Online Grid Replication Optimizers to Improve System Reliability

Ming Lei ,Susan V. Vrbsky and Zijie,Qi

Department of Computer Science
University of Alabama
Tuscaloosa, AL 35487-0290
{mlei, vrbsky,zqi}@cs.ua.edu

Abstract

In a data intensive Grid system, many data replica schemes and models have been proposed to improve the system response time or data consistency, but little attention has been paid to the system reliability. In this paper, we investigate how these data replica schemes will impact the system reliability. We use several metrics of system reliability we previously proposed (System Bytes Missing Rate and System File Missing Rate), and we model the system availability problem assuming limited replica storage and different sized files. In order to achieve higher system data availability, in this paper we propose two new replica optimizers, MinDmr-β and MinDmr-γ, to minimize the Data Miss Rate (MinDmr). A comparison of our replica optimizers using a simulation on the OptorSim demonstrates that by utilizing our strategy, we can increase the data availability for files with different sizes.

1. Introduction

Many of today's scientific applications, such as high energy physics [16] and the Compact Muon Solenoid (CMS) Data Grid [14], will generate huge amounts of data. Millions of files will be generated from these scientific experiments and researchers around the world need access to this data. How to make the data more accessible and available in such a large data set that is distributed in different geographic locations is very challenging, and has attracted a great deal of interest.

Data replication in a Data Grid is the most common solution to improve file access time and availability. Earlier work on data replication [1-6] placed most of the attention on decreasing the data access latency and the network bandwidth assumption. Other researchers have proposed some data replica schemes or models for maintaining the data consistency [17]. In many Data Grid systems, update operations will not occur often. At the same time, as bandwidth and computing capacity have become relatively cheaper due to hardware technology advances, the data access latency can drop dramatically, and how to improve the system reliability and availability become the focus.

The dynamic behavior of a Grid user makes it difficult to make decisions concerning data replications to meet the system availability goal [7]. In a Data Grid system, there are hundreds of clients physically distributed across the globe who will submit their job requests. Usually, a Grid job will access multiple files to do some type of analysis. In data-intensive applications, when a job accesses a massive-size file, the unavailability of that file can cause the whole job to hang up and the potential delay of the job can be unbounded. In large-scale data-intensive systems, hundreds of nodes will be involved and any node failure or network outage can cause potential file unavailability. As a result, there has been an increase in research focusing on how to maximize the file availability. Data replication strategies to improve the data availability have been proposed [7, 8], but have assumed unlimited storage for replicas.

To the best of our knowledge, the system data availability metrics, System Bytes Missing Rate and System File Missing Rate, we presented in [15] are the only such metrics which have been proposed for Data Grid Systems, thus far. We briefly describe these two metrics to measure the data reliability in the Data Grid. We model the system availability in the system where the storage space is limited and file size varies. In this paper, we propose two new greedy replica optimizers, MinDmr-β and MinDmr-γ, and we also describe our previously proposed replica optimizer MinDmr-α. For the replica optimizers, we introduce the file weight and prediction functions and discuss the details of how and why they work. In [15] we demonstrated that MinDmr-α performed better than existing optimizers for same sized files for the System File Missing Rate. In this paper, our test results using the OptorSim [9] show that our three replica schemes can increase the data availability for files with different sizes as measured by the System Bytes Missing Rate and System File missing Rate.

The rest of the paper is organized as follows. We describe related work in Grid systems in Section 2. Section 3 provides simple introductions of the two metrics: the system file missing rate and the system bytes missing rate, and a discussion of the system model. We

1-4244-0910-1/07/$20.00 ©2007 IEEE

present our analytical model and the new dynamic replica algorithms in Section 4. In Section 5, we describe our simulation results based on the OptorSim, a simulator designed by the European Data Grid Project [10]. In Section 6, we conclude and describe future work.

2. Related Work

Work on data availability in Grid systems initially focused on decreasing the data access latency and the network bandwidth assumption. In [1], the six replica strategies: No Replica, Best Client, Cascading Replication, Plain Caching, Caching plus Cascading Replica and Fast Spread are simulated for the three user access patterns: random access, small temporal locality, and small geographical and temporal locality. The simulation results show that the best strategy has significant savings in latency and bandwidth consumption if the access patterns contain a moderate amount of geographical locality. In [2, 3], a replica scheme based on the economical-model has been proposed. The authors use an auction protocol to make the replica decision for long-term optimization. They show the scheme outperforms other replica strategies with sequential file access patterns.

In [4], Szymaniak et al. present the HotZone algorithm to place replicas in a wide-area network, so that the client-to-replica latency is minimized. They use the GNP [5] technique to model the Internet as an M-dimensional space and all nodes are landmarked into different network regions. Hotzone places replicas on nodes that, along with their neighboring nodes, generate the highest load. Sang-Min Park et al. in [6] propose a dynamic replica replication strategy, called HBR, to reduce data access time by avoiding networking congestion in a Data-Grid network. The HBR algorithm benefits from 'network-level locality', which indicates that the required file is located at the site which has the broadest bandwidth to the site of the job execution.

More recent work has focused on maximizing file availability in a Grid system. Schintke and Reinefeld present an analytical model in [7] for determining the optimal number of replica servers, catalog servers and catalog sizes to guarantee a given overall reliability in the face of unreliable components. In [8], Ranganathan shows a dynamic model-driven replication approach in which peers create replicas automatically in a decentralized fashion. Both [7] and [8] propose algorithms 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 arbitrary number of copies needed to achieve its availability goal without any discrimination, even if the file will be accessed only one time in its entire life span.

3. Data availability

While the previous measures of mean available bandwidth and percentage of computer usage are important to the overall functioning of an efficient Grid system, the Grid user is concerned with completing a job with correct data. Any data file access failure can lead to an incorrect result or a job crash. To protect the user from such risk, the Grid system will be compelled to make the data availability as high as possible. In our previous work [15], we introduced two metrics: the System Data Missing Rate and the System File Missing Rate, to measure how data-reliable the system is as follows:

System File Missing Rate SFMR- represents the ratio of the number of files potentially unavailable and the number of all the files requested by all the jobs.

S*ystem Bytes Missing Rate SBMR* - represents the ratio of the number of bytes potentially unavailable and the total number of bytes requested by all jobs.

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

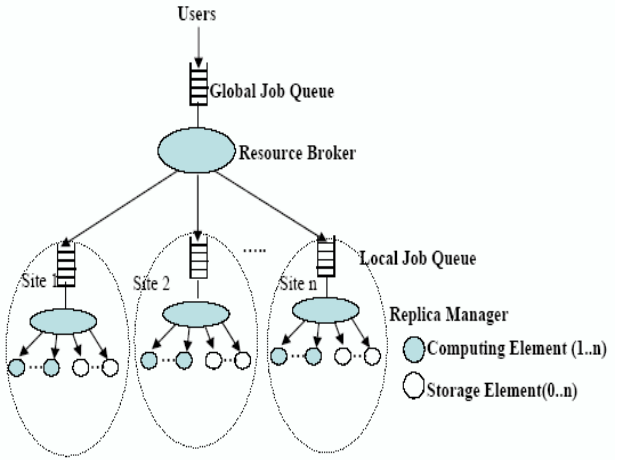

Figure 1. Simulated Data Grid Architecture

Based on the data grid architecture in Figure 1, we define the SFMR (*System File Missing Rate*) as:

$$\mathrm{SFMR} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{k}(1-P_j)}{\sum_{i=1}^{n}k_i} \qquad (1)$$

where *n* denotes the total number of jobs, each of which will access *k* different files (*k* file access operations). P_j indicates file f_j*'s* availability, and is defined as $P_j = 1 - \prod_{s=1}^{k}(1-P_s)$, where the P_s denotes the file availability in storage element s, and k is the number of storage elements where the file is located.

In the same way, we define the SBMR (S*ystem Bytes Missing Rate*) as:

$$SBMR = \frac{\sum_{i=1}^{n}\sum_{j=1}^{k}(1-P_j)*S_j}{\sum_{i=1}^{n}\sum_{j=1}^{k}S_j} \quad (2)$$

where n, k and P_j have the same meaning as in equation (2) and S_j denotes the size of file f_j. From equations (2) and (3), it is apparent that the values for SFMR and SBMR will have a fixed relationship if all the accessed file sizes are the same.

3.1 Problem model

To better model the replica problem, which concerns each file access rather than an individual job process, without any loss of generality, we can then model any job request into multiple file request operations. Suppose there is a sequence of file requests stream $O = (r_1, r_2, r_3 \ldots, r_N)$, with no assumptions about whether all of these file requests are distinct or not. For any file request r_i, it will associate a file f_i, with an instant availability of P_i, so it will contribute to the SFMR and SBMR with (1- P_i) and (1- P_i) * S_i, respectively. Hence, $SFMR = \sum_{i \in o}(1-P_i)/|O|$ and $SBMR = (\sum_{i \in O}(1-P_i)*S_i)/\sum_{i \in O}S_i$. The best system data availability results from minimizing the SFMR and SBMR above, subject to $\sum_{i=1}^{w} C_i * S_i \le \mathbb{Z}$ at any time, where w denotes the number of distinct files stored in the data grid at any instant, C_i denotes the number of copies of f_i, S_i denotes the size of the f_i and $\mathbb{Z}$ is the total storage available.

We can achieve the optimal solution to minimize the SFMR and SBMR by some off-line algorithms if we are aware of the future file request information. However, for any replica manager at the specified instant T, only the short term file requests, which are queued in the job queue, can be observed. For this online optimization problem, we will introduce some heuristics based on the file weight, which will be discussed in later sections.

3.2 File Value V_i

For these online optimization problems, the replica manager can not make any decisions on whether or not to replicate the current requested file based only on its local storage element status and the global file catalog, which record the file distribution information in the Data Grid system. In order to make a positive decision regarding long term performance, for every file in the system, we can predict its future value via the prediction function. For each file access operation r_i, at instant T, we associate it with an important variable V_i, which will be set to the number of times this file will be accessed in the future, i.e. ≥ T. Such a value cannot be calculated, but predicted as we just mentioned.

In this paper, we will try to make such a prediction via four kinds of prediction functions:

- No Prediction: We make no predictions of the file at instant T, which means the V_i will always be 1. It is the same strategy as in [8].
- Bio Prediction: As in [13], V_i is based on the file access history to predict the value of the file by a binomial distribution.
- Zipf Prediction: Same as in [13], V_i is based on the file access history to predict the value of the file by a Zipf distribution.
- Queue Prediction: The current job queue is used to predict the value of the file. If the queue is empty, this Queue Prediction function will work the same as No Prediction.

The value of V_i is dependent on the assumptions made about future file access. Therefore, we have chosen the above four functions to represent a range of possibilities for V_i and will be used to study the effectiveness of our strategy under varying assumptions.

For the later three prediction functions, the value of the file will be predicted based on either the access history or the job queue. Due to the fact that each site has its own file access history and job queue, in order to predict the file access times in the future, there are two options that can be chosen. Either the prediction can only consider the local file access history and local job queue, or both the local and remote sites will be taken into account. In current Grid systems, we cannot assume that there will be a central control point where all the file accesses or each sites' job queue is maintained. The more commonly used structure is as described in Figure 1, which is used in the European Data-Grid project [10]. In such an architecture, when predicting the future access time of one specified file, we can send the file information to all the sites and request each site return their prediction back to the requester. In this way, we can achieve Global Prediction. We believe that it is a tradeoff to choose local prediction vs global prediction, as illustrated in the Table 1.

4. Online optimization algorithms

We can restate the online replication problem on each site in the following way:

At any given time T, given a fixed space $\mathbb{Z}$ (sum of the size of the site's storage elements SE), for all the files stored in site F = {f_1, f_2,,,f_n) and for each file f_i in the storage, we associate a size s_i and v_i. Now assume the new requested file is τ, which is also associated with

Local Prediction	Pros:	. Easy to access the information needed. . Easy to implement.
	Cons:	. The prediction result is not as accurate as with global prediction.
Global Prediction	Pros:	. The prediction result is more accurate.
	Cons:	. Brings big prediction overhead. . Messaging delay will slow down the performance of replica manager. . Multiple points of failure. . Scalability issues. . Long delay in the prediction will also decrease the accuracy of the result.

Table 1 Local Prediction vs Global Prediction.

a size and value v_τ. Next we choose a file set $\delta = \{f_1, f_2, .., f_k\}$ from the file set $\{F\} + \{\tau\}$ to achieve the maximum $\sum_{i=1}^{k} P_i * V_i$ or $\sum_{i=1}^{k} P_i * S_i * V_i$. If τ is in δ, then we need to replicate the file.

The above optimal problem is a classic *Knapsack* problem by consideration that we can aggregate each file replicas storage costs together as the "*weight*" of the item `i` (in our problem, it is f_i). In [12], the relationship between the *Knapsack problem* and the *fractional knapsack problem* in the cache optimization problem is discussed. Based on the same ideas as those presented in [12], we can also convert our optimization problem to an approximate *fractional knapsack problem* because the amount of space left after storing the maximum number of files is negligible compared to the total storage space.

4.1 MinDmr-α replica optimizer

We now present three heuristic optimizers to optimize the system reliability. The first optimizer is MinDmr-α, which makes replica and placement decisions based on the benefits received from replicating the file in the long term as the file missing rate is concerned and first appeared in [15] as MinDmr. In our greedy algorithm, we introduce the file weight as:

$W_\alpha = (P_j * V_j) / (C_j * S_j)$

where the `P,V,C,S` have the same meaning as in the previous sections. The optimizer will sort the current file in the storage element based on the file weight in ascending order, then make the replication decision based on each file's value. The algorithm appears in Figure 2.

MinDmr-α ():
Input: $R = \{f_1, f_2, .., f_n\}$, $C = \{c_1, c_2, .., c_n\}$, $\mathbb{Z}$ and τ.
Where $\mathbb{R}$ is the file set stored in the current store element, C is the size of file $\mathbb{R}$, $\mathbb{Z}$ is the total space of the current store element, τ is the new request file.
Output: ψ = yes/no
{ if $\tau \in \mathbb{R}$ then
ψ = yes
return ψ // file τ exists in current storage
if $\tau \notin \mathbb{R}$ && ($\mathbb{Z}$ -sum(C)) ≥ size of(τ) then
ψ = yes // if free space is large enough to store
return ψ // the new file, replicate it.
else
{ Sort the files in $\mathbb{R}$ by W_α in ascending order
$\Delta = \{\}$
Repeat
{ If file f is deleteable then
$\Delta = \Delta \cup \{f\}$
} Until size of sum(Δ) ≥ size of(τ)
If (value-gained-α by replicating file > accumulative value-loss-α of removing Δ)
ψ = yes
else
ψ = no
// value-gained-α = $P_{incr} * V_\tau$,
// accumulative value-loss-α = $\sum_{i \in \Delta} P_{desc} * V_i$
return ψ }
}

Figure 2. MinDrm-α algorithm pseudo-code

4.2 MinDmr-β replica optimizer

We now propose a new optimizer named MinDmr-β, which will replicate data based on the file weight:

$W_\beta = (P_j * V_j * S_j) / (C_j * S_j) = (P_j * V_j) / C_j$

where the P, V, C, S have the same meaning as in the previous section. The weight is calculated based on: 1) file availability, 2) number of times of future accesses of this file, and 3) number of copies of the file. File size is not a factor. In MinDmr-β, the main logic is the same as in MinDmr-α, but there are two differences between them. One is that the file weight used to sort the files in the storage element is W_β, the other is that the value-gain and accumulative value-loss is: *value-gain-β=$P_{incr} * V_\tau * C_\tau$ and value-loss-β = $\sum_{i \in \Delta} P_{incr} * V_i * C_i$*, respectively.

4.3 MinDmr-γ replica optimizer

The two optimizers MinDmr-α and MinDmr-β make replication decisions based on whether the replication will

result in some gain in value (ΔP_i *V_i and ΔP_i *V_i * S_i, respectively). Therefore, it is straightforward to combine these two ideas to get one new replica optimizer that determines whether to replicate the file based on either the gain in the file missing value or a gain in the bytes missing value. In a similar way, we define the file weight:

$$W_\gamma = W_\alpha * W_\beta$$

We argue that when ranking the file based on the W_γ, we can achieve a balance on the file missing and bytes missing. Therefore, it is straightforward to obtain the MinDmr-γ optimizer algorithms by replacing the file weight as W_γ and basing the replication decision on either (value-gain-α where replicating file > accumulative value-loss-α of removing Δ) or ((value-gain-β where replicating file > accumulative value- loss-β removing Δ).

4.4 Discussion of optimizer

We have addressed the three replica optimizers: MinDmr-α**,** MinDmr-β and MinDmr-γ. In all the three optimizers, we associate each file with a weight W to incorporate the important aspects for determining data availability. For example, in W_α we consider not only the availability of the file but also the number of times the file will be referenced in the future. Similarly, we consider not only the size of a file, but the number of copies stored.

Since the weight calculation considers V_i, the weight of the file will be greatly affected by the file's popularity. We note that for hot data, V_i will be higher than for cold data. The files with the smallest weight (cold data) are considered for deletion first, so the hotter the file, the more possible it will be replicated.

As shown by the pseudo code for MinDmr-α**,** MinDmr-β and MinDmr-γ, the optimizers will take four steps to conduct the replica decision. First, if the requested file already exists in the storage element SE, it will not be replicated again. As mentioned previously, multiple copies of the file in the same SE cannot improve the data availability [15]. Instead, it can potentially lessen the system level data availability because such replication will waste the storage space in the SE. Second, when the free storage is large enough to hold the requested file, then the replication of the file will always be conducted. Thirdly, when there is not enough storage, the potential candidates for replacement will be chosen according to their file weight. This is valuable for long term consideration, since a less valuable file will be replaced by a more valuable file. The fourth, and the most difficult step, is to guarantee the replica gain will be greater than the replacement loss.

When choosing the potential replacement candidates, we note there are some files which will be not considered, and there are two kinds of possibilities. First, the file may be the master file. The master file is the original file in the system, which cannot be deleted. Secondly, the file may be pinned, which means it is being accessed by another job. There is more than one computing element CE in the Grid, which may lead some files to be accessed by multiple jobs instantaneously. The replica candidates must not only be the least valuable files, to ensure sure each replica will gain some value, but we must also eliminate files not under consideration.

The complexity of this algorithm is O(n log n), where n denotes the number of the files stored in this site, if we choose a good sorting algorithm to sort the files by their weight.

5. Performance Results

We evaluate the performance of our MinDmr replica and replacement strategy using the OptorSim, which was developed by the EU DataGrid Project [10] 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 as shown in Figure 3. We assume in our simulation there are a CE and SE at each site except the CERN site. The corresponding storage size and the bandwidth between two sites are marked in Figure 3. 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 different files in the Grid and we simulate 10000 jobs in the Grid to measure the SFMR and SBMR. Each job accesses 3~20 files. As illustrated in Figure 3, the storage available at an SE ranges from 100M to 10G. The job scheduler strategy is based on Queue Access Cost. In the first part of the experiments, we will simulate where the file size of each file is the same, at 1G in our setting, and we can say that the SFMR will be equal to SBMR. In the second part of the simulation, the file size of these 200 files will vary from 8M to 2G, which will lead to the unequal SFMR and SBMR. We completed the simulation on a Dell desktop with 2.8G CPU and 1G RAM.

5.1 Results for equal size files

This experiment demonstrates the performance of our replica optimizers for the metric SFMR when the sizes of the files are all the same. We compare our MinDmr optimizer to the existing strategies of the least-frequently-used LFU strategy and the EcoBio and EcoZipf of the economic model in the OptorSim [11]. In the economical

model (denoted as Eco), a file is replicated if it results in maximizing the profit of an SE. Further details of the Eco evaluator function can be found in [11] and differences between the Eco optimizer and MinDmr appear in [15].

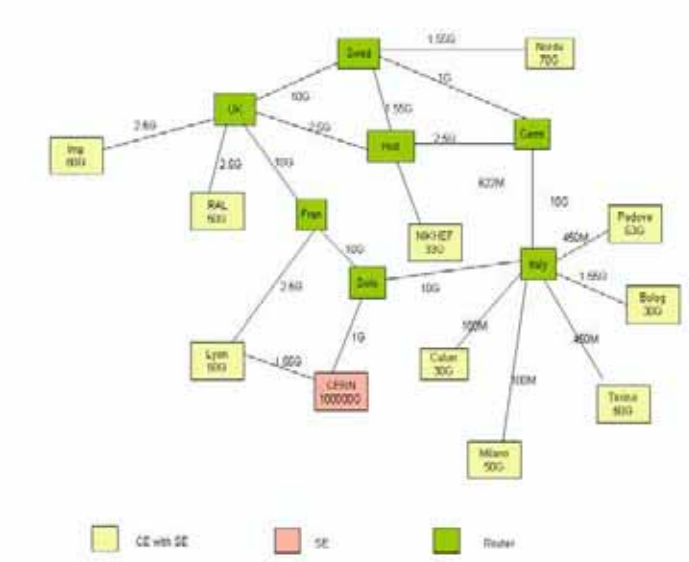

Figure 3. Grid Topology in the simulation

We compare the three existing replica schemes: LFU, EcoBio and EcoZipf, with our four replica schemes: BioMinDmr, ZipfMinDmr, MinDmrNoPred and MinDmrQueuePred, which utilize the EcoBio Prediction function, EcoZipf Prediction function, No Prediction function and Queue Prediction function, respectively. We have discussed the details of these four prediction functions in Section 3.2. We do not go deeply into an analysis of the EcoBio and EcoZipf predictions function here (details appear in [11]).

The order of the files requested is determined by the access pattern, and we consider four access patterns: Random, Random walk Gaussian, Sequential and Random Walk Zipf.

Figure 4 shows the SFMR using the MinDmr-α optimizer for the seven replica schemes and four access patterns. As illustrated in Figure 4, our replica schemes BioMinDmr, ZipfMinDmr, MinDmrNoPred and MinDmrQueuePred, perform better than the EcoBio and EcoZipf replica schemes for all access patterns. The EcoBio has the highest SFMR and the EcoZipf has the second highest SFMR for all access patterns. The SFMR for the Eco strategies is up to 200 times greater than the MinDmr-α strategies. In fact, the SFMR values for the BioMinDmr, MinDmrNoPred and MinDmrQueuePred are too small to see in Figure 4. LFU has lower rates than the two Eco strategies but it has a higher SFMR than the four MinDmr strategies, except for the ZipfMinDmr with a sequential access pattern. Figure 4 illustrates the MinDmr-α strategy performs better than the Eco and LRU strategies, even when there is no prediction function. This indicates the MinDmr is not dependent on the prediction function used.

The ZipfMinDmr does not have a lower missing rate for the sequential access pattern when compared to LFU. The reason lies in that our MinDmr replica managers decide 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 EcoZipf prediction function is not as accurate for the sequential access pattern, so it brings some inaccuracy into the calculation of the file weight. This in turn, will cause the replica scheme to fail to work as well as the other three MinDmr replica schemes for the sequential access pattern.

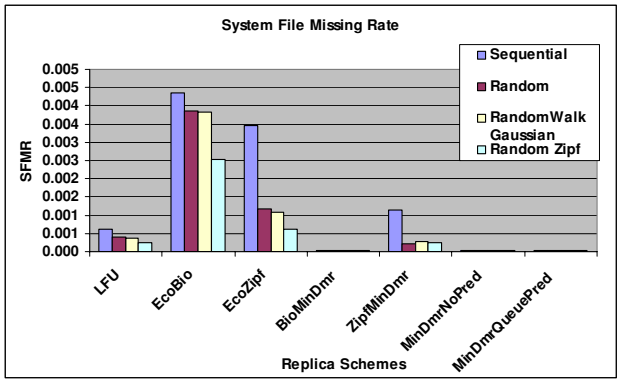

Figure 4. The SFMR with varying replica optimizers

5.2 Non-equal file sizes

In the following experiment, the sizes of the files in the grid are different and will vary from 8M to 2G. In this non-equal file size simulation, we only focus on the four schemes we designed, BioMinDmr, ZipfMinDmr, MinDmrNoPred and MinDmrQueuePred, and compare how the optimizers MinDmr-α, MinDmr-β, MinDmr-γ perform. All results for SFMR and SBMR are 0.0000001 per unit and are displayed in table format for readability.

Table 2 demonstrates the results from the different optimizers for the four replica strategies for a Random Gaussian access pattern. Table 2, indicates that the optimizer MinDmr-β can help to reduce the SBMR, which matches the weight and value-gain definition in MinDmr-β, expect for the ZipfMinDmr. We also find that the MinDmr-γ works best for both SFMR and SBMR in almost all the strategies (lowest values are indicated in bold). In the MinDmrNoPred strategy, the optimizer MinDmr-β achieves the best in both metrics, which gains almost 45% more than MinDmr-α and 10% more than MinDmr-γ for both the SFMR and SBMR. The MinDmrQueuePred with MinDmr-γ works best in the random Gaussian access pattern for all the replica policies. Another fact that should be mentioned is

Table 2. SFMR and SBMR with Random Gaussian access pattern (DIFF = SBMR –SFMR)

Optimizer	Replica Schemes											
	BioMinDmr			ZipfMinDmr			MinDmrNoPred			MinDmrQueuePred		
	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF
MinDmr-α	22.89	24.10	1.21	276.53	332.56	56.03	35.55	40.84	5.29	22.07	23.28	1.21
MinDmr-β	22.89	23.12	0.22	304.28	366.58	62.31	**22.86**	**21.43**	-1.43	21.14	21.23	0.09
MinDmr-γ	**21.11**	**21.58**	0.47	**225.93**	**247.57**	21.64	27.05	25.18	-1.87	**19.68**	**20.07**	0.39

Table 3. SFMR and SBMR with Sequential access pattern (DIFF = SBMR –SFMR)

Optimizer	Replica Scheme											
	BioMinDmr			ZipfMinDmr			MinDmrNoPred			MinDmrQueuePred		
	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF
MinDmr-α	21.3873	22.18	0.79	774.6	999.4	225	32.39	34.05	1.66	23.07	24.65	1.58
MinDmr-β	21.3361	21.52	0.18	913.1	1192	279	25.03	23.74	-1.3	22.72	22.73	0.01
MinDmr-γ	**20.5349**	**20.69**	0.16	**728.6**	**945.3**	217	**21.64**	**21.26**	-0.4	**21.68**	**22.24**	0.55

Table 4. SFMR and SBMR with Random access pattern (DIFF = SBMR –SFMR)

Optimizer	Replica Schemes											
	BioMinDmr			ZipfMinDmr			MinDmrNoPred			MinDmrQueuePred		
	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF
MinDmr-α	20.65	21.33	0.68	227.83	257.27	29.44	31.70	34.13	2.43	21.25	22.82	1.56
MinDmr-β	***20.88***	**20.90**	0.02	352.53	437.86	85.34	**22.86**	**21.88**	-0.98	**21.10**	**20.89**	-0.21
MinDmr-γ	23.07	24.43	1.36	**196.73**	**226.19**	29.46	26.58	25.46	-1.13	21.12	21.59	0.47

Table 5. SFMR and SBMR with Zipf access pattern (DIFF = SBMR –SFMR)

Optimizer	Replica Scheme											
	BioMinDmr			ZipfMinDmr			MinDmrNoPred			MinDmrQueuePred		
	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF	SFMR	SBMR	DIFF
MinDmr-α	**23.2116**	**24.99**	1.78	240.7	314.2	73.5	32.1	35.13	3.03	21.53	22.66	1.13
MinDmr-β	24.2965	25.14	0.85	220.1	254.8	34.7	23	22.3	-0.7	21.38	21.24	-0.1
MinDmr-γ	24.2165	25.28	1.06	**170.4**	**188.7**	18.3	**20.72**	**20.69**	-0.01	**20.68**	**20.77**	0.09

that the difference (DIFF) between the SFMR and SBMR always exists. This is because all of the replica algorithms prefer to store small-size files in the replica space, which will result in the large-size files always being replaced by smaller ones. This disparity varies with different replica schemes.

For the strategy ZipfMinDmr, the MinDmr-β does not help decrease the SBMR but instead it increases the both the SFMR and SBMR. This contradicts the file weight and value-gain definition in MinDmr-β. The reason behind this is again believed to be the fact that the prediction function in the ZipfMinDmr does not work as well as other three, and this directly causes the optimizer to make the wrong decision. The prediction function is responsible for outputting the Vi, which is the key input of the file weight function.

In Table 3, we illustrate the simulation results for the sequential access pattern. For this pattern, MinDmr-γ improves all four replica schemes for both SFMR and SBMR. BioMinDmr with MinDmr-γ performs the best for this access pattern. As noted for the previous table, the ZipfMinDmr performs the worst for all the replica schemes and MinDmr-β does not help to improve the SBMR.

In Table 4, we list the results with a random access pattern. In most of the cases, the SBMR is larger than SFMR, but MinDmr-β can help shorten the DIFF and even make the SBMR less than SFMR. MinDmr-β can help to reduce the SBMR, which matches the weight and value-gain definition in MinDmr-β, with the exception of the strategy ZipfMinDmr. MinDmr-γ can help decrease both the SFMR and SBMR at the same time, but it performs worse for the BioMinDmr strategy. The DIFF is around 1~5% in BioMinDmr, MinDmrNoPred, and MinDmrQueuePred. However, in the ZipfMinDmr the SBMR is about 10~40% higher than SFMR.

Table 5 presents the simulation results for the Zipf access pattern. Compared to the previous three access

patterns, the DIFF is the smallest. The MinDmr-γ still performs the best except for in BioMinDmr, where the MinDmr-γ results in the smallest decrease in the data availability. ZipfMinDmr still performs the worst against the other three replica schemes as in other access patterns. Both MinDmrNoPred and MinDmrQueuePred with MinDmr-γ perform almost the same and are the best of all the replica policies.

In summary, MinDmr-α and MinDmr-β always help the replica strategies to improve the SFMR and SBMR, respectively, with the exception of the ZipfMinDmr, due to its poorly performing prediction function. The MinDmr-γ performs the best overall for all the replica policies and for all access patterns. There always exists a difference between SFMR and SBMR and, in general, SFMR will always be smaller than SBMR.

6. Conclusions and future work

We discussed how we model the system availability problem and how to transfer the data availability to an approximate fractional knapsack problem, a classic optimal problem. With the assumption that the Grid storage space is limited and different file size, we proposed three replica optimizers to take full advantage of the limited storage resource to make the data availability as high as possible.

We evaluate how our replica strategies work in the situation where the data file size is non-equal. We evaluated our two new replica optimizers for the four prediction functions and demonstrated that our new strategies perform well overall in terms of data availability. Results indicate the performance of MinDmr is better than others with varying prediction functions, job schedulers and file access patterns, as far as the data availability is concerned. The results demonstrate that the optimizer MinDmr-β always helps to improve the SBMR except for the ZipfMinDmr. The MinDmr-γ performs the best overall.

In future work, we plan to move on to the next step to improve the Quality of Service in a Data Grid with a service-orientation design. Instead of considering the data replication policy, we will turn to the job scheduler and improve the current scheduler to provide QoS to clients.

References

[1] Kavitha Ranganathan and Ian Foster.: Identifying Dynamic Replication Strategies for a High Performance Data Grid. International Workshop on Grid Computing, Denver, November 2001.

[2] 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, May 2003. IEEE Computer Society Press.

[3] 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.

[4] Michal Szymaniak,Guillaume Pierre and Maarten van Steen.: Latency-Driven Replica Placement. 2005 Symposium on Applications and the Internet (SAINT'05) pp. 399-405.

[5] T. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *21st IEEE INFOCOM Conference*, June 2002.

[6] 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 CooperativeComputing (GCC'2003) in Shanghai, China, Dec 2003.

[7] F. Schintke, A. Reinefeld. Modeling Replica Availability in Large Data Grids. Journal of Grid Computing V1, N2. 2003, Kluwer

[8] 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.

[9] OptorSim – A Replica Optimizer Simulation: http://edg-wp2.web.cern.ch/edgwp2/optimization/ optorsim.html

[10] EU Data Grid Project: http://www.eu-datagrid.org

[11] 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.

[12] E. Otoo E, A. Shoshani: Accurate modeling of cache replacement policies in a Data-Grid. Mass Storage Systems and Technologies, 2003. (MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference.

[13] David G. Cameron, Ruben Carvajal-Schiaffino, A. Paul Millar, Caitriana Nicholson, and Kurt Stockinger Floriano Zini. Evaluating Scheduling and Replica Optimisation Strategies in OptorSim. In 4th International Workshop on Grid Computing (Grid2003), Phoenix, Arizona, November 17, 2003. IEEE Computer Society Press.

[14] GriPhyN: The Grid Physics Network Project http://www.griphyn.org

[15] Lei, M, S. Vrbsky, X. Hong, "A Dynamic Data Grid Replication Strategy to Minimize the Data Missed," *3rd International Workshop on Networks for Grid Applications (GridNets 2006),* San Jose, CA, Oct. 2006.

[16] PPDG, http://www.ppdg.net

[17] Dullmann, D. Hoschek, W. Jaen-Martinez, J.Segal, B. Samar, A.Stockinger, H. Stockinger, K. Models for replica synchronisation and consistency in a data grid .In High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on.