns (workloads) currently supported by OptorSim.

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 the optimisation algorithm deems it to be worthwhile. 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 whether to replicate and if so which file should be deleted differentiates optimisation algorithms. In the following, we briefly present three simple algorithms (two of which are based on traditional cache replacement algorithms) and a more sophisticated one in greater detail. These algorithms have been implemented into OptorSim.

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

org.edg.data.replication.optor
Interface Optimisable

All Known Implementing Classes:
AlwaysReplicateOptimiser, DeleteLeastAccessedOptimiser, EcoModelOptimiser, SimpleOptimiser

public interface **Optimisable**

Figure 6: Optimisation algorithms currently supported by OptorSim.

3.1 Simple Algorithms

3.1.1 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 the PFN with the fastest access time. Since the network load varies during the simulation, the optimal PFN may change.

3.1.2 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.

3.1.3 Unconditional replication, least accessed file deleted

This algorithms 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 file purchase cost, while Storage Elements have the goal of maximising profits.

This economic model is utilised in deciding if replication should occur and in the selection of the expendable file(s) when creating space for a new replica. This mechanism is described in the following section.

3.2.1 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 maximising the profit of the Storage Element.

In order to make this decision, in [9] we propose that the Replica Optimiser keeps track of the file requests it receives and uses this history as input to an evaluation function. This functions returns the predicted revenue that a Storage Element will earn for a file over a time window in the future based on the income it has recently earned for that and similar files. While evaluating a file that is not locally stored, the function takes into consideration the cost of local replication of the file.

Currently in OptorSim we use a simplified version of the evaluation function $E(f, r, n)$. It 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

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

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 (see Figure 7) in the space of integer file identifiers[1)]. In the random walk, the identifier of the next requested

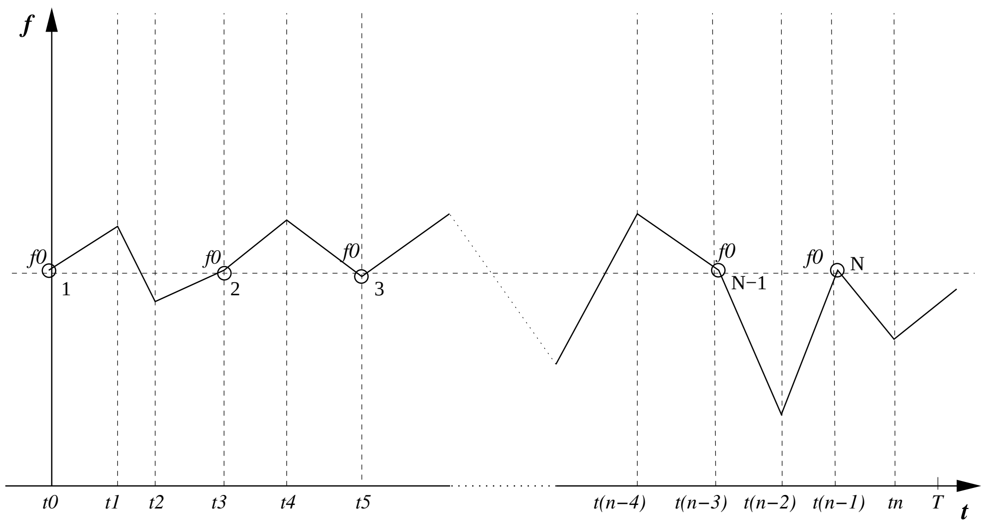

Figure 7: File access history as a random-walk in space $\{f\}$ of file identifiers.

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 symmetric binomial distribution of the steps, the random variable f, representing the file identifier requested at a generic step of the random walk, has also a symmetric binomial distribution. Thus, the probability of receiving a request for file f at step i of the random walk is given by the equation

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

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

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

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

$$n = r\frac{\delta t'}{\delta t} \tag{3}$$

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. In fact, it can be estimated on the basis of the past, taking into account the variance of the distribution of the last r file requests. The mean value $\bar{f}$ is obtained from the recent values in the random walk of the file identifiers. In particular, $\bar{f}$ is calculated as the weighted average of the last r file requests, 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 in Figure 8. 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[2)] 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.

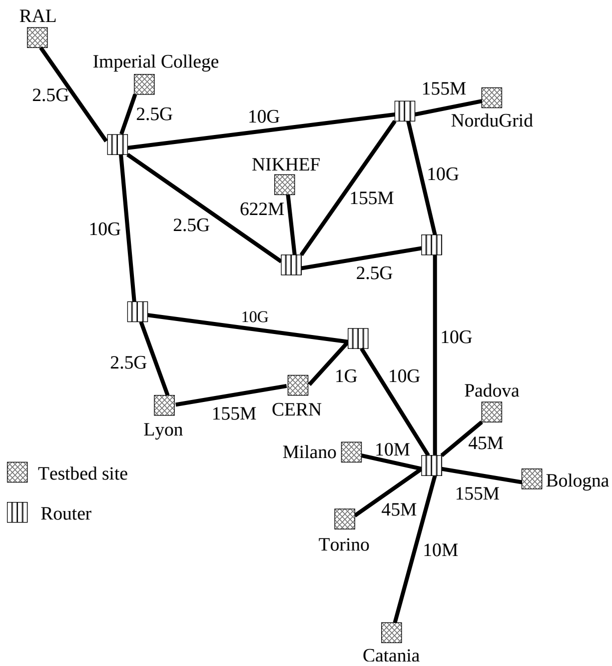

Figure 8: The EU DataGrid Testbed 1 sites and the approximate network geometry. The numbers indicate the bandwidth between two sites in Mbits/s (M) or Gbit/s (G).

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 [13]. There were six job types, with no overlap between the set of files each job accessed.

[2)] A master file contains the original copy of some data sample and cannot be deleted.

Site Name	Bologna	Catania	CERN	Imperial College	Lyon
Size (GB)	30	30	10000	80	50

Site Name	Milano	NIKHEF	NorduGrid	Padova	RAL	Torino
Size (GB)	50	70	63	50	50	50

Table 1: A list of Storage Element resources allocated to the Testbed 1 sites, from which the results in this paper were generated. The size of Storage Elements are in GigaBytes.

The total size of the file accessed by any job type were estimated in [13] and are summarised in Table 2. Each set of files was assumed to be composed of 1GByte 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 (GB)
Central J/ψ	10
High p_t leptons	2
Inclusive electrons	50
Inclusive muons	14
High E_t photons	58
$Z^0 \rightarrow b\bar{b}$	6

Table 2: Estimated sizes of CDF secondary data sets (from [13]).

5 Results

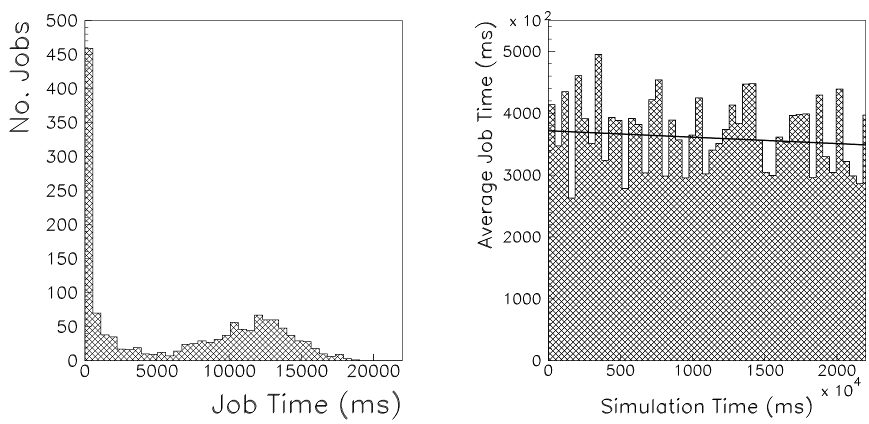

Figure 9: A histogram of job duration (left) and the progression of job duration over course of the simulation (right).

In this section we first show the behaviour of various jobs over the run time of the simulation. Next, we study the impact of different access patterns (work loads) on the performance of the optimisation algorithms presented in Section 3.

The left histogram in Figure 9 shows a typical spread of job duration for a single job type at a selected Computing Element over the course of a simulation run. There were one thousand jobs under consideration, evaluating the economic model's performance with the sequential access pattern. 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 9 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.

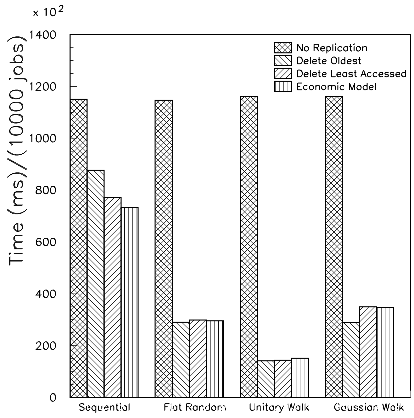

Figure 10: Integrated running times for 10000 jobs using each access pattern and replica optimisation algorithm.

Further tests were conducted simulating 10000 jobs using each of the four optimisation 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 10 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. They show similar performance for Random, Unitary random walk and Gaussian random walk.

For sequential access patterns, the total job execution time is at least 10% faster using the Economic Model algorithm than the other algorithms. This result was expected as the Economic Model assumes a sequential access pattern in its estimation of file values.

The performance of the Economic Model algorithm is roughly equal to that of the others for non-sequential access patterns. These access patterns result in a poorer estimation of the future file values, but the prediction function discussed in Section 3.2.1 can be adjusted to match the observed distribution, if needed. This will be part of our future work.

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 [14]. 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 [2].

MicroGrid [19] 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 [10, 5].

In [18] 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, 4] 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 and implementation 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 a simple remote access heuristic and two traditional cache replacement algorithms (oldest file deletion and least accessed file deletion). We then compared these algorithms 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.

In [7] we present an extended version of our economic model that includes an auction protocol. This is used to perform optimal replica selection and to achieve automatic replication to third-party sites. Future work will include the extensive testing of this additional functionality and evaluating the accuracy to which it migrates files to match demand.

Acknowledgments

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

This work was partially funded by the EU DataGrid Project, GridPP and the ScotGRID Project.

The research was conducted whilst Luigi Capozza was at ITC-irst. He thanks ITC-irst for his funding.

References

[1] The Condor Project. `http://www.cs.wisc.edu/condor/`.

[2] The DataGrid Project. `http://www.eu-datagrid.org`.

[3] The Globus Project. `http://www.globus.org`.

[4] D. Abramson, R. Buuya, and J. Giddy. A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. *Future Generation Computer Systems*, 18(8), October 2002.

[5] 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.

[6] W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. Design of a Replica Optimisation Framework. Technical Report DataGrid-02-TED-021215, CERN, Geneva, Switzerland, December 2002. EU DataGrid Project.

[7] W.H. Bell, D.G. Cameron, R. Carvajal-Schiaffino, A.P. Millar, K. Stockinger, and F. Zini. Evaluation of an Economy-Based File Replication Strategy for a Data Grid. In *Proceedings of the 3rd International Workshop on Agent based Cluster and Grid Computing at International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan, May 2003. IEEE Computer Society Press.

[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] M. Carman, F. Zini, L. Serafini, and K. Stockinger. Towards an Economy-Based Optimisation of File Access and Replication on a Data Grid. In *Proceedings of the 2nd 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.

[10] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proceedings of SuperComputing 2000*, pages 75–76, Dallas, Texas, USA, November 2000.

[11] A. Chervenak, E. Deelman, I. Foster, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, H. Stockinger, K. Stockinger, and B. Tierney. Giggle: A Framework for Constructing Scalable Replica Location Services. In *Proceedings of the International IEEE/ACM Supercomputing Conference (SC2002)*, Baltimore, USA, November 2002.

[12] L. Guy, P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. Replica Management in Data Grids. Technical report, GGF5 Working Draft, July 2002.

[13] B. T. Huffman et al. The CDF/D0 UK GridPP Project. CDF Internal Note CDF/DOC/COMP_UPG/5858, February 2002.

[14] I. C. Legrand. Multi-Threaded, Discrete Event Simulation of Distributed Computing Systems. In *Proceedings of Computing in High Energy Physics (CHEP 2000)*, Padova, Italy, February 2000.

[15] EU DataGrid Project. The DataGrid Architecture. Technical Report DataGrid-12-D12.4-333671-3-0, CERN, Geneva, Switzerland, 2001.

[16] K. Ranganathan and I. Foster. Identifying Dynamic Replication Strategies for a High Performance Data Grid. In *Proceedings of the International Grid Computing Workshop*, Denver, Colorado, USA, November 2001.

[17] K. Ranganathan 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.

[18] 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.

[19] 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.