Future Generation Computer Systems, vol.23, no. 1, January 2007, pp. 109-115.

Decentralized Data Management Framework for Data Grids

Houda Lamehamedi and Boleslaw K. Szymanski

Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

Abstract

A new class of data intensive applications has led to increased demand for cost-efficient resource sharing approaches. Yet, providing efficient access to widely distributed data for large numbers of users poses considerable challenges. Most existing Grid systems are centrally managed, thus hindering their scalable expansion. We introduce a new distributed, adaptive, and scalable middleware that provides transparent access to data in Data Grids. Our approach relies on dynamic techniques that adapt replica creation to continuously changing network connectivity and users behavior. Results from simulations and deployment of our middleware show that our solution provides better data access performance than static approaches.

Keywords: Data Grids, Replication, Middleware, simulation.

1 Introduction

The data requirements for both scientific and commercial applications in many areas such as genomics [6], drug discovery [1], astrophysics [3], geology [2], or climate change modeling [10] are very intensive and stress the increased need for highly efficient and cost-effective resource sharing and management approaches [4]. In response, Data Grids were developed based on multiple middleware solutions that transparently provide access to distributed resources for large number of users. Yet, providing efficient access to widely distributed data is still a considerable challenge.

In currently deployed Grid systems, static replication is used to copy data to sites where it is most requested and popular. There is no support for automatic replica creation and data placement. Additionally, most existing and deployed Grid systems and platforms are centrally managed and are quite difficult to set up and maintain. In these systems, control of resources is centralized and usually handled by system administrators. Such configurations hinder dynamic and scalable expansion of the Grid infrastructure and resources.

To address these issues, new data placement algorithms and replication techniques are needed to ensure efficient access and distribution of data based on real time users' and applications' demands. To that end, we introduce a new distributed, adaptive, and scalable middleware that provides transparent, fast, and reliable access to data and storage resources in distributed resource sharing environments such as Data Grids.

Our approach provides a lightweight framework that supports research and scientific collaborations and enables the creation of larger communities with less overhead than incurred today. This framework supports also the creation of small to medium scale Data Grids with large number of users taking advantage of under-utilized resources. Our approach is inspired by the P2P techniques that require no centralized management and advocate self organization. We evaluate the benefit and applicability of our solution via analytical models, simulations, and emulation. Results from the simulation and the deployment of our middleware prototype show that our solution provides better data access performance with lower resource consumption rates than the currently used static approaches.

2 Background and Related Work

Replication has been studied extensively and different distributed replica management strategies have been proposed in the literature [12,13,20]. Replication has traditionally been used to improve system performance by increasing reliability and fault tolerance. In the context of Data Grids, replication ensures that all users have access to the required data, enables scalability across multiple locations, and optimizes the use and consumption of network resources. A leading effort in building Computational and Data Grids was spearheaded by the Globus team [9] soon followed by additional groups, including the EU DataGrid [8,4,13].

Many studies have revealed that commonly used Grid systems and toolkits do not yet provide sufficient support to a large community of users as originally intended [7]. The Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL) are increasingly adopted as a means to support communication in distributed environments [7,18]. The Open Grid Service Architecture (OGSA) aims to define a common, standard, and open architecture for the Grid [18]. The resulting Grid infrastructure combines and integrates existing Grid technologies [9] and Web services technologies to create a distributed Grid computing framework based on the Open Grid Service Infrastructure (OGSI) [18]. The resulting communication tools however, are not strong enough to support high performance distributed applications.

In some existing Data Grids [8], dedicated nodes are responsible for storing and maintaining replica location indices throughout the system. One major drawback to such an approach is the centralized management and decision making. Dynamic replication presents a more attractive approach as decisions are made based on the current access patterns and the availability of resources.In this paper, we provide a stand-alone framework that is used to provide efficient access to data but could also be integrated, if needed, with a job scheduler. The main goal of our approach is to create a distributed and decentralized mechanism to replicate and manage access to data that is based on continuous and dynamic evaluation of resource utilization and access performance.

3 Decentralized Replica Management

Data access models are shaped by the social organization of the data-sharing community and are affected by the location of data, direction of data flow, and users' access patterns. To ensure efficient and easy access to all users, an organized structure is needed to navigate through the participating sites and locate requested data. In our approach, based on surveys of data sharing networks and existing Data Grid models, we use a combination of spanning tree and ring topologies. Most scientific applications and collaborative research enterprises that we surveyed fit a combination of hierarchical tree topologies and flat topologies.

Data on the Grid needs to be easily accessible to users regardless of location, so to provide easy access, we use a Replica Catalog to support transparent access to data at each Grid node for local and remote users. The Replica Catalog provides an indexing of available data sets and a mapping between file names and their physical location. Each newly created data file or replica is registered in the catalog. Information stored in the catalog is also replicated in some cases at a node's parent or ancestor. Thus, the catalog helps to locate locally stored data or route data access requests to the appropriate Grid node.

Data Organization Models And Topologies used in our approach are based on using application level overlay networks to enable scalable growth of Data Grids and tolerate large number of participants. Topology represents the connectivity graph formed by the *overlay network* defined by the set of application level connections between the participating nodes in the Data Grid. The resulting graph defines the communication paths that can be used to navigate through the network and locate data. However, data transfers are the responsibility of the physical network. The middleware's responsibility is limited to locating data sources and the end points of the transfer. Our proposed framework provides incremental scalability and robustness to 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. 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. Ad hoc spanning trees have been tried and proven to perform and scale well in P2P systems.

The Grid structure is constructed starting from a root node. When joining the grid, the new node is added through an existing grid node by attaching to it as a child node or a sibling. 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 proximity of a node to which the new node needs to attach. For example, a node can choose a parent or sibling 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, siblings, and children. The parent removes the departing node from its children's list as do its siblings. The child nodes contact the departing node's 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. Each node maintains a list of connections to its parent, sibling, 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. In case both a node's parent and ancestor fail, the node rejoins the tree by choosing a new node to attach to from the list of published nodes, using the same mechanism used by nodes joining the tree for the first time.

Replica Location and Access are initiated when access to a data set is needed and the proper request is issued. This request starts a search process that reaches all the possible nodes that may have a copy of this data set. When multiple locations are discovered, all are reported back to the requester who chooses the most appropriate source node. In existing Data Grid implementations, dedicated nodes store information about the locations of possible sources [5]. In a more dynamic platform, new nodes might join the grid and some nodes might leave. This creates the need for an adaptive, dynamic approach to discover, locate, and access data. Similar attempts were used to develop search protocols for peer-to-peer (P2P) data sharing, including the flooding algorithm used in Gnutella [11], the centralized algorithm used in Napster [15], and the distributed hash-table based protocol used in CAN, Pastry and Tapestry [19]. A super peer model was introduced in [16] to study Grid resource discovery based on a P2P model. Many studies have shown that using a combination of flat and hierarchical topologies give the best performance for message broadcasting.

The data search and location process 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, siblings, 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. At the sibling node the request is treated as being received from a parent node and forwarded down the local subtree. 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. The system, in addition to replicating data, supports and enables the replication of meta-data stored in the catalogs. This way nodes can publish the list of data sets stored locally, and send that information to their 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.

Cost Model relies mainly on the frequency of data access requests that are maintained by the resource monitoring service, RNS, at each node. Most existing studies on access patterns in data grids show that majority of access requests are reads. Accordingly, in this work, we only consider read-only data (we will consider write costs in our future work).

In addition to parameters listed above, the replica management service, RMS, also takes into account the storage capacity and availability at a given grid node. The frequency of cost estimation calculations is dynamically adjusted based on the history of the data accesses requested.

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 $cost_{v,i} = (\lambda_{v,i} + \mu_{v,i}) size(i) d(v,r)$, 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 $d(v,r) = 1/bandwidth(v,r)$, where $bandwidth(v_1, v_2)$ is the total available bandwidth between nodes v_1, v_2. Bandwidth is dynamically computed at runtime by computing the time to send and receive a sample files from v to r.

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.

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 and estimates the total data transfer cost based on its local view as follows:

$$Cost_{v,i} = f_v(i)size(i)d(v, c(v, r)) \tag{1}$$

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. τ is calculated based on cost of data transfers between v and a designated node u. The data transfer cost is calculated using the bandwidth estimation using the sample file between v and u and the accumulated access request count. The storage cost is computed based on the state of the data objects, their request frequencies, and their sizes. The least recently and least frequently used replicas are removed to enable the creation of new ones.

4 Middleware Architecture and Framework

The components of our middleware can be organized in a layered architecture in which each layer builds on top of and uses the services offered by the lower layers. Each layer encapsulates a set of services that are described below.

The Communication Layer consists of data transfer and authentication protocols as well as data organization and overlay support. The transfer and authentication protocols are used to ensure security, verify users identities, and maintain data integrity during data transfers. This layer contains the Routing and Connectivity Service.

The Resource Access Layer consists of basic software and tools that provide access to available resources and monitors their usage and availability. This layer also contains the Replica Catalog. The major services encapsulated in this layer are the Replica Monitoring and Location Services.

The Replica Management Layer consists of services that support the management and transferring of data between Grid nodes and the creation of new replicas. This layer's components track users' access patterns, monitor data popularity, and use input from the lower layers to decide if local replica creation is needed or not. The major service offered by this layer is the Replica Creation Service.

A participating Grid node can be any hardware/software that qualifies as a computing unit such as a supercomputer, workstation or desk-top. The middleware consists of autonomous and distributed components that run at each participating Grid node. Each agent is composed of the set of services described below and shown in Figure 1.

Resource Monitoring Service monitors resource availability at a given Grid node and collects statistics about resource usage and data access requests. Data collected by this service is fed to the Replica Creation Service.

Replica Creation Service creates local replicas based on accumulated access statistics and an evaluation of the incurred cost of creating a local replica.

Replica Location Service manages the local replica Catalog. Each newly created file is registered in the catalog. This service is also responsible for locating requested data that is not available locally by using the Routing and Connectivity service to route data access requests. We will refer to the Replica Location and Replica Creation services as Replica Management Service collectively.

Resource Allocation Service allocates space for newly created replicas and deallocates space from the least frequently and least recently accessed local replicas.

Routing/Connectivity Service routes outgoing request messages, handles incoming messages, manages data transfers, as well as monitors a node's connectivity to its neighbors. This service maintains a list of neighbors to which the local node is connected to.

5 Simulation Experiments And Results

To study and validate our approach, we started with simulating different replication strategies to guide our development and deployment of the middleware. To that end we designed and developed a Data Grid simulation framework: GridNet [14], through which we modeled realistic scenarios. The simulations allow us to verify the design and evaluate the performance of our strategies. The simulation environment enabled us to run multiple tests and supported an easy framework to change system parameters and experiment scenarios with very small overhead.

GridNet was developed in C++ and was built on top of the network simulator NS [17]. Each Data Grid node in the simulation 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. The simulator allows us to specify different types of nodes: *client*, *server* and *cache* nodes. A server node represents a main storage site, where all or part of the data within the Data Grid is stored. A cache node corresponds to an intermediate storage site that is expected to have high storage capacity and to replicate part of the data stored on the main storage site. A client node represents a site where data access requests originate and are generated. As shown in Figure 2, each GridNet node consists of a basic NS node, a storage element, a replica manager or a monitoring agent, and maintains a replica routing table.

Experiment Setup And Results were based on a three-tier binary tree Data Grid topology, with a single root server, two levels of caches, and 8 leaves representing clients. We used 28 sites with 90 data sets and 1000 requests. Bandwidth between cache client was 10Mb/s while cache-cache and server-cache were 100MB/s. The simulated events represent access requests, and the presented results were obtained with simulation of only read requests. The simulation generates random background traffic in the network and the stream of requests for the Grid data. We use response time as our performance metric as it incorporates and represents both the data transfer costs and gains of the replica placement strategy.

In Figure 2 we show the results obtained from comparing three different scenarios using different replication policies and storage/cache capacities at different levels of the hierarchy. In Scenario 1, static replication is used with popular files placed at different *cache nodes*. In scenario 2, our dynamic replication is applied with small storage capacities allocated to cache nodes. In scenario 3, the same dynamic replication is used with larger storage capacities at cache nodes. The guiding factor of the replication mechanism is the frequency and origin of access requests from *client nodes*. We used six data groups for our experiments: the first four with sizes ranging over 100Mbytes and starting value ranging from 100Mbytes (for the first data set) to 600Mbytes for data set four. The last two groups varied sizes over 50Mbytes with starting values 800 and 950 Mbytes, respectively. The workload at each node consists of running tasks that require both file access operations and the processing of the accessed data. The results shown in Figure 2 indicate that replication improves overall workload performance by up to 60%. With higher network bandwidths, the performance of high workload scenarios increases noticeably, and the gains and benefits are more considerable for larger file sizes.

6 Middleware Deployment

The testbed for our experimentations consists of a cluster of 40 Linux machines with dual processors and 20 Sun workstations. Both systems were used to emulate virtual organizations running across multiple organizational domains. The workstations are running the FreeBSD operating system. As a member of a virtual collaboration and organization, each machine runs the data management middleware and contributes some of its local storage space. The data shared by the members of the Grid is placed in the local disk of each machine. The access permissions on the files determine the users who have the right to access them. Due to the lack of real Data Grid trace data, we designed the experiments in a way to emulate existing access patterns within the scientific community.

For our experiments we use two models, a *top down* and *bottom up* models. The top down model, shown in Figure 3, generates data access requests mostly at the bottom layer of the hierarchy. All data is originally stored at the root of the hierarchy. Access requests drive the dynamic replication of popular data files at different levels of the hierarchy. The data requests are emulated using a group of 100 files of sizes ranging from 50MBytes to 500Mbytes. User applications generate data access requests using a Poisson distribution. Among the 100 files used in the experimentations, we designated a group of 20 data sets as *Interesting Files* that most users are interested in accessing when it is published. Subsequently, the spike in users' interest may shift to other groups of *interesting files* as they become available. The access requests are modeled in a train fashion; i.e., each set of random requests to regular files is followed by a set of requests to interesting files.

In the *bottom up* architecture, shown in Figure 4, the nodes at the bottom of the hierarchy are the original sources of data. We designed the architecture in such a way that the leftmost and rightmost nodes at the lower level of the tree store two groups of *Interesting Files.* The remaining nodes at this level each store a different group of data sets. Data access requests are generated by these same nodes at the lower level. The same data sets described above and used for the *top down* model are also used in these experiments. A similar access pattern as described above was also used in these experiments.

In addition to running the data management agent, each leaf node runs an application that consists of taking as input a list of data sets, and then posting read requests for these data sets according to a selected access pattern, then processing the files. While each node has the ability to issue requests, during these experiments only leaf nodes do so to maximize the amount of traffic load generated.

Experiments Results compare the access response rates at different levels of the hierarchy using different cost thresholds for the replication algorithm. If the cost of not replicating a requested data file exceeds the threshold then the file is replicated. The threshold values used in the experiments were chosen empirically to represent an estimated cost of data transfers between two different nodes in the Grid.

In the *top down* experiments we measure the data hits and responses registered before data is replicated at the nodes at the lower level of the tree. The results shown on the right of Figure 3 indicate that replicating data using higher cost thresholds delays the propagation of replicas to lower levels of the hierarchy since most hits occur at its top level. The results also show that Lowering the cost threshold triggers data replication at lower levels of the hierarchy at a faster rate, thus lowering network bandwidth consumption at higher levels of the structure and decreasing contention and creation of bottlenecks at the root.

In the next set of experiments we use the *bottom up* model. In this model initially, only leaf nodes store data, on average 20 files per node. The total number of regular files is 80. Throughout different stages of the experiments, results are collected for different scenarios where both storage availability at each node and replication threshold are updated. Figure 4 shows the response rate at both the lower and upper levels of the tree with two different storage capacity values and different cost values. The results show that an increased storage capacity decreases considerably resource consumption by replicating more data at locations closer to frequent users. Access requests at data sources decrease from over 80% to less than 20%. Since data requests are originating from other leaf nodes, this means that less bandwidth is consumed when data is dynamically replicated based on user demands.

The plots in Figure 4 show the performance of the system under two different storage space capacities at the top of the tree. With higher storage capacity available at the top level (*storage2*), more requests are answered at the top level. Decreasing that capacity pushes nodes at lower levels to replicate more of the requested and popular data, thus decreasing the need for requests to travel to the top of the tree. Higher cost threshold values decrease the capacity of the Grid nodes to replicate larger number of files at a faster rate. Figure 4 shows that with a higher storage capacity, more data is replicated at different levels of the tree thus decreasing traffic load to the original data sources and the overall bandwidth consumption. The results also show that with higher cost threshold values about 50% of the access requests get a hit at a lower level of the tree with a higher storage capacity. Additionally, the plots show that a lower storage availability at the lower level of the tree triggers replication at higher levels of the tree at a faster rate, thus reducing traffic load at the data sources. Higher storage capacity enables more nodes to replicate more files, thereby, reducing the overall traffic load in the network. The overall results show that our decentralized, dynamic, cost based replica management middleware and services improve data access performance by up to 30%.

7 Conclusion

In this paper we introduce a new distributed and decentralized data management middleware for Data Grids. This is an adaptive and scalable lightweight framework that enables users to dynamically join and leave the grid. Our replication management algorithm intelligently and transparently places data in strategic locations improving the overall data access performance. At the core of our system lies an analytical model that enables each participating node to decide when and what resources to contribute. The middleware enables each node to monitor and control its local storage space and capacity, access to locally stored files, use of network resources, as well as any other available local resources. Our approach takes also advantage of data organization models that are commonly used and popular in data sharing environments and data intensive applications.

The results of our simulations and experiments from deploying the middleware show that dynamic cost based replication outperforms static user initiated replication. The results also show that the choice of parameters and system variables used to compute and evaluate data access costs are key to ensuring good access performance.

References

[1] 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.

[2] R. Blaheta, P. Byczanski, O. Jakl, R. Kohut, A. Kolcun, K. Krecmer and J. Star, "Large scale parallel FEM computations of far/near stress field changes in rocks," Future Generation Computer Systems, 22(4):449-459, 2006.

[3] R. Bondarescu, G. Allen, G. Daues, I. Kelley, M. Russell, E. Seidel, J. Shalf and M. Tobias, "The Astrophysics Simulation Collaboratory Portal: a framework for effective distributed research," Future Generation Computer Systems, 21(2):259-270, 2005.

[4] D. Bosio, J. Casey, A. Frohner, L. Guy et al, "Next Generation EU DataGrid Data Management Services", *Computing in High Energy Physics (CHEP2003).*

[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] C. Chen and B. Schmidt, “An adaptive grid implementation of DNA sequence alignment,” Future Generation Computer Systems, 21(7):988-1003, 2005.

[7] D. Davis and M. Parashar, “Latency Performance of SOAP Implementations”, *Proceedings of the IEEE CCGrid 2002.*

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

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

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

[11] “The Gnutella protocol specification”, http://www.gnutella.com.

[12] R. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek, ”Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication,” *Workshop on Mobile Data Access*, November 1998.

[13] P. Kunszt, E. Laure, H. Stockinger and K. Stockinger, “File-based replica management,” Future Generation Computer Systems, 21(1):115-123 2005.

[14] H. Lamehamedi, B. K. Szymanski, Z. Shentu, E. Deelman, “Simulation of Dynamic Data Replication Strategies in Data Grids” *to appear in Proceedings of IPDPS’03, Homogeneous Computing Workshop*, Nice, France April 2003.

[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] C. Mastroianni, D. Talia and O. Verta, ”A super-peer model for resource discovery services in large-scale Grids”, Future Generation Computer Systems, Volume 21, Issue 8, October 2005, Pages 1235-1248.

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

[18] K. Czajkowski, A. Dan, J. Rofrano, S. Tuecke, and M. Xu, “Agreement-based Grid Service Management (OGSI-Agreement)”, *Global Grid Forum, GRAAP-WG Author Contribution, June 2003.*

[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] M. Tang, B.-S. Lee, X. Tang and C.-K. Yeo, “The impact of data replication on job scheduling performance in the Data Grid, Future Generation Computer Systems, 22(3):254-268, 2006.

[21] B. Tierney, W. Johnston, J. Lee and M. Thompson, ”A data intensive distributed computing architecture for ”Grid” applications”, Future Generation Computer Systems, Volume 16, Issue 5, March 2000, Pages 473-48.

[22] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. Miller, D. Long, and T. McLarty, “File system workload analysis for large scale scientific computing applications”, *In Proc. of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies, pp 139.152, April 2004.*

Biographies

Houda Lamehamedi received her Ph.D. degree in Computer Science at Rensselaer Polytechnic Institute on November 2005 under the supervision of Prof. B. Szymanski. She received her MS in computer Science from Al-Akhawayn University in Ifrane, Morocco in 1998. She recently joined Oracle. Her work involves investigating resource and workload management in distributed enterprise applications.

Boleslaw K. Szymanski is a Professor at the Department of Computer Science. He received his Ph.D. in Computer Science from National Academy of Sciences in Warsaw, Poland, in 1976. 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 parallel computing networking.

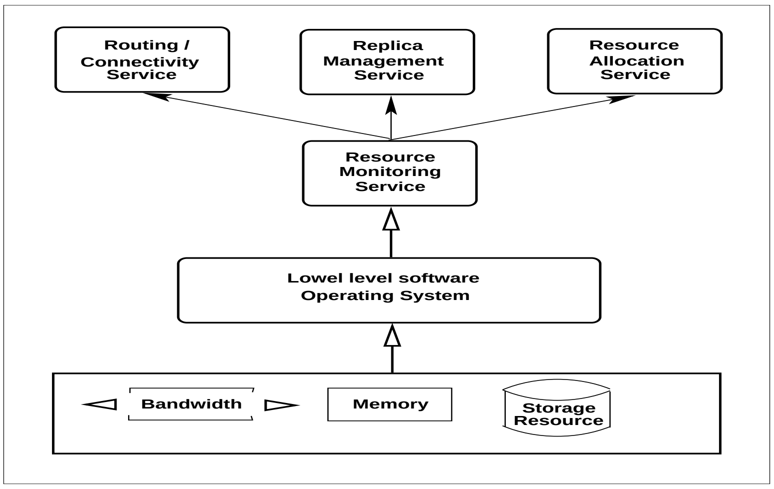
Routing / Connectivity Service
Replica Management Service
Resource Allocation Service
Resource Monitoring Service
Lowel level software Operating System
Bandwidth
Memory
Storage Resource

Fig. 1. Data Management Services

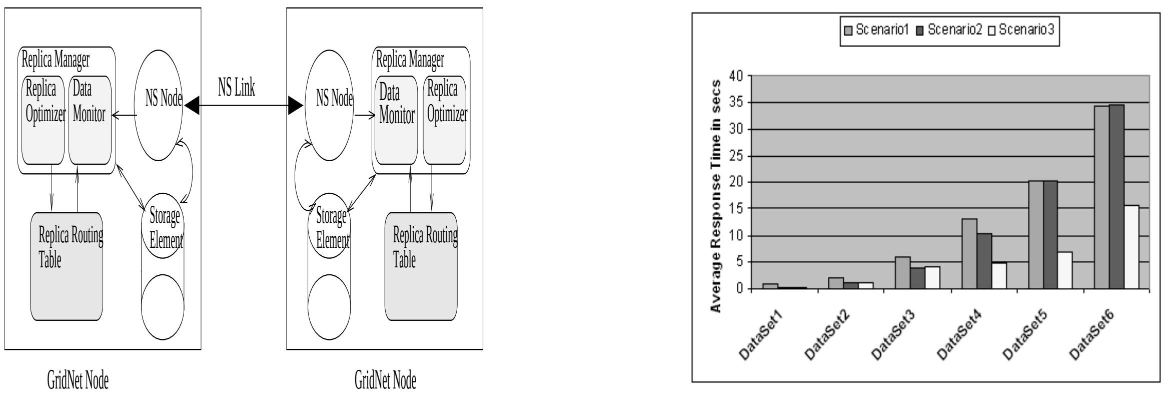

Fig. 2. Simulation Architecture And Simulation Results

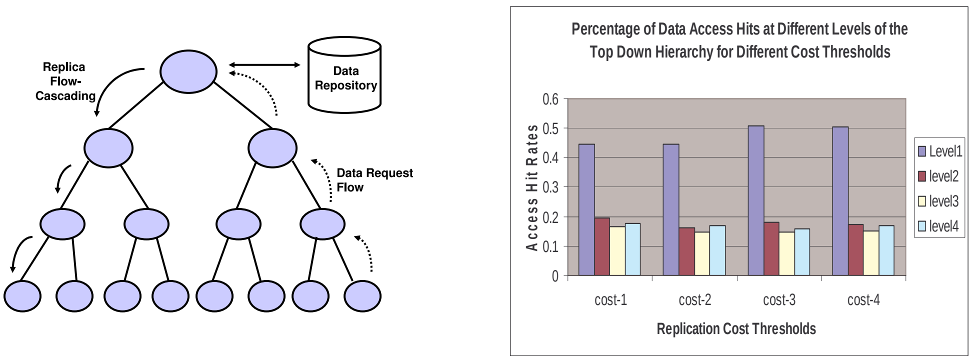

Fig. 3. Experiment Set Up and Results For Top Down Data Model

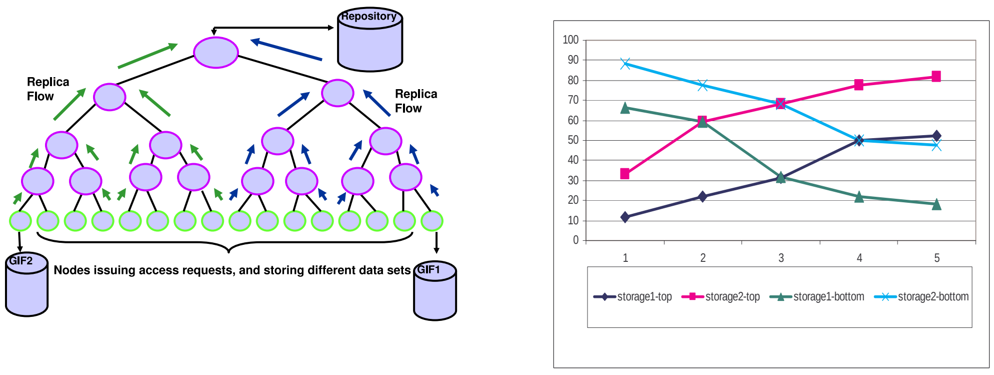

Fig. 4. Experiment Set Up and Results For Bottom Up Data Model