DECENTRALIZED DATA MANAGEMENT FRAMEWORK FOR DATA GRIDS

By

Houda Lamehamedi

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved by the
Examining Committee:

Boleslaw K. Szymanski, Thesis Adviser

Joseph E. Flaherty, Member

James H. Kaufman, Member

David R. Musser, Member

Carlos A. Varela, Member

Rensselaer Polytechnic Institute
Troy, New York

November 2005
(For Graduation December 2005)

DECENTRALIZED DATA MANAGEMENT FRAMEWORK FOR DATA GRIDS

By

Houda Lamehamedi

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Boleslaw K. Szymanski, Thesis Adviser
Joseph E. Flaherty, Member
James H. Kaufman, Member
David R. Musser, Member
Carlos A. Varela, Member

Rensselaer Polytechnic Institute
Troy, New York

November 2005
(For Graduation December 2005)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

An emergence of a new generation of data intensive universal co-operation and collaborative applications has led to increased demand for highly efficient and cost-effective resource sharing and problem solving. Data Grids provide an environment and a framework that supports and coordinates the access of widely distributed storage and compute resources to large numbers of users. In Data Grids data and data management utilities are treated as first class citizens. The main focus of Data Grids is providing users with an infrastructure that enables and facilitates reliable access and sharing of data, access to storage resources, and data transfer services that can scale across widely distributed locations. Yet, providing efficient access to huge and widely distributed data is still a considerable challenge.

Most existing and deployed Grid systems and platforms are centrally managed and are quite difficult to set up and maintain. Proper access to software and hardware resources requires meticulous installation, configuration, and testing of different components across all the participating Grid nodes. In such systems, 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. The tremendous growth in data requirements for both scientific and commercial applications in the last few years stresses the need for new data placement algorithms and access tools that can break administrative and geographical barriers. This new generation of universal co-operation and collaborative applications require new approaches to ensure efficient access and distribution of data and resources based on real time users' and applications' demand.

In this thesis we propose new lightweight 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. Strategically placing data near the user and her application offers considerable benefits and is key to our solution. At the core of our approach are dynamic data placement and replica location techniques that adapt replica creation and location to the continuously changing network connectivity and users behavior. The corresponding framework is fully distributed, self configuring, scalable to the large numbers of users, and supportive of dynamic growth of the underlying infrastructure.

We evaluate the benefit and applicability of our proposed solution via analytical models, simulations, and emulation. Results from the simulation and the deployment of our middleware prototype using widely observed and popular data access patterns show that our solution provides better data access performance with lower resource consumption rates than the static approaches.

Acknowledgments

I am grateful to my advisor Dr. Boleslaw Szymanski for his enthusiastic and effective supervision and support. Dr. Szymanski provided me with many opportunities to travel, seek and forge new collaborations, publish, and present. He created an environment that helped me carry out effective research and address challenging problems. His encouragements and guidance have made this work possible.

Throughout this work I have met and worked with many fine and great people. Dr. James Kaufman from IBM Almaden Research center was a great mentor to me. Working with him and his team, Dr. Toby Lehman and Glenn Deen, at IBM research gave me the opportunity to explore new ideas and improve my software development skills. I am very grateful to Dr. Kaufman for believing in me and in this work and for supporting it.

Parts of this work would not have been possible without the active contribution of my colleagues Zujun Shentu and Brenden Conte. I thank them for the long discussions we had, the papers they helped write, and their valuable feedback.

Deep gratitude to my doctoral committee members for their valuable suggestions and comments. Their proactive support, advice, and ideas have been critically essential to this research.

I am very thankful to my colleague David Bauer for his uninterrupted help and support with using the cluster and system related issues. I am also very thankful to Lindsay Todd for providing me with access to additional machines, and helping install all the necessary software and tools. Their help was critical to getting timely results.

I am grateful to the Fullbright Scholarship and IBM research for providing support for my academic education and academic training through academic fellowships and grants.

I thank all the RPI Computer Science Department's staff for making the department such an enjoyable place to work. Special thanks to Christine Coonrad, Pamela Paslow, Shannon Carrothers, Jacky Carley, and Terry Hayden for their assistance and help with all the non technical issues.

Last but not least, I am forever indebted to my parents, my late grand mother,

and my husband, Jamal Faik for their patience, inspiration, comprehension and encouragement when it was most needed.
This thesis is dedicated to them.

CHAPTER 1
Introduction

In this chapter we present the motivation and inspiration behind computational and Data Grids, present an overview of existing Data Grid technologies, and highlight the need for a replication management middleware.

1.1 Motivation for Data Grids

The motivation for Grids was initially driven by large-scale, resource intensive applications that require more resources than available in a single computing unit, be it a workstation, a supercomputer, or even a cluster within a single administrative domain. The emerging trend in scientific applications in many areas such as high energy physics, data mining, and large scale simulations suggests and shows that these applications process and produce large amounts of data [17, 16, 15, 14, 28, 27, 46, 37, 75]. The resulting output data needs in turn to be stored for further analysis and shared with collaborating researchers within the scientific community who are spread around the world. Facilitating collaborative research requires a new computing paradigm that can break administrative domains and organizational barriers in order to enable multi organizational co-operations and collaborations that benefit scientific research and the community at large. Grid Computing is a computing paradigm that enables the aggregation of large scale computing, storage, and networking resources. A Grid provides an environment where a widely distributed scientific community shares its resources, across different administrative and organizational domains, to solve large-scale compute- and data-intensive applications and collaborate on a wide variety of disciplines. A Grid, therefore, enables the creation of a virtual environment encompassing a pool of physical resources across different administrative domains; these resources are then abstracted into computing or storage units that can be transparently accessed and shared by large numbers of remote users.

The concepts used in Grid computing are not new. The invention of networking and the introduction of distributed operating systems enabled the access of resources in geographically distributed locations [30]. More technological advances brought up by parallel processing and distributed computing allowed not only the remote access of resources but also the simultaneous access and sharing of these distributed resources by different remote users [30]. Parallel processing enabled different tasks to be run simultaneously on different computers, usually homogeneous computers, and to compete for access to computational resources. In distributed computing, users are able to access and use widely distributed heterogeneous computers to run jobs that require more resources than available in local networks and laboratories. The emerging need for using more resources and collaborative problem solving in cost efficient ways led to the development of middleware solutions that transparently provide access to distributed resources and route data from back-end sources to end-user applications in a seamless scalable manner; this became known as meta-computing and later computing on the Grid [39]. Similar to the electrical power Grid, the underlying infrastructure of computational Grids aims to provide reliable, pervasive, and easy-to-use access to resources in widely distributed environments. The Grid offers an integrated architecture that enables and coordinates resource sharing for large numbers of users and applications with dynamic behaviors and continuously changing access patterns [39]. Peer-to-Peer (P2P) systems provide another resource sharing environment that benefits from these technological advances as well. These systems however have a different objective, to facilitate access to widely distributed autonomous resources that are intermittently unavailable and unreliable [6, 74, 80]. These systems have grown tremendously in their user base, forming communities of millions of participants across the world. However, the Grid and P2P target different communities with different service requirements and security guarantees. In contrast to P2P, current Grid participants are well known scientific organizations and institutes and well established scientists in their communities and share high levels of trust and accountability. Resources in the Grid are mostly dedicated to scientific research with high rates of availability. But the scale and success of P2P systems can help the scientific community to apply and adapt tested and proven approaches to design a more flexible and dynamic large resource sharing platform.

In many areas such as genomics [16], drug discovery [15], high energy physics [46], astrophysics [75], and climate change modeling [17, 37], the amount of data required to simulate natural phenomena and conduct experiments is very large. These experiments in turn generate large data sets that need to be collected and appropriately stored in geographically distributed designated data centers to be accessed for further processing at different additional sites [14, 23, 28, 27, 46, 48, 84]. With increasingly growing levels of global collaborations and cooperations between researchers, data storage and collection centers are fast spreading around the world. Conducting scientific experiments requires, in most cases, extensive computations and access to sophisticated instruments, which are in turn located in remotely distributed laboratories. Experiments in high energy particle physics such as those running at the European Center for Nuclear Research (CERN) the Compact Muon Selenoid (CMS) detector and ATLAS [46, 4, 7, 79] designed to study particle physics produce and collect massive amounts of data and involve thousands of researchers from all around the world. The goal of these experiments is to find rare events that are produced from the decay of massive new particles. In Astronomy the Sloan Digital Sky Survey (SDSS) is the most ambitious experiment in this field with a goal of producing a detailed image of a quarter of the sky and determining the positions and brightness of more than 100 million celestial objects [7]. These experiments and scenarios emphasize the challenges introduced by combining the management of large amounts of data and computing resources in Grid environments. Massive amounts of data are currently being produced by the aforementioned research for scientific analysis; some experiments indeed produce over a petabyte of data in one year. To effectively and efficiently address these challenges, a framework that enables the transparent access and sharing of widely distributed large data sets and computing resources is needed. A general design framework has been proposed as the Data Grid architecture and was introduced by Foster et al in [39]. The Data Grid is a Grid where data and data management utilities and resources are treated as first class citizens. A Data Grid is mainly focused on providing users with an infrastructure that enables and facilitates reliable access and sharing of data management resources, and data access and transfer services that can scale across widely distributed locations and reach across wide area networks and break administrative and geographical barriers. This new generation of universal co-operation and collaborative applications requires new approaches to ensure efficient access and distribution of data resources based on real time users' and applications' demands. Thus, intelligent resource allocation and scheduling are needed to enable users to take maximum advantage of the Data Grid infrastructure.

This thesis aims to introduce a new data management middleware that enables the aggregation of widely distributed storage resources to facilitate resource sharing on the scale of P2P systems while maintaining the reliability and levels of service offered by a Grid. Our solution is inspired by P2P approaches but draws its specificity from the requirements and needs of a large scientific community that is geared towards large-scale data and compute intensive collaborative applications.

1.2 Data Management Architecture in Data Grids

A Data Grid connects a collection of hundreds of geographically distributed computers and storage resources to facilitate sharing of data, storage resources, and computational power [17]. Collaborating scientists can form a Virtual Organization (VO) [40], which enables them to access different resources over Wide Area Networks (WANs) without regard to their own organizational and administrative domains. The size of the data that needs to be accessed in these VOs is on the order of petabytes today and is fast growing [46, 7]. In addition to accessing large amounts of data, most collaborative applications running across VOs in grid environments require simultaneous and coordinated access to extensive computational power to process and analyze this data. Ensuring efficient and reliable access to such huge and widely distributed data is a major challenge to network and Grid designers. The major barrier to fast data access in a Grid is the high latency of communication in WANs, which impacts scalability and fault tolerance of applications running on the Grid. To address these problems and enhance performance, replication has been widely used to place copies of data sets across different domains within the Grid. The users, however, still need to discover the closest replica locations to optimize the use of network resources and improve access performance. Current technologies and initiatives to address data grid issues include among others: The Globus project [38], the European DataGrid initiative [4], the GriPhyN (Grid Physics Network) project [7], and PPDG (Particle Physics Data Grid) [7]. The Globus toolkit [38] enables the deployment of a metacomputing infrastructure, which provides basic capabilities and interfaces to facilitate communication, resource location, resource scheduling, authentication, and data access. Figure 1.1 gives an overview of the Grid infrastructure and the services it provides and how they interact. We will describe and provide further details about the existing Globus data management architecture in Chapter 2. The Data Grid architecture in Globus [28] addresses some of these issues by providing some data management services. The main components of that architecture are a file transport protocol, GridFTP, and a replica location service. The current implementation, however, uses only static or user-driven replication services to manually replicate data files using GridFTP. Globus services do not provide any support for automatic replica creation or management.

In most high energy physics applications, data distribution follows a tier hierarchy. In the CMS and ATLAS experiments the Data Grid system spans over worldwide distributed locations, and is organized in “Tiers”. Tier 0 represents the main site located at CERN, Tier 1 encompasses national centers, Tier 2 represents regional centers that cover one region of a large country such as a state in the US or a smaller country, Tier 3 represents workgroup servers, and Tier 4 the (thousands of) researchers workstations and desktops. In this scenario all data is collected at CERN, the European Center for Nuclear Research located in Geneva Switzerland. It is preprocessed online and stored in the CERN computer center, also known as Tier 0 in the data grid hierarchy. Subsets of that data are then replicated at national centers in France, Germany, Italy and the US, in particular at the Fermi Laboratory. Meanwhile, smaller subsets of the data are replicated at individual institutions such as Caltech. Existing systems such as the Globus Replica Location Service (RLS) [27] enable users to locate replicas of a given data file based on its assigned logical filename. A physicist working at Caltech can use the RLS to find the location of data

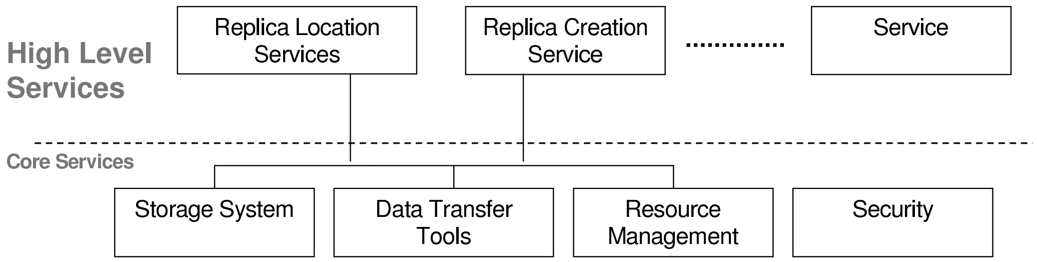

Figure 1.1: Overview Of Grid Architecture And Services.

originally collected at the experiment site. It is possible that the data is located on a storage server at Caltech, in which case access to the data is fast, but if the data is located only at CERN, network latencies are likely to result in slow data access. Clearly a good replication strategy is needed to anticipate and/or analyze the users' requests for data and to place subsets of the data and replicas at strategic locations.

Given the size of the data sets, it is impossible for a human to make decisions about where the data needs to be placed. An automated system is needed that can take into account the data access patterns across multiple users and applications, guided by a cost model for data replication. Not all experiments fit the high-energy physics data model. In some cases, data can be collected at multiple sites, replicated to other locations, and then shared among collaborators. This type of replication scenario can be found, for example, in the gravitational wave community, where there are multiple detectors (two in the US and two in Europe) [7]. These different replication schemes can be classified into hierarchical and peer-to-peer–like patterns, and in order to achieve good performance in a replication system these patterns need to be reflected in the replica management system.

In this thesis, we propose to improve the Grid's data management infrastructure by employing highly distributed intelligent replication and caching of objects at strategic sites. Experience from distributed system design shows that replication offers many advantages over non-replicated data systems. Those advantages include high data availability, low bandwidth consumption, increased fault tolerance, and improved scalability of the system. The performance of replication-based systems depends on a variety of factors, such as the data placement policy. To address the data placement policy we use a cost model to evaluate the gains and losses of replicating data objects before deciding when and where to create and place new replicas. The cost model is formulated as an optimization problem where different performance metrics are evaluated against the different optimization goals and quality of service requirements. To support replica consistency, the data distribution graphs used to connect Grid nodes enable scalable replica distribution and propagation under user-specific guidelines. Update propagations are only instantiated by direct user-defined directives. Our approach provides a more general and robust architecture based on the deployment of dynamic replication using a set of data management middleware services. These services enable the automatic and dynamic replication of data when needed by dynamically adapting to changes in user request access patterns and network behavior. An important aspect of replication management is the replica placement policy. Different system parameters such as network and user behavior, compute and storage resource availability should be taken into consideration before deciding when and where to place new replicas. It is expected that data updates will be infrequent in the physics applications. However, it is necessary to guarantee that the updates will be eventually propagated and that the users will have access to consistent copies of the data.

The data management middleware we introduce in this thesis 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 access cost of the replication scheme is calculated based on the most important factors, such as accumulated read/write statistics, network latency, response time, bandwidth, replica size, and storage availability and capacity. Our replication cost model is formulated as an optimization problem that minimizes the total sum of the data access and replica maintenance cost on a given node in the Grid. The replica maintenance costs include among others the cost of memory or disk space occupied by replicas. To accommodate the wide-ranging requirements of replication for applications such as those described above, our system supports a combination of hierarchical and P2P replication models. These models are used as an overlay structure to virtually connect participating nodes in the data grid. Our approach exploits different structures to decrease data access time and to lower bandwidth consumption. In the early stages of our research, we developed a Grid simulator, GridNet [57, 56] in order evaluate the performance and scalability of our approach. GridNet provides a generic and modular simulation framework through which we can model different Data Grid configurations and resource specifications. As an initial study, we have implemented replication scenarios to evaluate our approach and reported results in [57, 56]. The results of the simulation are very promising, and show that dynamic replication outperforms the static approach. The core of the simulator is based on a decentralized and distributed replication model, which dynamically adapts to both user and network behavior while improving the performance of the overall system. This same model forms the basis of our approach to developing middleware that enables the creation of small to medium scale Data Grids and the formation of virtual organizations. This middleware can foster and encourage collaborative problem solving between multiple individual researchers or research organizations without the added overhead of centralized management.

1.2.1 Data Grid Design Requirements

In the previous sections we outlined some of the aspects of the replication management system we will address. An important aspect of replication-based systems is the protocol used to maintain consistency among object replicas. The main issue in such systems is maintaining scalability with large numbers of replicas distributed over wide area networks while maintaining the same view of all replicas. This issue has been addressed in previous distributed file systems, databases, content distribution networks, and web applications by the use of optimistic consistency protocols [13, 42, 43, 70, 69, 76]. However the consistency issues have not been addressed on the scale of the Grid environment, across multiple organizational domains. Additionally, some of the requirements for the above systems are different from those of the applications targeted in this work. The sizes of the data stored in the Grid are much bigger than those supported by existing distributed file systems, and the replica granularity is much higher. Moreover, in the Grid environment we assume that updates to the data are infrequent and that the consistency can be more relaxed than in high-performance commercial databases. Given these requirements we have used an approach that achieves greater scalability and reaches over wide area networks while making modest compromises in terms of update propagation and replica synchronization. Replica synchronization is initiated by direct user-defined semantics. In the next chapter we will explain in further detail the advantages of using optimistic replication protocols vs. strong or conservative protocols, and present a short survey of the different computing areas where such protocols are used.

1.2.2 Data Grid Applications

To address this massive data challenge, efficient schedulers are needed to appropriately allocate resources by coordinating computations with data access requests. The resource management and scheduling services need to continuously adapt to changes in the availability of resources. A number of Grid implementations, such as Globus [50, 38], have addressed many of these issues with the exception of providing dynamic replica management services. Moreover, the emerging need to solve larger-scale computational problems creates the associated need and challenge to efficiently store and locate the data needed for these computations.

We have surveyed a large number of projects to gather real trace data and data access patterns for large scientific applications running on Data Grids. Such projects include GriPhyN (Grid Physics Network) [7], an NSF-funded project, which

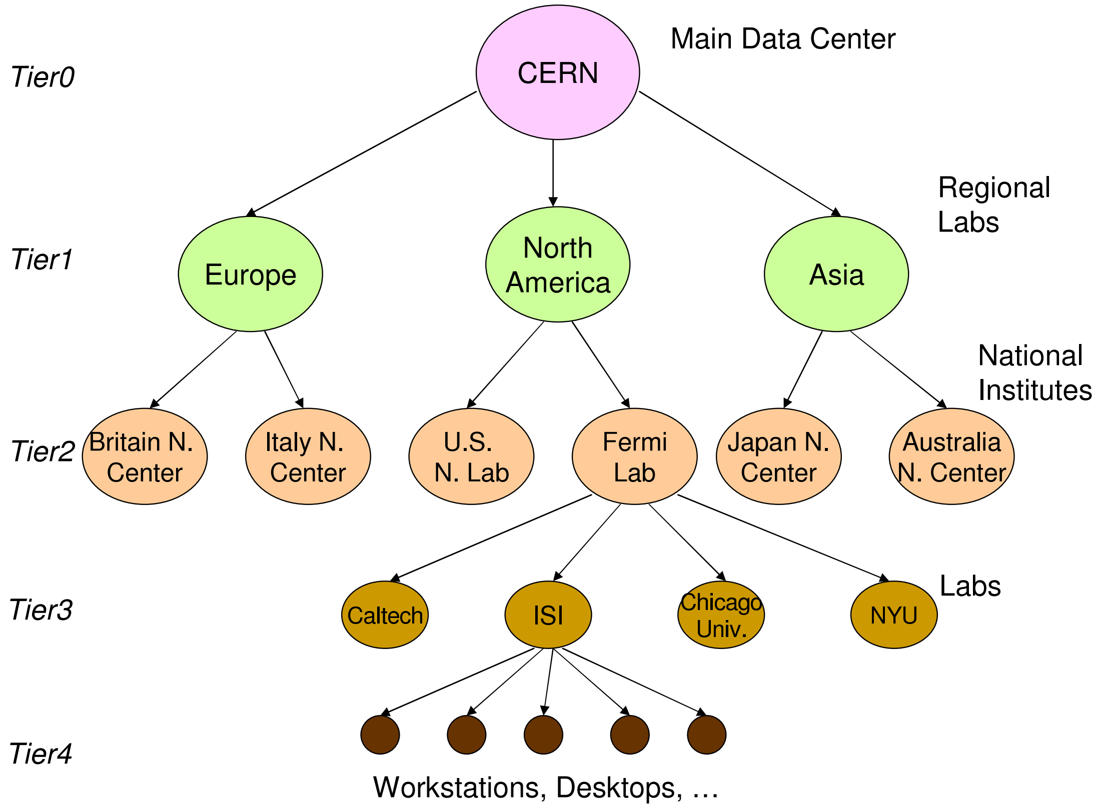

Figure 1.2: Hierarchical Data Distribution And Organization

is developing the concept of a Virtual Data Grid [33, 34, 35], where users can request data without needing to know whether it is available on some storage server or whether it needs to be computed. Another such project is PPDG (Particle Physics Data Grid), a DOE-sponsored project, which is designing Grid-enabled tools for the data-intensive requirements of particle and nuclear physics. Applications involved in these projects are the high-energy physics experiments CMS and Atlas, and the gravitational-wave physics experiment Laser Interferometer Gravitational Observatory LIGO and astronomy surveys such as Sloan Digital Sky Survey SDDS, which deal with large amounts of data. In particular, we used the requirements defined by these applications to guide our design decisions. These applications share the need to access huge amounts of data distributed over WANs, yet they have different data access patterns. In high-energy physics experiments, data is collected at a single

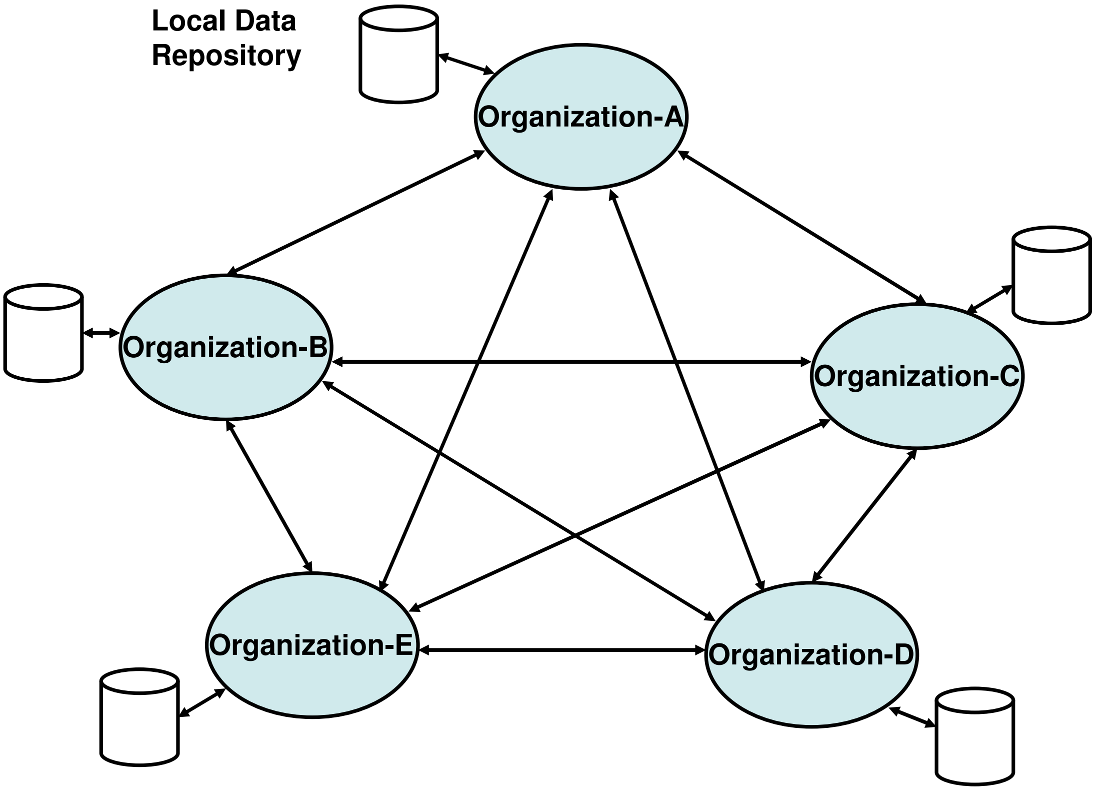

Figure 1.3: Data Sharing Model in a Federation Organization

location, where the detector is located, for example CERN, and then shared by collaborators in Europe and the US [46]. This model fits a hierarchical data distribution and organization model as described in Figure 1.2. In LIGO, data is collected at two locations, at the instrument sites in Washington State and Louisiana, and then replicated at two other sites, Caltech and University of Wisconsin Milwaukee [35]. The data distribution and organization patterns of this application fit the pattern and model displayed in Figure 1.3.

1.3 Thesis Roadmap

The main focus of this thesis is the design and development of a fully decentralized data management solution to support large scale data and compute intensive applications running on the Data Grid. Our proposed solution uses concepts that are inspired by P2P networks and shaped to fit the demands and needs of a more complex and larger scientific community. At the core of our solution is the use of autonomous replica and resource management components running at each participating Grid node. Our proposed framework can be easily integrated with existing low level Grid services and adheres to established standards in Grid computing.

Most existing and deployed grid systems and platforms are centrally managed [14, 23, 27], and are quite difficult to set up and maintain. Enabling dynamic node addition and deletion would provide efficient access to resources on the data grid but presents considerable challenges to system designers. A number of studies and surveys have shown that most existing Grid systems and toolkits do not yet provide sufficient support to a large community of users as originally intended by the designers and developers of these systems [32]. To alleviate and address some of these issues a more adaptive and scalable lightweight data management framework that enables users to dynamically join and leave the grid is needed. In this thesis we show that our proposed middleware can be efficiently deployed in a data grid and can provide efficient, transparent, fast, and reliable access to data and storage resources on the Grid. Our approach is inspired by P2P techniques that require no centralized management and advocate self organization. The P2P approach has many attractive features that make it very suitable for grid computing. The simplicity of the approach makes it easy to deploy on large numbers of nodes. Actions taken by nodes in a P2P system are based on local information, an important feature that enables P2P systems to scale well.

This thesis is organized as follows:

Chapter 2: Data Management And Replication in Data-Centric Networks. In this chapter we survey standard and currently used Data Grid architectures and discuss their infrastructure and the services they offer. We also compare and contrast the use of replication in Data Grids vs. distributed data-centric environments and networks.

Chapter 3: Distributed Data And Replica Management Framework. We present the motivations and goals of our proposed work. We describe our solution and approaches to addressing the replica management issues. We then present and describe the main components of the proposed system and its design.

Chapter 4: Distributed Replica Management Middleware Architecture and Development. We provide more details about low level and architectural design decisions, as well as a thorough description of key components and their implementation.

Chapter5: Data Grid Simulation Study. We present the results we obtained through simulations of our proposed middleware. In order to evaluate our approach, we developed a Data Grid simulator GridNet. This simulator provides a modular simulation framework through which we can model different Data Grid configurations and resource specifications. We present the simulator and the algorithms and mechanisms we used to model and simulate real Data Grid environments.

Chapter 6: Performance Study. We present the different scenarios we used to conduct our simulations, the testbed we used to conduct the experiments, and the models we used to represent realistic Data Grid environments. We also study the different replication models and give a comparative overview of the results.

Chapter7: Conclusion. We conclude with a discussion of the benefits and advantages of our proposed solution and middleware, present the contributions of our work, and discuss future work.

CHAPTER 2
Data Management and Replication in Data-Centric Networks

Replication is an important tool that has been widely used and supported by different data centric systems and networks to improve data accessibility and overall system performance. In this chapter we survey standard and currently used Data Grid architectures and discuss their infrastructure and the services they offer. We also compare and contrast the use of replication in Data Grids vs. distributed data-centric environments and networks. Such environments include distributed systems, databases, and web applications. We then discuss existing replication management solutions for Data Grids and overview some of the problems and issues that we will address in this thesis.

2.1 Data Grid Architecture

Many initiatives to build computational and data grids were undertaken by different groups of researchers from different institutions and universities in the late nineties. A leading effort in that area was undertaken by the Globus team [38, 39, 40, 36]. The Globus project provides a framework for building Grids based on a service-oriented architecture. The services the framework offers are: Security, Information Services, Resource Management, and Data Management [38, 39, 40]. The toolkit relies on a Grid Security Infrastructure (GSI), which provides a set of security features to allow users to authenticate their communication and use single sign-ons to access grid resources and services. The Information Services provide information about the status of Grid resources using a notification approach where resources publish their status and subscribers receive updates about specific instruments, machines, or storage components they want to access. This allows the monitoring as well as the discovery of resources. The Resource Management component uses input from the Information Services to enable users to access available resources and to allow the system to schedule resource allocations. The Data Management or the

Data Grid provides the ability to access and manage data and data resources on the grid [14, 28]. The Globus toolkit provides several components to move, copy, and locate data. The major components are: GridFTP, RFT, and RLS [14, 40, 36]. GridFTP provides tools for fast, secure, and parallel data transfers in the Grid. It is a data transfer protocol that extends the FTP protocol and provides secure, parallel, and reliable mechanisms to move data on the Grid. RFT, the Replica File Transfer service provides reliable management of multiple GridFTP transfers. RLS, the Replica Location Service, maintains and provides access to information about the location of data available within the Data Grid [14, 28, 27]. The RLS uses Replica Catalogs to register, index, and locate data. The Replica Catalog is a registry in which users maintain records for all the shared files and objects in the Grid. Each record contains the locations of all the object replicas, and provides a mapping between object names and its replicas. The aforementioned components of the Data Grid can be organized in a layered architecture as shown in Figure 2.1 reproduced from descriptions in [15, 28]. This architecture is derived from definitions and proposals introduced in [15, 14, 28, 40].

The layers outlined in Figure 2.1 represent the different components that make up the Data Grid infrastructure. Components at the same level can co-operate to offer certain services, and components at higher levels use services and components offered at lower levels and build on top of them. The Grid Fabric consists of computing resources such as workstations and supercomputers, storage resources such as disks and tapes, as well as scientific instruments. Most of these resources are widely distributed and are in turn connected by high bandwidth and wide area networks. These computing, storage and networking resources represent the physical layer of the Data Grid. The operating systems and software that manage these heterogeneous resources represent the basic software layer of the Data Grid. The Connectivity layer consists of data transfer protocols to copy data from resources in the Grid Fabric layer. The data transfer protocols are are based on TCP/IP communication protocol and authentication protocols to verify users identity and ensure security and data integrity. The Data Grid Services layer consists of core services such as replication and resource monitoring that provide transparent dis-

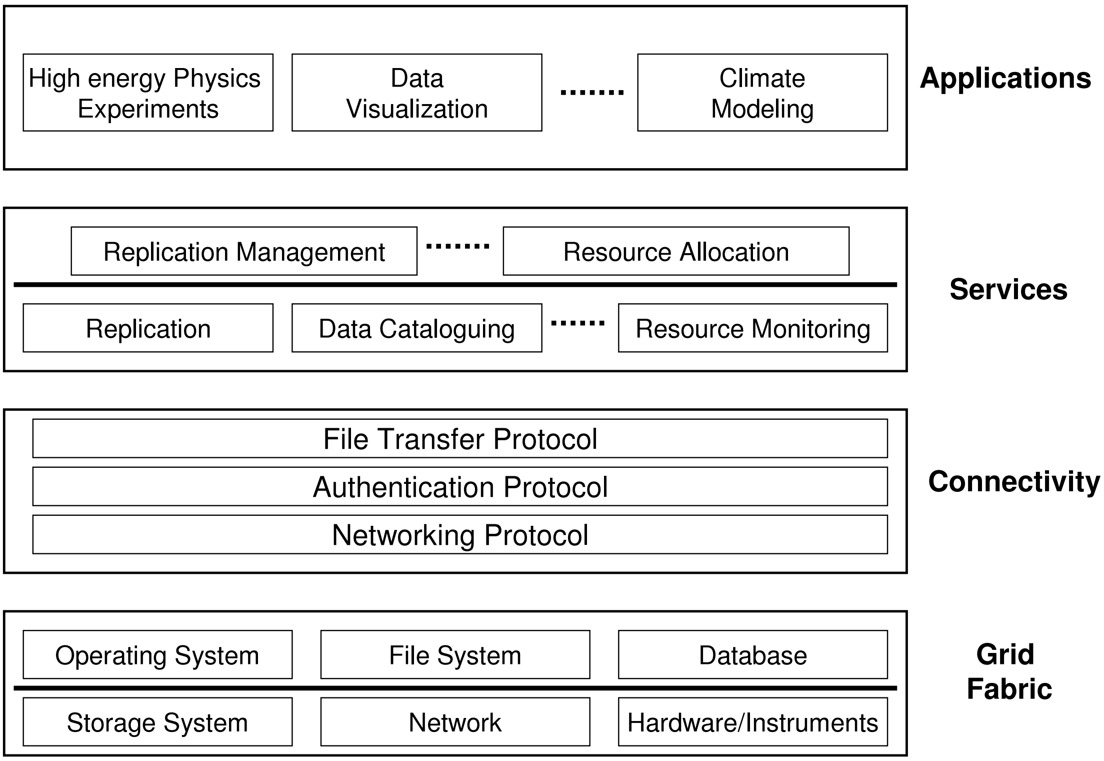

Figure 2.1: Overview of Grid Architecture.

covery, location, and access to data and compute resources. These services can be used to contribute resources to the Grid and define access policies of these resources. The monitoring service is equivalent to the Information Services in the Globus architecture and implementation [40]. The next components of the Services layer provide higher level services that use the lower level services to enable efficient management and allocation of replicas and data resources on the Data Grid. The collective set of services provided at this layer represents the Data Grid middleware. The middleware abstracts hides the complexity of managing access to resources and provides API's for users and applications to transparently take full advantage of the utilities available in the Data Grid. The Applications layer provides services and access interfaces that are specific to a community or Virtual Organization.

2.2 Replication in Data-Centric Environments

Replication has been studied extensively and different distributed replica management strategies have been proposed in the literature [30, 42, 43, 60, 61, 62, 70, 76, 81, 83]. Replication has traditionally been used to improve system performance by increasing its reliability and fault tolerance. In the context of Grids, data replication is used to reduce access latency and bandwidth consumption. In this thesis, replication will also be used to improve data availability, data autonomy, host and network traffic load balancing, and data access performance.

2.2.1 Replication in Databases

As a means to improve scalability, improve data availability and provide fault tolerance, replication has also been an area of interest to databases. In [93], the authors survey the concepts and solutions developed for managing data in distributed environments versus database systems. While there are many similarities in the solutions and services provided in both communities for enabling remote and scalable access, there are many differences in the requirements and properties of the problem spaces [93]. A fundamental difference is the nature of the protocols used in both communities [93, 92, 91]. While databases use blocking protocols, distributed systems usually use non-blocking protocols. Databases protocol specifications put more emphasis on ensuring safety, rather than keeping the application live or running [91]. Examples of these specifications are seen through the implementation of transactions. Transactions are defined as a set of instructions that are treated as a single execution unit. The transaction's results are accepted and committed only after ensuring that each single instruction in the transaction has processed and completed successfully. To ensure the correctness of transaction processing a set of rules collectively called ACID are enforced. ACID stands for Atomicity, Consistency, Isolation, and Durability [91]. Those rules respectively ensure that all the transaction instructions have completed successfully (otherwise the whole transaction is aborted), enforce the correctness of the transaction execution, ensure the consistency of parallel transaction executions, and finally ensure that transaction results are saved and would be retrieved even in the event of failures. The main replication techniques used in databases are: Update Everywhere and Primary Site. To guarantee consistency and enforce the ACID rules, these techniques use different voting algorithms depending on the application requirements and design. Using distributed voting schemes, however, degrades the performance of replication. To remedy to that, optimistic and weak consistency protocols have been introduced. These protocols rely heavily on the use of broadcasting techniques that ensure the correctness of the order of execution of distributed transactions [45, 51, 73, 91].

2.2.2 Replication in Content Distribution Networks

Content Distribution Networks (CDN's) are targeted for speeding up the delivery of normal Web content and reduce the load on the origin servers as well as the network. CDN's, such as Akamai [1] or Digital Island [3], distribute content by placing it on content servers, which are located near the users. A content provider can sign up for the service and have its content placed on the content servers. The content is replicated either on-demand when users request it, or it can be replicated beforehand, by pushing the content on the content servers [52]. Another form of content distribution is peer-to-peer content distribution. This form is mainly used to share individual files between users. In peer-to-peer networks, such as Napster [8], or Gnutella [6], individual users decide to share files with others. With the help of a directory service, users can determine where different files can be downloaded from. CDN's effectively reduce the client-perceived latency and balance the load on the servers and the network by distributing and providing content from sites closer to clients. One of the foremost problems in CDN's is to decide where to place site contents in the CDN infrastructure. This problem has been proven to be NP-Complete, and several caching and replica placement algorithms have been used and proposed in the literature based on heuristic and relaxation algorithms [52, 53, 54]. However, most existing algorithms do not scale well for larger systems. For optimal performance, as in Data Grids, decentralized approaches need to be considered as well as optimization and heuristic approaches to maximize the use of network resources and meet the users' quality of service requirements.

2.2.3 Replication in Web Applications

Data replication is also a basis for Web caching, which is used to store copies of frequently requested documents at nearby servers. It reduces network and server loads and decreases latency of Web responses. However, Web caching may return stale pages to clients. Different cooperative Web caching approaches have been studied to improve hit rates and response times [20, 78]. In hierarchical caching, caches are located at different network levels. Such a caching scheme is used by Netscape and MS Explorer and implemented by Squid [9]. In distributed caching, caches cooperate to serve client requests. A popular mechanism used to share documents is based on broadcast probes using the Inter Cache Protocol [90]. In [62] a set of optimistic replication techniques for Web documents have been proposed, where weak consistency algorithms are used. However, data accesses in the Web and in Grids have different patterns. In the Web, updates are very frequent and requested data (documents) are small compared to Data Grids and their access patterns. Hence, these two environments require different validation mechanisms. However, a decentralized optimization model for a general approach to replica management might be useful in improving the performance of replication techniques on the Web.

2.2.4 Replication in Distributed Systems

Different replication systems have come out of the research community, such as the replication file systems Coda [77], Ficus [42], Rumor [43], and Roam [70]; the Bayou storage system [82]; and the Locus operating system [63] among many others. An important aspect of replication-based systems is the protocol used to maintain consistency among object replicas. The main issue in such systems is maintaining scalability with large numbers of replicas distributed over wide area networks. This issue has been addressed in previous distributed file systems both for standard networks and mobile computers by the use of optimistic consistency protocols [13, 42, 43, 70, 77]. However the consistency issues have not been addressed on the scale of the Grid environment, across multiple organizational domains. Additionally, the requirements for the above systems are different from those of the applications targeted in this thesis. The sizes of the data stored in the Grid are much bigger than those supported by existing distributed file systems, and the replica granularity is much higher. Moreover, in the Grid environment we assume that updates to the data are infrequent and that the consistency can be more relaxed than in high-performance commercial databases such as Oracle. Given these requirements, we have designed a system that can achieve greater scalability and reach over the wide area networks.

2.2.5 Replication in Data Grids

The existing components and services in Data Grid implementations do not provide any automated replica placement support. The data and replica management services do not implement full replica management functionality and do not enforce any replication semantics. The existing systems only provide the users with tools to replicate data at different locations under user-specific definitions without enforcing the user's assertions [15].

With the massively increasing challenge of managing larger and larger amounts of data, there has been an equivalent rise in interest in modeling Data Grid requirements and simulating different data replication techniques [79]. Different studies were conducted to model scientific experiments settings and configurations, such as the CMS and ATLAS experiments [7, 79]. Many projects, such as the GriPhyN [7] and the EU Data Grid [4], are developing Data Grids. Other projects have focused more on simulations. In [65, 64, 66] the authors have studied the performance of job scheduling algorithms and their combination with data placement algorithms. The simulations show that optimal results were obtained by using loosely coupled replication decisions and scheduling policies. The replication decisions were predicted based on local data access patterns. In [67] the authors present a P2P replication approach where data is replicated based on global data access patterns to guarantee global data availability. Another simulator is presented in [22] OptorSim, which provides a framework for studying the combination of data access optimization algorithms and job execution. The authors use an economic model to evaluate the data access costs using a P2P auction approach. Data replication decisions are predicted based on past access patterns.

To satisfy the growing need to provide more storage allocation and management services within the Grid community, many storage systems were developed. Examples of such systems are the Distributed Parallel Storage System DPSS [83] and the High Performance Storage System (HPSS) [89]. These systems provide high performance data transport tools by supporting parallel data transfers. This is achieved by aggregating the use of multiple TCP streams and by partitioning data across multiple servers and supporting striped data transfers. In the Microsoft Distributed File System (DFS) collections of files located in different servers and are linked through a DFS root share [9]. The DFS roots are in turn replicated to provide fault tolerance, and files are also replicated in different servers to provide faster access. Another example of distributed storage systems is the Storage Resource Broker (SRB) [21]. SRB is a client server system that provides access to different types of data storage across distributed heterogeneous platforms and maintains metadata about each stored object. Access to data is provided through a Metadata Catalog (MCAT) that uses metadata attributes rather than physical names to query the servers and locate files. The Google file System (GFS) [41] is a distributed file system for large distributed data-intensive applications using commodity hardware. GFS uses replication to provide fault tolerance and support large-scale data processing workloads. The GFS architecture is based on a single master node within a cluster which still requires and relies on global cluster knowledge to make data placement decisions.

In this thesis, we develop a distributed replication system that is highly decentralized and capable of performing automatic and optimal data replication based on the user and application demands. We demonstrate that this replica management system performs better than static replication approaches thanks to its highly dynamic replica placement and creation strategies and scalability.

In the next section, we overview the existing replication protocols and models and discuss their advantages and disadvantages.

2.3 Replication Models

An important challenge in replication systems is how to handle updates to multiple copies simultaneously and how to maintain the same view of all replicas all the time. Conservative update replication systems prevent all concurrent updates and require locks and votes from other replicas before committing any changes to a data item. Thus, they consume network bandwidth to check consistency at every update. On the other hand, optimistic replication systems allow any machine storing a replica to perform an update locally. Consistency checks are only performed if inconsistencies are discovered or if requested by the user, thus reducing the amount of data exchanged and the bandwidth used to update replicas. But given the type of data stored on data grids, conflicts are rare. Most of the time, updates on the data stored on the Data Grid affect the application and replica metadata. Application metadata is a high level description of a file that also contains attributes of the data stored such as keywords describing the nature of the data and the description of the origin of the data and its production. Replica metadata provides a mapping between a file name and one or more physical locations of the file replicas. Thus, it is necessary to guarantee that the updates will be eventually propagated and that the users will have access to consistent copies of the data. The replication system must be able to maintain a desired level of consistency in the Data Grid environment. But given the number of replicas anticipated on the Data Grid and their geographical distribution, scalability is an important issue. We implement scalable replica consistency algorithms to support user-defined semantics, and mainly use a relaxed consistency approach. The consistency algorithms will use scalable replica distribution and propagation graphs for efficient replica synchronization. There are three replication models that have been utilized in practically all replication systems: master-slave, client-server, and peer-to-peer models. In the following subsections we survey these most commonly used models and describe them in terms of their replica communication and synchronization characteristics.

2.3.1 Master-slave

The master-slave model labels one replica *master*; all other replicas are slaves. The replication paradigm is that slaves should always be identical to the master. The model is very simple, yielding a simple implementation, and has been used in many replication packages such as Laplink [59] and RDIST [29]. However, with the simple model comes limited functionality: the slave is essentially read-only. Most master-slave services ignore all updates or modifications performed at the slave and "undo" the update during synchronization making the slave identical to the master. While some provide crude abilities for some notion of update generation at the slave, all limit the set of operations that can be performed. For instance, object removal cannot be performed at the slave in Laplink, as the object will be reinstated by the replication service as part of making the slave identical to the master. In general, modifications can only be reliably performed at the master, and slaves must synchronize directly with the master.

2.3.2 Client-server

The client-server model is similar to the master-slave in that it designates one server, which serves multiple clients. However, the functionality of the clients is greatly improved, and multiple inter-communicating servers are permitted. All types of data modifications and updates can be generated at the client. The client-server model has been successfully implemented in replication systems such as Coda [77] and Little Work [47]. Clients however cannot intercommunicate or synchronize with each other, limiting their functionality. They must rather communicate with a server. Scaling is not typically a problem in the model; in proposed solutions multiple servers can be recursively combined in a client server arrangement. However, doing so increases the system's overall dependence on the servers at the upper levels of the hierarchy. The failure of a server can isolate all clients served by it, which impacts the reliability of the overall system.

2.3.3 Peer-to-Peer

The peer-to-peer model takes a very different approach from both the master-slave and the client-server models. The peer model is not class based: all replicas are equals, or peers. Any replica can synchronize with any other replica, and any file system modification or update can be applied at any accessible replica. The peer model has been implemented in systems such as Locus [63] Bayou [82], Ficus [42, 71], Rumor [43], and Roam [70] and more generally been used in other distributed environments such as xFS [18]. The peer model provides a robust communication framework. However, it has typically suffered from scaling problems. Peer models have traditionally been implemented by storing all necessary replication knowledge at every site; each replica thereby has full knowledge about everyone else, and synchronization and communication is permissible between any two hosts. Such an approach results in exceedingly large replicated data structures, leading to potential scaling problems. A recent study in [13] shows that updating replicas with greater access requests first, in a peer-to-peer model, improves the performance of weak and relaxed consistency without incurring the additional costs of strict and strong consistency.

In the client server approach, maintaining replica consistency is simpler that in the peer-to-peer approach. In the former model, there is one central location to which all updates must be posted. This solution substantially simplifies replication consistency maintenance. However, the full benefit of this solution is only achieved when there is a single replica server. Such a solution has poor reliability, because a failure of the server makes it impossible for any other replicas to receive new updates or disseminate their own updates to others.

2.4 Synchronization

Another important aspect of maintaining replica consistency is the replica synchronization process and the frequency of consistency checks. In both weak and strong consistency models, updates to any replica should be made known to the replicas. These changes are either immediately propagated to all replicas or later reconciled with updates at other locations. Update propagation-based replication attempts to propagate updates made at one replica to the other replicas immediately, either directly or through some propagation graph spanning the overall set of replicas that minimizes communication costs. An alternative is an update reconciliation-based replication, in which no attempt is made to propagate updates automatically. Instead, all changes made to the replicated data are batched together and periodically sent to other sites storing replicas. These batched changes can be sent during periods of high bandwidth availability.

In Grid environments, latency might be very high given the sizes of the data items stored on a Data Grid. In some cases, propagating the updates to all replicas is unnecessary. The main challenge though is scalability. Applications running on the Grid require access to Terabytes or even Petabytes of data in wide-area, distributed computing environments. Propagating data updates of this magnitude to a large number of replicas every time a data item replica is modified greatly affects the performance of the entire system. Update reconciliation is better suited for this environment. Used with relaxed consistency requirements, this approach allows replicas to initiate synchronization and reconciliation sessions with other replicas and only propagate updates when changes do occur. The replica communication patterns and the physical location of the replicas affect the performance of the reconciliation process. In our approach, we rely on user-defined replication semantics without enforcing any particular replication or synchronization approach. The replica consistency algorithm when initiated will use a high level overlay network to use the least number of exchanged messages among replicas and the least amount of time to reach consistency. This approach is decentralized and does not require the system to keep any global information or view about replica locations. This approach has been shown to scale very well in distributed environments [13, 42, 43, 44, 70, 76, 89].

CHAPTER 3
Distributed Data And Replica Management Framework

In this Chapter we present the motivations and goals of our proposed work and argue its benefits. We describe our solution and approaches to addressing the replica management issues. To that end, we present the design of the middleware, provide a general framework and description of the main components, and give an overview of the high level algorithms and services provided.

3.1 Data And Replica Management Concepts And Requirements

Data Grids provide a platform for the scientific community to share, access, transfer, and process large collections of data distributed worldwide. This platform enables the aggregation and provisioning of high performance computing resources, high performance networking and large scale storage resources and management technologies. The rise in popularity of Data Grids has led to the introduction of several Data Grid solutions. These solutions aim to enable users to take full advantage of the Data Grid infrastructure. As mentioned in the previous Chapter, we have surveyed different mechanisms that provide similar support for distributed data intensive platforms. These mechanisms include: Content Delivery Networks, P2P systems, and Distributed Database systems. These data-centric networks have developed various solutions that have matured over the years and which could be applied to address outstanding issues in Data Grids. These solutions could be either wholly adopted by the Grid community or modified and adapted to fit the specific needs and requirements of Data Grids. Our approach adopts some concepts from these existing systems to take advantage of their proved successes and better improve the quality of services and tools available in the Data Grid. However the needs of the Data Grid community make it unique and quite different from existing data-intensive networks and systems. One of the major properties of the Data Grid which makes it unique is the aggregation of a diverse set of resources. This set of resources includes not only storage resources but also specialized instruments and hardware that produce and generate the needed data, as well as high performance computational resources and software that are equally needed to manage, process, analyze, and store large amounts of data. On the one hand, the sharing of these resources through the Data Grid platform enables more users to access them and take advantage of their utilities. On the other hand, the intensive demand for processing, transfer, and storage might give rise to situations where multiple users are competing for access to few available resources. Resource allocation and scheduling is thus a very necessary mechanism to ensure efficient and fair access to the users. In [40] Foster et al. proposed a Grid architecture that is based on the concept of Virtual Organizations (VOs). The VOs are set up and formed by a group of entities or organizations willing to share their resources in order to work and collaborate on problems and applications with common interest to all participating parties. The VO defines the resources donated for sharing by each participating organization, as well as the rules dictating the use of these resources. In order to overcome variations and incompatibilities in infrastructures used at each organization, common middleware running across these different infrastructures is needed to ensure transparent and reliable access to the shared resources, and to manage access and usage of these resources according to the defined access rules. To ensure security, a VO should use some sort of authentication mechanism to facilitate access to the different users from different administrative domains. One major drawback to that proposed VO management approach is the central management and decision making of such an approach, which to a high degree hinders the growth and scalability of Data Grids. With an emerging and growing interest in collaborative problem-solving across the globe larger VOs are needed to reach larger numbers of collaborators and organizations. Additionally, important components of commonly used and popular data management middleware used by the Data Grid community, such as the Replica Location Service (RLS), do not scale well across large numbers of sites [27]. The RLS is used to register the physical names and locations of files and data sets stored on the Data Grid. Using the same logical name space, each file is assigned a unique logical file name. An entry for that logical name is created in the Replica Catalog, and the logical name is then mapped to all the physical locations of that data file. This logical name is thus used to index, locate, and access that data file. The RLS uses a hierarchically distributed system to catalog the available data within the Data Grid and place Replica Location Indices at predetermined Grid nodes.

Most existing and deployed grid systems and platforms are centrally managed [14, 23, 27] and are quite difficult to set up and maintain. Proper access to software and hardware resources requires meticulous installation, configuration, and testing of different components across all the participating Grid nodes. Enabling dynamic node addition and deletion would provide efficient access to resources on the Data Grid but presents considerable challenges to system designers. Many studies have revealed that commonly used Grid systems and toolkits do not provide yet sufficient support to a large community of users as originally intended by the designers and developers of these systems [32]. The Open Grid Service Architecture (OGSA) [31] defines a Grid infrastructure that combines and integrates existing Grid technologies [40, 38, 36] and Web services technologies to create a distributed Grid computing framework based on the Open Grid Service Infrastructure (OGSI) [31]. In this infrastructure "a Grid service instance is a (potentially transient) service that conforms to a set of conventions, expressed as Web Service Definition Language (WSDL) interfaces, extensions, and behaviors" [31]. A Grid service is thus a Web service that maintains its state to match the requirements of large and complex distributed applications. The Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL) are a protocol and an interface language that enable distributed applications on the Web to communicate and exchange messages [32]. SOAP and WSDL are increasingly adopted as a means to support communication in distributed environments [32, 31, 36]. Soap is an XML based protocol and most of its existing implementations use HTTP, but it is not strong enough to support high performance distributed applications. The inefficiency of SOAP is mostly caused by the time spent in parsing and formatting the XML messages, as well as the complexity of its system calls to deliver messages [32]. This constitutes a major drawback to achieving expected performance levels by Grid implementations that adopted this Grid service oriented architecture and computing paradigm, such as Globus [36].

In this thesis we attempt to address some of these issues. To that end, we introduce new highly distributed and decentralized data management middleware for Data Grids. We develop an adaptive and scalable lightweight data management framework that enables users to dynamically join and leave the grid. Our solution provides replication management services that intelligently and transparently place data in strategic locations in order to improve the overall data access performance. In our proposed system, we advocate a local and autonomous approach to replica management at each participating node in the Grid. 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. Providing such lightweight middleware to support research and scientific collaborations enables the creation of larger communities with less overhead. This leads to supporting the creation of small to medium scale Data Grids with large numbers of users taking advantage of larger numbers of under-utilized resources.

Our approach is inspired by the P2P techniques that require no centralized management and advocate self organizing. The P2P approach has many attractive features that make it very suitable for grid computing. The simplicity of the approach makes it easy to deploy on large numbers of nodes. Actions taken by nodes in a P2P system are based on local information, which is an important feature that enables P2P networks to scale well. In order for that management to be efficient, however, it has to be based on information and intelligence collected at each location about the availability of resources and how much of its resources it can contribute to the overall system. We achieve this by evaluating replica creation and placement by evaluating data access costs as well as data popularity and its spatial and temporal locality based on user demands.

3.2 Data Organization And Replication Models

Data on the Grid needs to be easily accessible to users regardless of location. Data access models are formed by community-oriented social organizations and are affected by the direction of data flow and users' access patterns. The location and number of data sources as well as data size play an important role in shaping the map of the community sharing access to that data. A thorough study and survey of data flow and organization in Data Grids presented in [87] showed that there are four common models that are currently being used in Data Grids:*Hierarchical*, *Federation*, *Bottom-Up* and *Hybrid*.

Hierarchical: This model is used in an environment where there is a single source of data, and that data has then to be distributed and copied to multiple locations to be shared by a large community of collaborators. This model is shown in Figure 3.1, reproduced from [87]. It shows the data distribution model in the CERN experiments where data is first generated and stored in CERN, and later copied to different distribution and regional centers. From these centers the data is then distributed to different labs worldwide to give access to scientists from around the world.

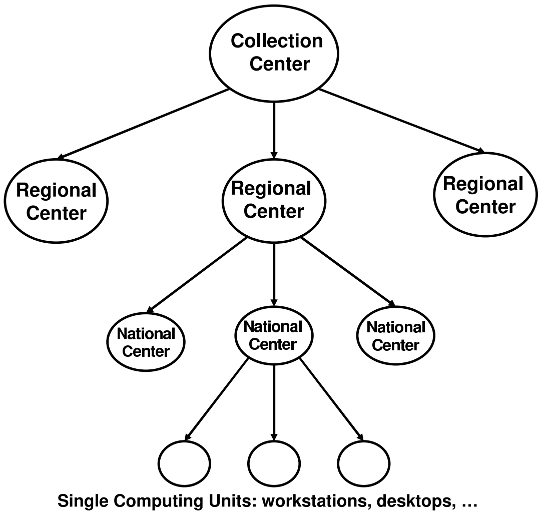

Figure 3.1: Hierarchical Data Model

Federation: This model is formed by organizations and institutions who share their resources to collaborate and co-operate on research projects forming a specialized P2P network. Each institution allows users from the participating institutions to access its locally stored data. This model is shown in Figure 3.2.

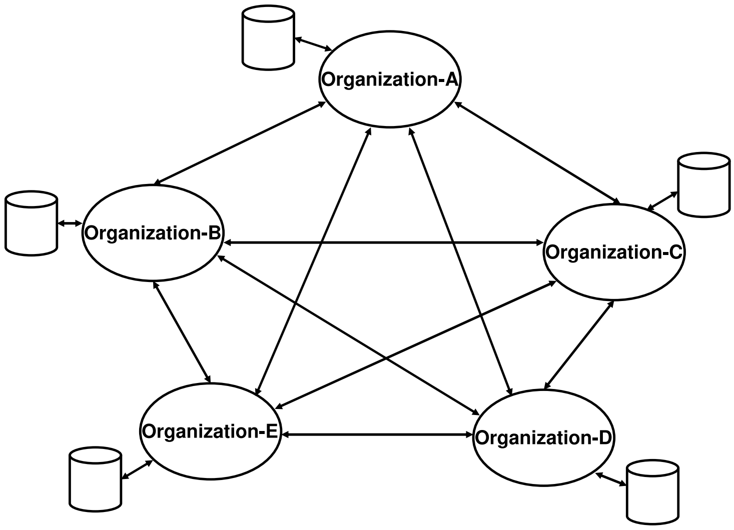

Figure 3.2: Federation Data Model

Bottom-up: This model is used in scenarios where multiple data collection points, such as sensors, are distributed in different locations. The data is collected in a bottom-up fashion and stored at a central location or database where it can be accessed by more users. This model is shown in Figure 3.3.

Hybrid: This model is used when both the Hierarchical and Federation models are combined. It is shown in Figure 3.4.

The scope of this thesis covers scientific Data Grids, where a number of researchers from different institutes share their resources to collaborate on solving scientific problems. The most prevalent data model used in these settings is the hierarchical model. To some extent a P2P model has also been adopted as a model to form collaborations between smaller entities. Our middleware supports the hierarchical, P2P, and hybrid models. Each model requires a different replication strategy. In the next subsection we will highlight these strategies and the basis of our approach.

3.2.1 Data Access And Location

Replication is a necessary requirement for Data Grids to ensure all users have access to the required data, to maintain scalability across multiple locations and

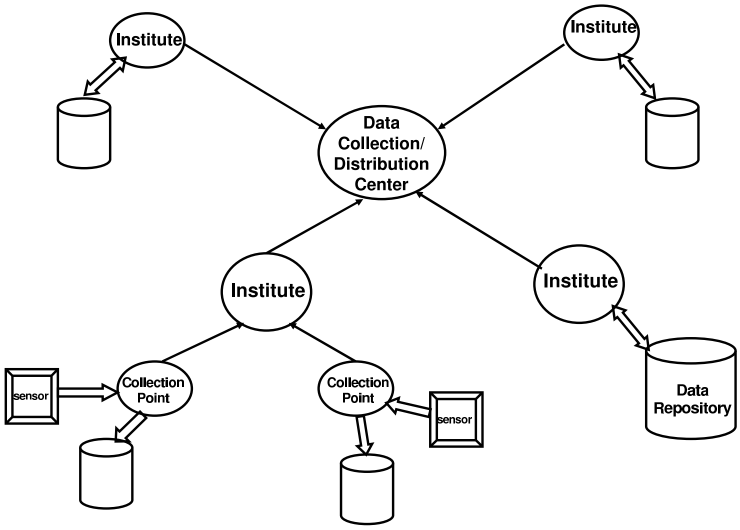

Figure 3.3: Bottom-Up Data Model

domains, and to optimize the use and consumption of network resources. The replication mechanism creates copies of popular data sets and manages the number and locations of these copies at multiple locations providing better access to users. The replication mechanism is guided by cost/benefit estimation strategies that take into account current demand for the data sets, locality of requests, network resources availability, and storage capacity to create copies of the data. The two major strategies for managing replicas are *Centralized* and *Decentralized*. In a centralized model all replicas are copied from a central location where the data is originally collected or generated, and then propagated following the data flow and demands using a cascading model. This model inherently creates a hierarchical topology that maps Grid nodes where data and replicas are located onto a graph creating a tree structure (see Figure 3.1 and Figure 3.3). In a decentralized or P2P model data can be originally generated at different locations and then shared between a set of equal and peer nodes participating in the Data Grid (see Figure 3.2). The organization and distribution of data locations in this model creates a mesh topology. However, to ensure efficient access to data a less complex structure is needed to navigate through the different data locations and locate requested data. Such a structure is the ring topology. This topology can be created when nodes establish connections

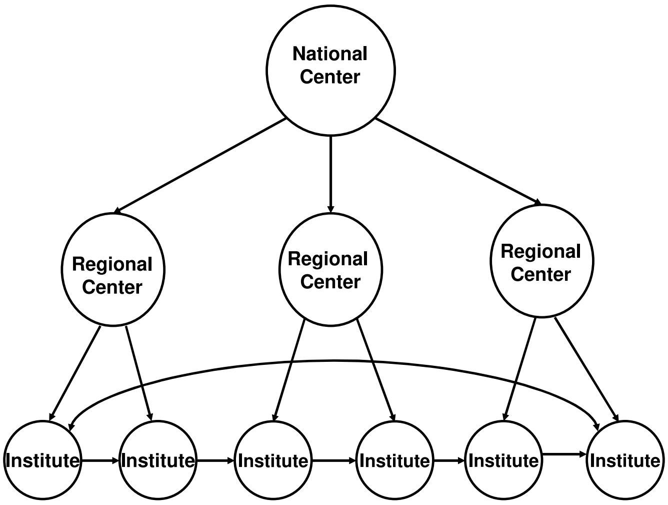

Figure 3.4: Hybrid Data Model

with at most two of their neighbors and peers. Using this structure would avoid the complexity of data location algorithms that are used in some P2P networks. Additionally, extensive surveys of Data Grid models that we have covered in this and previous Chapters suggest that such a model is not commonly used. Most scientific applications and collaborative research enterprises running on or requiring the use of Data Grids fit a hybrid model (see Figure 3.4). A hybrid model is a combination of a *hierarchy* and a *federation*.

Topology means and represents the connectivity graph formed by the *overlay network*. The overlay network consists of the application-level (not physical level) connections between the nodes, as each node in this network has connections to a number of other nodes also called neighbors. Using structured overlay networks forms the basis of our approach to building scalable replica management middleware. These graphs or networks define the communication paths that can be used to navigate through the network and locate data. To transfer data, however, the physical network takes care of properly routing data. The middleware's responsibility is limited to locating data sources and the end points of the transfer. In the next sections we will discuss in further detail the key architectural and design concepts of our proposed middleware.

3.2.2 Replica Management And Consistency Support

In many existing replication systems and contexts, data replication is associated with semantics that define the replica consistency as maintaining replicas as exact copies of the original files and data sets. This approach requires the use of distributed locking mechanisms to ensure replica consistency at all times similar to the mechanisms used in databases [91]. Contrary to that approach, our replica management middleware does not enforce any replica semantics and does not guarantee absolute replica consistency. The system does not keep track of original or source locations. When a logical data collection is registered into the system, and subsequent replicas of that entity are created, the system only maintains information about the locations of all existing copies of that logical data collection. The system does, however, provide support to user-defined and user-initiated replica semantics. We support replica synchronization through a separate mechanism that is orthogonal and independent of replica creation and replica management services. Data Grids mostly deal with read-only data, and enforcing consistency checks would only bring down the performance of the replica management services.

3.3 Replica Management Framework

The major goal of our work is to promote a more decentralized and distributed approach to managing Data Grids. Our attempt at creating a more dynamic and versatile Data Grid Architecture is supported by the distribution and the delegation of management and decision making to all participating Grid nodes. In our approach each participating node is responsible for managing access to its local and contributed resources, thus limiting centralized management and control over widely distributed resources, contrary to currently and commonly used approaches in Data Grids. To support this distributed and decentralized approach, we introduce and deploy distributed middleware that can operate independently and make decisions based on local knowledge and collected statistics at each Grid node.

3.3.1 Framework Overview

A participating Grid node can be any hardware/software that qualifies as a computing unit such as a supercomputer, workstation or desktop computer. The middleware consists of autonomous and distributed components that run at each participating Grid node. Each agent is composed of:

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

Replica Creation Service This service is responsible for creating local replicas based on accumulated statistics about data popularity and an evaluation of the incurred cost of creating a local replica. A cost function is used to evaluate the cost of creating a local replica vsṫhe cost of transferring data. This cost/benefit analysis and calculation is based on the popularity of the data, network resource availability, size of data, and storage space availability. These statistics and information are provided by the Resource Monitoring Service.

Replica Location Service This service is responsible for managing the local replica Catalog. Each node maintains a catalog to access locally stored data and possibly track remotely located data. Each newly created file is registered in the catalog. In case the data is not available locally and the node has information about different locations where that data is available, that information is also stored in the catalog. That information is kept from previous data requests initiated locally; i.e., the results of previous searches are thus kept and used to speed up the data look-up process. This service is also responsible for locating requested data that is not available locally. We will refer to the Replica Location and Replica Creation services collectively as the Replica Management Service.

Resource Allocation Service This service is responsible for allocating space for newly created replicas and de-allocating space from the least frequently and last accessed locally stored replicas.

Routing/Connectivity Service Responsible for routing outgoing request messages and handling incoming messages, managing data transfers, as well as monitoring a node's connectivity to its neighbors, this service maintains a list of neighbors to which the local node is connected.

The system architecture is shown in Figure 3.5. 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.

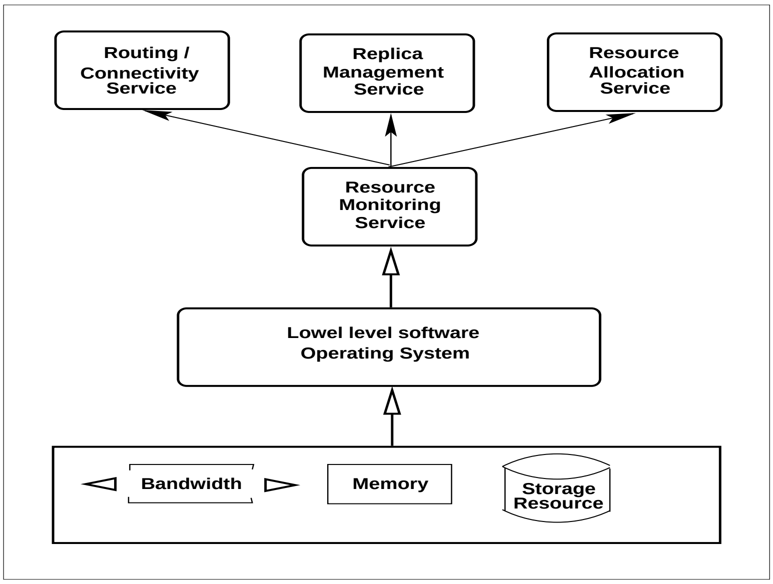
Routing / Connectivity Service
Replica Management Service
Resource Allocation Service
Resource Monitoring Service
Lowel level software Operating System
Bandwidth
Memory
Storage Resource

Figure 3.5: Architecture and Design of the Data Management Middleware

The distributed replication service determines the placement and location of replicas in the system independently at each node. The cost function evaluates at runtime the maintenance cost and access performance gains of creating a local replica. The monitoring service feeds the collected statistics about resource availability and data/replica locality to the decision-making component at each node. This latter component takes into consideration data access frequency, data locations, latency, and bandwidth availability before deciding whether or not to replicate the requested data or not.

3.3.2 Replica Synchronization

In many existing replication systems and contexts, data replication is associated with semantics that define the replica consistency where a replica has to be maintained as an exact copy of the original file or data set. This approach requires the use of distributed locking mechanisms to ensure replica consistency at all times similar to databases [91]. Contrary to that approach and as stated in Section 2.3, our replica management middleware does not enforce any replica semantics and does not guarantee absolute replica consistency. The system does not keep track of original or source locations. When a logical data collection is registered into the system, and subsequent replicas of that entity are created, the system only maintains information about the locations of all existing copies of that logical data collection. The system, however, does provide support to user-defined and user-initiated replica semantics. We support replica synchronization through a separate service that is independent of replica creation and replica management services. Data Grids mostly deal with read-only data, and enforcing consistency checks would only bring down the performance of the replica management services.

3.3.3 Replica Distribution Topology

Our approach is based on using application level overlay networks to enable scalable growth of Data Grids and tolerate larger numbers of participants. The overlay network is formed by the set of connections between the participating nodes in the Data Grid. Topology means and represents the connectivity graph formed by the *overlay network*. The overlay network consists of the application-level (not physical level) connections between the nodes, as each node in this network has connections to a number of other nodes also called neighbors. Using structured overlay networks forms the basis of our approach to building scalable replica management middleware. These graphs or networks define the communication paths that can be used to navigate through the network and locate data. To transfer data, however, the physical network takes care of properly routing data.

3.3.4 Replica Creation And Replacement Algorithm

In addition to the cost analysis, determining whether to create a replica or not needs to include more factors and system parameters. Such parameters that are 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 last 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, previously collected read and write count values are averaged out and the actual read and write count values are annulled. The read and write count averages represent the rate of read operations and write operations from the sum of accumulated read and write counts. These values are used to predict future tendencies in access patterns. Different prediction tools could be used 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. This index is relative to the total available storage space.

The storage cost is a linear combination of these parameters. Each factor in the combination is assigned a weight depending on the application's 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 locally stored 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 removed to make space for the new replica. When a replica is created at a given node, that information is stored in the local data catalog. Similar approaches have been used by operating systems to manage memory caches. The most popular technique is based on access frequency: Frequency Based Replacement using a Least Recently Used (LRU) list to replace least used data [72].

3.3.5 Replica Location And Access

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 [27]. 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 [6], the centralized algorithm used in Napster [58], and the distributed hash-table based protocol used in CAN, Pastry and Tapestry [68, 74, 80]. Many studies have shown that using a combination of flat and hierarchical topologies gives the best performance for message broadcasting [58, 55].

In our work we use a combination of spanning trees and rings to build the overlay network to route access requests and locate data in a dynamic Grid platform. These structures can be easily adapted to different topologies and changed dynamically by adding new nodes and dropping unused connections. Ad hoc span-

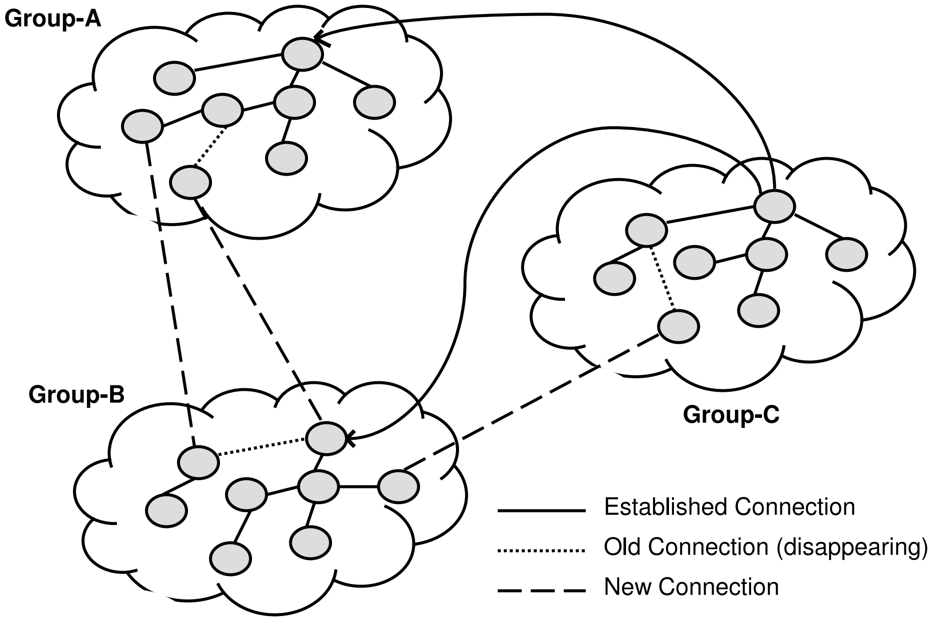

Figure 3.6: Replica/Data Search Process

ning trees have been tried and proven to perform and scale well in P2P systems [55]. To build the distributed overlay network two algorithms are needed. The first is an algorithm for maintaining the tree and accounting for inclusion of new nodes joining the tree and deletion of nodes leaving. The second is a search algorithm for locating data. The use of tree and ring topologies yields a mostly hierarchical structure that can be represented as a tree. Figure 3.6 shows three groups of clustered nodes. The clustering is defined based on similar interests. Users interested in the same data and aspects of research end up connected. The Figure also shows the creation of new links and the removal of existing links, thus creating new groupings. This adaptivity is based on the node's interest and request response history. Each node maintains a list of *preferred neighbors*. This list consists of node addresses from which the local node has previously received responses to data requests. When a given site in the list becomes the source of an increasing number of access responses, a new connection to that site is created. In a parent-child connection the old connection is deleted while the new connection is created as shown in Figure 3.6. In a peer connection, the new connection is added replacing a less used existing peer connection. In the remainder of this subsection we describe the algorithms used to add/remove new nodes to/from the overlay structure and locate data on the Grid.

Node Insertion The tree is constructed starting from a root node that stays online. When joining the grid, a 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.

Node Removal 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 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, 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.

Data Search Searching for and locating data starts by a data access request at a given node. The process is described in Figure 3.7. 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, 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

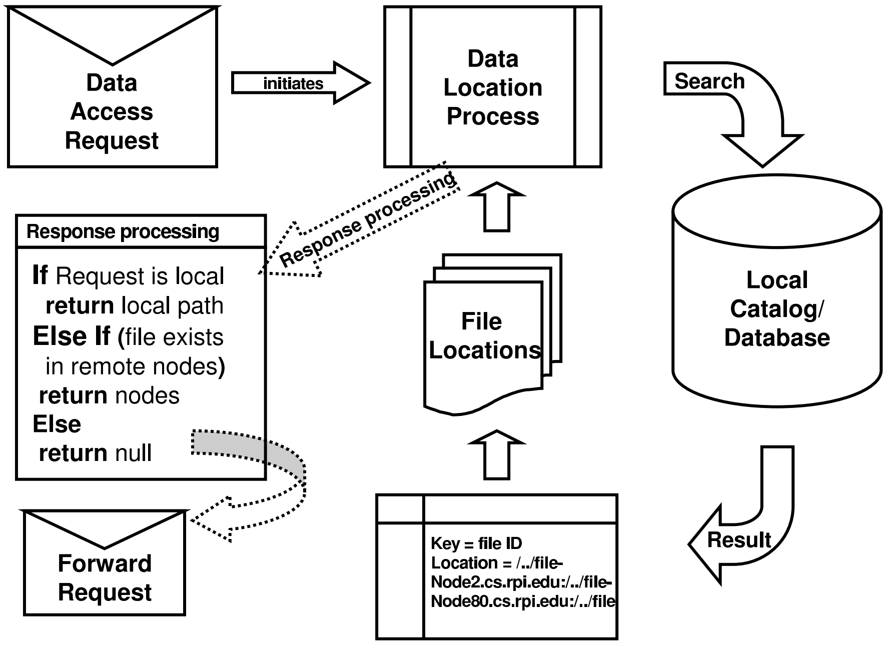

Figure 3.7: Replica/Data Search Process

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, thereby, 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.

3.3.6 Resource Monitoring

The replication mechanism is guided by cost/benefit analysis that is based on current demand for the data sets, locality of requests, network resources availability, and storage capacity to create copies of the data. To get accurate estimates and measurement of resource availability and levels of consumption we use multiple monitoring tools. To monitor storage space availability we use operating systems tools to measure disk space availability. To measure data access frequency, the Resource Monitoring Service keeps track of accumulated access requests to the data. To measure bandwidth availability, the middleware provides a mechanism that enables each member of the Data Grid to estimate bandwidth availability to other members.

To monitor network connectivity each node monitors its connections and links to its immediate neighbors. Periodically each node probes its immediate neighbors to check if they are online or if there have been any node failures. The periodicity is based on estimated up-time of Grid participants. In our current implementation the system performs the checks daily.

Each node uses a local database to collect statistics about outgoing access requests and to monitor the origin of responses, as well as the path traveled by the requests if desired. Keeping track of the path of the requests enables the nodes to identify the list of *preferred neighbors*. As described earlier this list enables each member node to adjust its connections depending on the data flow and users' interests. Details about the database implementation are provided in Chapter 4.

3.3.7 Cost Model

In this section we outline the cost model we use to evaluate the costs and benefits of replica creation and placement. 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. Let $C_w(i)$ be the write cost for a given object i and $C_r(i)$ be the read cost for the same object. Then, $C_w(i)/C_r(i) = \alpha_i$ is the ratio of the write cost to the read 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

$$cost_{v,i} = (\lambda_{v,i} + \alpha_i \mu_{v,i}) size(i) d(v,r) \tag{3.1}$$

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 $bandwidth(v, r)$ is the total available bandwidth between nodes v, r.

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 node of 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:

$$\frac{cost^i(N, R_i, v)}{size(i)d(v, c(v, R_i))} = -\lambda^t_{v,i} + \alpha_i(\mu^t_{r,i} - \mu^t_{v,i}) \tag{3.2}$$

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 $N - T_v$ 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 replica set R_i is given by:

$$Cost(N, R_i) = cost(N, r) + \sum_{v \in R_i - \{r\}} cost^i(N, R_i, v) \tag{3.3}$$

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 3.3 is constant. Hence, we only need to consider the problem of optimizing the following cost:

$$Cost'(N, R_i) = \sum_{v \in R_i - \{r\}} cost^i(N, R_i, v) \tag{3.4}$$

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.

CHAPTER 4
Distributed Replica Management Middleware Architecture and Development

In the previous chapters we laid down the motivation and rationale for our work, and provided a high level overview and description of our design decisions and overall approach. In this chapter we provide more details about low level and architectural design decisions, as well as a thorough description of key components and their implementation.

4.1 Replica Management Middleware Architecture

The main motivation behind our work is to provide a lightweight replica management middleware that can be easily deployed in large numbers of distributed nodes and that requires the least amount possible of central control or management. To that end we have designed and developed a fully distributed middleware that takes advantage of existing infrastructure: large scale networks of distributed computing and storage resources that are connected by reliable networks using widely adopted communication protocols such as TCP/IP. Our middleware creates a virtual machine at each participating Grid node, providing users with seamless access to a larger pool of remote resources and hiding access and control complexities. We have developed our prototype using Java 1.5 SDK.

The components of our middleware can be organized in a layered architecture as shown in Figure 4.1. These layers are: a Resource Access and Control layer, a Communication layer, and a Management layer. These layers provide the services described in Section 3.3. Each layer builds on top of and uses the services offered by the lower layers. We can describe the layers as follows:

- The Communication Layer consists of data transfer and authentication protocols as data organization and overlay support. The transfer and authentication protocols are used to ensure security, verify users identities, and maintain data

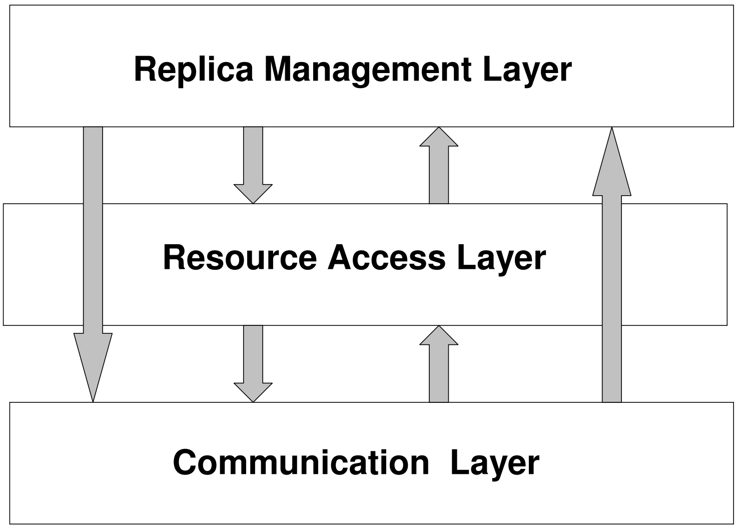

Figure 4.1: Lightweight Replica Management Middleware Architecture

integrity during data transfers. In our implementation we use SSH (Secure Shell) in combination with PKI (Public Key Infrastructure) to authenticate users and control access to resources, and SFTP (SSH File Transfer Protocol) [11]. Only authenticated users are allowed access to data and resources on the participating Grid nodes. This layer also provides support for the overlay network structure. The overlay structure enables nodes to keep lists of direct neighbors. As we outlined in Section 3.2, our middleware implementation supports different overlay structures to manage communication and data location in the Grid. Data access requests are routed in the Grid using the overlay structure to take advantage of data organization and location on the Grid.

- The Resource Access Layer consists of basic software and tools that provide access to available resources and monitors their usage and availability. Each resource present in a Grid node runs software such as a file system or database to provide an access interface to users. Locally stored data at each node is managed by the local file system. To monitor local storage space availability and capacity the system relies on data provided through the tools provided by the local operating system. This layer also includes a Replica Catalog component 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. The Catalog thus helps to locate locally stored data or route data access requests to the appropriate Grid node. The Catalog is implemented using the Berkeley Database which is available under the BSD license.

- The Resource 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. These services provide transparent and efficient access to distributed data through simple and easy to use interfaces and APIs.

Each layer in the architecture provides a set of services that build on top of and use services offered by the lower layers. In Section 3.2 we provided a general framework of our Data Grid middleware. In that section we outline the major high level services offered by the middleware and their functionalities. Each layer in our architecture includes and supports some of these services. In the next subsections we provide further details about each layer and its components and map the services offered by the middleware to the different layers.

4.1.1 Data Transport And Communication

Data transport is an important component in the Data Grid middleware. In addition to managing and controlling data movement between different Grid nodes it also provides different security mechanisms to protect data integrity, ensure users authenticity, and control access to Grid resources. The Communication layer in our architecture can be further organized into three major components. The lowest level component is the data transfer service that uses the Internet protocol TCP/IP to transfer data between two physical locations. Most Data Grid implementation rely on the use of FTP and GridFTP protocols [87]. In our implementation we use the SSH File Transfer Protocol (SFTP) [11]. SFTP is a network protocol that provides secure file transfer over the Secure Shell (SSH) protocol. SFTP is platform independent, provides reliable data manipulation over a secure stream, and acts as a remote file system protocol. The SFTP protocol provides a range of operations on remote files such as remote file deletion, directory listing and file transfer resumption. We use SFTP over SSH-2 to provide reliability. To support a Java interface to the SFTP protocol we use SSHTools [10]. SSHTools is a suite of Java SSH applications that provide SSH Java libraries and APIs supporting SFTP Client among other SSH based applications. SSHTools is available under a GNU General Public License (GPL).

The next component in the Communication layer is the Overlay Network used to route data access requests and locate data in the Grid. This service is used to manage each node's connectivity to its neighbors. As we mentioned in Section 3.2, the overlay network consists of predefined application-level connections between neighboring nodes. To structure these overlay networks we use two different topologies: tree and ring. The use of these topologies forms the basis of our approach to building scalable replica management middleware. The graphs created by these structures define the communication paths that can be used to navigate through the network and locate data. To transfer data however, the physical network takes care of properly routing data using the underlying data transfer protocol.

The third major component and functionality offered by the Communication layer is security. It is imperative in a distributed environment such as the Grid to guard against malicious attacks, ensure user authenticity, protect data integrity, and user and application confidentiality. Security mainly supports two levels of protections: user authenticity and data transfer protection or encryption. User authenticity is provided through the use of Secure Shell (SSH) in combination with Public Key Infrastructure (PKI). SSH is a network protocol that is used to connect to remote computers and run commands over the network. SSH provides "secure encrypted communications between two untrusted hosts over an insecure network" [10]. To support user access control we use public key based authentication. A list of public keys corresponding to all users is distributed and replicated in all participating Grid nodes to check each users access credentials. To support data transfer security, SSH is used in combination with SFTP. Access control is provided through file permissions and limiting the number of files that can be accessed by users. SFTP enables data transfers to be resumed starting from the last data byte that was acknowledged. This feature provides added fault tolerance to the system especially for large data transfers.

The Communication layer provides many low level services that support the high level services we described in our general framework in Section 3.2. The one service directly supported by this layer is the Routing and Connectivity service.

4.1.2 Replica Management

Data Grids provide a platform for the scientific community to share, access, transfer, and process large collections of data distributed worldwide. Replication is a key feature that is needed to provide efficient access to all participating Grid users, ensure scalability of the Data Grid, and preserve bandwidth usage and consumption. Replica creation is guided the user needs and data popularity as well as storage and bandwidth availability. In our work we adopt a distributed approach to replica management. Each member node of the Data Grid runs its own Replica Manager. This manager controls the creation of new replicas to ensure access to popular data to local users. The replica manager also manages local storage resources. Data is only replicated if there is high demand for it locally, and storage space is available. The replica manager uses the replica *Catalog* to keep track of replica locations. Each Grid node has its own catalog to register available local data sets and newly created replicas. Not all requested data at a given node is necessarily replicated; data can sometimes be transferred for a lesser cost than creating a local replica. High performance network resources available in Data Grids and the data transfer protocols that are used contribute to the reduction of the cost of data transfer for infrequently requested data sets. The replica catalog is a component in the Resource Access layer, but it is mainly used by the replica management services.

In addition to keeping a record of locally available data, the catalog keeps track of the locations of remote data collections that have been previously requested from a given location. Keeping track of that data decreases data look up time and improves data access performance. In addition, data sets can be replicated at multiple locations and multiple records are then registered for each of these locations. The catalog is queried by users and applications to look up requested data sets and discover their locations. To choose among multiple locations for a given data access request or transfer, bandwidth availability helps determine the best source node. Because of the distributed nature of our catalog implementation the replication strategy and mechanisms used play an important role in the performance of data access for the Data Grid. The overlay network structure supported by the Communication layer provides a model and topology that guides data discovery and that determines how the Grid nodes are organized and how they communicate.

Our replica management middleware supports two different levels of replication: *data replication* and *catalog replication*.

- Data Replication: Data is replicated in the Data Grid based on users' demands and storage availability. Replica creation is also based on replica maintenance cost compared to data access benefits. As outlined in Chapter 3, the replica management services uses a cost function to calculate costs associated with creating a replica and evaluate the gains from preserving bandwidth usage and placing popular data closer to users. The replica management services are located on the Replica Management Layer of the architecture.

- Catalog Replication: Because of the distributed nature of the replica management middleware, the replica management service also supports catalog replication. Each node in the Grid is mostly responsible for publishing and keeping track of locally available data. However, to improve data discovery and access performance, this data needs to be distributed and shared with other nodes. We will discuss in further detail in the next section different catalog replication mechanisms.

4.1.3 Resource Access

The Resource Access layer provides support for access to resources at each member node of the Data Grid. The layer consists of basic software and tools that provide API's enabling users and applications to access available data sets and local storage resources. The layer also supports the monitoring of resource usage and availability. Each resource present in a Grid node is managed by software such as a file system or database to provide an access interface to users. The local file system controls access to locally stored data, and access is granted based on predefined and assigned permissions. On top of the file system a local data repository is used to enable users to locate the data they request. The data registry is a *catalog* that keeps track of data locations.

For the catalog implementation we use the Berkeley Database (DB) [2]. DB is a high performance database library with Java bindings as well as for many other languages. It has a simple architecture and provides simple data access and management API's but still supports many advanced database features including management and control of large amounts of data up to 256 terabytes [2]. DB supports multiple data items for each single key as well as different key/data formats and types.

To keep track of data and replica locations, each data set or file name is mapped to the physical locations where that file is stored. The location of a requested data set is fetched from the catalog using its logical file name as key. If the data is available at the local node, the search returns the path for the file, and if the local node is aware of the remote locations where copies of the requested data are available that information is also stored in the catalog. The remote node's address is mapped in the catalog to the file's logical name. In the presence of multiple remote locations for a requested data set, a performance metric is used to choose what remote node would provide the most efficient and fastest access to that data. The performance metric is based on bandwidth availability estimations.

To monitor resources availability the middleware uses tools provided by the underlying Operating System to capture statistics about each resource usage. Through Java interfaces the system can check storage space capacity, available storage space, and access permissions and rights. Additionally, each node can monitor its connectivity to its neighbors. The monitoring mechanisms are described in Section 3.3.6.

4.2 Middleware Implementation

The middleware is organized in three layers as shown in Figure 4.1. As mentioned in previous chapters, the middleware consists of different components. Each component provides a service that is either high level and used directly by the users, or a low level service that is used by other services to access local resources. The major classes implementing these components are illustrated in Figure 4.2. Each class implements some or part of a service. Table 4.1 lists the major classes and maps each class to the corresponding service.

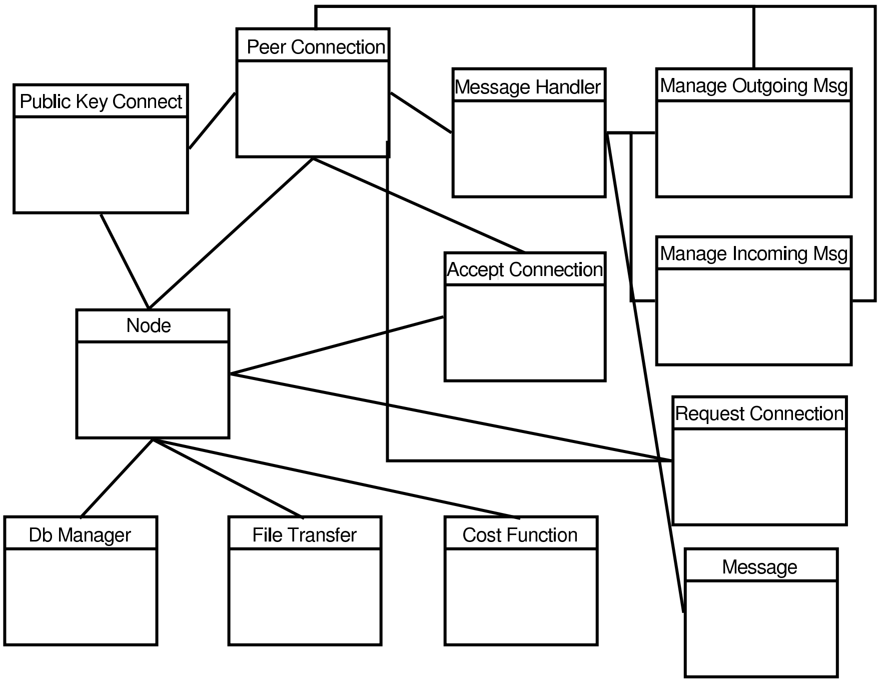

Figure 4.2: Overview of Middleware Implementation

4.3 Communication

In this thesis we develop a distributed data management middleware where components of the system run on distributed and networked participating computers. A communication mechanism is thus needed to support messaging, remote data access, data discovery and search in such a distributed environment. Additionally, each participating Grid node running the middleware needs to check its connectivity

Classes	**Layer**	**Interactions**	**Services**
Node	Management	Resource Access	Main User Access Interface connect-to/access the Grid
Peer Connection	Resource Access	Management Transport	Handling Remote Connections
Request Connect	Transport	Resource Access	Handling Outgoing Connection Requests
Accept Connect	Transport	Resource Access	Handling Incoming Connection Requests
Message Handler	Transport	Resource Access	Processing Incoming requests and generating Outgoing responses/messages
Manage Incg Msgs	Resource Access	Transport	Managing message queue Dequeuing incoming messages
Manage Outg Msgs	Resource Access	Transport	Managing outgoing queue Queuing outgoing messages
File Transfer	Transport	Resource Access	Managing data transfers
Public Key Con	Resource Access	Management	Handling users authentication
Db Manager	Resource Access	Management	Manages local data catalog Stores local monitoring statistics
Cost Function	Management	Resource Access	Replica and storage management

Table 4.1: Class Implementation and Organization Overview

to its neighbors to maintain and monitor its membership and participation in the Data Grid. The middleware provides listeners at each Grid node that use uniform communication interfaces to send, receive, and process messages.

To support reliable communication between the Grid nodes, we use TCP sockets to provide end-to-end communication channels between the different Grid nodes. The TCP communication establishes a channel between a server program and a client program. Each one of these two programs binds a socket to its end of the connection [12]. To communicate, the client and the server each reads from and writes to the socket bound to the connection. The communication interface at each node provides listeners which in turn implement a server socket to listen to and receive incoming messages. To communicate with a remote node, the communication interface at each node uses a client socket to initiate a communication channel with a remote socket server and establish a connection with the remote node.

Our communication mechanisms support scalable expansion and deployment of the middleware. New nodes can join the Grid and leave without disturbing the underlying connectivity fabric of the overall Data Grid. This mechanism provides fault tolerance support in the presence of node failures. Following the occurrence of a node failure, the node's neighbors detect the failure when their messages are no longer accepted. This leads the neighbors to change their connectivity and update their neighbors lists. We will address in further detail the algorithms used to support runtime adaptivity and support to node failures. We should highlight however the fact that very few steps are involved to detect a node's failure using this communication mechanism. Since each message sent over the socket channel is acknowledged, in the event a message is not acknowledged and connection is refused the node at the opposite end is notified that the connection was closed. Before a given node drops a possibly failed node from its list of neighbors, another communication attempt is made and network monitoring tools are used to check if the suspected failed node is still online. If these attempts reveal the suspected node is indeed offline then all its neighbors drop it from their connectivity list. In addition to node failures, nodes can dynamically leave and join the Grid in small scale collaborations. In case a node needs to leave and cease its participation in the collaborative Grid environment, it informs its immediate neighbors and notifies them. The neighbors in turn update their connectivity lists by dropping the node from their routing list.

The TCP sockets provide an added security layer to support users' authenticity and security. The communication channels provided by the TCP sockets are reliable; no messages exchanged over the channel are dropped, and all data sent arrives at the receiving end in the same order. The Grid Join and Leave processes are thus ensured to be reliable and provide support for verifying users credentials and access rights.

4.3.1 Node Insertion And Removal Algorithms

The topologies used to overlay data on the Grid support dynamic node addition and deletion. When new nodes join or leave the collaborative environment the topologies are adapted to the changes and dynamically adjusted. The initial topology is constructed starting from one node first. In the case of a *Top down* hierarchical topology (see Figure 3.1), this node would represent a major storage site. More sites joining the Grid are added as child nodes of the main storage site. Depending on the number of organizational domains the Data Grid spans through, more levels of the topology, i.e., tiers, and nodes are added the same way. Nodes are added to the Grid by attaching to an existing Grid node as a child node, thus creating a *spanning tree*.

Node Insertion. The tree is constructed starting from a root node. When joining the grid, a node is added by attaching to an existing grid node as a child or a sibling at any level of the exiting tree. Existing grid nodes are published in a Web page or could be accessed through a Web service. Before joining the Grid, each joining node needs to download the midldeware binaries. The binaries are available through the same site publishing the existing members; the binaries are also accessible at each existing member which can in turn publish them and make them accessible to more future participants. 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.

Node Removal. 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. As described in Section 3.3.4 to avoid having a disconnected tree in case a node fails and disconnects before sending any notification messages, each node periodically checks if its immediate neighbors are still alive. 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 and a grand child node. However in the event of the failure of both the parent and grandparent nodes, to reattach to the Grid a new parent node is selected from the list of existing members.

In the case of a Bottom Up hierarchical topology (see Figure 3.3, the overlay is constructed starting from the data collection nodes which constitute the leaf nodes of the hierarchy. Nodes interested in joining the collaboration by sharing and distributing the data, attach to the data collection points as parents. The same way depending on the number of organizational domains more levels are added to the spanning tree. Similar to the Top Down hierarchy, each node keeps track of its parent and child nodes. In the case of node failures or node detachments, connections are adjusted to reconnect children of the failed or departing nodes.

In a hybrid model, the types of connections include in addition to parent-child connections, sibling connections (see Figure 3.2). A node can connect to the Grid by being added as a sibling or parent or child node. Each node keeps track of its sibling, parent, and child connections. In case of a node failure of a sibling node, the connection to the failing node is removed.

When a node leaves the Grid it informs its immediate neighbors by sending a leave notification. The neighbors readjust their connections accordingly by removing the link to the departing node from their contact list. Each node in the network keeps track of its siblings, parent, and child connections. In case the departing node is a parent node, its child nodes update their connection list and attach to the grandparent node. In case the departing node has sibling connections, once informed of the departure its immediate neighbors update their connection lists and attach to the next sibling.

4.3.2 Messaging

The communication mechanisms outlined above provide the necessary tools for sending, receiving, and processing messages. A messaging mechanism is still needed to exchange messages between the Grid nodes and to request access to services and resources. To support messaging and formulate the messages to be used for communication between the Grid nodes we use XML (Extensible Markup Language) [5]. XML has been established as a standard for representing text-based structured data. Different types of information and data can be encapsulated in an XML document. XML also provides support for describing different types of data. In our implementation we use XML to describe different types of requests and

Message type	**Category**	**Description**
Hello	System Message	Initiating a connection channel with a peer node
Bye	System Message	Closing the connection channel
FORWARD	Data Search	Forwarding a data request for a data set not found locally
DATA OFFER	Data Search	Response to data request message from a node where data was located
DATA ACCEPT	Data Search	Response to data offer message to start data transfer
RESOURCE UPDATE	Replication Message	Catalog replication

Table 4.2: Messages Overview

responses exchanged between the Grid nodes. The major message categories are:

- System Messages: joining the Grid, leaving the Grid
- Data Search Messages: data access requests, forwarded requests
- Replication Messages: catalog replication and update propagation
- Data Location/Reception Responses: acknowledgement for data location, acknowledgement for data reception

To formulate the different message types we created template XML documents that describe different attributes and fields of the message. We use the attributes to define the types of requests issued at each node. The fields of the message are also used to track the path each message traverses from the source originating the request to the end node where the requested data is located. Table 4.2 provides a listing of the main message types used within the middleware. To process the XML documents we use JDOM [49]. JDOM provides lightweight means for reading and writing XML documents while inter-operating with existing standards and without the complexity and cost of other solutions. JDOM provides API's for accessing, manipulating, and producing XML format data from Java code. It is a Java-based document object model for XML that builds XML documents by using XML parsers and integrating with Document Object Model (DOM) and Simple API for XML (SAX).

CHAPTER 5
Data Grid Simulation Study

In order to evaluate our approach, we developed a Data Grid simulator called GridNet. This simulator provides a modular simulation framework through which we can model different Data Grid configurations and resource specifications. In this chapter we present the framework and design of our simulation, outline the data models simulated, and the algorithms used to manage replica creation and location.

5.1 Background And Related Work

The steady increase of collaborative research and problem solving has led to the rise of data production volumes and data sharing needs. Simulation presents system designers with unique opportunities to study different approaches and implementation paradigms as well as their performance before a costly deployment on a real infrastructure. The simulation environment provides the users with the ability to run multiple tests and an easy framework to change system parameters and experiment scenarios with very small overhead. Redeploying a real middleware and updating control and access API's of the underlying infrastructure requires more time to update every distributed component of the system and re-launching the middleware. The attractive features of simulation led to a rise in interest in modeling Data Grid environments and simulating different data replication techniques as well as basic file replication protocols to support development of Data Grid middleware [80].

Different studies were conducted to model scientific experiments settings and configurations, such as the CMS and ATLAS experiments [80, 7]. A simulation framework was introduced in [22] where data replication is combined with job scheduling. In contrast to our approach, introduced first in [57, 56], this simulator uses a prediction function based on spatial and time locality regardless of the overall data access cost in the Data Grid. In [64], an approach is proposed for automatically creating replicas in a typical decentralized P2P network. The goal is to create a certain number of replicas on a given site to guarantee minimal availability requirements. Different replication and caching strategies within simulated Grid environments are discussed in [65] and their combination with scheduling algorithms is studied in [64]. The replication algorithms used 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 and data access costs as well as data popularity along with its spatial and temporal locality based on user demands.

In [66] the authors have studied different replication strategies coupled with job scheduling techniques in a grid. 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 [67] the authors study the performance of a somewhat distributed and dynamic replica creation mechanism in P2P 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 [25]. 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. The simulations allowed the authors of this research to perform initial verifications of the design and evaluate the performance of their strategies. The results show that a dynamic replication technique yields better performance with larger file sizes and with an appropriate allocation of storage space. The results 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 data management middleware that actively 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. Our simulator was also among the first simulators to be introduced in the field to enable the study and analysis of Data Grid replication mechanisms and their performance. The simulator has been used by multiple research groups who were investigating replica management techniques in Data Grids.

5.2 Simulation Design

In order to validate our approach we designed a Data Grid simulation Framework through which we can model realistic scenarios. To that end we developed a modular simulator called GridNet. The simulator was developed in C++ and was built on top of the network simulator NS [85]. NS provides the ability to abstract and model basic Grid network components, i.e., nodes and links, as well as the ability to model messages and data transfers. 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 application level data that is passed down to the NS node as packets. In addition, there are also packets representing grid user requests as well as the start and the end of grid data transmissions in NS that transfer control of the simulation to GridNet. The latter 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 layer, the grid node and replication enables us to use the existing package, NS, and add only the Data Grid specific elements to the simulation.

One of the main considerations in designing our simulator was to model the 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 High Energy Physics experiments conducted in CERN, CMS and ATLAS [46]. We assume a multi-tier Grid topology for our simulations.

5.3 Overview of the Network Simulator NS

Ns is a discrete event network simulator that allows the simulation different network topologies and components [85, 19]. In ns, a simulation scenario defines the input configuration for a simulation run. Simulations are constructed using several components: a network topology, which describes the physical interconnections between nodes and the static characteristics of links and nodes, traffic models, which define the network usage patterns and locations of nodes initiating communication through unicast and multicast messages, test generation, which creates events such as multicast group distribution, and network dynamics (node and link failure) [19]. NS supports both pre-defined and automatically generated network topologies. Pre-defined topologies may be created manually or chosen from a topology library. NS provides a wide variety of traffic models that support different transport protocols. At present, supported protocols include reliable delivery transport (e.g. several TCP variants), and unreliable transports (e.g. UDP). FTP applications can be modeled on ns using the former model, while the latter models telnet-like applications. Through the simulation interface, scenarios are defined in a scripting language (Tcl/Tk) while the underlying processing is performed in the back-end simulator and written in a system language. Mainly the simulation kernel is implemented in a compiled language (C++), while simulations are defined and configured by a simulation program written in the Tcl scripting language [19]. For an object oriented design support, an extension to Tcl, the Otcl scripting language is used.

5.4 GridNet Architecture

Grids are made up of a large collection of computers connected by wide area networks. Most grid nodes provide both computational (CPU) and storage (disk) resources. As such, our simulation is based on that model. Grids are composed of a connected set of nodes that provide either computational or data resources, or both. 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 [7]. Figure 5.1 shows an example of the architecture

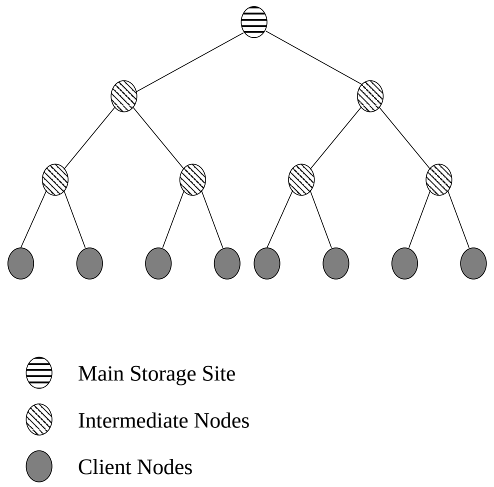

Figure 5.1: Network

as used in the CMS experiment, it also shows the simulation model and topology used in our experiments. 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.

Packets representing grid user requests as well as the start and the end of grid data transmission in NS, transfer control of the simulation to GridNet. The latter simulates the replication decision at each node and generates new NS traffic (forwarding requests other nodes or sending the requested data to the client). Given the configuration of the adopted model, the simulator allows us to specify different types of nodes: *client*, *server* and *cache* nodes.

- Server Node: represents a main storage site, where all or part of the data within the Data Grid is stored. This site represents a collection site in both the CMS and LIGO experiments and is the root of the grid hierarchy.

- Cache Node: represents an intermediate storage site, for example a regional site in the CMS experiment. Such sites would have high storage capacity and would replicate part of the data stored on the main storage site. In Figure 5.1, these nodes represent tier-1 and tier-2 nodes (intermediate levels of the tree) on the grid hierarchy.

- Client Node: represents a site where data access requests originate and are generated. The client nodes are always at the leaf level of the grid hierarchy.

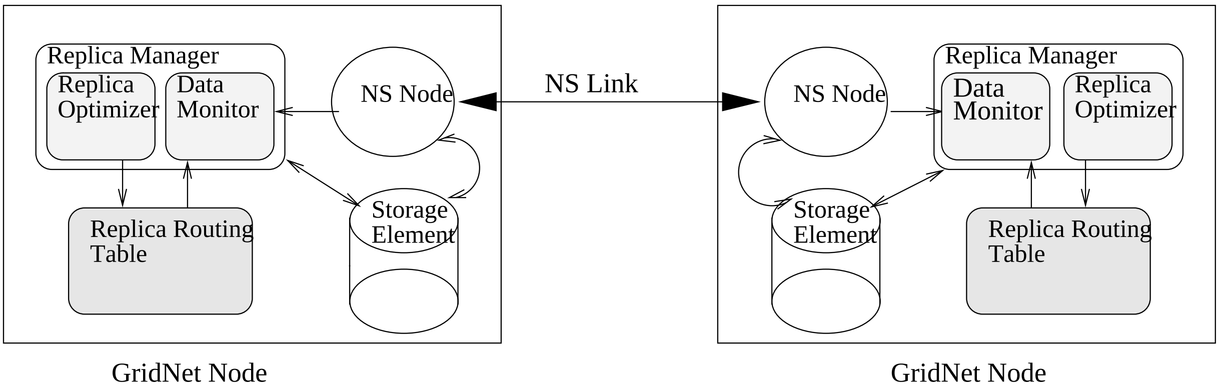

Figure 5.2: Simulation Architecture

The network interface model is specified in the link object that is provided in NS. The link object is used to model the parameters of physical interconnections in the simulated network, such as link bandwidth and latency. As shown in Figure 5.2, 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. The network interface model is specified in the link object that is provided in NS. The link object is used to model the parameters of physical interconnections in the simulated network, such as link bandwidth and latency.

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 include 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 connections to its neighbors. The data collected is then evaluated by the replica placement algorithm or optimizer using the cost function formulated in Section 3.3.7 (equations 3.3 and 3.4).

Algortithm: Process Read Request

```
log-traffic;
log-count;
If (has a local copy and is valid){
    Send a GPT_READ_ACK
    to the original sender directly;
 }
 Else{
    Forward GPT_READ_REQUEST to parent;
 }
```

Figure 5.3: Description of Read Request Processing

The simulation is constructed assuming that each node in a 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 a newer version of data is available,
- GPT_READ_ACK used when the server or cache node confirms the read request
- 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 sequence of commands at the *cache* shown as pseudo-code in Figure 5.3.

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 adopting a bottom up data model hierarchy. Each client runs a set of jobs that require accessing certain data files.

The jobs running-time is simulated by associating a processing time with each file. Each client runs a sequence of jobs concurrently with the other clients. 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 subsection explains further how these tables are updated and used.

5.5 Replication Model

Given the hierarchical property of the Data Grid, we organized our simulations into a multi-tier topology. Nodes placed in higher levels have higher storage capacity. Client nodes are placed in the lowest level. Using the data cost model described in the previous chapter, the replica manager evaluates the access request frequencies against the size of the data object and the network connectivity and bandwidth. As a first attempt to model our approach, we use the read rate (including statistics of read requests received from other nodes) and the available storage space as the basis for estimating the replica creation benefits. And we use the write rate, the data transfer rate, and the replica size as the basis for estimating the replica creation cost. Using equation 3.4 (in Chapter3), the replica manager computes the costs and the gains of creating a new replica. A local replica is created if the gains outweigh the costs. The data transfer cost represents the cost of bandwidth consumed by transferring data from one node to another. If the replica manager decides to create a local replica and no space is available, then another replica is deleted to make space for the new one. The replica manager deletes the least recently accessed replica. It makes that decision using an evaluation formula that rates the access requests of that replica and the number of other nodes requesting that replica. The algorithm used by the replica manager is shown in pseudo-code in Figure 5.4.

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

Algortithm: Local Replica Creation

```
log read_count;
log write_count;
Check_requests_cost() {
    for (i=0; i < log_entry; i++) {
        if (write_count[i] > 0) {
            cost = (read_count[i] * size[i]/band(node,replica))
            + (write_count[i] * size[i]/band(node,replica));
            gain = \sum_{known replica sites}(write_count[i] *
            size[i]/band(node,replica));
            if (gain > cost) Create_replica(file[i]);
        }
        else {
            if (read_count[i] > Threshold)
                Create_replica(file[i]);
        }
    }
}
Create_replica(file){
    if (no storage space available) {
        \* Order Local Replicas by access Rate *\
        for (j=0; j<local_replicas_nbs; j++) {
            pick local_replica with least access_rate[j];
            if (access_count_replica > access_rate[j]) {
                delete local_replica;
                add local_replica for file[i];
            }
        }
    }
    else add local_replica for file[i];
}
```

Figure 5.4: Cost Based Replica Creation Algorithm

path, the bandwidths of the node's connections are evaluated to determine the more 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 implemented by the Data Management Component of the Globus toolkit [28, 38, 36]. These tables provide a mapping between logical file names and their physical locations on the Data Grid.

CHAPTER 6
Performance Study

In this chapter we study the performance of our approach, both through the simulation and deployment of the middleware. We start by describing the testing environment we use to deploy our prototype and conduct our experiments. We then list the different scenarios we used to run the experiments and describe the data models and patterns that characterize the target environment for our middleware. Finally we show and provide an analysis of the results obtained from our testings and experimentations.

6.1 Simulation Study

In this thesis we address problems pertaining to supporting medium to large scale Data Grids to foster collaborative problem solving between researchers in different institutes and organizations. To that end, we have introduced new middleware for managing data in distributed dynamic platforms. To further study and analyze our approach, our first research interests focused on simulating different replication strategies before developing and deploying the middleware. As outlined in Chapter 5, to validate our approach we designed a Data Grid simulation framework through which we modeled realistic scenarios to test the performance of our approach. The simulations allow us to verify the design and evaluate the performance of our strategies. Simulation helps us to study different approaches and implementation paradigms as well as their performance before a costly deployment on a real infrastructure. The simulation environment provides us the ability to run multiple tests and an easy framework to change system parameters and experiment scenarios with very small overhead.

6.1.1 Simulation Configuration

In the absence of real trace data in the early stages of our research, we carried out numerous simulations using different synthetically generated data and hierar-

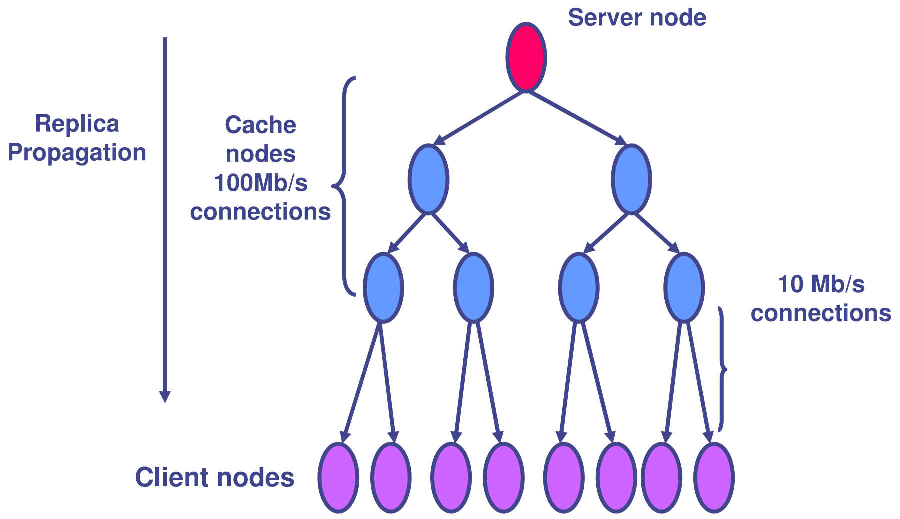

Figure 6.1: Simulation Model Topology

chical models. In these experiments we will use a case study of a three-tier Data Grid topology as shown in Figure 6.1. Table 6.1 shows the parameters used in our study. Essentially, 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. The dataset sizes range from 100MB to 1GB.

Number of sites	28
Total number of Datasets	90
Connectivity bandwidth	server-cache, cache-cache 100Mb/s cache-client 10Mb/s
Total number of requests	1000

Table 6.1: Simulation parameters

Initially only one replica per dataset exists in the system. Users are mapped evenly across sites and submit a number of sequential data access requests (only read requests are used). Requests are processed sequentially and each user accesses only one file at a time. However, collectively, users utilize some files more than others. 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.

6.1.2 Experiments and Results

Our experiments consisted of running different simulation scenarios of the model described above. The topology consisted of a 3-tier tree, with three cache nodes in the first tier. Each of these nodes has cache nodes as its children. In addition, each cache node in tier 1 has three children client nodes. In total, we used 18 client nodes, two cache levels with 9 cache nodes, and one main storage site or server node. The goal of the simulations was to evaluate our replication approach against cases where static replication, and no replication were used. In static replication, subsets of popular replicas were placed in cache nodes closer to the request-generating nodes. Because of storage capacity limitations, these subsets did not contain all requested datasets and did not satisfy all requests. The file sizes used in the simulations ranged from 100Mb to 1Gb. 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. Figure 6.2 shows the average response time of read requests for multiple file sizes comparing scenarios using static or no-replication vs dynamic replication policies. In the static replication scenario the data was placed only in *server nodes*, and no caching was activated at the *cache nodes.* In the static replication scenario popular files were pre-staged in strategic locations at different *cache nodes.* The pre-staging was based on the popularity of data requested by the user applications. Prior to running the simulation, popular files were replicated accordingly at *cache nodes.* The workload at each node consists of running tasks that require both file access operations and the processing of the accessed data.

In the simulation, each file is assigned a processing time that corresponds to its size. The response time measures the combination of IO and processing times. The results show that dynamic replication outperforms static replication by up to 30%.

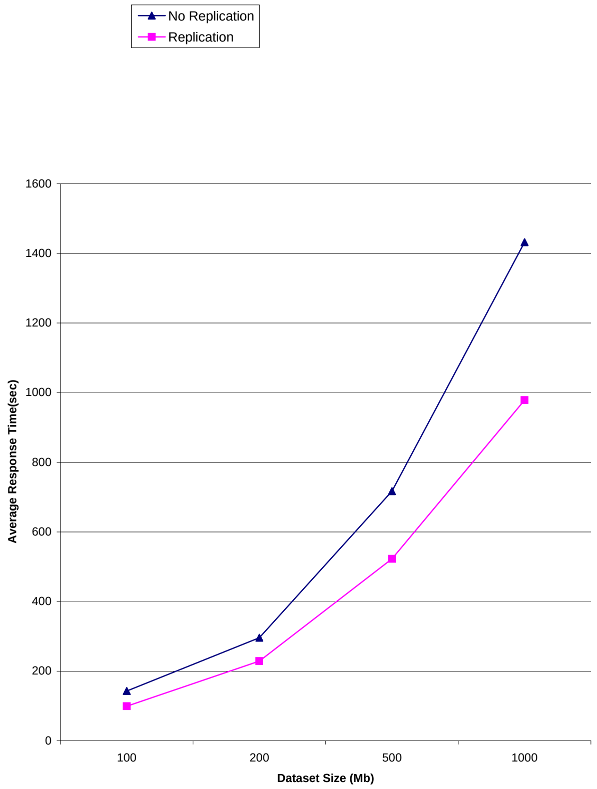

Figure 6.2: Average Response Time For Different Data Access Loads for Replication vs. No Replication

In Figure 6.3 we show the results obtained from comparing three different scenarios using different replication policies and storage/cache capacities at different levels of the hierarchy. The goal of the simulations was to evaluate our replication approach and the affect of cache storage space availability on performance against the case where no replication is 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 scenario 3, dynamic replication is used with larger storage capacities at cache nodes. The no-replication or static replication policy is similar to the policy we used and described in the previous experiment. The dynamic replication policy is based on the cost model described in Section 5.5. The guiding factor of the replication mechanism is the frequency and origin of access requests from *client nodes*. We used 6 data groups for our experiments. Table 6.1.2 shows the range of data file sizes within each data group.

Data Sets Used	**File Size range**
Data set 1	100MByte to 200MByte
Data set 2	200MByte to 300KByte
Data set 3	400MByte to 500MByte
Data set 4	600MByte to 700MByte
Data set 5	800MByte to 850MByte
Data set 6	950MByte to 1000MByte

Table 6.2: Data File Sizes Used in the Experiments

The results in 6.3 show that replication improves overall workload performance by up to 60%. The plots in the Figure show an improvement of up to 60% for Dataset6 from scenario2 to scenario3, and approximately 64% for Dataset5. The results also show that with higher network bandwidths, the performance of high workload scenarios increases noticeably, and that the gains and benefits are more considerable for larger file sizes. The plots in both Figures 6.2 and 6.3 show that dynamic replication yields large improvement in the overall data transfer and response times within the Data Grid. The results of the second simulation tests also show that the use of small caches at intermediate nodes does not improve the data access performance of large data sets in the Grid. With small storage space available, *cache 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 improvement is higher than that of existing ones.

The plot in Figure 6.3 also shows that the average response times of data requests for scenario 2 and scenario 3 are similar when the size of data files is smaller than 500MB. In these cases cache sizes do not much affect the overall data transfer in the Data Grid. However, for file sizes larger than 500MB, scenario 1

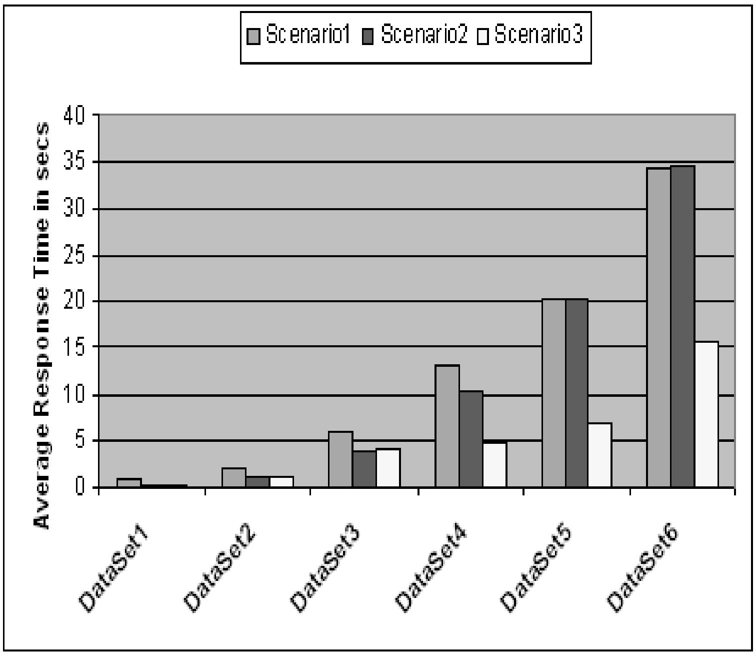

Figure 6.3: Replication Performance With Different Storage Availabilities And Different Data Sizes

and scenario 2 produce the same results while the use of larger storage spaces at cache nodes yields better performance. The experiments results 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 over 60% from scenario 2 to scenario 3.

With the introduction of additional traffic on the grid, we created an environment in which the connection network is not solely dedicated to the grid data transfer. This additional traffic uses part of the bandwidth and introduces extra delays. Our results also show that performance gains increase with the size of data used. As data file sizes in Data Grid environments are reported to reach the Terabyte scale, we expect that a larger scale use of our model will yield better performance results. While running the simulation, we also found 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.

These results are very promising, but they are based on automatically generated workloads and specific grid scenarios. In the follow up of our research, we investigated different scenarios using different topologies and workloads. Based on surveys of scientific applications and user identified requirements for Data Grids we identified popular access patterns and scenarios in collaborative environments. These patterns were detected and extracted from experiments run on the Grid and popular applications. In the next section we will show the results we obtained from deploying our proposed middleware.

6.2 Middleware Deployment

The testbed of our experimentations consists of a cluster of 40 Linux machines with dual processors running the distributed file system NFS. In addition to the Linux machines, workstations from the Computer Science department were also used to emulate virtual organizations running across multiple organizational domains. These workstations are running the FreeBSD operating system and the distributed file system NFS. Each machine though has a local disk that is not accessible through NFS. 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. The models used for our experiments are based on data access patterns observed in popular scientific applications [88] and documented in surveys and studies of existing Data Grid applications. The applications we mostly based our testing on are the CERN High Energy Physics applications such as CMS and Atlas [46, 7]. The data organization models we adopted are based on the models we discussed in Section 3.2 and shown in Figures 3.1 and 3.3.

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 data management agent, each leaf node runs a client host that runs a script that 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.

6.2.1 Experiments Modeling

Scientific applications running on the Data Grid have different workloads, but most applications share the same data access and data flow patterns. In our survey of applications running on the Data Grid specifically, and high performance distributed applications in general, we discovered that most applications have one or two typical data request sizes [17, 14, 7, 88, 46]. The survey also showed that scientific applications demonstrate a bursty behavior. The spikes and bursts in access patterns in scientific environments reflect a rise in interest in certain data files. Such patterns are mostly contributed to and caused by the publications of new results or discoveries. Within a single application, the bursts are mainly caused by the nature of scientific computing where computations are data intensive and require multiple IO operations. These operations mostly occur at the start and end of iterative computational phases. In both tightly and loosely coupled parallel and distributed applications data synchronizations cause the output and input of large amounts of data. In most applications, requests for data sets with similar sizes account for 90% of the overall data requests [88]. A survey of scientists' access patterns show that most of their requests are for file sizes with similar sizes and similar content. Bursty behavior is attributed to the discovery of newly introduced or added files to the community repository.

Defining and determining commonly used access patterns help us model our experiments as realistically as possible. While deploying the middleware with real Data Grid applications would provide a better testing environment, we hope that our personal experience working with and developing distributed and parallel scientific applications such as earthquake simulations and Plasma simulations [26] gives us a unique perspective in shaping and modeling scientific collaborative environments as close to reality as possible. In the next section we will provide further details about the models we used in our testings.

6.2.2 Experiment Set up

As outlined in the previous section, the most popular data organization and collaborative models used by the scientific community are the hierarchical models. The two models we use in our experiments are the *top-down* model and the *bottom-up* model. These models are respectively represented in Figure 3.1 and Figure 3.3. The first data model is used in an environment where there is a single source of data, and that data has then to be distributed and copied to multiple locations to be shared by a large community of collaborators. It shows the data distribution model in the CERN experiments where data is first generated and stored in CERN, and later copied to different distribution and regional centers. From these centers the data is then distributed to different labs worldwide to give access to scientists from around the world.

The model we use in our experiments is shown in Figure 6.4. Data access requests are mostly generated at the nodes in 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. To model collaborative environments as we described above 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. I/O operations follow the patterns we descried in the previous section. Among the 100 files used in the experimentations, we designated a group of 20 data sets as *Interesting Files*. This labeling means that most users would be highly interested in getting accessing to those files at a certain time and analyzing the data. This pattern mimics a natural human characteristic: curiosity. When interesting data is published most scientists would be highly interested in checking it and using it. But their interest might shift to another group of *Interesting Files* afterwards. The

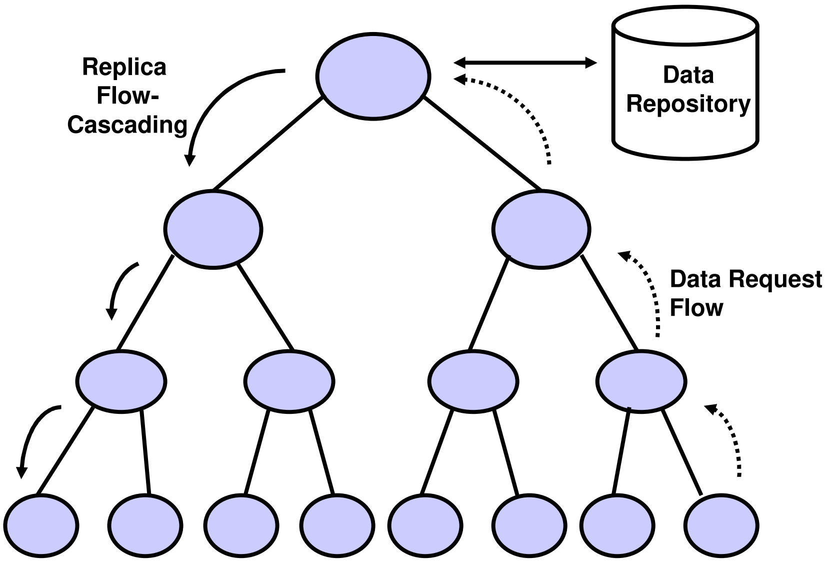

Figure 6.4: Hierarchical Top Down Experiment Model

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.

Figure 6.5 shows the model used to emulate the *bottom up* architecture and evaluate the performance of our approach. Nodes at the bottom of the hierarchy represent the original sources of data. As shown in Figure 6.5 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. We provide further details about the experiments and their results in the next section.

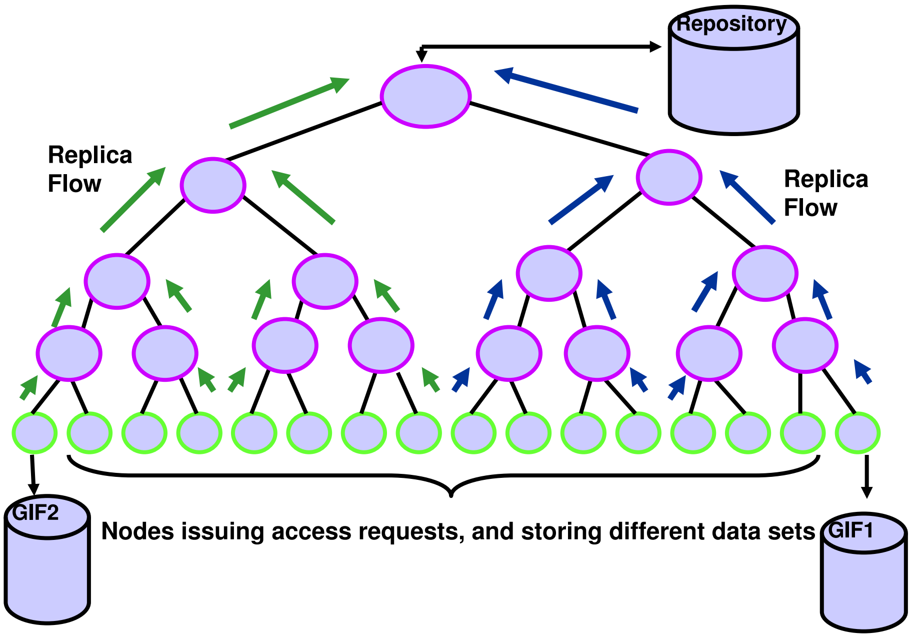
Repository
Replica Flow
Replica Flow
GIF2
Nodes issuing access requests, and storing different data sets
GIF1

Figure 6.5: Hierarchical Bottom Up Experiment Model

6.2.3 Performance Evaluation

The topologies and experimental setup we described in the previous section were modeled according to patterns observed in users' behavior, popular applications, and collaborative environments. The goal of these experiments is to test the efficiency and performance of the overlay network on a dynamic grid platform using cost based replication. The experiments consisted of using a 31 node Grid and placing the nodes issuing access requests at the bottom of the tree. 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. The processing consists of executing a set of operations depending on the data set. The purpose of the processing is to associate and couple a computational task to each data access request. We use the Poisson distribution to model request generation with different arrival rates at each node. 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. Each node has some storage space available where it can store a number of data sets. The storage space availability dictates the number of local replicas each node can create and store.

6.2.3.1 Top Down Model

In the *top down* model data is initially placed at the top of the hierarchy. In the first set of experiments we compare the access response rates at different levels of the hierarchy using different cost thresholds for the replication algorithm as described in Chapter 3. In these experiments we measure the percentage of requests answered at each level of the hierarchy while the data is being dynamically replicated based on different cost thresholds. The cost threshold determines the value that guides the replication. 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 based on multiple trials. The chosen cost values represent each an estimated cost of data transfers between two different nodes in the Grid.

An important element in designing the experiments is deciding the frequency of calling the cost function which in turn triggers the invocation of resource monitoring tools. These calculations incur some overhead. However throughout the experiments we found that these calls do not actually cause a noticeable overhead and do not affect the overall performance of the system. The distributed and decentralized nature of the system helps reduce this overhead. Each Grid node is only responsible for monitoring its local resources and performing local computations. The call to the cost function in the Replica Management Service is triggered by the accumulated number of access requests. After each invocation, the accumulated value is reset to zero. Given the normal CPU usage and consumption in large scale scientific applications, the cost function computations are very small. Additionally CPU consumption of the cost function is not relative to the size of the application, only to the number of data sets used by the application. Most applications and users have an interest in a limited number of data sets. Large scale applications such as earthquake simulations, high energy physics, and climate change modeling [17, 26, 46, 37] require

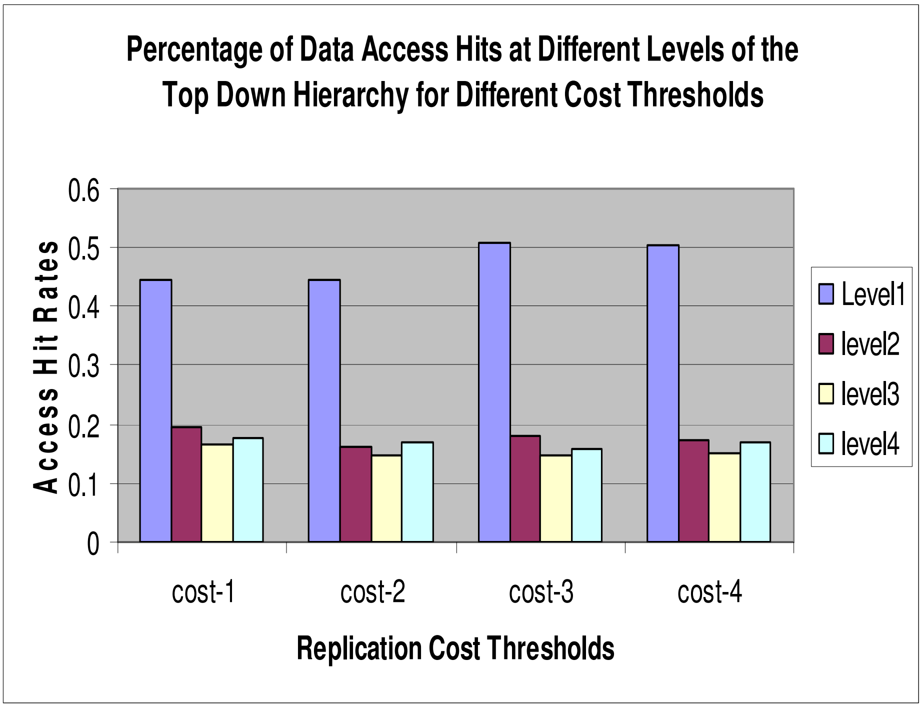

Figure 6.6: Top Down Model Experiment Results

access to a limited number of data sets.

The cost threshold values used during the experiments are organized in increasing order from cost-1 to cost-4. The measurements only take into consideration the data hits and responses registered before data is replicated at the nodes at the lower level of the tree. These nodes as we mentioned earlier represent the origin of the requests. The reason we only account for these data hits is to assess the performance of the replication scheme and how fast data is replicated.

The results of the experiments are shown in Figure 6.6. The results show that replicating data using higher cost thresholds delays the propagation of data to lower levels of the hierarchy since most hits occur at the top level of the hierarchy. With higher cost thresholds more requests are answered at the top levels before data is replicated, thus increasing resource consumption levels. The plots 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.

6.2.3.2 Bottom Up Model

In the next set of experiments we use the *bottom up* model. In this model initially, only leaf nodes store different sets of data each. All the nodes at the lower level store each a set of files, with an average of 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 6.7 and Figure 6.8 show the response rate at both the lower and upper (root) levels of the tree with two different storage capacity values and different cost values. The values chosen for space allocated to caching and storing replicas were decided based on empirical testing of different scenarios and commonly known and dedicated storage spaces in real Data Grids and Grid applications relative to the sizes and number of files used in the experiments. 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 6.7 also show and represent the performance of the system under two different storage space capacities at the top of the tree. The first storage capacity level, *storage1*, is lower than the second storage capacity level, *storage2*. The results show that with higher storage capacity available at the top level 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 6.8 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

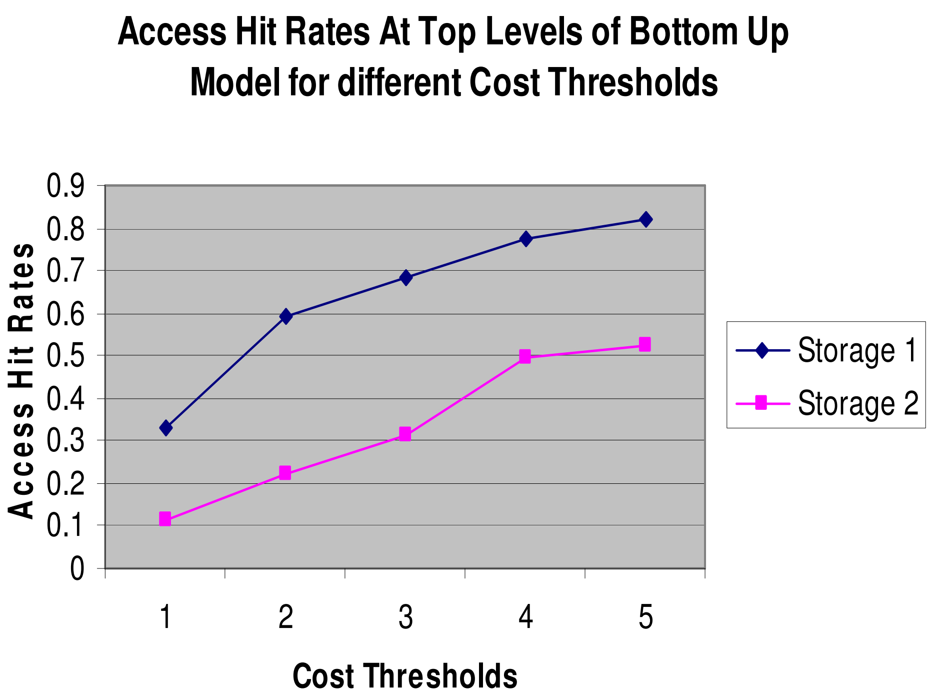

Figure 6.7: Bottom Up Model Experiment Results

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 thus reducing the overall traffic load in the network.

The results shown in Figure 6.9 summarizes the access performance of different data sets based on their access rates and replica location on the hierarchy. The access rates represent levels of data popularity. A 10% access frequency means that 10% of the overall requests were requests to access that specific data set or group of data sets. The replication levels represent levels of the hierarchy where data is replicated and access requests get a hit. *Rep1* represents the top level of the hierarchy, *Rep2* represents the second level of the hierarchy, and so on. The results were accumulated over thousands of data access requests during multiple runs averaged over time.

The plots show that regardless of access frequencies, dynamic replication of popular data closer to the origin of access requests improves access performance by more than 20% and up to 30%and a minimum of 10% compared to static user

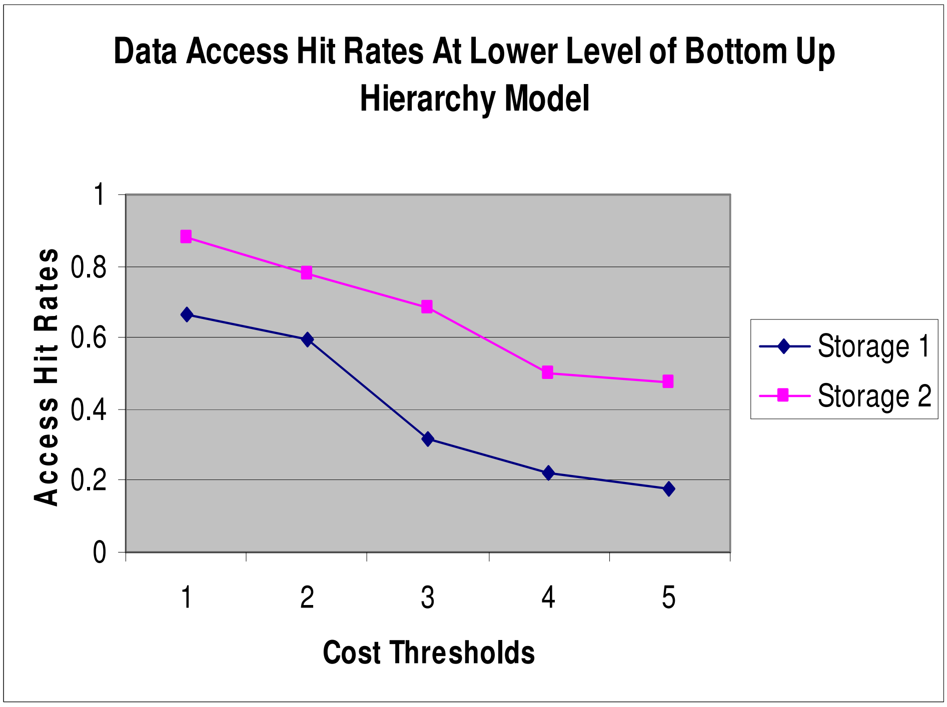

Figure 6.8: Bottom Up Model Experiment Results

initiated replication. The results also show that popular data files benefit the most from dynamic replication. The results do not show a uniform distribution in access performance for the different popularity rates at different levels of the hierarchy. This is due to the fact that when data is replicated at one level of the tree, it does not restrict and limit the replication of this data at other levels of the tree. For example, for data sets with access frequencies of 50%, the presence of replicas at the top of the hierarchy yields a 16% improvement while replicas at the second level of the hierarchy only improved access performance by 10%. The presence of data at the second level of the hierarchy does not necessarily mean that this data has been replicated at all the locations on this level. Since the topology of the experiment represents a *bottom up* model, it means that the requests originate from the lower level of the hierarchy and the access requests do not get a hit in some cases until the requests go through $2\log(n)$ nodes.

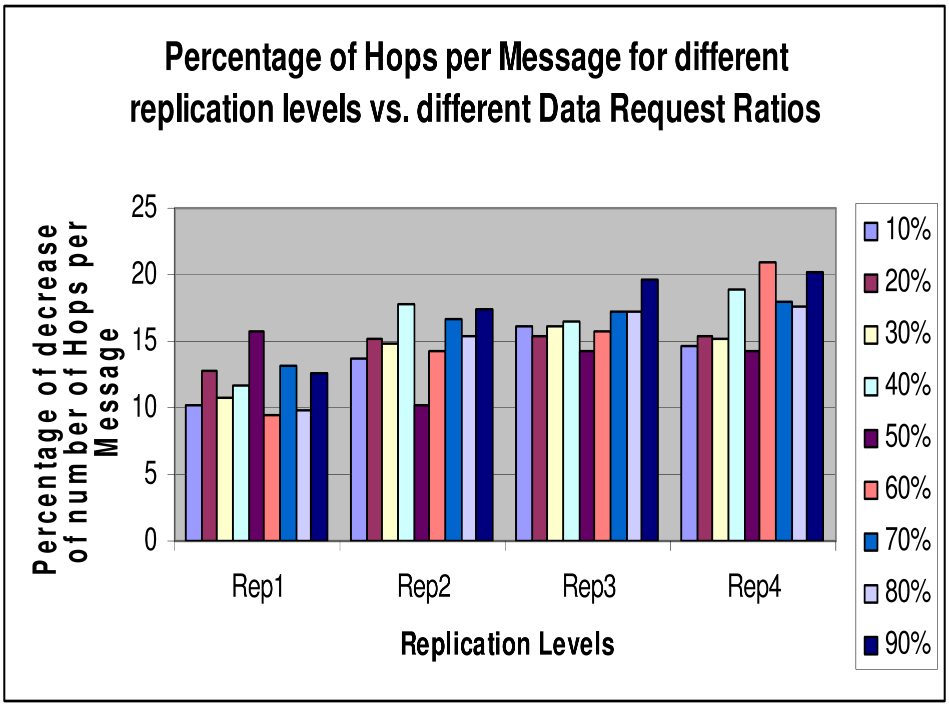

Figure 6.9: Data Access Performance for Different Data Popularity Rates at Different Levels of the Hierarchy

6.3 Summary

In summary, the results presented in this Chapter show that cost guided dynamic replication improves data access performance by up to 30% and a minimum of 10% compared to static user initiated replication. Simulation results show an overall improvement of up to 60% in the overall response time in Data Grids. The results show that the availability of larger storages contributes significantly to improving access performance and lowering network resource consumption. Higher storage availability increases the chances of replicating popular and highly demanded data at different levels of the tree thus decreasing traffic load to the original data sources and overall bandwidth consumption.

During the experiments we discovered that the combination of parameter selection for cost evaluation and resource availability play a key role in influencing the performance of the system. In both the *bottom up* and *top down* models, lower cost thresholds trigger replication at faster rates. However, lower storage availability might lead to race conditions where popular data compete for storage space, thus decreasing performance access. Data collected from multiple experiments suggest that a good approximation of cost values yields better results. In the experiments we conducted we chose cost values based on empirical data and multiple trial runs. The chosen cost values represent estimated costs of data transfers between two different nodes in the Grid.

The results also show that popular data files benefit the most from dynamic replication. Popular data files are first replicated in locations where accumulated frequency rates are higher. The replication scheme follows a cascading model, where replicas are trickled either up or down the hierarchy depending on the location of the data sources. The topologies we use in our approach and experiments take advantage of data organization models that are commonly used and popular in data sharing environments and data intensive applications.

CHAPTER 7
Conclusion

In Chapter 2 we described major components of a Data Grid and outlined existing problems with the current approaches and deployed systems. In Chapter 3 we introduced our solution and approach to managing replicas in data sharing environments for data intensive applications. In Chapter 4 we provided a description of our design decisions, as well as a detailed description of key components and their implementation. We then presented in Chapter 5 our Data Grid: GridNet simulator which we designed and built to validate our approach. In Chapter 6 we studied the performance of our approach, both through the simulation and the deployment of our middleware. In this chapter we discuss the major contributions of our approach and how it can be applied to help create and foster small and medium size research collaborations. Our system is based on and takes advantage of usage patterns that are most common and popular in scientific computing and collaborative environments.

7.1 Discussion

In this thesis we introduce a new distributed and decentralized data management middleware for Data Grids. In our approach, we provide an adaptive and scalable lightweight data management framework that enables users to dynamically join and leave the grid. Our solution provides replication management services that intelligently and transparently place data in strategic locations in order to improve the overall data access performance. In our system, we advocate a local and autonomous approach to replica management at each participating node in the Grid. 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, and use of network resources, as well as any other available local resources. The basis of our approach is treating data as a first class citizen, i.e., prioritizing data queries and data management operations. The data replication, placement, and location techniques we use are based on the analytical model that treats data access performance in a distributed environment as an optimization problem. Our approach also takes 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 running 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. In our case dynamic replication outperforms static user-driven replication by up to 30%. And we believe that deploying the middleware in larger environments would yield even better performance. The results also show that popular data files benefit the most from dynamic replication. Additionally, the availability of larger amounts of storage at locations closer to users contributes significantly to improving access performance and lowering network resource consumption.

Most existing Data Grid implementations and platforms offer limited support to replication. Standard existing Data Grid implementations offer replica services that maintain and provide access to information about the location of data available within the Data Grid. However, these systems do not support automatic and dynamic replication. They only offer mechanisms to transfer and replicate data based on user initiatives. While the deployment of Data Grids remains limited to a number of scientific institutes and organizations, current trends suggest that there is an increasing demand for resource sharing mechanisms and collaborative problem solving platforms. With increasing levels of data production volumes and the need to address and tackle larger problems, we anticipate that in the near future more scientists and researchers will need to access larger collections of data. We believe our middleware fosters and supports the creation of small to medium scale data sharing Grids. It requires almost no administrative management overhead and provides simplified and easy to use API's.

7.2 Contributions

The major contributions of our work are as follows:

1. A solution that combines and adapts approaches previously used in different data sharing environments: the Grid and P2P systems. Exploring the problem from two different perspectives enables us to take advantage of previous experience and existing solutions. Various solutions that have been developed to address similar problems in P2P systems have matured over the years and can be applied to address outstanding issues in Data Grids. Existing Data Grid solutions have not focused on scalability and supporting dynamic and intermittent participation. If Data Grids are to grow their user base and membership participation, they need to provide scalable, distributed, and decentralized mechanisms to support larger numbers of users. Our approach is inspired by the P2P techniques that require no centralized management and advocate self organization. The P2P approach has many attractive features that make it very suitable for Grid environments.

2. A formulation of the replica creation and placement problem and its requirements using a mathematical model and treating it as an optimization problem. While the system does not provide absolute solutions, we use the model to approximate the best possible solutions. The replication mechanism is guided by a cost/benefit estimation strategy that takes into account current demand for the data, origin and locality of requests, network resources availability, and storage capacity. New replicas are created if the estimated accumulated remote access cost to the requested data is higher than the cost of creating a local replica. This approach has significant performance benefits. Storage and bandwidth are consumable commodities that have to be carefully managed in order to improve access performance for multiple and concurrent users.

3. The use of dynamic and adaptive overlay networks and data organization models to connect participating nodes in a Data Grid. The overlay networks are used both to locate and propagate data on the network. The data search process is based mostly on the structure of the overlay network to propagate

access requests. This approach yields better performance results than broadcasting or random propagation. Our work has been cited in multiple research publications including [25, 24, 86] as we were one of the first groups to investigate dynamic and adaptive data overlay networks. Most recently our work was cited in a taxonomy of Data Grids [87] as our approach remains unique in using adaptive and dynamic overlay networks.

4. The use of access patterns to model and organize the data network. Using the proper access patterns yields better performance than relying on random formations and connections. An important pattern in data sharing environments is the sharing and commonality of users' interests. This pattern guides the creation and formation of same-interest communities within a larger community. Taking advantage of that pattern improves data access performance by placing data closer to larger numbers of users and lowering storage and bandwidth consumption.

5. A generic Data Grid simulator, GridNet. The simulator provides a modular simulation framework through which we can model different Data Grid configurations and resource specifications. The simulator validated our approach to replica management in intensive data sharing environments. The results obtained from the simulations show that dynamic replication in combination with our proposed cost model greatly improves data access performance on the Grid. Our simulator has been used by multiple research groups to investigate replica management issues in Data Grids. The simulator was distributed along with a user manual that enabled the users to write simulations as well as update the internals of the simulator to support additional features or functionalities. We received very positive feedback about the usability and benefits of the simulator.

6. New data and replica management middleware that scales with the number of users and supports dynamic and intermittent user participation. The middleware exploits commonly observed and popular patterns in data access and user behavior. Our middleware uses a decentralized mechanism to create and

locate replicas. The decentralized mechanism maps users' interests and behaviors onto an overlay network. The overlay network is dynamically adaptable to changing users' interests. The middleware satisfies Data Gird requirements and provides efficient algorithms and techniques to locate data on the network under user-controlled access permissions. The techniques employed scale with the number of users and Grid nodes. Additionally the system supports dynamic data insertion as well as dynamic and intermittent node participation in the Grid.

7.3 Future Work

In this thesis we address the problem of replication in Data Grid environments by investigating the use of decentralized dynamic replication services used to improve data access time, data availability, bandwidth consumption, and scalability of the overall system. There are still more problems that need to be addressed with regard to data and replica management in Data Grids and in data sharing networks in general. Some future research directions are a natural continuation of our work, while others deal with more general and outstanding issues in data management.

A natural continuation of our work would be to investigate additional user access patterns and overlay structures and study data access performance under these settings. Interest based clusters and formations could be further investigated to uncover and develop new data search and location techniques.

Additionally, new cost models could be investigated and tested using different combinations of parameters and cost evaluation techniques. Different applications require and stress the use of different resources. An important research direction is to study the dynamic deployment and selection of a combination of different replication techniques and cost models. This approach would enable the system to select different algorithms based on current conditions and adapt to changes in users' and applications' behavior.

Deploying the middleware in a larger environment with larger applications would enable us to test the performance of our approach in more realistic settings. During our experimentations we used large numbers of machines under different administrative domains, but security issues prevented us from deploying the middleware across different institutes. We hope in the future to be able to tackle this issue and secure the participation and contribution of different scientists in larger experiments over long periods of time.

An important research problem in distributed systems in general and Grids specifically is identifying the relation and effect of data and replica management on other services. In our current approach we have decoupled replica management from other services such as storage management. This approach has clearly more benefits and advantages for the overall data access performance. However, integrating our services with additional existing Grid services would enable us to reach wider communities and uncover new patterns that can improve data access performance. Many storage systems have been developed to satisfy the growing need to provide more storage allocation and management services within the Grid community. These storage systems provide access to different types of data storage across distributed heterogeneous platforms and maintain metadata about available data collections. Supporting access to such systems and providing mechanisms to handle, replicate, and manage metadata would widen the scope of our work and enable us to deploy our middleware in larger environments.

LITERATURE CITED

[1] Akamai, cambridge ma, usa. http://www.akamai.com.

[2] Berkeley database. http://www.sleepycat.com/.

[3] Digital island. http://www.digitalisland.com.

[4] The european data grid project, the datagrid architecture 2001. http://eu-datagrid.web.cern.ch/eu-datagrid/.

[5] Extended markup language (xml). http://www.w3.org/XML/.

[6] The gnutella protocol specification. http://www.gnutella.com.

[7] Grid physics network (griphyn). http://www.griphyn.org.

[8] Napster. http://www.napster.com.

[9] Squid internet object cache. http://squid.nlanr.net/.

[10] Ssh authentication tools. http://www.sshtools.com/.

[11] Ssh file transfer protocol. http://www.sshtools.com/.

[12] Tcp sockets in java. http://java.sun.com/docs/books/tutorial/index.html/.

[13] J. Acosta-Elias and L. Navarro-Moldes. A demand based algorithm for rapid updating of replicas. In *IEEE Workshop on Resource Sharing in Massively Distributed Systems (RESH 2002)*, 2002.

[14] 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(3):749–771, May 2002.

[15] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *IEEE Mass Storage Conference*, 2001.

[16] W. Allcock, 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*, 23:187–200, 2001.

[17] W. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Lee, 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 *Proceedings of SC 2001*, November 2001.

[18] T. Anderson, T. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless network file systems. In ACM, editor, *Proceedings of the 15th Symposium on Operating Systems Principles*, 1995.

[19] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala. Improving simulation for network research. Technical Report 99-702, University of Southern California, 99-702 1999.

[20] G. Barish and K. Obraczka. World wide web caching, trends and techniques. In *IEEE Communications, Internet Technology Series*, May 2000.

[21] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The sdsc storage resource broker. In *Proceedings of CASCON'98*, 1998.

[22] W. Bell, D. Cameron, L. Capozza, A. Millar, K. Stockinger, and F. Zini. Simulation of dynamic grid replication startegies in optorsim. In *Proc. Of the 3rd Int'l IEEE workshop on Grid Computing (Grid 2002)*, 2002.

[23] D. Bosio, J. Casey, A. Frohner, and L. G. et al. Next generation eu datagrid data management services. In *Computing in High Energy Physics (CHEP2003)*, March 2003.

[24] D. Cameron, J. Casey, L. Guy, P. Kunszt, S. Lemaitre, G. McCance, H. Stockinger, K. S. G. Andronico, W. Bell, I. Ben-Akiva, D. Bosio, R. Chytracek, A. Domenici, F. Donno, W. Hoschek, E. Laure, L. Lucio, P. Millar, L. Salconi, B. Segal, and M. Silander. Replica management in the european datagrid project. *Journal of Grid Computing*, vol. 2(iss. 4):pp. 341–351(11), December 2004.

[25] 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. In *CHEP 2004, Interlaken*, September 2004.

[26] G. Cheng, H. Lamehamedi, B. K. Szymanski, and A. Vargun. Web-enabled and speculative performance computing. In *Worldwide Conference of past, present, and future SGI Users: SGII'2000*. 2000.

[27] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Giggle: A framework for constructing sclable replica location services. In *Proceedings of Supercomputing (SC2002)*, 2002.

[28] 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*, 23:187–200, 2001.

[29] M. A. Cooper. Overhauling rdist for the 90's. In *Proceedings of the Sixth USENIX Systems Administration Conference*, 1992.

[30] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems, Concepts and Designs.* Addison Wesley, 2001.

[31] K. Czajkowski, A. Dan, J. Rofrano, S. Tuecke, and M. Xu. Agreement-based grid service management (ogsi-agreement). In *Global Grid Forum, GRAAP-WG Author Contribution,*, June 2003.

[32] D. Davis and M. Parashar. Latency performance of soap implementations. In *Proceedings of the IEEE CCGrid*, 2002.

[33] E. Deelman, I. Foster, C. Kesselman, and M. Livny. Representing virtual data: A catalog architecture for location and materialization transparency - draft of january 26, 2001. technical report GRIPHYN 2001-13, GriPhyN, 2001.

[34] E. Deelman, C. Kesselman, S. Koranda, A. Lazzarini, and R. Williams. Applications of virtual data in the ligo experiment. In *Proceedings of the Fourth International Conference on Parallel Processing and Applied Mathematics (PPAM'2001)*, volume 2328, pages 23–34. Springer-Verlag, September 2002.

[35] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, and S. Koranda. Griphyn and ligo, building a virtual data grid for gravitational wave scientists. In *Proceedings of High Performance Distributed Computing*, 2002.

[36] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, volume 3779, pages 2–13. Springer-Verlag LNCS, 2005.

[37] 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. In *Proc. of the American Meterologcal Society Conference*, 2001.

[38] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl JOurnal of Supercomputer Applications*, 11(2):115–128, 1997.

[39] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufman Publishers Inc., 1999.

[40] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal Supercomputer Applications*, 15(3), 2001.

[41] S. Ghemawat, H. Gobioff, and S. T. Leung. The google file system. In ACM, editor, *19th ACM Symposium on Operating Systems Principles SOSP'03*, 2003.

[42] R. Guy, J. Heidmenn, W. Mak, T. P. Jr., G. Popek, and D. Rothmeier. Implementation of the ficus replicated file system. In *Proceedings of the summer Usenix Conference*, 1990.

[43] R. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek. Rumor: Mobile data access through optimistic peer-to-peer replication. In *Workshop on Mobile Data Access*, November 1998.

[44] D. Hagimont and D. Louvegnies. Javanaise: Distributed shared objects for internet cooperative applications. In *IFTP International Conference on Distributed Systems, Platforms, and Open Distributed Processing Middleware98*, 1998.

[45] J. Holliday. Replicated database recovery using multicast communications. In *IEEE Symposium on Network Computing and Applications*, October 2001.

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

[47] P. Honeyman, L. Huston, J. Rees, and D. Bachmann. The little work project. In *Proceedings of the Third Workshop on Workstation Operating Systems.* IEEE, April 1992.

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

[49] J. Hunter and B. McLaughlin. Java document object model. http://www.jdom.org.

[50] C. K. I. Foster. A data grid reference architecture - draft of february 1, 2001. GriPhyN technical report 2001-12, GriPhyn, 2001.

[51] R. Jimenez-Peris, M. Patio-Martnez, B. Kemme, and G. Alonso. Improving the scalability of fault-tolerant database clusters. In *IEEE 22nd Int. Conf. on Distributed Computing Systems, ICDCS*, pages 477–484, July 2002.

[52] J. Kangasharju, J. Roberts, and K. Ross. Object replication strategies in content distribution networks. In *Computer Communications*, volume 25, pages 367–383. IEEE, March 2002.

[53] M. Karlsson, C. Karamanolis, and M. Mahalingam. A framework for evaluating replica placement algorithms. Tech. Rep. HPL-2002, HP Laboratories, July 2002.

[54] M. Karlsson and M. Mahalingam. Do we need replica placement algorithms in content delivery networks? In *Proceedings of the International Workshop on Web Content Caching and Distribution (WCW)*, August 2002.

[55] N. Kaushik and S. Figueira. Spanning trees for distributed search in p2p systems.

[56] H. Lamehamedi, Z. Shentu, B. Szymanski, and E. Deelman. Simulation of dynamic data replication strategies in data grids. In *Proc. 12th Heterogeneous Computing Workshop (HCW2003) Nice, France.* IEEE Computer Science Press, April 2003.

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

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

[59] B. Nance. File transfer on steroids. In *Byte Magazine.* February 1995.

[60] N. Narasimhan. *Transparent Fault Tolerance For Java Remote Method Invocation.* Ph.d dissertation, Department of Electrical and Computer Engineering, University of California, Santa Barbara, June 2001.

[61] T. Page, R. Guy, J. Heidemann, D. Ratner, P. Reiher, A. Goel, G. Kuenning, and G. Popek. Perspectives on optimistically replicated, peer-to-peer filing. Software - Practice and Experience, December 1997.

[62] G. Pierre, M. van Steen, and A. Tanenbaum. Dynamically selecting optimal distribution strategies for web documents. In *IEEE Transactions on Computers*, volume 51. IEEE Press, June 2002.

[63] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, and G. Thiel. Locus: A network transparent high reliability distributed system. In *Proceedings of the Eighth Symposium on Operating Systems Principles*, pages 169–177. ACM, 1981.

[64] K. Ranganathan and I. Foster. Design and evaluation of replication strategies for a high performance data grid. In *International Conference on Computing in High Energy and Nuclear Physics*, September 2001.

[65] K. Ranganathan and I. Foster. Identifying dynamic replication strategies for a high performance data grid. In *Proceedings of the International Grid Computing Workshop*, November 2001.

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

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

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

[69] D. Ratner, P. Reiher, and G. Popek. Roam: A scalable replication system for mobile computing. In *Workshop on Mobile Databases and Distributed Systems (MDDS)*, September 1999.

[70] D. H. Ratner. *Roam: A Scalable Replication System for Mobile and Disconnected Computing.* Phdthesis, University of California, Los Angeles, Los Angeles CA, January 1998.

[71] P. Reiher, J. Heidemann, D. Ratner, G. Skinner, and G. Popek. Resolving file conflicts in the ficus file system. In *Proceedings of the summer Usenix Conference*, 1994.

[72] J. Robinson and M. Devarakonda. Data cache management using frequency-based replacement. In *In Proc. SIGMETRICS*, 1990.

[73] L. Rodrigues, H. Miranda, R. Almeida, J. Martins, and P. Vicente. Strong replication in the globdata middleware. In *Proceedings Workshop on Dependable Middleware-Based Systems*, pages G96–G104, June 2002. (Suplemental Volume of the 2002 Dependable Systems and Networks Conference, DSN 2002).

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

[75] 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. In *Cluster Computing*, number 5(3), pages 297–304, 2002.

[76] Y. Saito and H. Levy. Optimistic replication for internet data services. In *Proc. of the 14th Intl. Conf. on Distributed Computing*, pages 297–314, October 2000.

[77] M. Satyanarayanan, J. Kister, P. Kumar, M. Okasaki, E. Siegel, and D. Steere. Coda: A highly available file system for a distributed workstation environment. In *IEEE Transactions on Computers*, volume 39, pages 447–459. IEEE, 1990.

[78] V. J. Sosa and L. Navarro. Influence of the document validation/replication methods on cooperative web proxy caching architectures. In *Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS)*, pages 238–245, January 2002.

[79] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney. File and object replication in data grids. In *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10).* IEEE Press, August 2001.

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

[81] A. S. Tanenbaum and M. van Steen. *Distributed Systems, Principles and Paradigms.* Prentice Hall, 2002.

[82] D. Terry, M. Theimer, K. Peterson, A. Demers, M. Spreitzer, and C. Hausen. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings of the fifteenth Symposium on Operating systems Principles*, pages 49–70. ACM, October 1983.

[83] B. Tierney, W. Johnston, L. Chen, H. Herzog, G. Hoo, G. Jin, and J. Lee. Distributed parallel data storage systems: A scalable approach to high-speed image servers. In *Proceedings of ACM Multimedia.* ACM Press, 1994.

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

[85] University of Southern California. *Virtual InterNetworks Testbed, VINT Project.*

[86] S. Vazhkudai, S. Tuecke, and I. Foster. Replica selection in the globus data grid. In *Proceedings of the First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID)*, pages 106–113. IEEE Computer Society Press, May 2001.

[87] S. Venugopal, R. Buyya, and K. Ramamohanarao. A taxonomy of data grids for distributed data sharing, management and processing. *ACM Computing Surveys*, 1(10), June 2005.

[88] 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 *In Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies*, page 139152, April 2004.

[89] R. W. Watson and R. A. Coyne. The parallel i/o architecture of the high-performance storage system (hpss). In *Proceedings of the IEEE MSS Symposium*, 1995.

[90] D. Wessels and K. Claffy. Application of internet cache protocol (icp), version 2. Internet Draft IETF, May 1997.

[91] M. Wiesmann. *Group Communications and Database Replication: Techniques, Issues and Performance*. PhD thesis, Ecole Polytechnique Federale de Lausanne, Switzerland, May 2002.

[92] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Database replication techniques: a three parameter classification. In *Proceedings of 19th IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2000.

[93] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS)*, 2000.