

# A Data Replication Strategy to Increase Data Availability in Data Grids

Ming Lei and Susan V. Vrbsky  
Department of Computer Science  
University of Alabama  
Tuscaloosa, AL 35487-0290  
mlei@cs.ua.edu, vrbsky@cs.ua.edu

## Abstract

*Data Grid systems require the management of a massive number of data sets to make the data available to the many clients accessing the data. Data is typically replicated in these large scale applications to improve the data availability and job response time. Because the behavior of the thousands of Grid users is very dynamic, it is a challenging problem to determine where and when to make data replications to increase the system availability. Strategies for data replication have previously been proposed, but assume unlimited storage is available for replicas. In this paper, we introduce a new distributed strategy to maximize the data availability given limited replica storage. Our simulation shows our strategy can achieve better data availability and has a shorter total job time.*

**Keywords:** data availability, data Grid, limited storage, replica strategy, file missing rate

## 1. Introduction

As large-scale geographically distributed systems become more and more popular in data-intensive scientific applications, new strategies for managing such data are needed. Examples of Grid systems are the Particle Physics Data Grid (PPDG), and the Compact Muon Solenoid (CMS) Data Grid [1], in which millions of files will be generated from these scientific experiments and thousands of clients world-wide will access these data. This huge volume of data sets requires new strategies to determine how to make the data more available. Replication of data is typically viewed as the solution to improve file access time and reliability. Data replication schemes involve determining how many replicas are needed and where to place the data copies in the distributed nodes.

Earlier work on data replication has concentrated on decreasing the data access latency and the network bandwidth consumption. The work in [2] shows that the best replica strategy has

significant savings in latency and bandwidth consumption if the access patterns contain a moderate amount of geographical locality. In [3,4], a replica scheme based on the economical-model has been proposed that uses an auction protocol to make the replica decision for long-term optimization. The author shows the scheme outperforms other replica strategies with sequential file access patterns. In [5], an algorithm places replicas in a wide-area network, that along with their neighboring nodes, generate the highest load [6]. A dynamic replica replication strategy is proposed in [7] that benefits from 'network-level locality'.

As bandwidth and computing capacity have become relatively cheaper, the data access latency can decrease dramatically. How to improve the system reliability and availability then becomes the focal point. In large-scale data-intensive systems, hundreds of nodes are involved and any node failure or network outage can cause potential file unavailability. As a result, there has been an increase in research [8, 9] focusing on how to maximize the file availability.

An analytical model for determining the optimal number of replica servers is presented in [8]. In [9], a dynamic model-driven replication approach is presented in which peers create replicas automatically in a decentralized fashion. Both algorithms propose to meet the data availability goal based on the assumption that the total system replica storage is large enough to hold all the data replica copies. Each file will be replicated to the number of copies needed to achieve its availability goal without any discrimination. We feel this can waste system resources, since a replicated file may be accessed only one time in its whole life span

In this paper, we propose a replica strategy to increase the system data availability when storage resources are limited without sacrificing the data access latency. We introduce two new data availability metrics, the File Missing Rate (FMR) and the Bytes Missing Rate (BMR). Our test results on the simulator OptorSim [10] show that, in

general, our replica schemes perform better in terms of these two metrics and the total job time.

The rest of the paper is organized as follows. Section 2 introduces the two measures: the file missing rate and the bytes missing rate. We present our dynamic replica strategy in section 3. In section 4, we describe our simulation results based on the OptorSim, a simulator designed by the European Data Grid Project [11]. In the final section, we conclude our paper and describe future work.

## 2. System Data Availability

In a Data Grid system, the hundreds of clients who are physically distributed around the world will submit their job requests. Usually, such a Grid job will access multiple files to do some type of analysis, where any data file access failure will lead to an incorrect result or even a job crash. To protect the user from such risk, the Grid system will be required to make the data availability as high as possible. Furthermore, the system level data availability is more important than a single individual file's availability.

The Grid system will hold many physically distributed data files and their replicas, which will be the critical inputs for many jobs. A more reliable system will provide more of a benefit than the availability of a single file, especially when there are some files that are "hot" and other files that are relatively "cold". (Hot data is accessed more frequently than cold data.) At the same time, the storage elements cannot guarantee there are enough resources to make corresponding replicas of each single data file to make sure of its availability. However, as we demonstrate later, we can treat the hot and cold data differently to provide system level reliability to the end user.

Previous studies in Data Grid systems have measured such aspects as the mean available bandwidth and percentage of computer usage. In order to better represent the system data availability, we introduce two new measurements: the file missing rate and the bytes missing rate, which are defined as follows.

*File Missing Rate* - represents the number of files potentially unavailable for all the files requested by all the jobs.

*Bytes Missing Rate* - represents the number of bytes potentially unavailable of the total number of bytes requested by all jobs.

These two measurements will have a fixed relationship when all the file sizes in the system are the same, but will be different when the file sizes are unique in the system.

Figure 1 illustrates the whole system architecture of the EU Data Grid System as in [11]. A Data Grid typically consists of job execution sites containing the components: the computing element CE, the storage element SE and a replica manger containing a replica optimizer. Given a set of jobs,  $\mathcal{J} = (j_1, j_2, j_3, \dots, j_N)$ , they are submitted to the resource broker agent through a user interface. The resource broker agent then schedules them to the appropriate computing sites, where the jobs may be queued in the site's job queue. Meanwhile, each job will request more than one file to complete its own task from the file set,  $\mathcal{F} = (f_1, f_2, \dots, f_k)$ , where each file  $f_i$  is of size  $S_i$ . For better performance and reliability, the data files are usually replicated in different storage elements.

We describe each physical file's availability as a probability  $P_{\text{avai}}$ . Each file  $f_j$  must be placed in a Storage Element, which is the only place where a file can be stored. We assume that there is only one copy for a file in each Storage Element and any two files in the same storage element will have the same availability. Increasing the file copies in the same SE doesn't help improve the file availability, because if the SE fails, all the files on the SE will fail together. This can potentially harm the system level data availability because such redundant replications will waste the storage space in the SE. However, there may be more than one copy of a file in the Data Grid, which will be stored in different SEs. We define the  $P_{\text{avai}}$  of file  $f_j$  as:

$$P_{\text{avai}} = 1 - \prod_{i=1}^k (1 - p_i) \quad (1)$$

where  $k$  denotes there are  $k$  copies of the file  $f_j$ , each in different storage elements, and each copy with availability  $p_i$ .

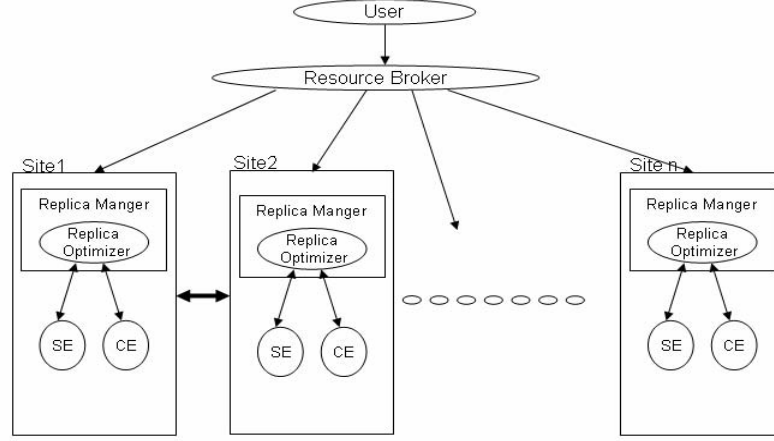


Figure1 Simulated DataGrid Architecture

For each file accessing operation that only accesses one file, the potential unsuccessfulness is  $1 - P_{avai}$ , with the assumption that any two file access operations are independent and each job will have several such file accessing operations. We can define the File Missing Rate as:

$$FMR = \sum_{i=1}^n \sum_{j=1}^k (1 - p_j) \quad (2)$$

where  $n$  denotes there are a total of  $n$  jobs, each of which will access  $k$  different files ( $k$  file accessing operations).  $p_j$  denotes file  $j$ 's availability as defined in equation (1) at the file access instant.

In the same way, we define the BMR (*Bytes Missing Rate*) as:

$$BMR = \sum_{i=1}^n \sum_{j=1}^k (1 - p_j) * S_j \quad (3)$$

where  $n$ ,  $k$  and  $p_j$  have the same meaning as in equation (2) and  $S_j$  denotes the size of file  $j$ .

From equations (2) and (3), it is apparent that the values for FMR and BMR will have a fixed

relationship if all the accessed file sizes are the same.

We note that because we assume the storage space is limited, we must consider:

$$\sum_{i=1}^m C_i * S_i \leq S \quad (4)$$

where  $m$  denotes the number of the different files occurring in the request operation set  $O$ ,  $C_i$  denotes the number of copies of  $f_i$  and  $S$  is the total storage space available.

### 3. Replication Strategy

In figure 2, we present our replication strategy to minimize the data missed rate. In order to minimize the FMR and BMR in equations (2) and (3), our strategy makes the replica and placement decisions based on the benefits received from replicating the file in the long term. If the requested file is not at the site, it is replicated at the site if there is enough storage space. If there is not enough free space to store the replica, an existing file must be replaced and the strategy must weigh the benefits of storing the file with the cost of deleting a file.

The strategy uses the value  $V_i$  which is calculated as follows. We assume we have a file  $f_i$  in storage at instant  $t$ , and associate such a file with a value  $V_i$ , which will be set to the number of times this file will be accessed in the future. For some access pattern, based on the file access history, we can make a prediction of the  $V_i$ . We will not go deeply into an analysis of the prediction function here; in [12] the details of the prediction function have been discussed.  $V_i$  predicts the future access times of a file.

A weight is then calculated based on: 1)  $V_i$ , 2) the availability of the file, 3) the size of the file and 4) the number of copies of the file. We note that for hot data,  $V_i$  will be higher than for cold data. Since the weight calculation considers  $V_i$  as well, the weight of the file will be greatly affected by the file's popularity. The files with the smallest weight are considered for deletion first. Files are deleted and replaced by the new replica, only if the replica's value is greater than the value of the loss resulting from deleting the existing files.

Replication Strategy ():

1. if (needed file ( $f_i$ ) of the request operation isn't at the site) // Step 1  
     Get the file from other sites
2. if the SE.freeSpace >  $f_i$ .Size()  
     Store  $f_i$  in this sites
3. else  
     {     for each file in the SE  
         predict the  $V_i$  (the future access times of this file)  
         //this prediction function has been discussed in [12]  
         calculate the file weight  $w_j = (P_j * V_j) / (C_j * S_j)$   
         sort the file in the this SE by the file weight in ascending order  
         accumulativeSpace = 0  
         accumulativeLoss = 0  
         while (accumulativeSpace <  $f_i$ .size)  
             {  
                 Get File  $f_{potential}$  from the head of the sorted list  
                 //the file must not be locked or master file.  
                 Delete it from the sorted list  
                 AccumulativeSpace+ = File.size.  
                 AccumulativeLoss =  $(P_j - P_j') * V_{potential}$   
                 // the  $P_j$  is this file's current availability  
                 // the  $P_j'$  is this file's availability if it delete from this SE  
                 Add  $f_{potential}$  into the deletableList  
             }  
         }  
     }
4.  
      $f_i$ 's value =  $(P_i' - P_i) * V_i$                       // Step 2  
     //  $P_i'$  is the file  $f_i$ 's availability if  $f_i$  is replicated in the SE.  
     //  $P_i$  is the file  $f_i$ 's current availability  
     if  $f_i$ 's value > AccumulativeLoss              // Step 3  
         delete all the file in the deletableList  
         store  $f_i$
5. update the history log.

Figure 2. Replication strategy.

The strategy takes three steps to conduct the replica decision. First, if the requested file exists in the SE, it will not be replicated again. Second, the potential candidates will be chosen according to their weight. This is valuable for long term consideration, since a less valuable file will be replaced by a more valuable file. Thirdly, the strategy guarantees the replica gain will be greater than the replacement loss. The complexity of this algorithm is  $O(n \log n)$ .

#### 4. Results

We evaluate the performance of our replica and replacement strategy using the OptorSim, which was developed by the EU DataGrid Project [11] to test dynamic replica schemes. The OptorSim consists of several parts: the CE components, SE components, resource brokers and Replica Optimizers. We implement our replica strategy into the OptorSim by adding the new strategies to Our Optimizers. Our simulations are done on the EDG test bed Grid and the network topology of the EDG test bed (Figure 3). As shown in figure 3, we assume in our simulation there are a CE and SE at each site except the CERN site (see [11] for further details). Initially, we assume that all the master files are stored at the CERN site, which has a huge SE

but not a CE. The replica managers at each site will then decide when and how to make the replications based on the weighted value for a file. As a result, the files will eventually be distributed across the Grid.

We assume there are 200 files in the Grid and we simulate 10000 jobs in the Grid to measure the FMR and BMR. Each job accesses 3 ~20 files and each file size is 1G. We completed the simulation on a Dell desktop with 2.8G CPU and 1G RAM.

We compare the performance of our replica strategy, denoted as Repl, for the metrics FMR, BMR and the Total Job Time. We compare our replica schemes to the least-frequently-used strategy (LFU) and the economical-model [2], denoted as Eco. Specifically, we compare the performance of five replica schemes: LFU, EcoBio, EcoZipf, ReplBio and ReplZipf, where Bio indicates the Optimizer uses the Binomial-based prediction function and Zipf indicates the Optimizer uses the Zipf-based prediction function. We consider the four access patterns: Random, Gaussian, Sequential and Zipf.

Figure 4 illustrates the FMR for these five replica schemes and four access patterns. As illustrated in figure 4, our replica schemes ReplBio and ReplZipf perform better than the EcoBio

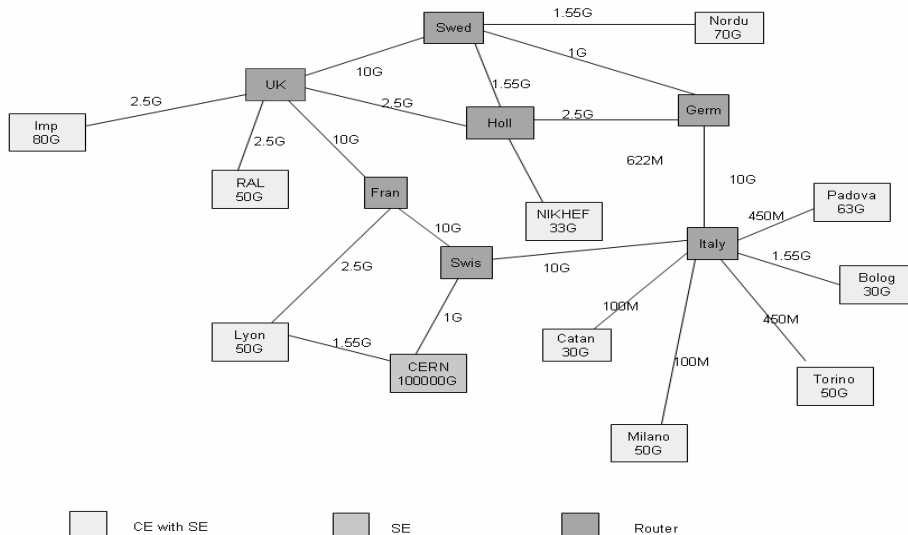


Figure 3. Grid Topology in the simulation.

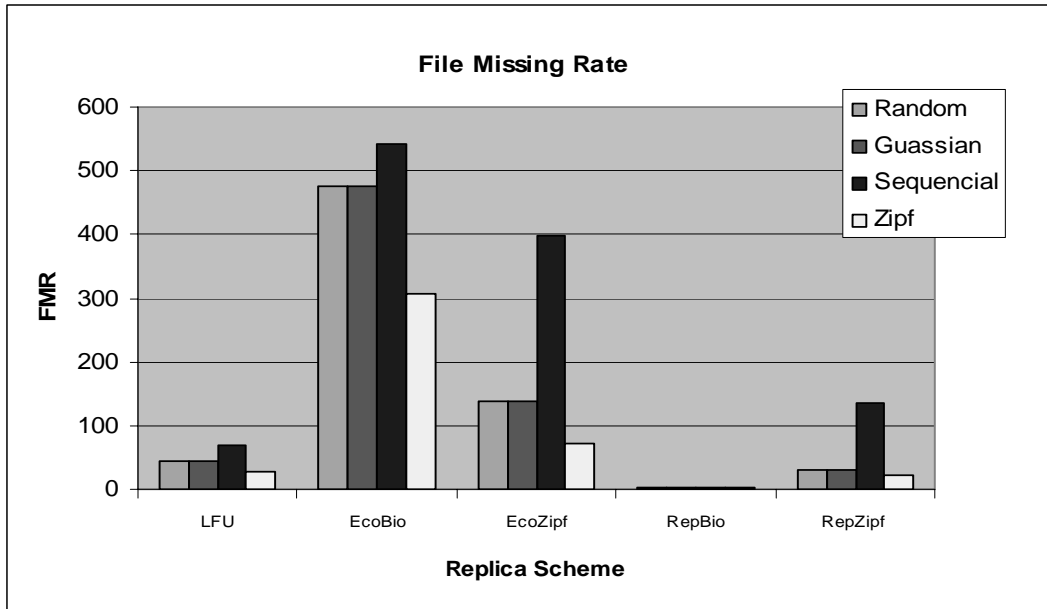


Figure 4. The File Missing Number with varying replica optimizers

and EcoZipf replica schemes for all access patterns. They have the lowest FMR values. The ReplBio and ReplZipf perform better than the LFU for all access patterns, except for the sequential access pattern.

In figure 4, the ReplZipf does not have the lowest missing rate for a sequential access pattern when compared to LFU. In our two optimizers, the replica manager decides to make the replica only when the gain of the value from the replicated file is greater than the loss of the value of the replaced file. However, the OptorSim Zipf prediction function is not as accurate for the sequential access pattern, so the calculation of the file weight has some incorrectness. Since the ReplZipf optimizer uses this prediction to calculate the weight of a file, the ReplZipf does not work as well as ReplBio.

Figure 5 illustrates the total job time (in seconds) for the five replication schemes for sequential access. The ReplBio and ReplZipf replica schemes have the shortest total job time when compared to the EcoBio, EcoZipf and LFU replica schemes. Even though the ReplZipf has a higher FMR than LFU, ReplZipf still has a shorter total job time than LFU, as illustrated in figure 5.

In our simulation, we measure the system bytes missing rate using the size of the missed file. We assume all the file sizes are 1G. Since we use Gigabytes as the measurement unit when we measure the bytes missing, the File Missing and

Bytes Missing rates will have the same value as those in figure 4. Hence, we can state that the ReplBio and ReplZipf in the simulation display a better performance on the system bytes missing rate as well.

## 5. Conclusions and further work

We have introduced two metrics to directly measure the availability of the data in a Data Grid system. With the assumption that the Grid storage space is limited, we have also developed a replica strategy that makes decisions about which data to replicate to take advantage of the limited storage resource to make the data available. Our replica strategy treats the hot and cold data file differently. It uses a weighting factor and a prediction function for the replacement scheme. We evaluated our strategy and demonstrate that our new strategy will outperform all others in most cases.

In future work, we plan to study how to enhance our algorithms so that we can differentiate the file missing rate and bytes missing rate in the Grid when the file size is not unique. This paper is our initial step in understanding how the replica scheme will affect the system availability. We believe more and more intelligent schemes will be proposed for Grid systems to make them more reliable and available.

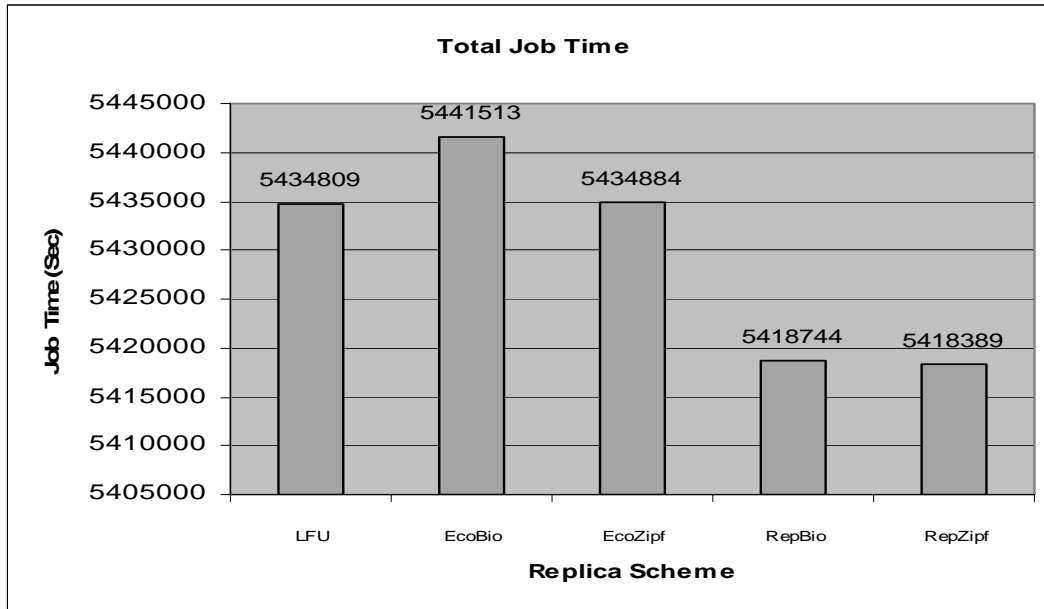


Figure 5. The Total job time with varying replica scheme, sequential access

## References

- [1] GriPhyN: The Grid Physics Network Project <http://www.griphyn.org>
- [2] Kavitha Ranganathan and Ian Foster.: Identifying Dynamic Replication Strategies for a High Performance Data Grid. International Workshop on Grid Computing, Denver, November 2001.
- [3] William H. Bell, David G. Cameron, Ruben Carvajal-Schiaffino, A. Paul Millar, Kurt Stockinger, and Floriano Zini.: Evaluation of an Economy-Based File Replication Strategy for a Data Grid. In International Workshop on Agent based Cluster and Grid Computing at CCGrid 2003, Tokyo, Japan, May 2003. IEEE Computer Society Press.
- [4] Mark Carman, Floriano Zini, Luciano Serafini, and Kurt 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.
- [5] Michal Szymaniak, Guillaume Pierre and Maarten van Steen.: Latency-Driven Replica Placement. 2005 Symposium on Applications and the Internet (SAINT'05) pp. 399-405.
- [6] T. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *21st IEEE INFOCOM Conference*, June 2002.
- [7] Sang-Min Park, Jai-Hoon Kim, Young-Bae Ko, and Won-Sik Yoon: Dynamic Data Grid Replication Strategy based on Internet Hierarchy. Second International Workshop on Grid and Cooperative Computing(GCC'2003) in Shanghai, China, Dec 2003.
- [8] F. Schintke, A. Reinefeld. Modeling Replica Availability in Large Data Grids. *Journal of Grid Computing* V1, N2. 2003, Kluwer Publishers.
- [9] Kavitha Ranganathan, Adriana Iamnitchi and Ian Foster: Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities, Proceedings of the Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems, Berlin, May 2002.
- [10] OptorSim – A Replica Optimizer Simulation: <http://edg-wp2.web.cern.ch/edgwp2/optimization/optorsim.html>
- [11] EU Data Grid Project: <http://www.eu-datagrid.org>
- [12] William H. Bell, David G. Cameron, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini: OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *International Journal of High Performance Computing Applications*, 17(4), 2003.