Distributed Data Management Services for Dynamic Data Grids

Houda Lamehamedi, Boleslaw K. Szymanski, Brenden Conte
Department of Computer Science,
Rensselaer Polytechnic Institute,
Troy, NY 12180
{lamehh, conteb, szymansk}@cs.rpi.edu

Abstract

Data grids are middleware systems that enable users and applications to locate, access, and place large numbers of data sets in geographically distributed storage sites. In most existing and deployed grid systems however, control of the resources is centralized and usually handled by system administrators. Such configurations hinder dynamic and scalable expansion of the Grid infrastructure and resources. We propose a new lightweight distributed, adaptive, and scalable data Grid middleware that provides transparent, fast, and reliable access to data and storage resources in data grids. At the core of our approach are dynamic data and replica location and placement techniques that adapt replica location and access to the continuously changing network connectivity and users behavior. The system is fully distributed and self configuring. In this paper we present the design of the system, the algorithms we use to implement the data management services, and demonstrate the scalability and performance of the overall system.

1 Introduction

Most of scientific applications require accessing, analyzing, and storing large amounts of data. Such applications warrant special treatment to scale across distributed computing platforms like the grid and avoid data access bottlenecks. In areas such as genomics [2], drug discovery [2], high energy physics [9], astrophysics [22], and climate change modelling [2, 8], large data sets are generated, collected, and stored in geographically distributed locations [1, 3, 6, 7, 9, 10, 24]. This data is then accessed for further processing at different additional sites. Due to the actual, and in fact growing, size of such data sets, a management framework is needed to ensure fast and reliable access to users in remote and distributed locations.

In most existing and deployed grid systems and platforms, software and hardware resources are centrally managed [1, 3, 7]. System administrators are responsible for installing software, granting and controlling access to resources. Such configurations, hinder dynamic and scalable expansion of the Grid infrastructure and resources. New nodes cannot be added to the grid without the authorization and the intervention of system administrators, nor can they leave the grid without prior system pre-configuration. In the case of data grids storage resources are dedicated resources and cannot be taken off-line without prior configuration and notice to the users. Enabling dynamic node addition and deletion while providing efficient access to resources on the data grid presents considerable challenges to system designers. Grids are made up of a diverse set of machines and instruments with different capacities and purposes. However, to increase the power of the grid, and increase its versatility, the grid needs to be extended to incorporate in addition to supercomputers and dedicated instruments available in research centers, a much larger number of available commodity computers. Doing so, on the one hand, enables the formation of an ad-hoc configuration and potentially the exploitation of more available resources based on demand. On the other hand this configuration requires a more flexible and extensible approach to manage a dynamic infrastructure.

To alleviate and address some of these issues we propose an adaptive and scalable lightweight data management framework that enables users to dynamically join and leave the grid. The middleware provides transparent, fast, and reliable access to data and storage resources on the Grid. It relies on an application level overlay network structure to identify the fastest paths for locating and accessing data. In our approach, the application level overlay networks are incrementally built and dynamically updated to adapt to changes in users's and applications's behavior and access patterns. We use an ad hoc dynamic spanning tree overlay network as it can be easily adapted to different topologies and changed dynamically. The overlay network is a virtual connectivity graph in which the computational nodes are mapped as Vertices and their connection links as edges. Abstracting the underlying infrastructure into these overlay connections enables more flexible management and deployment of higher level techniques.

Each node in the graph maintains a routing table that contains a list of neighbors to which data access requests are forwarded. When a node first joins the platform, it attaches itself to a parent node. Thus, a hierarchical graph is created and a spanning tree connecting the grid nodes is built. The routing table contents at each node are dynamically updated at runtime depending on network resource availability and the location of popular data sets. Each participating node or member of the network is aware of the location of its closest neighbors but does not have a global view of where data is located or how far and distributed are the rest of the nodes. Similar to the peer-to-peer networks approach [28, 11, 15, 20, 21, 23], and based on the topology of the overlay network, each node monitors its connectivity to and activity with its neighbors and collects statistics about incoming access requests. This statistical data is then used to decide whether or not to create local replicas of the requested data. The middleware we have designed consists of autonomous agents running at each node. Each agent is composed of a monitor, a replica manager, and a storage manager. The monitors feed the collected statistics about resource availability and data/replica locality to the decision-making components at each node, the replica and storage managers. The replica manager takes into consideration data access patterns, data location, latency, and bandwidth availability before deciding on either replicating the requested data or not and where to fetch the data from. The replication serivce is driven by the location of storage resources, network resources availability, and user access patterns and behavior. We will explain in further details the concepts and theory behind our approach in Section 3, and present the architecture of the system. In this work however, we focus on the study of the replication service and the cost model we use to manage the creation and removal of replicas.

The benefits and applicability of our approach have been proved very successful and the results were reported in [14, 13]. In our previous work we developed a Grid Simulator, GridNet, to simulate data movements, user requests, and replication techniques in Data Grids. The simulator allowed us to evaluate our design and cost models, and our results have shown that proactive replication improved the overall performance of resource usage on the Grid by up to 30%.

Our proposed framework can be easily integrated with existing low level grid services and adheres to established standards in grid computing. The remainder of this paper is organized as follows: in Section 2 we present a summary of existing work and state of the art. In Section 3 we give an overview of our proposed system, present our approach and the analytical cost model we adopted. Section 4 provides results that show the system's performance in real-time environment, and we give our conclusions in Section 5.

2 Background And Related Work

With the clear increase in data production, archiving, analysis, and sharing needs within the scientific community, there has been a rise in interest in developing and extending data grid infrastructures to provide users with high level data management services and transparent access to this data [17, 3, 9, 16, 12, 22, 23]. 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 [29, 9, 23]. These studies led to the initiation of many projects such as the GriPhyN [29] and the EU DataGrid [27]. Many projects, such as the GriPhyN [29] and the EU Data Grid [3] have developed data grid platforms. Other projects have focused more on simulations [10, 1].

In existing implementations of data grid services in Globus [7, 5, 25, 26] and the EUDataGrid [10, 3], dedicated nodes are responsible for storing and maintaining replica location indices throughout the system. These implementations though, offer a static and centralized approach to managing replicas on the grid. Dynamic replication presents a more attractive approach as decisions are made based on current access patterns and availability of resources. This however incurs some costs from the creation of new replicas and the continuous evaluation of resources availability. In [18] the authors have studied different replication strategies coupled with job scheduling techniques. Their results show that taking into consideration the location of data instead of focusing only on available and idle cpu cycles yields better performance. In [19] the authors study the performance of a somewhat distributed and dynamic replica creation mechanism in peer to peer environments. Their approach outperformed static replication in most cases but with the risk of creating more replicas than necessary, thus consuming even more resources. Optorsim is a grid simulator that was developed to study the efficiency of different replication algorithms in a grid [4]. The studies conducted with Optorsim combined job scheduling with data access optimization, and showed that scheduling techniques that take as input the availability and location of data outperform the classical scheduling methods.

In our work we take a different approach by decoupling data and replica management from computational job scheduling, providing a stand-alone framework that is mainly used to provide efficient access to data but could also be integrated with a job scheduler if needed. The main goal of our approach is to create a distributed mechanism to replicate and manage access to data that is based on evaluating costs and gains of resource utilization and access performance. The work we present in this paper first started with our initial investigations of dynamic replication strategies in grid environments [13, 14]. Which then led us to the development of a data grid simulator GridNet. This simulator provides a modular simulation framework through which we can model different Data Grid configurations and resource specifications. The simulations allowed us to perform initial verifications of the design and evaluate the performance of our strategies. The results show that our replication technique yields better performance with larger file sizes and with an appropriate allocation of storage space. Our results are very promising and show that dynamic replication improves dramatically the performance of the overall Data Grid.

As stated above most existing systems have focused on job scheduling vs. data location. In such scenarios, Data placement is coupled with computation and job scheduling. While computational jobs are an important factor in deciding where to place data, building a data management middleware that proactively places and stages data in strategic locations and adapts data distribution to continuously changing user and network behavior, is a more general and comprehensive approach.

In the next sections we provide more details about the design and implementation of our proposed framework, and present the results of our experiments.

3 System Design

Our proposed framework provides incremental scalability and robustness to ad hoc dynamic data grids. The application level data management overlay, can handle a continuously changing number of participating nodes and failures without affecting the overall performance or efficiency of the replica management service. An added advantage of this approach is the elimination of centralized control of data and replica registration, and the balance of data discovery and lookup among different nodes. Our prototype was implemented in Java, using sockets to enable message passing between the distributed grid nodes. Our proposed replica management service offers transparent data replication based on a runtime system that evaluates the access cost and performance gains of replication before moving or copying any data. The goal being to lower users access time and optimize the usage of network and system resources: bandwidth and storage space.

The access cost of the replication scheme is calculated based on multiple factors, such as accumulated read/write statistics, network latency, response time, bandwidth, and replica size, averaged over selected time scales. This cost changes during the program execution and from one application to another, so it is constantly reevaluated to dynamically minimize data access costs by changing and adapting the number and placement of replicas.

3.1 System Architecture

The middleware consists of autonomous agents that run at each participating grid node. Each agent is composed of:

1. Resource Monitoring Service (RNS): responsible for collecting statistics about resource usage and data access requests,

2. Replica Management Service (RMS): responsible for creating local replicas based on a cost function and choosing the nearest node containing a replica of a requested data set,

3. Resource Allocation Service (RAS): responsible for allocating space for newly created replicas and de-allocating space from the least frequently and last accessed locally stored replicas,

4. Routing/Connectivity Service (RS): responsible for routing outgoing messages and managing incoming messages, managing data transfers, as well as monitoring and managing a node's connectivity to its neighbors.

The system architecture is shown in figure 1. The monitoring service feeds the collected statistics about resource availability and data/replica location to the decision-making components at each node, the replica and storage managers. The replica manager takes its input statistics from the monitoring service and uses a cost function to evaluate gains vs costs before deciding on whether to replicate the requested data or not.

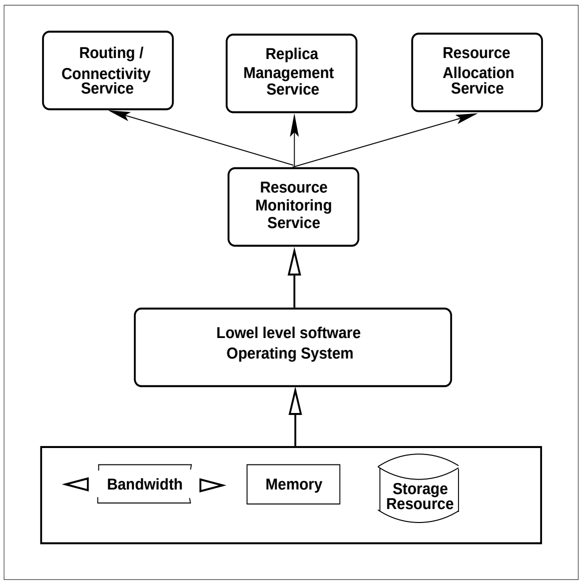
Routing / Connectivity Service
Replica Management Service
Resource Allocation Service
Resource Monitoring Service
Lowel level software Operating System
Bandwidth
Memory
Storage Resource

Figure 1: Architecture and Design of the Data Management Middleware

In our approach we treat replica management as an optimization problem in which important parameters of the system, such as read/write access patterns, network performance etc., are taken into consideration when deciding the creation and the placement of replicas. The replication service determines the placement, and location of replicas in the system. These decisions are guided by a cost model that, at runtime, evaluates the maintenance cost and access performance gains of creating a replica.

3.2 Data Access And Replica Location Algorithm

When access to a data set is needed, a request is issued. This request starts a search process. This search is supposed to reach all the possible nodes that have a copy of this data set. In case multiple locations are discovered the requester needs to be provided with all the locations of the required data and choose the appropriate source node. In existing data grid implementations, dedicated nodes store information about the locations of possible sources. In a more dynamic platform, new nodes might join the grid and some nodes might leave. Thus the need for an adaptive, more dynamic approach to discover, locate, and access data. Similar attempts were used to develop search protocols for peer-to-peer data sharing. Examples of such protocols are the flooding algorithm used in Gnutella [28], the centralized algorithm used in Napster [15], and the distributed hash-table based protocol used in CAN, Pastry and Tapestry [20, 21, 23]. Many studies have shown that using a combination of flat and hierarchical topologies give the best performance for message broadcasting [15, 11].

In our work we use an ad hoc dynamic spanning tree overlay network to route access requests and locate data in a dynamic grid platform as they can be easily adapted to different topologies and changed dynamically. Ad hoc spanning trees have been tried and proven to perform and scale well in peer-to-peer systems [11].

To build the distributed spanning tree overlay network two algorithms are needed. The first one is an algorithm for maintaining the tree and accounting for inclusion of new nodes joining the tree, and deletion of nodes leaving. Another one is a search algorithm for locating data in the tree.

The tree is constructed starting from a root node that always stays on line and alive. When joining the grid, a node is added through an existing grid node by attaching to it as a child node. Existing grid nodes are published in a web page or could be accessed through a web service. New nodes choose from the list of available nodes using some metric such as geographical proximity of a node to which the new node needs to attach. For example a node can choose a parent node which is in the same domain. This approach creates an inherent tree structure. When a node leaves the tree, it sends a notification message to its parent and children. The parent removes the departing node from its children's list. The children nodes contact the parent node (their grandparent), and rejoin the tree as its children nodes. To avoid having a disconnected tree in case a node fails and disconnects before sending any notification messages, each node periodically checks if its parent is still alive. If it is not then the node tries to rejoin the tree by attaching itself to its grandparent node. To rejoin the tree by choosing a new node to attach to from the list of published nodes would take more time and would be costlier. Each node maintains a list of connections to its parent and child nodes, along with their physical network properties such as bandwidth and latency. In addition to that, it also needs to keep track of its grandparent node.

Searching for and locating data starts by a data access request at a given node. The search starts at the local data catalog to check if the data is stored and available locally. If it is not, then the node sends a request message to its parent and children. Upon receiving a request the node first checks the source of the request. If the request is coming from a child node, then the search is continued up the tree, and the request is forwarded to the current node's parent. If the request was received from a parent node, then the search goes down the tree, and the request is forwarded to the current node's children. When the request reaches a node that contains the data, a notification message is sent to the initial requester before initiating data transfer.

The algorithm is also designed to include, within the notification message, a list of different target locations where the requested data could be retrieved. This case occurs in a situation where a node publishes the list of data sets that it stores locally, and sends that information to its parent node. Thus, creating a global view of data stored within a subtree at the root of that subtree, and creating a global catalog at the root of the tree. After receiving a list of possible locations, the local data management service uses network performance tools to choose the source that would yield the best data transfer performance. However, in this paper we concentrate our efforts on replicating data files and not replica location indices.

3.3 Replica Management Service

The replica service continuously adapts the configuration of the replica layout to the needs and requirements of the users and to the network performance reflected in the cost model. The replica management service is responsible for initiating data replication, when needed. In addition, once multiple replicas of files are distributed at multiple locations, the data management service at each node transparently locates replicas with the potentially shortest access time to the requested data. Obviously, given the dynamic nature of the Grid, the service can only provide its best estimate given its current view of the Grid. It also determines whether to access a replica or to create a new one. The Grid environment is highly dynamic. The resources availability and network performance change constantly and data access requests also vary with each application and each user. Accordingly, new replicas may be added at different locations and deleted from locations where they are no longer needed. In figure 2 shows the architecture and design of the replica management service.

3.4 Cost Model

Replica creation is mostly based on data access request statistics. These statistics are gathered by the resource monitoring service, RNS, at each node. The algorithm used by the replica management service, RMS, currently takes as input the frequency of data access requests by users. In this work we only consider read-only data sets. Our cost model initially introduced in [14], incorporates and takes into consideration more parameters. In this paper however we only use a limited version of the cost model.

In addition to accumulated access frequencies, 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 and the frequency of cost estimation. The latter parameter is estimated based on the history of the data

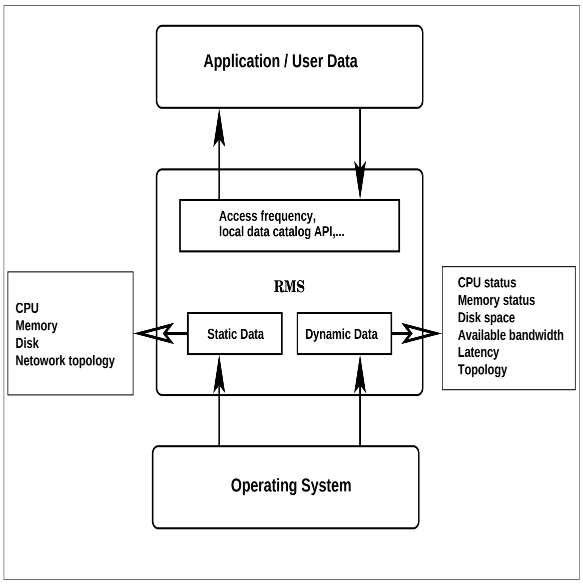

Figure 2: Replica Management Service Architecture

accesses requested at a given site.

To calculate data access cost at a given node in the grid for a given data object we associate each data object i at each node v 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 there is no local replica for object i, then the total data transfer cost for this object at node v is:

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

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.

Creating a replica at node v decreases the access cost for all nodes that belong to the same subtree. Let N be the set of all nodes, and let T_v be the partition of nodes that would be serviced by v for future access requests to object i. Indeed, creating a replica at v decreases the read cost of each node in T_v by $size(i)d(v, c(v, r))$ and increases the write cost of each node in the $N - T_v$ by $size(i)d(v, c(v, r))$ where r is the closest replica location, but it does not change other costs.

Because we currently do not consider write requests, the cost model only accounts for read access requests. Accumulated access requests at node v are added up and stored in the variable $f_v(i)$. To compute the cost of accessing a data set i, the current node treats all accumulated incoming transient requests as locally issued requests and estimates the total data transfer cost based on its local view. The estimated cost is:

$$Cost_{v,i} = f_v(i)size(i)d(v, c(v, r)) \tag{2}$$

if $Cost_(v, i) > \tau$ where τ is a threshold, then a local replica is created. Once a replica is created, $f_v(i)$ is re-initialized.

Another parameter taken into consideration is the storage cost. It 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. 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.

4 Experiments And Results

We conducted experiments to test the efficiency of the spanning tree overlay on a dynamic grid platform with and without replication enabled. The experiments consisted of using a 31 node grid and placing the nodes issuing access requests at the bottom of the tree as leaves. In addition to running the ad-hoc data management agent, each leaf node runs a client host that runs a script which takes as input a list of data sets, and then posts read requests for these data sets according to a selected access pattern. We use the Poisson distribution to model request generation with different arrival rates at each client host. While each node has the ability to issue requests, during these experiments only leaf nodes do so to make sure the traffic load generated travels along the longest paths. Figure 3 shows the modelling and set up of the experiments. The spanning tree of the grid nodes was constructed using the algorithms described in 3.2. To create the overlay network, the join algorithm used the evolving list of newly added nodes which creates a more balanced tree as shown in figure 3.

Each node has some storage space available where it can store a group of data sets. Initially only leaf nodes store different sets of data each. As shown in figure 3, the rightmost and leftmost nodes in the bottom level of the tree store each a Group of Interesting Files (GIF). Each GIF is composed of 10 files, with sizes ranging from 10 to 100MBytes. The rest of the nodes at the bottom level store each a set of files, with an average of 10 files per node. The total number of regular files is 80, with some of the files replicated at multiple nodes. Regular file sizes also ranged from 10 to 100MBytes. Throughout different stages of the experiments, results are collected for different levels of replication. The tests are conducted in multiple phases. In the first phase of testing, the

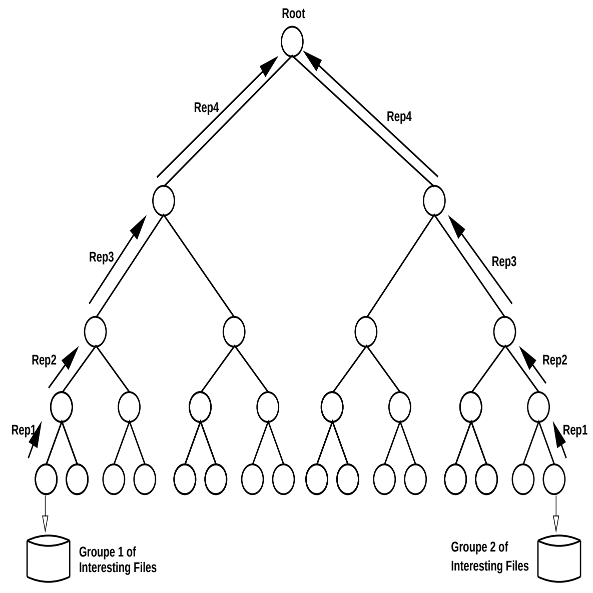

Figure 3: Experimentation Design

access request script is executed on the client hosts located at the leaf nodes. In the second phase when data replication is triggered by data popularity at the higher level in the tree, and new replicas creations are finished, the scripts are executed again with the different conditions for subsequent replica creation at different levels of the tree and results are compared. The total number of phases depends on the height of the tree, in our case the experiments are conducted in 4 phases. Popular data sets are first replicated one level up in the tree and then replicated at the second level, and so on until they reach the top of the tree.

In lack of real trace data, we designed the experiments in a way to emulate existing access patterns within the scientific community. We designated a group of data sets as Interesting Files, meaning that most users were highly interested in those files at a given time. This pattern follows a natural human characteristic, that is when interesting data is published most scientists would be highly interested in checking it out. But their interest might shift to another group of Interesting Files afterwards.

In the first set of experiments we study the improvement of access performance for popular data with different levels of replication vs. no replication. The performance is measured by the number of hops data access request messages go through before reaching a source node containing a copy of the requested data. The results are presented in figure 4. The results show that data access performance improves greatly for popular data with higher levels of replication.

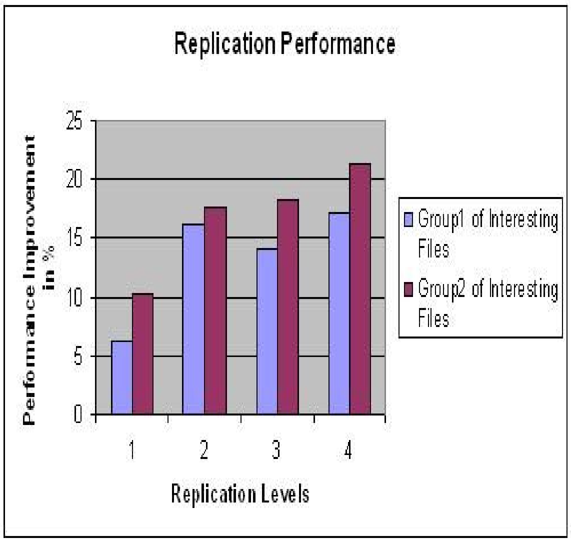

Figure 4: Replication Performance for Popular data

In the next set of experiments, we study the effect of replication on data access performance with different access ratios for popular data ranging from 10% to 90%. During the experiments client hosts running at leaf nodes used different access patterns for tow distinct groups of interesting files, and similarly for multiple other data sets stored at different nodes. The focus of the experiments was on the access ratio of the two Groups of Interesting Files vs. the access ratio of the remaining set of files. The results of different access scenarios using replication were compared to the same scenarios with no replication. A summarized plot in figure 5 shows the accumulated test results for the different access rations with different replication levels for different data sets.

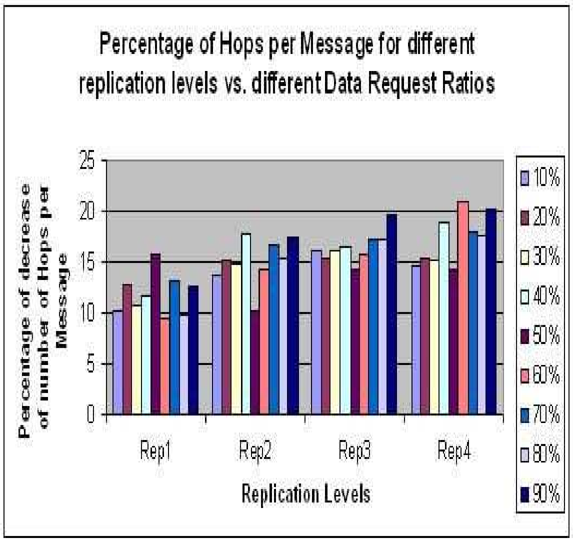

Figure 5: Replication Performance for Different Replication Levels

The results show that data access performance improves considerably with higher levels of replication. Replication improves data location and access performance of the overall requests issued in the system from a minimum of 10% to 22%. The results also show that when the ratio of access requests for a given GIF increases, and the group is replicated in the system accordingly, a subsequent drop in popularity of that group does not result in an immediate improvement in access for the later data sets.

5 Conclusion

In this paper we have introduced a new middleware for managing data in distributed dynamic platforms such as data grids. We have proposed the use of a spanning tree as an overlay network, developed the algorithms to build and maintain the tree. We have also introduced new data management services to dynamically manage replica creation and deletion in a grid environment. Additionally we have implemented the cost model used by the replica management services. The middleware has been tested in a real environment. The experiments we conducted show that the use of an ad-hoc spanning tree as an overlay network to access data combined with dynamic replication techniques is very cost efficient and produces considerable performance improvements. The performance gains ranged from a minimum of 10% to over 22%.

Our cost efficient replica management middleware is a framework for scientific collaborations where researchers from different locations can start a virtual grid to share their results and findings which they can join and leave at different times. In our future work we will continue developing our cost model to include and take into consideration more system parameters. We will also test our approach on larger environments and use different performance metrics such as bandwidth consumption and storage cost.

References

[1] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and S. Tuecke, *Data management and transfer in high performance computational grid environments*, Parallel Computing Journal **28** (2002), no. 3, 749–771.

[2] B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Leigh, A. Sim, A. Shoshani, B. Drach, and D. Williams, *High-performance remote access to climate simulation data: A challenge problem for data grid technologies*, In Proc. of the SuperComputing Conference, November 2001.

[3] D. Bosio, J. Casey, A. Frohner, and L. Guy et al, *Next generation eu datagrid data management services*, Computing in high energy physics (CHEP2003), March 2003.

[4] D. G. Cameron, A. P. Millar, C. Nicholson, R. Carvajal-Schiaffino, F. Zini, and K. Stockinger, *Optorsim: a simulation tool*

for scheduling and replica optimisation in data grids, CHEP 2004, Interlaken, September 2004.

[5] A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf, *Performance and scalability of a replica location service*, Proc. of the International IEEE Symposium on High Performance Distributed Computing (HPDC-13), June 2004.

[6] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and 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** (2001), 187–200.

[7] A. Chervenak et. al, *Giggle: A framework for constructing scalable replica location services*, Proc. of the ACM/ IEEE SuperComputing Conference, November 2002.

[8] I. Foster, E. Alpert, A. Chervenak, B. Drach, C. Kesselman, V. Nefedova, D. Middleton, A. Shoshani, A. Sim, and D. Williams, *The earth system grid ii: Turning climate datasets into community resources*, Proc. of the American Meterologcal Society Conference, 2001.

[9] K. Holtman, *Cms data grid system overview and requirements*, Tech. report, CERN, July 2001, CMS Note 2001/037.

[10] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, *Data management in an international data grid project*, Proc. of the first IEEE/ACM International Workshop on Grid Computing, 2000.

[11] N.R. Kaushik and S.M. Figueira, *Spanning trees for distributed search in p2p systems.*

[12] G. Kola, T. Kosar, J. Frey, M. Livny, R.J. Brunner, and M. Remijan, *Disc: A system for distributed data intensive scientific computing, san francisco, ca*, Proc. of First Workshop on Real, Large Distributed Systems, December 2004.

[13] H. Lamehamedi and Zujun Shentu, *Simulation of dynamic data replication strategies in data grids*, Proc. 12th Heterogeneous Computing Workshop (HCW2003) Nice, France, IEEE Computer Science Press, April 2003.

[14] H. Lamehamedi, B. Szymanski, Z. Shentu, and E. Deelman, *Data replication strategies in grid environments*, Proc. 5th International Conference on Algorithms and Architecture for Parallel Processing, Bejing, China pp. 378-383., IEEE Computer Science Press, October 2002.

[15] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, *Search and replication in unstructured peer-to-peer networks*, Proc. of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems, 2002, pp. 258–259.

[16] D. Nikolow, R. Slota, J. Kitowski, and L. Skital, *Virtual storage system for the grid environment*, International Conference on Computational Science, 2004, pp. 458–461.

[17] A. Rajasekar, M. Wan, and R. Moore, *Mysrb and srb: Components of a data grid*, Proc. of the 11th International Symposium on High Performance Distributed Computing (HPDC-11), July 2002.

[18] K. Ranganathan and I. Foster, *Decoupling computation and data scheduling in distributed data-intensive applications*, Proc. of 11th IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland, July 2002.

[19] K. Ranganathan, Adriana Iamnitchi, and I. Foster, *Improving data availability through dynamic model-driven replication in large peer-to-peer communities*, Proc. of Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop, Berlin, Germany, May 2002.

[20] S. Ratnasamy, S. Shenker, and I. Stoica, *Routing algorithms for dhts: Some open questions*, IPTPS, 2002.

[21] A. Rowstron and P. Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, Germany, November 2001.

[22] M. Russel, G. Allen, G. Daues, I. Foster, E. Seidel, J. Novotny, J. Shalf, and G. von Laszewski, *The astrophysics simulation collaboratory: A science portal enabling community software development*, Cluster Computing, no. 5(3), 2002, pp. 297–304.

[23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, *Chord: A scalable peer-to-peer lookup service for internet applications*, Proc. of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, 2001.

[24] R. Tuchinda, S. Thakkar, Y. Gil, and E. Deelman, *Artemis: Integrating scientific data on the grid*, Proc. of the Sixteenth Innovative Applications of Artificial Intelligence, July 2004.

[25] S. Vazhkudai and J. Schopf, *Using disk throughput data in predictions of end-to-end grid transfers*, Proc. of the 3rd International Workshop on Grid Computing, November 2002, Baltimore, MD.

[26] S. Vazhkudai and J. Schopf., *Using regression techniques to predict large data transfers*, The International Journal of High Performance Computing Applications, special issue on Grid Computing: Infrastructure and Applications **17** (2003), no. 3.

[27] *The european data grid project, the datagrid architecture 2001*, http://www.eu-datagrid.org.

[28] *The gnutella protocol specification*, http://www.gnutella.com.

[29] *Grid physics network (griphyn)*, http://www.griphyn.org.