Proc. 12th Heterogeneous Computing Workshop (HCW2003) Nice, France, April 2003, IEEE Computer Science Press, Los Alamitos, CA, 2003

Simulation of Dynamic Data Replication Strategies in Data Grids *

Houda Lamehamedi, Zujun Shentu, and Boleslaw Szymanski
Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180
lamehh, shentu, szymansk@cs.rpi.edu

Ewa Deelman
Information Sciences Institute, University of Southern California, Marina Del Rey, CA 90292
deelman@isi.edu

Abstract

Data Grids provide geographically distributed resources for large-scale data-intensive applications that generate large data sets. However, ensuring efficient access to such huge and widely distributed data is hindered by the high latencies of the Internet. We address these challenges by employing intelligent replication and caching of objects at strategic locations. In our approach, replication decisions are based on a cost-estimation model and driven by the estimation of the data access gains and the replica's creation and maintenance costs. These costs are in turn based on factors such as runtime accumulated read/write statistics, network latency, bandwidth, and replica size. To support large numbers of users who continuously change their data and processing needs, we introduce scalable replica distribution topologies that adapt replica placement to meet these needs. In this paper we present the design of our dynamic memory middleware and replication algorithm. To evaluate the performance of our approach, we developed a Data Grid simulator, called the GridNet. Simulation results demonstrate that replication improves dramatically the data access performance in Data Grids, and that the gain increases with the size of the datasets involved.

1. Introduction

A Data Grid connects a collection of hundreds of geographically distributed computers and storage resources located in different parts of the world to facilitate sharing of data and resources [2, 11, 10, 18]. Data Grids enable scientists from different universities and research laboratories to collaborate with one another to solve large-scale computational problems. The size of the data that needs to be accessed on the Data Grid is on the order of Terabytes today and will reach Petabytes in the near future.

The major barrier to supporting fast data access on a Grid are the high latencies of Wide Area Networks and the Internet. This barrier impacts the scalability of Grid systems. To overcome this barrier, large amounts of data need to be replicated in multiple copies at several world-wide distributed sites.

Replication is a well-established field in the distributed computing, Internet, and database communities. The Grid environment, however, introduces new and different challenges, such as high latencies and dynamic resource availability. Given the size of the data sets, it is impossible for a human to make decisions about where the data needs to be placed. Clearly, a good replication strategy is needed to anticipate and/or analyze user data requests and to place subsets of the data (replicas) accordingly.

Current technologies and initiatives use only static or user-driven replication services, to manually replicate data files. The Grid environment however, is highly dynamic. The resources availability and network performance change constantly and data access requests also vary with each application and each user. Two key challenges that need to be addressed are:

1. scalability, and
2. adaptability.

We address these challenges by employing intelligent replication and caching of objects at strategic locations. To this end, we have designed a dynamic memory middleware that allows Grid nodes to automatically replicate data when needed. To support large numbers of users that continuously change their data and processing requirements, we plan

*This research was supported in part by the IBM Fellowship Program, by a joint study fund from the IBM Almaden Research Center and by the National Science Foundation under Contract ITR-0086044 (GriPhyN). The content of this paper does not necessarily reflect the position or policy of the U.S. Government or IBM Corp.—no official endorsement should be inferred or implied.

to use dynamic techniques to adapt replica placement to changing users needs and requests. As a result, the replica management service must have a runtime component to dynamically monitor and evaluate the applicationss and userss needs and adapt the replica placement to meet these needs. Accordingly, new replicas may be added at different locations and deleted from locations where they are no longer needed.

In our approach, the replication runtime component evaluates the data access gains and compares them with the creation and maintenance costs of replicas, before moving or copying any data. The gains and costs of the replication decisions are calculated based on many factors, such as accumulated read/write statistics, network latency and bandwidth, replica size and system reliability. Our replication cost model is formulated as an optimization problem that minimizes the sum of the total access costs of data in a Grid and the replica creation and maintenance costs.

To address the scalability, we propose to overlay the replicated data in two alternative logical configurations or topologies: a ring and a tree. Both configurations can be organized on top of each other. The replica distribution spanning graph is chosen depending on the application's design and sharing patterns. The ring topology is used in a peer-to-peer replication approach in the presence of multiple master replicas. The tree topology on the other hand, exploits geographical locality and high bandwidth availability and provides a scalable expansion of the overall replica distribution topology. The collected Grid access patterns suggest that although data updates are infrequent in the Data Grid, they occasionally happen. It is thus necessary to guarantee that the updates are eventually propagated and that the users have access to consistent copies of the data. Our distribution topologies provide a scalable infrastructure to maintain replica consistency. In this work however, we only consider read only data.

To evaluate the benefits and applicability of our approach, we use a cost based analytical model and simulations. For that purpose, we developed a Data Grid simulation tool: GridNet [13]. This simulator provides a modular simulation framework through which we can model different Data Grid configurations and resource specifications. The simulations allow us to perform initial verifications of the design and evaluate the performance of our strategies. Our preliminary simulation results were presented first in [13], and were based on a trace file using static data replication. The contributions made in this paper are twofold: we present results for large network configurations and data files, and we introduce and use adaptive dynamic cost-driven replication.

The remainder of the paper is organized as follows. In Section 2, we overview existing work on modeling Data Grids and simulating difference data replication techniques. Section 3 describes our approach to deploying dynamic replication in Data Grids. In Section 4, we present the design concepts of the simulator GridNet. Section 5 describes our analytical model of the replication costs and benefits and its use in our dynamic replication system. In Section 6, we present the results of simulated dynamic replication techniques for the typical Data Grid architecture proposed at CERN (the European Organization for Nuclear Research) and implemented by the Grid Physics Network (GriPhyN) [12]. In Section 7 we provide the conclusions and argue that the replica management system that we propose offers better performance than existing approaches [3, 2, 4, 15, 16, 17, 19, 20] because it offers a highly dynamic and optimized solution using scalable replica distribution mechanisms.

2. Related Work

Recently, there has been a rise in interest in modeling Data Grid environments and simulating different data replication techniques as well as basic file replication protocols [19]. With the increase of the data production and data sharing needs of a widening global scientific community, there is growing need for improving data access performance and data availability.

Different studies were conducted to model scientific experiments settings and configurations, such as the CMS and ATLAS experiments at the Large Hadron Collider, the Laser Interferometer Gravitational Observatory and the Sloan Digital Sky Survey [19, 12]. These studies led to the initiation of many projects such as the GriPhyN [12] and the EU DataGrid [6]. Currently, these projects use static, or user-driven replication services provided by the Globus Toolkit [8, 10], such as Globus Replica Location Service [5].

A simulation framework OptorSim was introduced in [4], where data replication is combined with job scheduling. In contrast to our approach, introduced first in [13], OptorSim uses a prediction function based on spatial and time locality regardless of the overall data access cost on the Data Grid.

In [17], an approach is proposed for automatically creating replicas in a typical decentralized Peer-to-Peer network. The goal is to create a certain number of replicas on a given site to guarantee a minimal availability requirements. Different replication and caching strategies within simulated Grid environments are discussed in [15] and their combination with scheduling algorithms is studied in [16]. The replication algorithms proposed were based on the assumption that popular files in one site are also popular at other sites. We take a different approach by evaluating replica creation and placement using the network attributes as well as data popularity along with its spatial and time locality.

3. Replication Algorithm

3.1. Replica Distribution topologies

Given the sizes of data entities on the Data Grid, scalability is an important issue. To offer scalability, we use both hierarchical and flat topologies to organize replicas on the Data Grid. These topologies reflect the use of the client-server and the peer-to-peer approaches to exploit locality and higher bandwidth availability in search for faster data access time. In the client-server approach, the cost of maintaining replica consistency is lower than in the peer-to-peer approach. In the former model, there is one central location with a single replica server to which all updates must be posted. This solution substantially simplifies replication consistency maintenance. However, such a solution has poor reliability because a failure of the server makes it impossible for any other replicas to receive new updates or to disseminate their own updates to others. In our approach, we intend to use multiple replica servers to enhance data availability and to avoid single points of failures. Moreover, the particular communication patterns among the replicas form the replica distribution topologies. These topologies are the basis for the replica synchronization and reconciliation paths. The replica placement service uses these distribution topologies to overlay replicas on the Data Grid; thus improving data access and replica synchronization costs. In this work however, we only consider read-only data. We plan to investigate the performance of our approach with different consistency algorithms in future work.

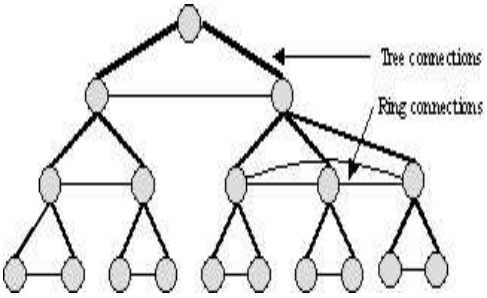

Figure 1. Hybrid Replica Distribution Topology

To take advantage of the hierarchical and flat topologies, we allow both topologies to be combined in multi-level hierarchies. Figure 1 shows a hybrid replica connection graph, that combines both the ring and tree topologies. This distribution model could be used in cases where the replica distribution spans over different partitions and be part of a hierarchical structure as well as part of a flat distribution. This approach improves both the data availability and the reliability of the flat topology and allows for a scalable expansion of the hierarchical distribution.

3.2. Replica Placement Algorithm

The topologies described earlier are adaptive. When new nodes join or leave the system the nodes connections are dynamically adjusted to reflect the new changes. The replica sites location within the topology is defined by the organizational domains these sites belong to, and by the replica placement algorithm. Initially, additional replica sites are connected to the main storage site as child nodes and to each other in a ring topology. Depending on the number of organizational domains that the Data Grid spans through, more levels of the topology and nodes could be added the same way. When an access request is generated from a grid node, that request is forwarded to the closest main data storage. Further requests are subsequently forwarded to that same server site. When the replica placement algorithm decides to create a new replica at the request-originating node, that node is attached to the existing topology as a child of the server site. If the server site has other children, the underlying sibling topology is adaptively adjusted to include the joining node. This process is enumerated in two steps:

1. add edge to parent,
2. add edges to parent's children (siblings in a ring),

Each newly added replica is registered in a local replica catalog, by adding a new entry in the catalog reflecting the physical-logical name mapping of the replicated object name to its new replica location. These changes are made visible to the entire grid through the propagation of the new entry in the local replica catalog and its synchronization with other remote replica catalogs. The catalogs are synchronized by forwarding new entries to the parent node and subsequently all the way to the root. To facilitate replica look up in a hybrid topology, the replica catalogs maintained at parent nodes contain all the entries in their child nodes catalogs.

In some cases adding a new replica might not necessarily mean adding an additional hierarchy level. An important criterion in deciding whether to attach a new replica site as sibling or as child, is the connection quality of the replica sites. Ideally, a decision about grouping should be based on:

- geographic location: replica nodes that are physically near each other should be grouped together. Since communication with a nearby site is more efficient that communication with a distant one, nearby neighbors are a good synchronization choice.
- expected bandwidth: nodes that are connected by high bandwidth links should be grouped together.

- connection latency: nodes with low latency connections should be grouped together. Those connected by high latency connections should not be linked together.

The replica catalogs are used to locate the closest replica site when receiving a data access request. When a replica site receives a data access request, it first checks its local replica catalog to see if it has an entry for the requested data. In case there is no entry, the replica look up request is forwarded to the parent node, and up the hierarchy until a replica entry is found. The data access request is then forward to the closest replica site.

4 Simulation

In order to evaluate our approach, we developed a Data Grid simulator GridNet [13]. This simulator provides a modular simulation framework through which we can model different Data Grid configurations and resource specifications. In this section we present our simulation study along with its results. Throughout the simulation, we assumed read-only data and did not include the consistency or write and update propagations costs in the study.

4.1. Simulation Design

GridNet is a modular simulator, written in C++, and built on top of the network simulator ns [14], which provides us with basic Grid network specification: nodes, links, and messages. GridNet introduces application level services implemented on top of the existing ns protocols. It allows us to specify different network configurations, different types of nodes, different node resources, a replication strategy, and a cost function and its parameters.

The GridNet simulator modules are composed of objects that are mapped into application level object classes in ns. Data exchanged between the GridNet nodes is defined as application level data that is passed down to the ns node as a stream of packets. In addition, there are also packets representing grid user requests as well as the packets indicating the start and the end of grid data transmission in ns that transfer control of the simulation to GridNet. The GridNet code simulates the replication decision at each node and generates new ns traffic (forwarding requests other nodes or sending the requested data to the client). This separation of the network simulation strata, the Grid nodes and the replication algorithm enables us to use the existing package, ns, and add only the grid specific elements to the simulation.

One of the main considerations in designing our simulator was to model a Data Grid architecture and the interactions of the individual Grid components as realistically as possible. Therefore, the simulation is based on the architecture proposed for CERN experiments. We assume a multi-tier Grid topology throughout this study.

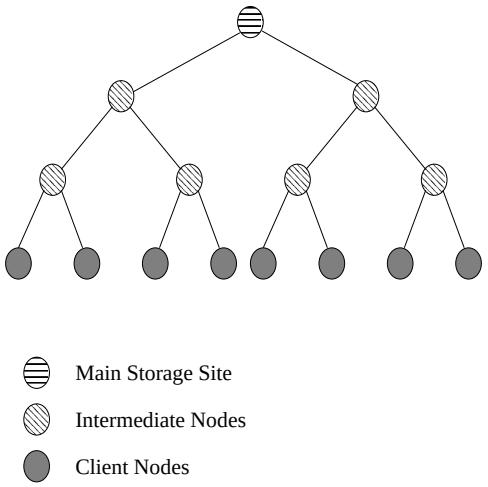

Figure 2. Simulation Model

4.2. Architecture

The simulation was constructed assuming that the Grid consists of several sites, each of which may provide computational and data-storage resources. In our study we adopt the typical Data Grid architecture used at CERN (the European Organization for Nuclear Research) and implemented by the Grid Physics Network GriPhyN [12].

To specify Data Grid nodes, the simulator extends the original semantics of a node object in ns. Each node is able to specify its storage capacity, organization of its local data files, its relative processor performance, and to maintain a list of its neighbors and peer replica nodes.

Figure 2 shows the simulation model and topology used in our experiments. Given the configuration of the adopted model, the simulator allows us to specify different types of nodes, namely *client*, *server* and *cache* nodes that are described below.

- Server node: represents a main storage site, where all or part of the data within the Data Grid is stored. This site represents the root of the grid hierarchy.
- Cache Node: represents an intermediate storage site, for example a regional storage site. Such sites would have high storage capacity and would replicate part of the data stored on the main storage site.
- Client node: represents a site where data access requests originate and are generated. The client nodes are always placed at the leaf level of the Grid hierarchy.

The network interface model is specified in the link object that is provided in ns. The link object is used to model

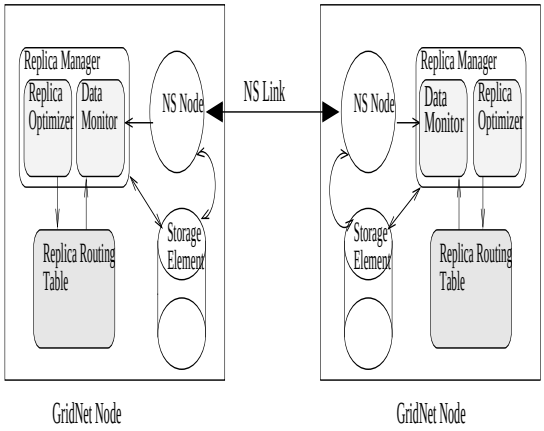

Figure 3. The GridNet Simulator Architecture

the parameters of physical interconnections in the simulated network, such as link bandwidth and latency.

As shown in Figure 3, each GridNet *node* consists of the following three elements:

1. a basic *ns* node,
2. a *storage element*,
3. a *replica manager* or a monitoring agent.

Each GridNet node also maintains a replica routing table.

The replica manager makes the decision about when to create and delete replicas. This decision is made using statistics collected at each site about data access requests and network characteristics. Such data are access frequencies per data file, connection bandwidth information, and storage space availability.

To collect the performance data, each replica manager contains a monitoring module or agent that is responsible for computing the number of data requests generated at each node, as well as the number of requests received from other nodes. To compute bandwidth availability, each node regularly polls its connections to its neighbors. The data collected is then evaluated by the replica placement algorithm or optimizer using the cost function formulated in Section 5 (equations (3) and (4)).

The simulation is constructed assuming that each node on the Data Grid has some computational power and may provide data-storage resources. The GridNet defines the simulation through a sequence of commands. Currently the following five commands are generated by GridNet nodes:

GPT_READ_REQUEST used by the client to initiate reading the data,

GPT_WRITE_REQUEST used when a client needs to write a file to the server,

GPT_WRITE_UPDATE used by a server or cache node to inform the cache nodes that the newer version of data is available,

GPT_READ_ACK used when the server or cache node confirms the read request, and

GPT_WRITE_ACK used by the server or cache node to confirm the write request.

Their implementation is quite simple. For example, GPT_READ_REQUEST is sent from the *client* node to a *cache* node and generates the following sequence of commands at the *cache*:

```
log-traffic;
log-count;
if (has a local copy and is valid){
    send a READ-ACK
    to the original sender directly;
 }
 else{
    forward GPT-READ_REQUEST to parent;
 }
```

Some of the parameters used in the simulation, like the frequency of bandwidth probing and statistics gathering are set by *ns* commands. In our simulation, only client nodes generate data access requests. Each client runs a set of jobs that require accessing certain data files. The job running-time is simulated by associating a processing time with each file. Clients run concurrently with each other.

Client nodes are assumed to have a limited storage space that they use for caching. Before generating a request, the client checks if the data requested is available locally. Each node maintains a *replica routing table* that allows it to locate the closest replica site and the node to which it should forward its request. Each time new replicas are created or deleted, replica routing tables are updated accordingly. The next Section explains further how these tables are updated and used.

5. Cost Model and Its Use in Replication

Given the hierarchical structure of the Data Grid, we organized our simulations into multi-layered trees. Nodes placed in higher levels have higher storage capacity. Client nodes are placed in the lowest level. At each level, nodes can be organized into ring topologies to facilitate data access among same level nodes (peer-to-peer model). Initially routing tables contain the address of each node's parent and siblings in the hierarchy. When initial requests are generated, they are forwarded to the higher level.

The replication placement policy is formulated as an optimization problem. Each node v in the overall system and a

data object i are associated with a nonnegative read rate $\lambda_{v,i}$ and a nonnegative write rate $\mu_{v,i}$ that represent the traffic generated within this node's local domain related to object i. If $C_w(i)$ is the write cost for a given object i and $C_r(i)$ is the read cost for the same object, then $C_w(i)/C_r(i) = \alpha_i$ is the ratio of the write cost for node i. If there are no replicas for object i in the system, then the total data transfer cost for this object at node v is:

$$cost_{v,i} = (\lambda_{v,i} + \alpha_i \mu_{v,i}) size(i) d(v,r) \quad (1)$$

where r is the node containing the object i and $d(v,r)$ is the sum of the edge costs along the path from v to r such that

$$d(v,r) = \frac{1}{bandwidth(v,r)},$$

where $bandwidth(v_1, v_2)$ is the total available bandwidth between nodes v_1, v_2.

Let N represent the set of all nodes in the system, R_i be the replica set of object i, and $c(v, R_i)$ denote the replica of object i closest to node v. So, if node v is to be added to R_i, $c(v, R_i)$ would become v's ancestor in the replica tree of object i. Let T_v be the partition of nodes that would be serviced by v for future access requests to object i, assuming that v is added to R_i. Let $\lambda^t_{v,i}$ represent the total read rate of all nodes at partition T_v, and $\mu^t_{v,i}$ represent the total write rate of the partition. The incremental data transfer resulting from placing a replica at v can be expressed by the following formula:

$$\frac{cost^i(N, R_i, v)}{size(i) d(v, c(v, R_i))} = -\lambda^t_{v,i} + \alpha_i (\mu^t_{r,i} - \mu^t_{v,i}) \quad (2)$$

Indeed, adding v to R_i decreases the read cost of each node in T_v by $size(i)d(v, c(v, R_i))$ and increases the write cost of each node in the $N - T_v$ by $size(i)d(v, c(v, R_i))$, but it does not change other costs. Thus, the total cost of data transfer for object i with a replica set R_i is given by:

$$Cost(N, R_i) = cost(N, r) + \sum_{v \in R_i - \{r\}} cost^i(N, R_i, v) \quad (3)$$

where $cost(N, r)$ represents the data transfer cost of object i from the root node r. Given the structure of the data grid and the associated read/write patterns for object i, the first term in Equation 3 is constant. Hence, we only need to consider the problem of optimizing the following cost:

$$Cost'(N, R_i) = \sum_{v \in R_i - \{r\}} cost^i(N, R_i, v) \quad (4)$$

The above formula expresses the data transfer cost for object i improved thanks to the placement of a set of replicas $R_i - \{r\}$. The runtime system uses access cost statistics to compare the replications gains to replication costs (update cost) and then informs the replica management service whether to place a replica on node v or not.

Other parameters that can also be taken into consideration while creating and placing replicas are: the storage capacity and availability at a given Grid node, the frequency of cost estimation, and the replica access patterns. The latter parameter is estimated based on the history of the data accesses requested at a given site.

The frequency of cost estimation depends on the load of the system as well as the number of nodes in the Data Grid. Each time the replication cost function is evoked, the read and write count values are annulled and previous collected values are averaged out. These values are used to predict future tendencies in access patterns. We plan to use different prediction tools to infer data access patterns and to tune the system parameters, such as: regression methods, moving average, and exponential smoothing.

The storage cost is computed based on the state of the data objects, their request frequencies, and their size. The state of data objects is defined as busy, active, passive, or obsolete. The first state is assumed when the local data replica is being accessed. The second state describes local replicas that have been accessed recently within a predefined time-frame window. Replicas that have not been accessed within that time-frame window are categorized as passive. If a file is found out of date following a consistency check, it is marked as obsolete. Each replica is also assigned a weight index that indicates how much space it is occupying.

The storage cost is a linear combination of these parameters. Each factor in the combination is assigned a weight depending on the applications properties and access patterns. When the replica management decides to create a local replica, it first checks whether storage space is needed and available. If it is needed but not available, then based on the ranking of existing replicas, the system decides whether to delete some of the existing replicas to make space for a new one or to decline the creation of the new replica. To compare the cost of storage used by existing replicas to the cost of storage needed by the new one, the system ranks the latter as busy and uses the method used for evaluating costs of new replicas to assign it a storage cost. If there are enough obsolete, passive or active replicas with lower improvements in data access time than the new replica can provide (as evaluated by the cost model), then these existing replicas are replaced to make space for the new replica.

When a replica is created at a given node, that information is propagated to the siblings and children of that node. Subsequent access requests to these replicas are forwarded to the closest replica site using the closest path. To determine that path, the bandwidth of the node's connections are evaluated to determine the most efficient routes to access

data. The replica routing tables are then updated accordingly to reflect that information.

The replica routing tables implement some of the functionality of the replica catalogs provided by the Data Management Component of the Globus toolkit [1, 5]. These tables provide a mapping between logical file names and their physical locations on the Data Grid. We are working to extend further these functionalities, and integrate our model with the Globus model.

6. Simulation Results

6.1. Simulation Configuration

Our study was carried out using the three-tier Data Grid topology shown in Figure 2. Table 6.1 defines the parameters used in our study.

Number of sites	28
Total number of Datasets	30
Connectivity bandwidth	server-cache 1Gb/s cache-cache 400Mb/s cache-client 100Mb/s
Total number of requests	1000

Table 1. Simulation parameters

Initially all files are placed on the Main Storage Site (root). Data access patterns are based on both temporal and geographical data locality. Recently accessed files are more likely to be accessed again and files accessed by a node are more likely to be accessed by its siblings and children.

Within this model, each site was allocated storage resources proportional to their location on the grid hierarchy. The main storage site was allocated a Storage Element to hold all of the master files. Client nodes, however, do not have any storage space.

6.2. Results

Our experiments consisted of running simulation of the model described above using three different scenarios. The topology consists of a 3-tier tree, with one root (tier-0) and three cache nodes (children) in the first tier (tier-1). Each of these nodes has two cache nodes as its children in tier-2. In addition, each cache node in tier-2 has three children client nodes. In total, we used 18 client nodes, two cache levels with 9 cache nodes, and one main storage site. The goal of the simulations was to evaluate our replication approach against the case where no replication was used. Scenario 1 represents the case where no replication is used, therefore all data access requests were forwarded and serviced by the main storage site. In scenario 2, dynamic replication is used with small storage capacities allocated to cache nodes. In scenario3, dynamic replication is used with larger storage capacities at cache nodes. We used 6 datasets for our experiments. Table 6.2 shows the range of data file sizes within each dataset.

Datasets Used	File Size range
Dataset 1	1KByte to 400KByte
Dataset 2	500KByte to 900KByte
Dataset 3	1MByte to 3MByte
Dataset 4	3.5MByte to 5.5MByte
Dataset 5	6MByte to 8MByte
Dataset 6	8.5MByte to 20 MByte

Table 2. Data File Sizes used in the Experiments

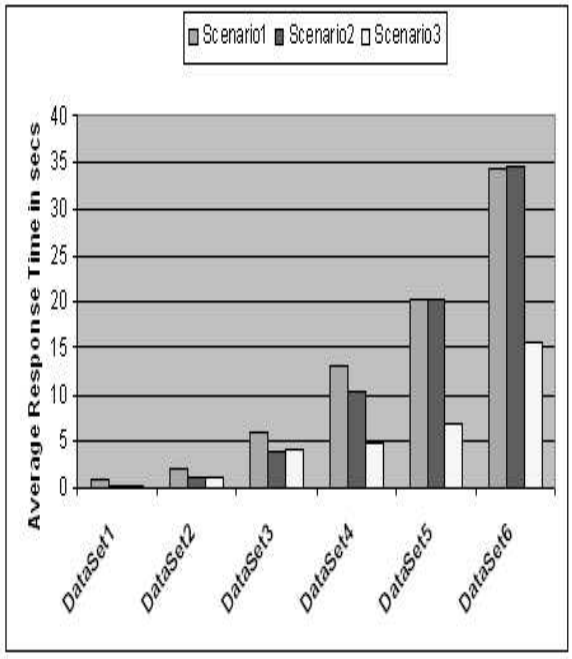

Figure 4. Average Response Time of both Dynamic Replication and No Replication Policy for different Dataset Sizes

Figure 4 shows the average response time of read requests for the files used in the simulation. The file sizes ranged from 100KB to 20GB. The results show that dynamic replication yields large improvement (60% on average) in the overall data transfer and response times within the Data Grid. The results also show that the use of small

caches at intermediate nodes does not improve the data access performance of large datasets on the Grid. With small storage space available, intermediate nodes spend a lot of time adding new replicas and deleting older ones. Since data accesses are based on both temporal and geographical locality, deleted replicas might be needed at a later time. The replica manager needs then to free space to add new replicas when their access cost is higher than that of existing ones.

The chart also shows that the average response time of data requests for scenario 1 and scenario 2 are similar when the size of data files is less than 4GB. In these cases cache sizes do not affect much the overall data transfer on the Data Grid. However, for file sizes larger than 4GB cache, scenario 1 and 2 produce the same results while the use of larger storage spaces at cache nodes yields better performance. In this latter case, the frequently requested replicas do not compete over available space.

Results from our experiments also show that data transfer costs and bandwidth consumption decrease dramatically with the use of dynamic replication. Our results show that bandwidth consumption decreases by almost 30% from scenario 2 to scenario 3.

While running the simulation, we also found out that with higher network bandwidths, the performance of high workload scenarios increases noticeably. Our replication technique yields better performance with the use of larger file sizes and with an appropriate allocation of storage space at cache nodes. Our results are very promising and show that dynamic replication improves dramatically the performance of the overall Data Grid.

7. Conclusions

In this paper we described the design of our Data Grid simulator GridNet. We gave a description of our simulation model and simulated configurations. We presented initial performance results of replication using a hierarchical grid topology with various scenarios.

We have begun addressing the problem of replication in Data Grid environments by investigating the use of a decentralized dynamic replication services that can be used to improve data access time, data availability, bandwidth consumption, and scalability of the overall system. We have also used a cost function that dynamically evaluates the replica placement policy by comparing the replica maintenance costs and data access gains of creating a replica at any given location.

Our results are very promising, and showed that dynamic replication decrease data transfer costs and improve data access performance on the Data Grid . We note that our approach has significant advantages. First it is based on decentralized and distributed computing model, and second it dynamically adapts to both user and network behavior while improving the performance of the overall system.

In future work we plan to validate our model on real Data Grids. We are also interested in exploring different replication algorithms and cost models. Additionally we plan on integrating our services with the Replica Location System [5] and the Globus Toolkit [3].

Acknowledgment

The authors express their gratitude to Dr. Carl Kesselman for his help and discussions of Data Grid architectures.

References

[1] W. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Lee, A. Sim, A. Shoshani, B. Drach, D. Williams. “High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies.” *Proceedings of SC 2001*, Denver, CO, November 2001.

[2] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, “Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing,” *IEEE Mass Storage Conference*, 2001.

[3] W. Allcock, A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, “The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets,” *Journal of Network and Computer Applications*, 23:187-200, 2001.

[4] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, F. Zini, “Simulation of Dynamic Grid Replication Startegies in OptorSim,” *Proc. Of the 3rd Int’l IEEE workshop on Grid Computing (Grid 2002)*, Baltimore, USA, November 2002.

[5] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripneau, B. Schwartzkopf, H. Stockinger, K. Stockinger, B. Tierney, “Giggle: A Framework for Constructing Scalable Replica Location Services.” *Proceedings of Supercomputing 2002 SC2002* November 2002.

[6] The European Data Grid Project. The DataGrid Architecture 2001. http://www.eu-datagrid.org

[7] I. Foster, C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit,” Intl J. Supercomputer Applications, 11(2):115-128, 1997.

[8] I. Foster, C. Kesselman, S. Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations." International J. Supercomputer Applications, 15(3), 2001.

[9] I. Foster, C. Kesselman, "A Data Grid Reference Architecture - Draft of February 1, 2001", GriPhyN technical report GRIPHYN 2001-12, 2001.

[10] I. Foster and C. Kesselman (eds.), "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann, 1999.

[11] I. Foster, "The Grid: A New Infrastructure for 21st Century Science", *Physics Today*, 54 (2). 2002.

[12] Grid Physics Network (GriPhyN). http://www.griphyn.org

[13] H. Lamehamedi, B. K. Szymanski, Z. Shentu, E. Deelman, "Data Replication Strategies in Grid Environments," *Proceedings of ICAP'03*, Beijing, China October 2002, IEEE Computer Science Press, Los Alamitos, CA, 2002, pp. 378-383.

[14] NS network simulator. *http://www-mash.cs.berkeley.edu/ns*.

[15] K. Ranganathan and I. Foster, "Identifying Dynamic Replication Strategies For a High performance Data Grid," Proceedings of the International Grid Computing Workshop, Denver, November 2001.

[16] K. Ranganathan and I. Foster, "Design and Evaluation of Replication Strategies for a High Performance Data Grid," International Conference on Computing in High Energy and Nuclear Physics, Beijing, September 2001.

[17] K. Ranganathan, A. Iamnitchi, and I. Foster, "Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities," *Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop*, Berlin, Germany, May 2002.

[18] R. Stevens, P. Woodward, T. DeFanti, and C. Catlett, "From the I-WAY to the National Technology Grid", *Communications of the ACM*, 40 (11). 50-61. 1997.

[19] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, B. Tierney , "File and Object Replication in Data Grids," *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*, IEEE Press, August 2001.

[20] S. Vazhkudai, S. Tuecke, I. Foster, "Replica Selection in the Globus Data Grid," *Proceedings of the First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001)*, pp. 106-113, IEEE Computer Society Press, May 2001.

Biographies

Houda Lamehamedi is a graduate student pursuing a Ph.D. degree in Computer Science at Rensselaer Polytechnic Institute under the supervision of Prof. B. Szymanski. She received her MS in computer Science from Al-Akhawayn University in Ifrane, Morocco in 1998. She is currently working on developing a Data Grid management middleware at Rensselaer Polytechnic Institute. She is also working on a joint research project with IBM Almaden Research Center to develop an autonomic grid management middleware. Her interests include distributed and parallel computing, and Grid computing.

Boleslaw K. Szymanski is a Professor at the Department of Computer Science and a member of the Scientific Computation Research Center, Rensselaer Polytechnic Institute. He received his Ph.D. in Computer Science from National Academy of Sciences in Warsaw, Poland, in 1976. He was a post-doctoral fellow at the Aberdeen University, Aberdeen, U.K. and on the faculty of the Department of Computer and Information Sciences at University of Pennsylvania. He is an author and co-author of more than hundred fifty scientific publications and an editor of three books. Dr. Szymanski is also an Editor-in-Chief of Scientific Programming journal and on the editorial boards of other journals. Dr. Szymanski is an IEEE fellow and a member of the IEEE Computer Society, and Association for Computing Machinery. Dr. Boleslaw Szymanski's interests include distributed and parallel computing and system modeling and simulation. His recent work includes network management, on-line network simulation, network monitoring and network security.

Zujun Shentu is a graduate student pursuing the Ph.D. degree in Computer Science at Rensselaer Polytechnic Institute, working under the supervision of Professor Szymanski. He obtained his MS and BE degree (both in Computer Science and Engineering) from Zhejiang University, P.R. China in 1999. When studying at Zhejiang University, his research interests included MRPII, Product Data representation and exchange, Artificial Intelligence and Engineering Database Systems. After his graduation, he joined Hangzhou Torren Software Company, where he led a group to develop a full-function Object Data Management System, which has been successfully integrated in a commercial Geography Information System. His current research interest include Distributed Computing, Data Grid

management and Network Simulation.

Ewa Deelman is a Computer Scientist at the Center for Grid Technologies of the Information Sciences Institute at USC. Dr. Deelman's research interests include data management, request planning and performance evaluation in Grid environments. At ISI, Dr. Deelman is part of the Globus project, which designs in implements middleware for the Grid. She received her PhD from the Rensselaer Polytechnic Institute in Computer Science in 1997 in the area of parallel discrete event simulation.