Dynamic and Scalable Storage Management in Grid Environment

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

By:
Ajay Kumar
(800831018)

Under the supervision of:
Dr. Seema Bawa
Professor, CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

July 2010

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Dynamic and Scalable Storage Management in Grid Environment"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Seema Bawa and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Ajay Kumar

(Ajay Kumar)
800831018

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Seema Bawa 10/07/2010

(Dr. SEEMA BAWA)
Professor
Computer Science and Engineering Department
Thapar University
Patiala, Punjab

Countersigned by

12/07/10

(Dr. RAJESH BHATIA)
Head
Computer Science & Engineering, Department
Thapar University
Patiala, Punjab

RK Sharma 12/7/10

(Dr. R. K. SHARMA)
Dean (Academic Affairs)
Thapar University,
Patiala., Punjab

Acknowledgement

The satisfaction and euphoria that accompany the successful completion of the thesis would be incomplete without the mention of the people who are responsible for the completion of this thesis work.

I consider it a privilege to express through the pages of this report, a few words of gratitude and respect to all those who guided and inspired me in completion of this thesis.

I sincerely acknowledge the guidance rendered to me by Dr. Seema Bawa, Professor, at Computer Science and Engineering Department, Thapar University, Patiala. Words are not enough to describe her invaluable support; inspiration, guidance, encouragement and making me understand easily anything, anytime during the thesis work.

I would like to thank to Dr. Rajesh Bhatia, Asst. Professor and Head of Computer Science and Engineering Department at, Thapar University, Patiala for co-operation in completing the thesis work.

I would like to thank to Dr. Maninder Singh, Professor, Thapar University, Patiala who really put me in a real technical frame and bent of mind – that allowed me to conceptualize and materialize this concept.

I am deeply indebted to our Sr. Grid Members (Ms Shashi, Ms Rajani, Ms Semu, Ms Pankajdeep Kaur, Ms Ratinder Kaur) who constantly encouraged me tender my utmost gratitude and appreciation for her invaluable guidance and suggestions.

Finally, I would like to Yuhui Deng professor at the Computer Science Department of Jinan University, China who gave some basic idea about Grid Storage system and give me permission to use my research papers.

With these words I share with the persons mentioned above the credit of my present humble achievement as well as hopeful success in future.

Abstract

Grid applications typically deal with large amounts of data. In traditional approaches high-performance dedicated servers are used for data storage and data replication. Data storage management is one of the most challenging issues for Grid resource management since large amount of data intensive applications frequently involve a high degree of data access locality. As computational grid focuses on the usage of idle resources from interconnected machines, a large amount of unused storage space can be used when the machines are idle or underutilized. This allows opportunistic grids to share not only the computational cycles, but also the storage space. In this thesis new mechanism for Dynamic and Scalable Storage Management (DSSM) in grid environments is proposed. The storage can be transparently accessed from any grid machine, allowing easy data sharing among grid users and applications. The concept of virtual ids that, allows the creation of virtual spaces has been introduced and used.

The DSSM divides all Grid Oriented Storage devices (nodes) into multiple geographically distributed domains and to facilitate the locality and simplify the intra-domain storage management. The DSSM architecture is validated by a proof-of-concept prototype system. Grid service based storage resources are adopted to stack simple modular service piece by piece as demand grows. To this end, we propose four axes that define: DSSM architecture and algorithms description, Storage resources and resource discovery into Grid service, Evaluate purpose prototype system, dynamically, scalability, and bandwidth, and Discuss results. Algorithms at bottom and upper level for standardization dynamic and scalable storage management, along with higher bandwidths have been designed. Findings with prototype infrastructure and application components, involving experiments coupling end-to-end resources for interactive storage management of large data in representative distributed environments have been reported.

Contents

List of Figures

List of Tables

Chapter 1
An Overview of Grid Computing

Grid environments are increasingly being used by applications such as particle physics, climate modeling, whether forecasting or astrophysics. These applications make use of unique, high-end supercomputers and storage-systems and produce large multi-dimensional data sets. This type of work is increasingly being performed in collaborative efforts between geographically distributed scientists and organizations, utilizing shared resources that are also widely distributed .The growth of data generated by information digitization have been identified as the key driver to escalate storage requirements. Network based storage systems such as Network Attached Storage (NAS) and Storage Area Network (SAN) offer a robust and easy method to control and access large amounts of storage. However, with the steady growth of client access demands and the required data sizes, it is a challenge to design an autonomous, dynamic, large-scale and scalable storage system which can consolidate distributed storage resources to satisfy both the bandwidth and storage capacity requirements [1].

A Grid is a collection of machines, sometimes referred to as "nodes", "resources", "members", "clients", "hosts", "engines", and many other terms. They all contribute any combination of resources to the Grid as a whole. Some resources may be used by all users of the Grid while others may have specific restrictions. Also, Grid computing is any distributed cluster or compute resources that provides an environment for the sharing and managing of the resource for the distribution of tasks based on configurable service-level policies.

1.1 Grid Fundamentals

Grid computing is the generic term given to techniques and technology design to make pools of distributed computer resources available on demand. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. The evolution in standard for sharing resources, along with higher bandwidths,

is driving in Grid computing. Also, a Grid computing environment enables sharing of loosely coupled resources and services required by various applications in a large-scale. In such an environment, one of the key challenges is to develop a flexible, scalable, and self-adaptive resource management system which would allow users to carry out their jobs by transparently accessing autonomous, distributed, and heterogeneous resources.

Grid computing was originally conceived by research scientists as a way of combining computer across a network to from a distributed supercomputer tackle complex problem or computational. In the commercial world Grid aims to maximize the utilization of an organization's computing resources by making them sharable across application's and potentially, provide computing on demand to the third parties as a utility service. Most important capabilities of Grid computing are [2]:

- Increasing the efficiency of resource usage
- Parallel capabilities
- Collaboration with virtual organization and virtual resources
- Better Resource balancing
- Better resource management
- Access to value added-services/resources.

1.1.1 Increasing the Efficiency of Resource Usage

The easiest use of Grid computing is to run an existing application on a different machine. The machine on which the application is normally run might be unusually busy due to an unusual peak in activity. The job in question could be run on an idle machine elsewhere on the Grid. There are at least two prerequisites for this scenario.

First, the application must be executable remotely and without undue overhead. Second, the remote machine must meet any special hardware, software, or resource requirements imposed by the application. For example, a batch job that spends a significant amount of time processing a set of input data to produce an output set is perhaps the most ideal and simple use for a Grid.

1.1.2 Parallel Capabilities

The potential for massive parallel CPU capacity is one of the most attractive features of a Grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the bio-medical field, financial modeling, oil exploration, motion picture animation, and many others.

The common attribute among such uses is that the applications have been written to use algorithms that can be partitioned into independently running parts. A CPU intensive Grid application can be thought of as many smaller "sub-jobs," each executing on a different machine in the Grid. To the extent that these sub-jobs do not need to communicate with each other, the more "scalable" the application becomes. A perfectly scalable application will, for example, finish 10 times faster if it uses 10 times the number of processors.

1.1.3 Collaboration with Virtual Organization and Virtual Resources

Another important Grid computing contribution is to enable and simplify collaboration among a wider audience. In the past, distributed computing promised this collaboration and achieved it to some extent. Grid computing takes these capabilities to an even wider audience, while offering important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of virtual resources. The users of the Grid can be organized dynamically into a number of virtual organizations, each with different policy requirements. These virtual organizations can share their resources collectively as a larger Grid. The participants and users of the Grid can be members of several real and virtual organizations.

1.1.4 Better Resource Balancing

The Grid can offer a resource balancing effect by scheduling Grid jobs on machines with low utilization. This feature can prove invaluable for handling occasional peak loads of activity in parts of a larger organization. This can happen in two ways:

- An unexpected peak can be routed to relatively idle machines in the Grid.
- If the Grid is already fully utilized, the lowest priority work being performed on the Grid can be temporarily suspended or even cancelled and performed again later to make room for the higher priority work.

Finally, a Grid provides excellent infrastructure for brokering resources. Individual resources can e profiled to determine their availability and their capacity, and this can be factored into scheduling on the Grid. Different organizations participating in the Grid can build up Grid credits and use them at times when they need additional resources. This can form the basis for Grid accounting and the ability to more fairly distribute work on the Grid.

1.1.5 Better Resource Management

The goal to virtualize the resources on the Grid and more uniformly handle heterogeneous systems will create new opportunities to better manage a larger, more disperse IT infrastructure. It will be easier to visualize capacity and utilization, making it easier for IT departments to control expenditures for computing resources over a larger organization. The Grid offers management of priorities among different projects. In the past, each project may have been responsible for its own IT resource hardware and the expenses associated with it. Often this hardware might be underutilized while another project finds itself in trouble, needing more resources due to unexpected events. With the larger view a Grid can offer, it becomes easier to control and manage such situations. Aggregating utilization data over a larger set of projects can enhance an organization's ability to project future upgrade needs. When maintenance is required, Grid work can be rerouted to other machines without crippling the projects involved.

1.1.6 Access to Value Aided-Services/Resources

A Grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services. The additional resources can be provided in additional numbers and/or capacity. For example, if a user needs to increase

his total bandwidth to the Internet to implement a data mining search engine, the work can be split among Grid machines that have independent connections to the Internet. In this way, the total searching capability is multiplied, since each machine has a separate connection to the Internet. If the machines had shared the connection to the Internet, there would not have been an effective increase in bandwidth. The Grid can enable more elaborate access, potentially to remote medical diagnostic and robotic surgery tools with two-way interaction from a distance. The variations are limited only by one's imagination.

Today, we have remote device drivers for printers. Eventually, we will see standards for Grid enabled device drivers to many unusual devices and resources. All of these will make the Grid look like a large virtual machine with a collection of virtual resources beyond what would be available on just one conventional machine.

1.2 Types of Grid

A Grid fundamentally consists of two parts [2]. These are:

- **Compute Grid**- provides the core resource and task management services for Grid computing: sharing, management, and distribution of task based on configurable service-level policies.
- **Data Grid** – provides the data management features to enable data access, synchronization, and distribution of a Grid

Grid research defines some other type of Grids. These are:

- Semantic Grid
- Knowledge Grid
- Service Grid

1.3 Baseline of Data Grid: Terms and Functions

There are many analogies in the development and adoption of Grid computing to those of client/server technology. Both are fundamental paradigm shifts in the way of computing

is performed. As client/server technology ushered in the broad acceptance of relational database technology, grid technology will usher in new data management paradigms to address the specific topology of the physical compute grid. To see how this is happening, it is best to untangle the concepts of data management in Grid from by drawing on a fundamental baseline that we are all familiar with. The people who are going to use Grid technology-developers, architects, and lines of business-are accustomed features within a client/server paradigm.

Figure 1.1 illustrates the parallel of the vocabulary and fundamentals between data management within relational databases and that within grid computing. This comparison is useful in two aspects [5]:

- It relates to terms that most are already vary familiar with and
- More importantly, it suggests that any data management system in grid computing must provide the same levels of service quality as within relational databases.

1.3.1 Relational Database Management

Relational database implementations have two fundamental components:

(1) The underlying engine that manages physical resources, in this case a disk and

(2) A layer on top of that to provide all the data management features and functionality that architects and developers would rely on for data management, querying, arrangement of data in highly ordered structures such as structures such as tables, the ability to transact on data, leveraging stored procedures, event triggering, and transacting in and out of the database with external systems.

These are the management features and functions that today are where our true interest lies. One of the data management policies within the data Grid is synchronization. There are two types of synchronization: intraregion and interresion. These respective synchronization types identify data synchronization within a data region with other data sources either external or internal to the region.

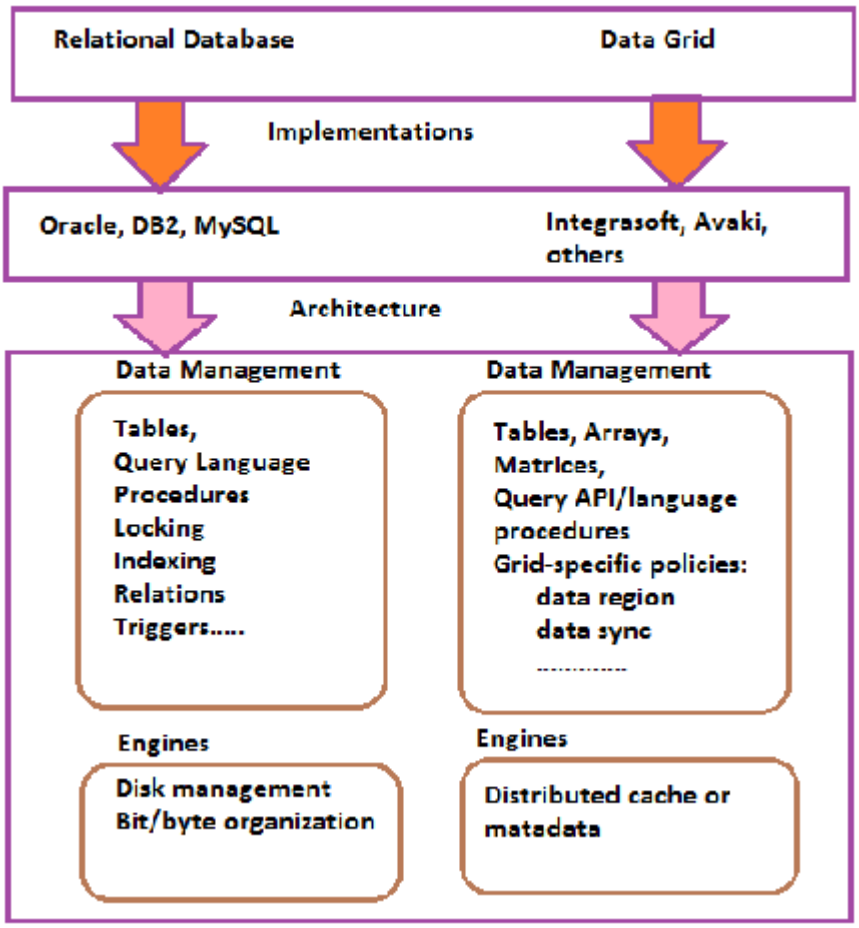

Figure -1.1: Baseline of terms and functions [5]

1.3.2 Data Grid

In same way that relational database is a generic term, so is data grid. Companies will offer implementations, products of their vision of what a data grid is. To analyze the differences between the products offers, it is possible to apply a baseline consisting of generic term, implementation, data management, and engine. That engine may be a metadata dictionary or a distributed cache. It will also handle the data management aspects of this data grid, defining how to structure data in tables, arrays, or, matrices; how to query data; and how to transact on the data. Depending on the exact implementation of this engine- whether it is metadata dictionary that routes requests to the true long-term persistent stores, or a distributed cache that spans all computers in the grid to form one virtual space-there are specific data management issues for t is new topology. How to

synchronize, how to transact on the data, how to address data affinity? These are all data management issues; issues that, no matter which the architect or application developer is, will need to be addressed within their applications.

Data Grid support for true data management extends to facilitation of the adoption and wide scale acceptance of Grid technology.

1.4 Grid Architecture

A Grid architecture that identifies fundamental system components, specifies the purpose and function of these components, and indicates how these components interact with one another. The Grid architecture is often described in terms of "layers", each layer providing a specific function. In general terms, the higher layer is user-centric, whereas the lower layers are more hardware-centric [2]. The layers of Grid architecture are:

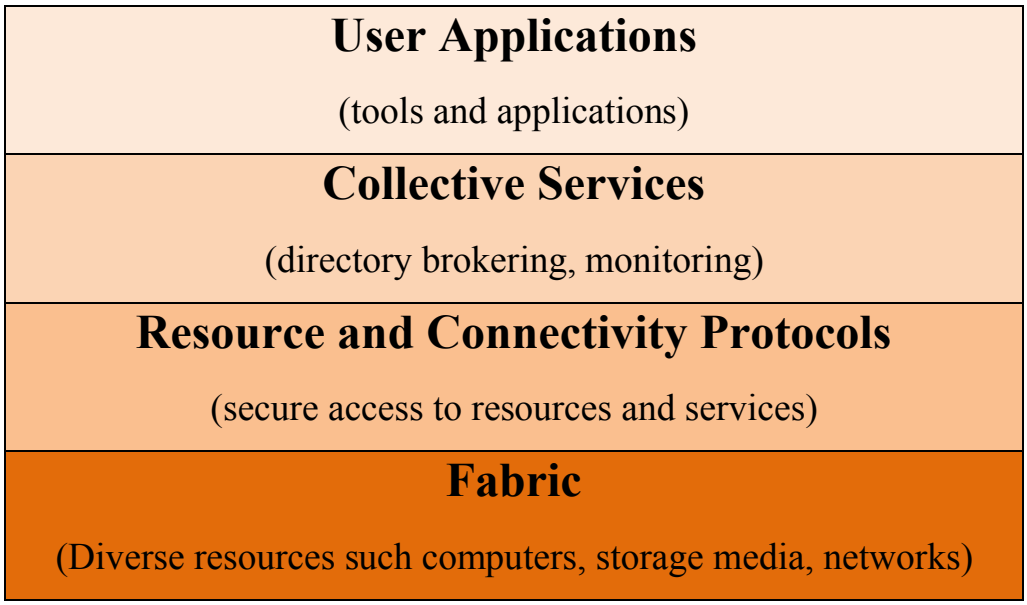

Figure -1.2: Different layers of Grid [2]

1.4.1 Fabric

The Grid fabric layer provides the resources to which shared access is mediated by Grid protocols, for example, computational resources, storage systems, catalogs, network

resources. Fabric components implement the local, resource-specific operations that occur on specific resources (whether physical or logical) as a result of sharing operations at higher levels. There is thus a tight and subtle interdependence between the functions implemented at the fabric level, on the one hand, and the sharing operations supported, on the other.

1.4.2 Resource and Connectivity Protocols

The connectivity layer defines core communication and authentication protocols required for Grid specific network transactions. Communication protocols required for Grid-specific network transactions. Communication protocols enable the exchange of data between fabric layer resources. Authentication protocols build on communication services to provide cryptographically secure mechanisms for verifying the identity of users and resources. Communication requirement include transport, routing, and naming. Although alternatives certainly exist, it is common to assume that these protocols are drawn from the TCP/IP protocol stack: specifically, the Internet (IP and ICMP), transport (TCP, UDP), and application (DNS, OSPF, RSVP, etc.) layers of the Internet – layered protocol architecture.

The complexity of the security problem makes it important that any connectivity layer security solutions be based on existing standards whenever possible. As with communication, many security standards developed within the context of the Internet protocol suite are applicable. Grid securities also provide flexible support for communication protection and enable stakeholder control over authorization decisions, including the ability to restrict the delegation of rights in various ways.

1.4.3 Collective Services

The collective layer coordinates multiple resources within Grid environment. This layer contains protocols and services not only for any one specific resource but instead capturing interactions across collections of resources. Some examples are:

- Directory services

- Monitoring and diagnostics services
- Data replication services
- Co-allocation, scheduling and brokering services
- Collaborating services

1.4.4 Applications

The final layer of Grid architecture which includes all different user applications (science, engineering, and business, financial), portals, and development toolkits supporting the applications those operate within a VO environment. Applications are constructed in term of, and by calling upon, services defined at any layer. This is the layer that users of the Grid will "see" and most of the time interacts through their browser.

1.5 Grid Topology

The Grid topology is a key factor as it determines how to manage it, to schedule jobs, to secure transactions. The simplest Grid consists of just a few machines on a local network, all of the same hardware architecture; then this model can be expended to a local areas network with heterogeneous systems; finally, the most complicated Grid with the Internet as the communication network and heterogeneous end systems.
A topology view covers the following spectrum of Grids:

Table-1.1: Classification of Grid topology [2]

Intra-Grids	Extra-Grids	Inter-Grids
• Single organizations • No partner integration • A single cluster	• Multiple organizations • Partner integration • Multiple clusters	• Many organizations • Multiple partners • Many multiple clusters

1.5.1 Intra-Grids

IntraGrid exists within a single organization, providing a basic set of Grid services. The single organization could e made up of a number of computers that share a common security domain, and share data internally on a private network. The main characteristics of an intraGrid are a single security provider, bandwidth on the private network is high and always available, and there is a single environment within a single network. It is easier to design and operate computational and data Grids and provides a relatively static set of computing resources and the intraGrid appropriate if the business has an initiative to gain economies of scale on internal job management. This Grid is also called as a "Software Grid" where resource sharing is fairly simple.

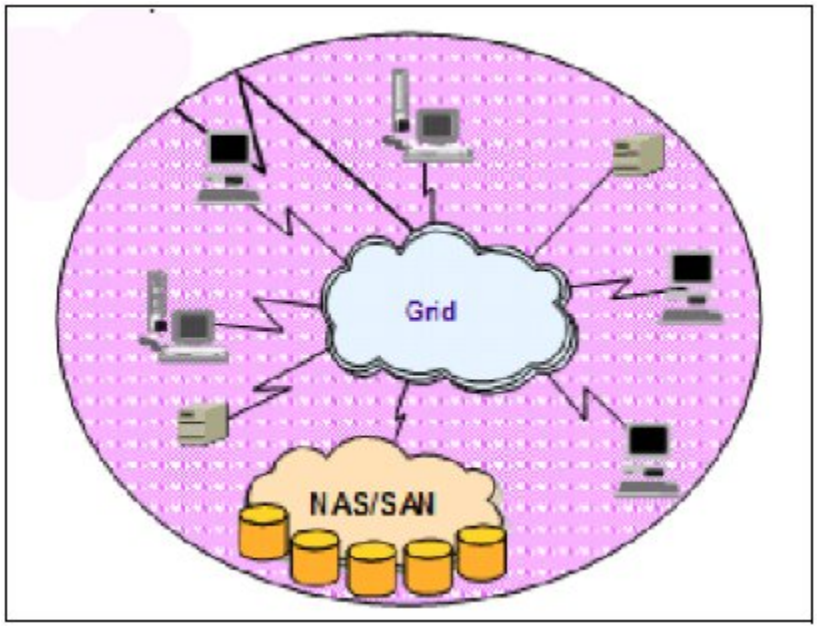

Figure-1.3: IntraGrid with set of Services [2]

1.5.2 ExtraGrid

The extraGrid expands on the concepts by bringing together two or more intraGrids. Here typically involves more than one security provider, and the level of management complexity increases. Multiple organizations, dispersed security, and remote/WAN connectivity are primary characteristics of Extra-Grid.

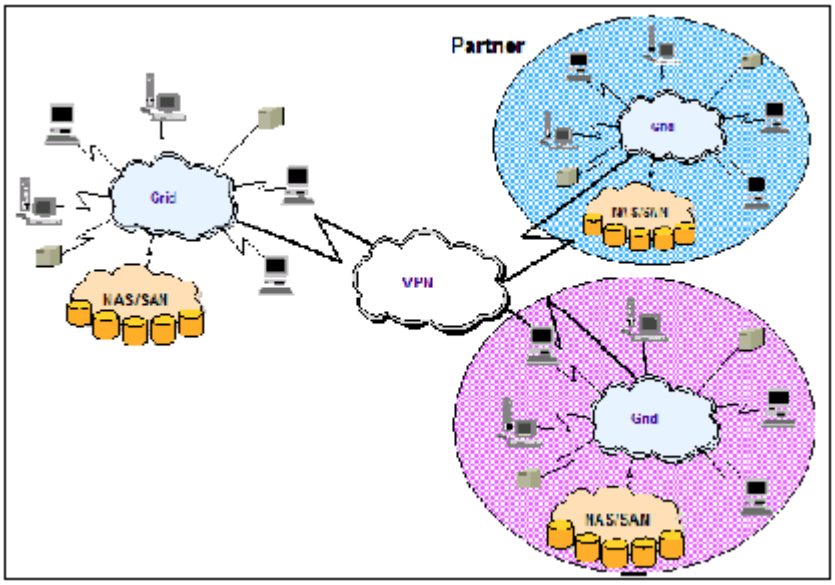

Figure-1.4: ExtraGrid with Multiple Organizations [2]

The design becomes more complemented and information service becomes relevant to ensure that Grid resources have access to workload management at run time.

1.5.3 InterGrid

An interGrid concern with dynamic integration of applications, resources, and services with patterns, customers, and any other authorized organizations that will obtain access to the Grid via the internet/WAN. An interGrid mainly used by engineering firms, life science industries, manufacturers, and by business in the financial industry. The main characteristics on an interGrid include dispersed security, multiple organizations, and remote connectivity.

1.6 Relationships with other technologies

There are many technologies and architectures that precede Grid computing in terms of attempting to “merge” computing power, share storage, and facilitate program-to-program communication. Comparison of these architectures is presented below.

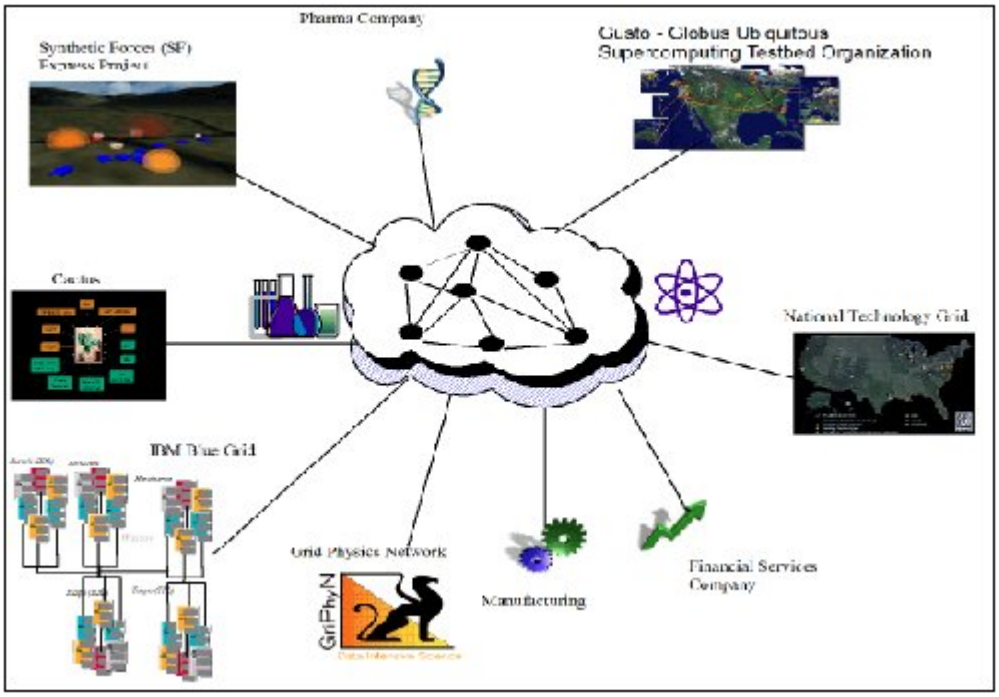

Figure-1.5: InterGrid with global Organizations [2]

1.6.1 Grid vs. Web

Some of the most significant differences between the web and the Grid are the following:

- While both are operated on the Internet they are currently driven by different needs and interests: Grids by e-science, web by e-commerce.
- While there is only one web, there are many separate and distinct Grids.
- Grids have fairly limited user communities by contrast the web addresses and potentially serves millions of people, i.e. the general public.
- Anyone can "join" the web; a Grid applies authentication and authorization mechanisms for accessing the Grid.
- Grids target processing and computations involving huge to gigantic datasets, whereas data volumes flowing across the web are far more modest.
- There is no inherently reliable and secure data transmission o the web reliability and security is key design issues for a Grid.

1.6.2 Grid vs. Distributed Computing

The different between Grid computing and traditional distributed computing lies in computation size, scope, heterogeneity and communication. Grid computing is distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and in some cases high performance orientation.

1.6.3 Grid vs. Cluster Computing

As clusters and Grids both operate on the same underlying principle that a group of computers acts as one but there are certain differences

- In clusters, a centralized resource manager performs the resource allocation and all nodes work together cooperatively as a single unified resource. Within a Grid, each node has its own resources manager and provision of a single system view is not a goal.
- While a Grid consists of heterogeneous resources, cluster computing is primarily concerned with homogeneous computational resources.
- Grid computing integrates storage, networking and computation resources. Clusters usually contain a single type of processor and operating system.
- Clusters typically contain a static number of processor s and resources. Resources are provided and removed from the Grid on an ongoing basis.

1.7 Thesis Outline

This thesis organized as following:

Chapter 1: An Overview of Grid Computing

Describe the fundamental ideas about Grid computing such as, Grid concepts, virtual organizations and the Grid, Grid classification, Grid architecture, relationships with other technologies and benefits.

Chapter 2: Data Management in Grid Computing

Explain the overview of data management, scaling in the Grid topology, traditional data management and some application characteristics for Grid.

Chapter 3: Literature Review

This chapter focuses our literature survey on data storage management in Grid environment. Mainly we explain different types of methods, tools, architecture that has been using in Grid environments. Finally, we formulate the problems also.

Chapter 4: Proposed DSSM Architecture

This chapter describes our proposed architecture for dynamic and scalable storage management in Grid environment that is known as DSSM architecture. Also explain the virtualization concepts in Grid environments in details

Chapter 5: Implementation and Experimental Evaluation

Discusses about the implementation of DSSM architecture. It also discuses the implementation environments, tools (such as, hardware, software, protocols etc.) and process of the experiment. At the end of this chapter we explain the experimental results.

Chapter 6: Conclusion and Future works

This is final chapter of our thesis that presents the conclusion of the thesis, describes the main contribution of the thesis and highlights future research direction.

1.8 Motivation

The motivation behind Grid Computing is coordinated resource sharing and problem solving in dynamic, multi-institutional Virtual Organizations. Grid is a robust, efficient, secure, scalable, coordinated resource sharing among dynamic collections of individual client, institutions, and resources. The objective is to provide resources and allow users and applications to access resources in a transparent manner. Storage Grid is a new model for deploying and managing storage resources geographically distributed

across multiple systems and networks, making efficient use of available storage capacity. Storage Resource Broker (SRB) [3] supports shared collections that can be distributed across multiple organizations and heterogeneous storage systems. The SRB offers some Grid functionalities including a virtual organization structure for data and information, handling a multiplicity of platforms, resource and data types, and seamless authorization and authentication to data and information which are stored in distributed sites, etc. Distributed Parallel Storage System (DPSS) [6] provides high speed parallel access to remote data and the DPSS servers are transparent to applications. It is currently being integrated into Globus and the SRB. High Performance Storage System (HPSS) [7] provides hierarchical storage management and services for very large storage environments. Storage elements (magnetic disk and tape) continue to improve in both capability and price-performance at an exponential rate. The rapid decrease in the cost of storage has significant implications for Grids. Not only do technology improvements make it possible and cost-effective to store large quantities of data, but the exploding storage capability in lowest-cost systems dictates that huge amounts of storage will be available on network of any size, and improving capabilities of high-speed networks will enable Grids to use that storage flexibly.

Chapter 2
Data Management in Grid Computing

In this chapter we will discuss the various levels of data management within the Grid and its enabling role in allowing the Grid topology to scale to its full potential. We will start with the evolution of the data management within the Grid and investigate the various implementations of the data Grids that are in place today as well as those currently emerging.

2.1 Scaling in the Grid Topology

In this section we will start with the evolution of the data management within the Grid and investigate the various implementations of the data Grids that are in place today as well as those currently emerging.

2.1.1 Evolution in Data Management

The evolution of computer science can be charted by the growth of the data management techniques and products. Initially, the mainframe and mini-computers utilized APL, network databases and hierarchical architecture. Today, client/servers are heavily in use and rely on both relational and object databases. The emerging technology for distributed computing relies on in-memory as well as relational databases. For our discussions, we are choosing the client/server architecture as a baseline for establishing an understanding to distributed computing because of its widespread use in industry. This allows us to draw on the common knowledge among managers, architects, and developers, and to leverage that knowledge so as to describe the data management needs within the Grid topology.

2.1.1.1 Client/Server Evolution

In the late 1980s and early 1990s, client/server technology began to gain wide acceptance. This change in compute topology, a change from mainframes and

minicomputers to a more distributed environment-where the separation of a program functions was made between “service” as a separate execution program, a “server” and the business “client” – started to usher in the need for relational data management systems. Relational database technology matured during the 1970s and 1980s, with the emergence of such companies as Oracle, Sybase, and IBM introducing the concept of relational database technology.

Issue such as how to structure the database both to support the business data and gain maximum performance began to receive more attention, while file and disk management lost focus. Specialist in relational data management quickly focused on system performance optimization, and how to provide the proper tools and methods to operate the system.

2.1.1.2 Grid Evolution

Data management within Grid computing has evolved with the applications of Grid technology. The earliest applications were complex analysis, often traversing large data sets. Data management of the static data sets is adequately handled via common data storage techniques such as file systems. The problem to resolve is how to take the data stored in files, typically large in size, and move that data to the nodes in the Grid that require the data in order to perform their designated tasks. Most common methods used today are File Transfer Protocol (FTP) and distributed file systems. Since these are the first generation data management tools for the Grid – and as such address the movement of a subset of data type, primarily data sets that are static in nature – they are considered a level 0 data Grid. As the use of Grid computing expands to support increasing fields of work and businesses, dynamic data sets become a more common problem that needs to be solved. Therefore, the level 0 data Grids, which work well for static data, will not work and scale for data sets that exhibit dynamic behavioral properties.

New data management techniques and infrastructures are required to address the dynamic data sets within the Grid environment. These level 1 data Grids, unlike the level 0 data Grid, take into account both static and dynamic data sets and address the unique data management issues posed when data are to be managed in the highly distributed compute environments of the Grid.

2.2 Different Implementations of Data Grids

2.2.1 Level 0 Data Grids

Level 0 data Grids were the earliest to address data requirements in a Grid topology. Their main function is the distribution of large, static data sets to the nodes in the Grid. They do not address data management issues such as updates, transactions, or integration with external systems.

FTP in Grid: The File Transfer Protocol (FTP) is one of the most widely used protocols for the movement of files across a network. It is amazing that it remains in heavy use even in today's technology advance society. Therefore, it is an obvious choice for data movement within a Grid environment. The standards, body Globus is investigating the use of data transfer protocol for a data Grid implementation, termed as GridFTP.

GridFTP, as a protocol engine, is a good choice for static data sets. It allows data to be transferred between nodes but does not address data management beyond data movement. Since it is based on the common used FTP, it is a common protocol that is well and work well. The physical architecture of the FTP protocol requires a central server to manage the FTP requests and the movement of data. Therefore, if a machine is to make public its files through the FTP protocol, that machine must be running a FTP server; thus, any machine requesting data must be a FTP client.

Distributed Filing Systems: Similar to GridFTP, a distributed file system is a way to manage the distribution of files across a network. From the user's perspective, a distributed file system is a common filing system with a directory tree structure. The architecture of distribute file system is server based; a server manages the mapping of the disparate physical file systems into a logical, unified file system. As the user of the distributed file system, the client in this case, traverses the tree structure, client requests are made to the server, which in turn negotiates with the true sources of data and pulls the data back to the client in the common directory tree view.

Faster servers: One of the fastest paths to maintaining quality of service for true data management within the Grid is to leverage the traditional data management techniques of client/server by leveraging existing relational databases and interfacing them into the Grid. This works well for smaller Grid implementations, where the numbers of CPUs are

relatively low. However, scaling becomes an issue as the size of the Grid increases, thus increasing in the number of CPUs. It is important the Grid nodes are required to know that they must connect to the database in order to query and transact on the data.

2.2.2 Level 1 Data Grid

Level 1 data Grids support data sets that are dynamic in nature: data sets that change daily. Hourly, minute-to-minute, second-to-second, or at any other intervals. Level 1data Grids addresses the distribution of and the ready access to data across the many nodes of the compute Grid. Measure things are in this level:

- Access method
- Management method
- Transactional method
- Synchronization method

Foundations: There are three basic technologies for providing level-1 data Grid: *JavaSpace, Global replication of data across the Grid,* and *Distributed Memory*.

Distribute memory: Distributed and shared memory is an effective mechanism for building a data grid. OpenMP designed for the splitting of large processing loops into smaller bits of work in a multithreaded, multiprocessor environment. It also creates a distributed memory space to eliminate the use of traditional network communication methods and middleware to allow the threads to share data.

2.3 Application Characteristics for Grid

The data Grid plane, which originated as a mechanism for distributing files to compute servers, is undergoing a substantial evolution. The evolution is centered on the needs of next-generation Grid applications, which include integrity and quality of service for both static and real-time or non-static data. Figure 2.1 shows the current evolution of the data Grid plane demands of an application.

There are numerous factors describing how the interdependencies of application data requirements, data integrity, and quality of service (QoS), data dynamics, and application

events interact with each other at various levels depending on the complexity of the application. I have defined some of the characteristics of an application with respect to the selection requirements.

The issue to resolve is how best to represent the parameters describing an application's properties and correlate them to the policies for data management in Grid computing. The use of tables, matrices, and graphs is not sufficient to visualize the various interdependencies, due to the multidimensional relationships that quickly from between the various parameters of work, data, time, geographic boundaries, and complexity of data analysis. The expression of an application's properties in mathematical notation is presented below.

Application (Work( ), Data( ), Time( ), Geography( ), Query( ))

Where the following are the driven parameters of the functions:

Work(batch/atomic, synchronous/nonsynchronous)
Data(overallsize, atomize, transactional, transient, queryable)
Time(real-time, not-real-time, near-real-time)
Geography(topology, network-bandwidth)
Query (basic, complex)

These methods provide a way for us to clearly and accurately define functional areas of an application and the parameterized arguments composing each part of the application. The definition for an application in a distributed environment can be defined as a function driven by of other function as indicated above.

2.4 Traditional Data Management

Electronic data management has a long and rich history dating back to the 1950s many data management systems have tried to make their way into the mainstream of information technology, some more successfully than others; hierarchical, network,

object, and in-memory are only a few examples. The most successful data management system has been the relational data management technology. I will start at this point to look at its development and what has made it succeed and also how far forward it has moved in comparison to any other data management system.

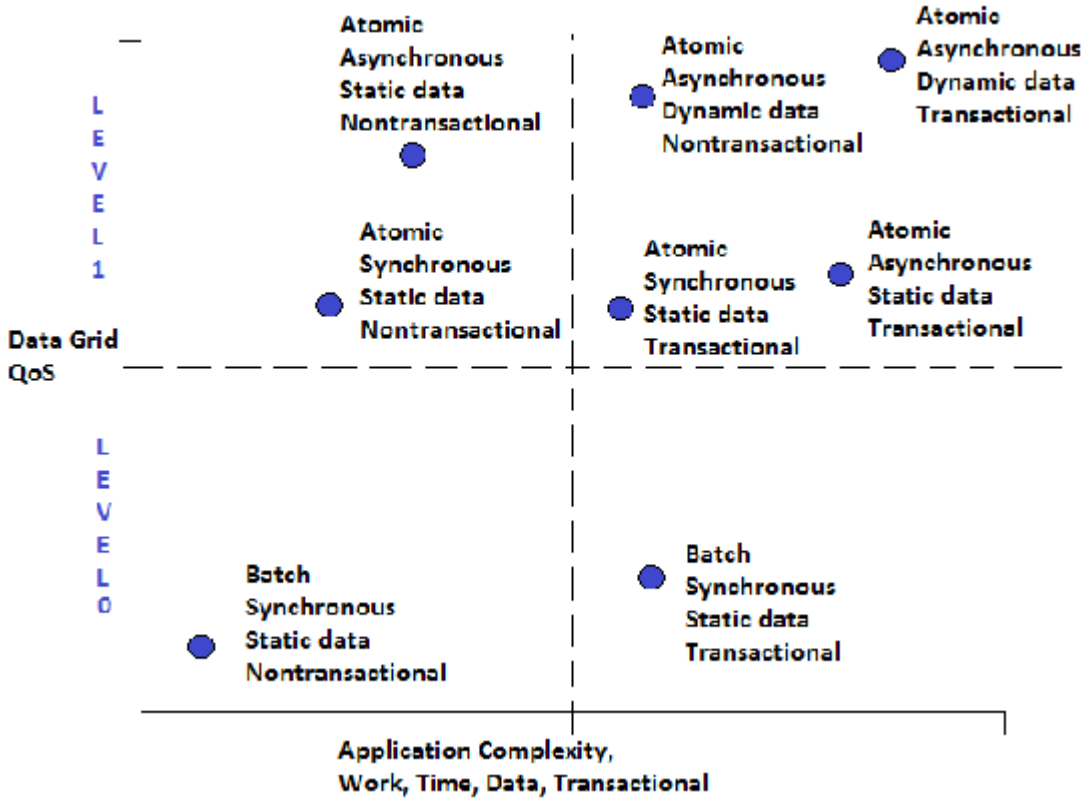

Figure 2.1: Data Grid Quality of Service vs. Application Demand and Requirement [5]

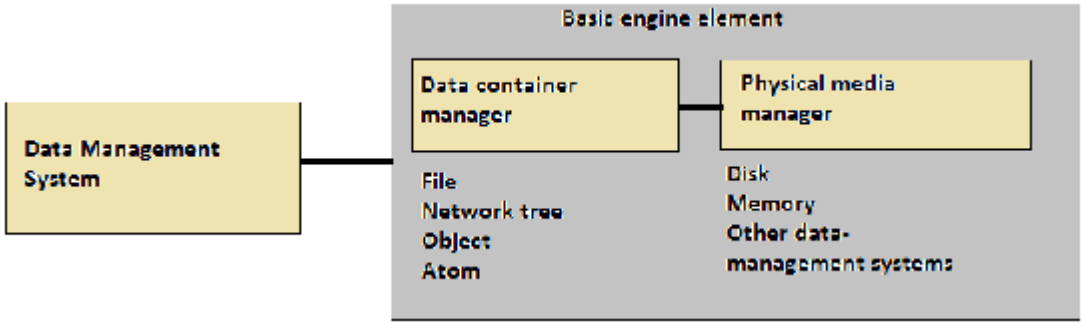

Figure 2.2: Data Management System and its Engine [5]

History

Dr. E. F. Codd, a researcher at IBM, defined the relational model in a 1970 paper entitled "A Relational Model of Data for Large Shared Data Banks" which initiated a chain reaction of research into the concepts, both internally at IBM and everywhere else. The research timeline included the following:

- 1974 – The System/R project gave birth to the Structural English Query Language (SEQUEL).
- 176 – 1977 – SEQUEL is extended to support multiple users, tables, and so on and later eventually renamed Structured Query Language (SQL).
- 1979 – Oracle is launched.
- 1983 – IBM introduces DB2.
- Other products are introduced to the marketplace during this same period, including Sybase.

Features

Some of the features of a data management system that are essential from the user perspective are:

- The mechanics or engine of the data management system
- Data integrity
- Transactional support
- Support for external events
- Backup, recovery, and availability of the data management system
- Security

Chapter 3
Literature Review

This chapter discusses all the research work done so far regarding Storage Management in Grid Environments. It examines the existing Storage Management System, their attributes, Storage Grid Architecture, and existing Grid Storage Middleware. Comparative study and analysis of existing Grid Middleware (Storage Resource Managers) has also been done for Grid storage management system. The Network Attached Storage Architecture has been discussed. Contemporary Grid Storage Devices have been discussed and compared. The related technologies and their impact on Grid Computing have also been discussed. The last section highlights the gaps in research and development work as identified during literature review. These gaps form the basis of problem formulation for this work.

3.1 Storage Resource Managers (SRMs): A Middleware Component for Grid Storage

The amount of scientific data generated by simulations or collected from large scale experiments have reached levels that cannot be stored in the researcher's workstation or even in his/her local computer centre. Such data are vital to large scientific collaborations dispersed over wide-area networks. In the past, the concepts of a Grid infrastructure mainly emphasized the computational aspect of supporting large distributed computational tasks, and managing the sharing of the network bandwidth by using bandwidth reservation techniques. The concept of Storage Resource Managers (SRMs) [32] support for the storage management of large distributed datasets. The access to data is becoming the main bottleneck in such "data intensive" applications because the data cannot be replicated in all sites. SRMs were designed to dynamically optimize the use of storage resources to help unclog this bottleneck.

The term "storage resource" refers to any storage system that can be shared by multiple clients. We use the term "client" here to refer to a user or a software program that runs on behalf of a user. Storage Resource Managers (SRMs) are middleware software modules

whose purpose is to manage in a dynamic fashion what resides on the storage resource at any one time. SRM do not perform file movement operations, but rather interact with operating systems, mass storage systems (MSSs) to perform file archiving and file staging, and invoke middleware components (such as GridFTP) to perform file transfer operations. There are several types of SRMs: Disk Resource Managers (DRMs), Tape Resource Managers (TRMs), and Hierarchical Resource Managers (HRMs). Unlike a storage system that allocates space to users in a static fashion, SRMs are design to allocate and reuse space dynamically. This is essential for the dynamic nature of shared resources on a Grid.

3.1.1 The role of SRMs in a Grid Environment

Suppose that a client runs an analysis program at some site and wishes to get data stored in files located in various sites on the Grid. First, the client must have some way of determining which files it needs to access. Checking a file catalog, using some index, or using a database system containing information about the file can accomplish this step. The information used in this step is often referred to as a metadata catalog. The result of this step is a set of logical file names that need to be accessed. The second step is to find out for each logical file where it physically resides or replicated. A single logical file can be replicated in multiple sites also. In a Grid environment, the information on the locations of replicated files exists in a replica catalog, a catalog that maps a single logical file name to multiple site-specific files. The site specific file name includes the mane to multiple site-specific files. The site-specific file name includes the name a machine and possibly port at the site, the directory path on that system, and the file name.

In many Grid environments today, the burden for the above work is being thrust on the clients. Therefore, it is now recognized that such tasks can delegated to middleware components to provide these services. A request manager is the term used to refer to such services. The request manager performs request planning based on some strategy, AIT then a request execution of the plan. This terminology is used by several Grid projects, notably PPDG, GriPhy and ESG. There are three options to consider for request planning: either moves the client's program and the data to another site for processing.

All three possibilities are valid, and much of the data middleware development addresses this issue. In all these cases, SRMs play an important role.

An analysis scenario where the computation is performed at the client's site and the needed files are in other sites on the Grid. This is common spatial case of Grid resource usage in many scientific communities. The Figure 3.1 has shown an analysis scenario diagram.

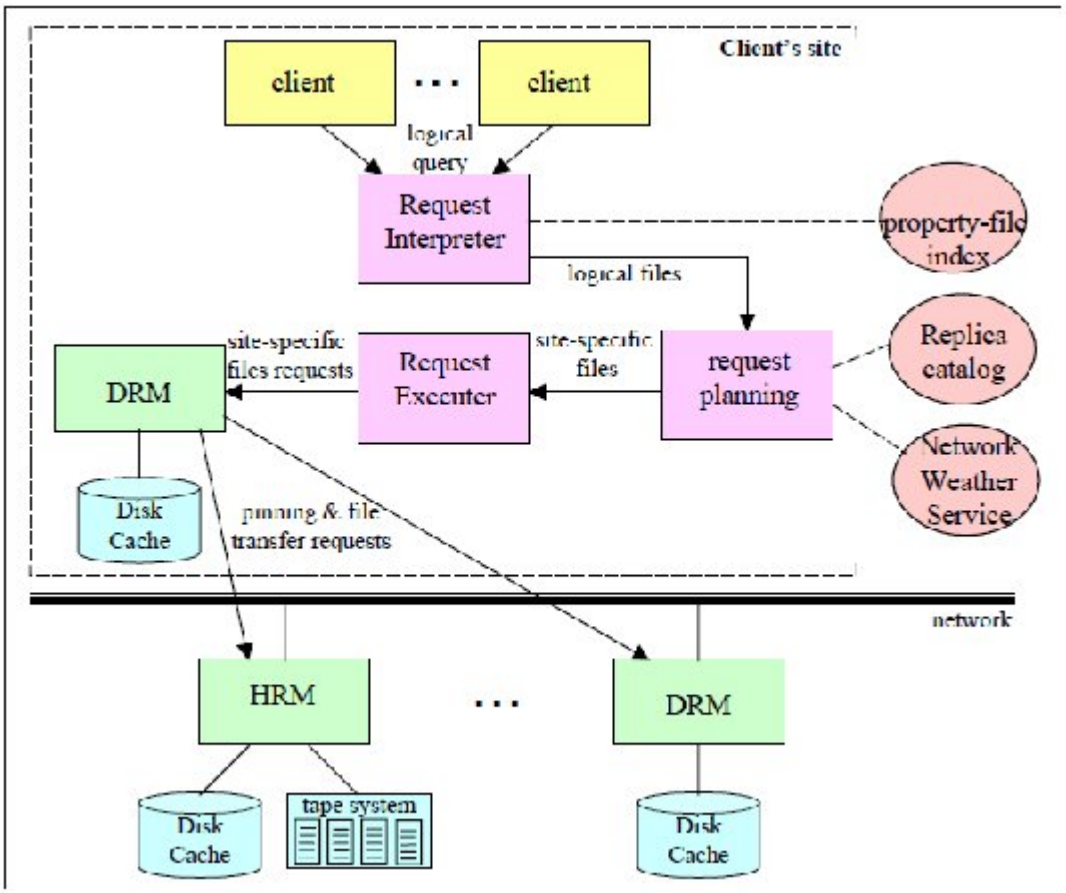

Figure 3.1: A Schematic Diagram of an Analysis Scenario [32]

3.1.2 The Implementation of the Analysis Scenario

From a client's point of view the system accepts a logical query request, and takes care of all the details of figuring out what files should be transferred, and where to get them from. The client can observe in a graphical display the progress of file transfers over

time. When a file that arrives is processed and released by the client, it may be removed DRM if it needs to make space for additional files.

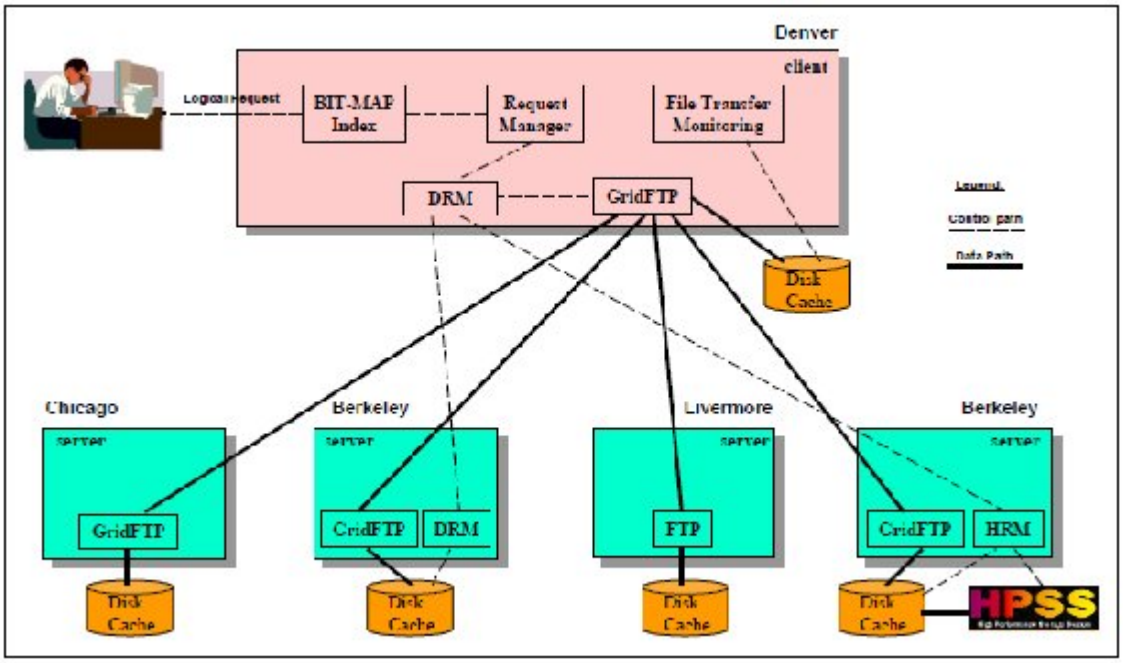

Figure 3.2: A Setup for Processing Logical Analysis Requests over the Grid [32]

3.1.3 Advantages and Limitations of using SRMs

There are following advantages and limitations of using SRMs:

Advantages:

- Eliminate unnecessary burden from the client.
- An advantage to the clients is that SRMs can insulate them from storage systems failures.
- SRMs can transparently deal with network failures and also monitor file transfers, and if failures occur, re-try the request.
- Enhance the efficiency of the Grid, eliminating unnecessary file transfers by sharing files.

Limitations:

- Does not achieve dynamically storage management

- Lack of scalability
- Lack of proper disk utilization
- Difficult to practical implementation

Finally we observed from this paper the concepts of Storage Resource Managers (SRMs), and argued that they have an important role in streamlining Grid functionality and making it possible for storage resources to be managed statically. SRMs make it possible to manage the storage resource based on the actual access patterns.

3.2 Network Attached Storage (NAS) System

Network Attached Storage (NAS) [33] has been gaining general acceptance, because it can be managed easily and files shared among many clients, which run different operating systems. The advent of Gigabit Ethernet and high speed transport protocols further facilitates the wide adoption of NAS. A distinct feature of NAS is that NAS involves both network I/O and file I/O. This paper analyzes the layered architecture of a typical NAS and the data flow, which travels through the layers. Several benchmarks are employed to explore the overhead involved in the layered NAS architecture and to identify system bottlenecks. The test results indicate that a Gigabit network is the system bottleneck due to the performance disparity between the storage stack and the network stack. The tests also demonstrate that the performance of NAS has lagged far behind that of the local storage subsystem, and the CPU utilization is not as high as imagined.

NAS adopts a Gigabit network: (1) the most effective method to alleviate the network bottleneck is increasing the physical network bandwidth or improving the utilization of network. For example, a more efficient network file system could boost the NAS performance. (2) It is unnecessary to employ specific hardware to increase the performance of the storage subsystem or the efficiency of the network stack because the hardware cannot contribute to the overall performance improvement. On the contrary, the hardware methods could have side effect on the throughput due to the small file accesses in NAS. (3) Adding more disk drives to an NAS when the aggregate performance reaches the saturation point can only contribute to storage capacity, but not performance.

The hierarchy of storage in current computer architectures is designed to take advantage of data access locality to improve overall performance. Each level of the hierarchy has higher speed, lower latency, and smaller size than lower levels. For decades, the hierarchical arrangement has suffered from significant bandwidth, latency, and cost gaps between the RAM and disk drive (Pugh, 1971). The performance gap has been widened to six orders of magnitude in 2000 and continues to widen by about 50% per year (Schlosser et al., 2000). The disk I/O subsystem is repeatedly identified as a major bottleneck to system performance in many computing systems. The bottleneck of disk I/O could significantly impact the application performance. NAS employs RAID subsystems, which adopt multiple disk drives working in parallel to achieve high performance, thus alleviating or even eliminating the I/O bottleneck. The performance of the storage subsystem scales with the number of disk drives. The performance bottleneck of a NAS could be migrated from disk I/O to other components with the increase in number of disk drives.

An NAS is a file server that normally presents a file interface to the network by employing Windows Common Internet Files System (CIFS) or Network File System (NFS). A typical NAS connects to a LAN and provides file sharing services to many clients (Riedel, 2003). Compared with the storage system of laptops and desktops, a salient feature of the storage subsystems in a file server is that multiple high performance disk drives are involved. The multiple disk drives are normally organized as a RAID with data distributed across the disk drives to aggregate the storage capacity, performance, and reliability (Riedel, 2003).

3.2.1 Architecture of NAS System

The architecture of a typical NAS has shown in Figure 3.3 and also illustrated the corresponding storage subsystem. The architecture consists of CPU, main memory, multiple disk drives, etc. The I/O Bridge has interfaces for the system bus and the memory bus, which are connected to CPU and memory, respectively. Multiple Small Computer System Interface (SCSI) bus adapters, which are inserted into the Peripheral Component Interconnect (PCI) slots, are used as initiators to connect the SCSI disk

drives. Each SCSI adapter can connect multiple SCSI disk drives. For example, an SCSI bus with 16 bits width can connect up to 15 SCSI disk drives, which act as target devices of the adapter. The disk drives can be organized as one or multiple large logical disk drives in terms of different RAID configurations. Data accesses are provided through the Network Interface Card (NIC) to the Ethernet.

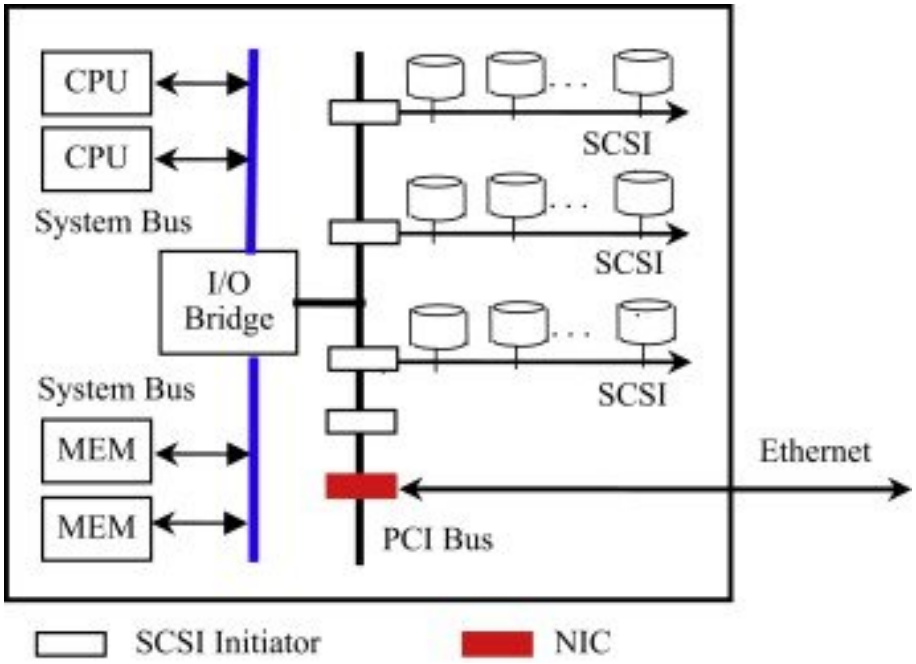

Figure 3.3: Architecture of typical Network Attached Storage System [33]

3.2.2 Data flow and key components of NAS

An NAS is a complex system composed of several components such as CPU, main memory, disk drive controllers, buses, and disk drives. It would be very complicated and unnecessary to analyze all the components which the data goes through, because it is difficult to extract the behaviors of the NAS from all the details. Varki et al. (2004) identified CPU, cache, and disk drives as the key components to model the performance of a real RAID. This is a quick way to isolate the potential problem areas with comparatively little effort. For simplicity, we have to identify the key components of an NAS. Figure 3.4 shows the basic data flow of file level I/O through a NAS.

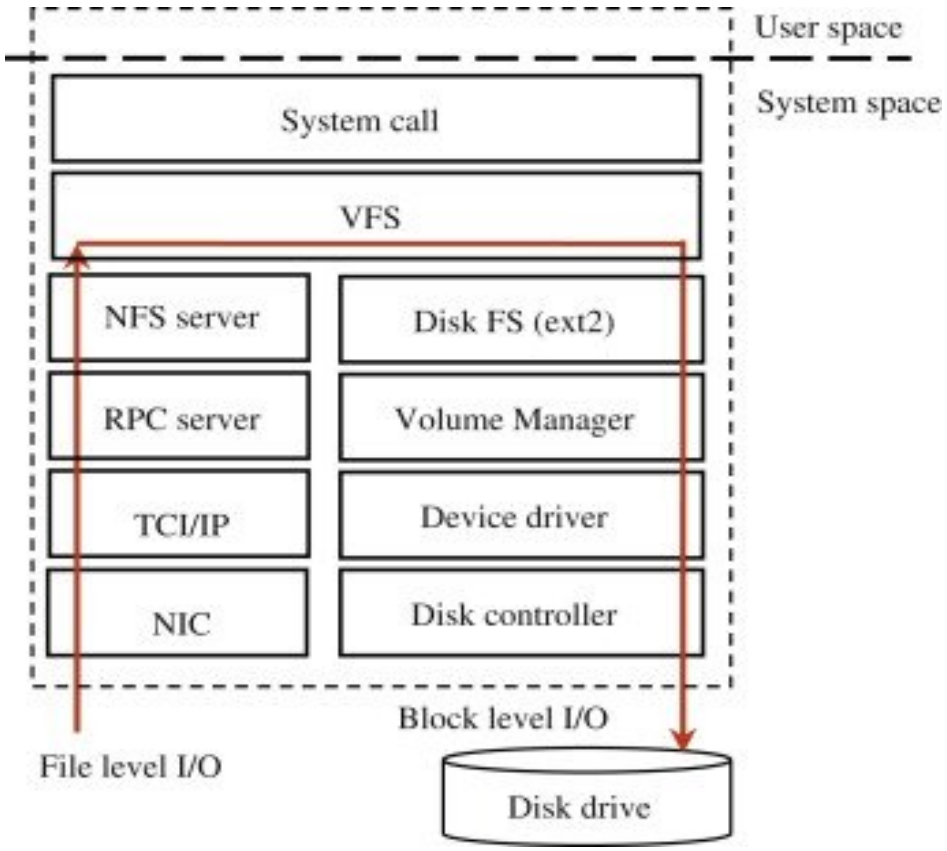

Figure 3.4: Data flow and key components of NAS [33]

NAS is unique because it contains the characteristics of both the storage and networking subsystems. The two subsystems are correlated with each other. NAS normally adopts CIFS or NFS to provide a file interface to the network. We use NFS in the following analysis and tests. NFS employs a client/server model and implements a standard network protocol, which provides file sharing to the remote clients transparently.

The data flow in an NAS normally takes following process: when a client application wishes to write a file to the NAS, it calls a write ( ) system call, which invokes a request to the NAS via the Remote Procedure Calls (RPC) layer. The parameters and data will be sent to the NAS as messages through TCP/IP stack by the RPC. The messages are then transferred to the RPC server from network to the system memory cache through the NIC and the system bus of the NAS. The RPC server unpacks and passes the messages to the NFS server protocol. The NFS server protocol passes the data to the local disk file system (e.g. ext2/ext3) of NAS through the VFS. According to write through policy, the data to be written will travel via the volume manager, device driver, PCI bus, disk controller, and

finally be written to the disk drives. If write back policy is chosen, the data in the memory cache will be written back to the disk drives when the data is evicted from the cache. A reply will be finally sent back to the client through the NFS, RPC, and TCP/IP protocol layers. When a read request from NAS client wishes to access data stored in the NAS, it first checks the memory cache. If the data can be found in the cache, the read request will be served and the corresponding data will be wrapped into messages and then sent to the client through TCP/IP stack by the RPC server immediately

3.3 Peer-to-Peer (P2P) Techniques

Peer-to-peer (P2P) was originally used to describe the communication of two peers and is analogous to a telephone conversion. Now P2P model offers a prospect of robustness, scalability and availability of a large pool of storage and computational resources [9] [10]. Recently, there has been some research efforts invested in designing P2P based large-scale and geographically distributed storage systems. However, the security, storage management, data consistency, etc. are all challenging problems to be solved. Fortunately, the P2P philosophy and techniques could be used to implement nonhierarchical and decentralized Grid systems. More recently, several research projects have investigated techniques for P2P Grid, which let us share resources (computers, databases, instruments, and so forth) distributed across multiple organizations. The P2P Grid is emerging as a promising platform for executing large-scale, resource intensive applications [12] and [13].

3.4 Gap Analysis and Problem Formulation

3.4.1 Gap Analysis

After conducting and extensive literature review, we have observed the following gaps:

- ✓ **Lack of scalability**

Several factors inhibit scalability in storage systems. These include:

- Cabling limitations

- Lack of connectivity
- Reliance on the server
- Bandwidth Management
- Lack of a common interface to storage
- Ability to horizontally scale throughput over many servers,
- Inefficient use of distributed indexes and RAM for data storage

✓ **Dynamic Storage management**

- Physical space management
- Virtual space organization
- Dynamically memory allocation and reallocation
- Dynamic memory release

✓ **Data Inconsistency**

3.4.2 Problem Formulation

It is proposed Dynamic and Scalable Storage Management (DSSM) architecture to organize Grid Oriented Service (GOS) devices into a large-scale and geographically distributed storage system to meet the requirements imposed by all kinds of Grid applications.

✓ DSSM divides GOS devices into multiple geographically distributed domains to facilitate the data access locality.

✓ DSSM consist two levels:

- Bottom level/physical level
- Upper level

✓ **Bottom level** adopts multicast to achieve dynamic, Scalable, and self-organized physical domains (intra-domain storage resource management)

✓ **Upper level** is a virtual domain that consists of geographically distributed and dynamic Grid Oriented Storage (GOS) agents selected from each physical domain.

- ✓ **Data locality** is a measure of how well data can be selected, retrieved, compactly stored, and reused for subsequent accesses.

Finally, we analysis and design the algorithms for above problems. The algorithms included in this thesis report are:

Algorithm#1: When a new GOS device wants to join the domain:

Algorithm#2: Leaving of a GOS device

Algorithm#3: Select agent from particular physical domain

Algorithm#4: Dynamic Storage Management and Access

Chapter 4
Proposed DSSM Architecture and Algorithm Description

The Proposed Dynamic and Scalable Storage Management (DSSM) architecture to organize Grid storage devices into a large-scale and geographically distributed storage system to meet the requirements imposed by all kinds of Grid applications. The DSSM divides Grid storage devices into multiple geographically distributed domains to facilitate the data access locality. The architecture consists of two levels. The bottom level adopts multicast to achieve dynamic, scalable, and self-organized physical domains. The method significantly simplifies the intra-domain storage resource management. The upper level is a virtual domain that consists of geographically distributed and dynamic GOS agents selected from each physical domain. P2P model is employed to discover the required storage resources. Due to the virtual two-tiered architecture, two-tiered parallelism and scalability of intra-domain and inter-domain are achieved [11]. The DSSM removes the traditional hierarchy or centralized approaches of conventional Grid architecture, avoids large-scale flat flooding of unstructured P2P systems, and inherits the dynamic scalability of P2P naturally. Storage resources of GOS devices are wrapped into Grid services which are adopted as basic building blocks to stack an infinite storage pool.

4.1 The DSSM architecture

A large-scale and geographically distributed storage system may involve hundreds or even thousands of storage nodes to match the requirements imposed by all kinds of applications, which is challenging the storage resource management with the increasing of the system scale.

4.1.1 Data Locality

Data locality is a measure of how well data can be selected, retrieved, compactly stored, and reused for subsequent accesses [14]. In general, there are two basic types of data locality: temporal and spatial. Temporal locality denotes that the data accessed at one

point in time will be accessed in the near future. Temporal locality relies on the access pattern of different applications and can therefore change dynamically. Spatial locality defines that the probability of accessing data is higher if the data near it was just accessed (e.g. prefetch). Unlike the temporal locality, spatial locality is inherent in the data managed by a storage system, and is relatively more stable and does not depend on applications, but rather on data organizations which is closely related to the system architecture. Data locality is a property of both the access pattern of applications and data organization or system architectures. Reshaping access patterns can be employed to improve temporal locality [15]. Data reorganization is normally adopted to improve spatial locality.

Many research efforts have been invested in exploiting the impact of access pattern and data organization of applications on the data locality to achieve performance gains [14] and [16]. DSSM divides GOS devices into multiple domains to facilitate the locality and reduce unnecessary network traffic. Fig.-1 illustrates the DSSM architecture.

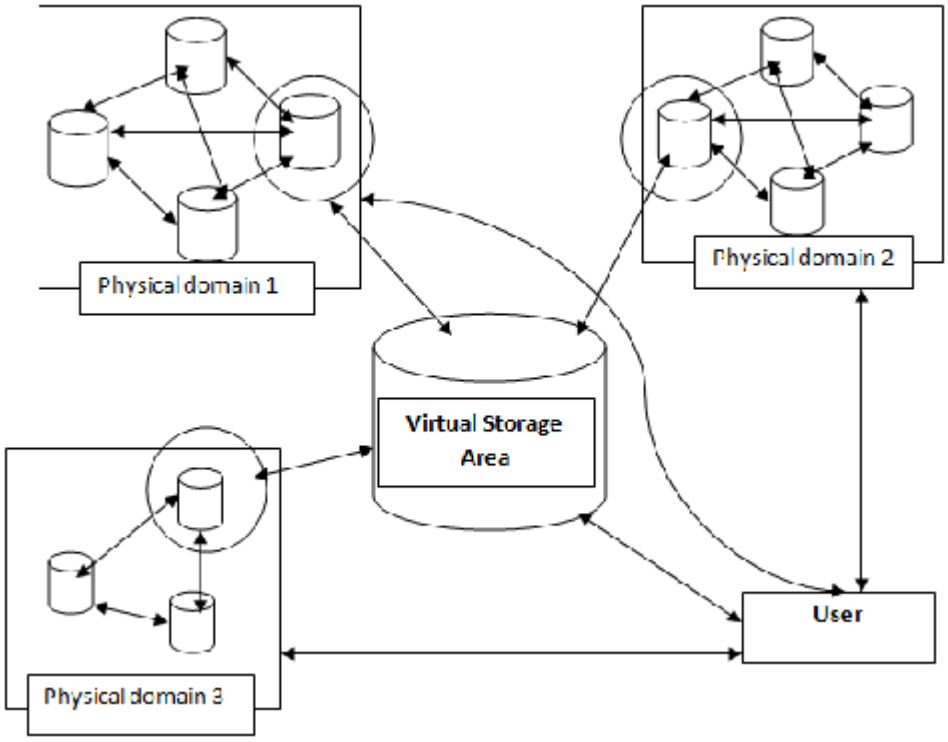

Figure 4.1: Proposed Architecture for DSSM

4.1.2 Domain Division

Choosing a suitable criterion to form GOS nodes into domains is an important factor of the DSSM architecture. DSSM employs the distance criterion in a geographical way to divide GOS nodes into multiple domains to facilitate the data access locality and limit the amount of communication traffic seen by each node with performance guarantee. GOS devices belonging to the same geographical area (e.g. the same enterprise or the same LAN) are formed into a domain, because the GOS devices in the area are normally placed geographically close.

GOS nodes divide into multiple domains based on:

- Distance
- Quantitative (bandwidth)
- Qualitative (performance analysis)

With a domain based GOS Network, the scalability is achieved from a two-tiered architecture, namely, intra-domain and inter-domain. Within a domain, each GOS device is equal in functions and capabilities, and can leave or join the domain dynamically [4][6]. Any node can communicate with anyone else directly. Each domain selects a domain agent from the domain members to cooperate with other domains according to domain formation algorithm.

With a domain based GOS network, the scalability is achieved from a two-tiered architecture, namely, intra-domain and inter-domain, as shown in Fig-1 Within a domain, each GOS device is equal in functions and capabilities, and can leave or join the domain dynamically. Any node can communicate with anyone else directly. Multicast is employed to achieve self-organizing and self-discovery features when the GOS devices leave or join the domains. Since DSSM is based on stable and trusted GOS devices that are less dynamic than pure P2P nodes, multicast does not result in much communication traffic. All GOS devices in a domain work collaboratively to construct a scalable storage system. Hence, the domain based DSSM architecture inherently facilitates data access locality. In addition, each domain selects a domain agent from the domain members to

cooperate with other domains according to a domain formation algorithm. As only the domain agents who form a virtual agent domain participate in the inter-domain communication, system wide information dissemination can be done far more efficiently than flat flooding [17].

The DSSM can be regarded as a virtual two-tiered hierarchy, but it is different to the traditional hierarchical architecture. It replaces a few root nodes of the traditional hierarchy with many GOS agents which can be expanded to a large-scale, thus avoiding the single point of failure and the potential performance bottleneck [18]. The DSSM can achieve near-infinite dynamic scalability due to the two-tiered architecture. The intra-domain scalability is obtained by expanding the system storage capacity incrementally with additional GOS devices along with associated network interfaces that expand the data transmission rate proportionally. Additional domain can join the DSSM to achieve the inter-domain scalability.

4.2 Domain Formation Algorithm of DSSM

The DSSM architecture consists of two levels. The bottom level adopts multicast to achieve dynamic, scalable, and self-organized physical domains. The upper level is a virtual domain that consists of geographically distributed and dynamic GOS agents selected from all domains. The objective of the domain formation algorithm is to manage the geographically distributed GOS devices across the Internet effectively. A good domain formation algorithm should not change the domain configuration too drastically when a few nodes are leaving or joining. Otherwise, the domain agents cannot control their domains efficiently and thus lose their roles as local coordinators.

The algorithm consists of knowledge of the neighbors of each node in a domain to organize the domain members and select a domain agent from all candidate members. Several distributed domain formation algorithms have been devised over the years. One is the lowest-ID algorithm [19]. The other is the highest-connectivity (degree) algorithm [20].

4.2.1 Algorithm Formulation at Bottom Level

DSSM architecture organized as a scalable storage system. Assign with a multicast IP address to provide a single entry point the storage space. Each GOS device maintains an Adjacent Information Table (AIT) which keeps a list of all active GOS devices with their related resource information. Information consists of IP address, reminder storage capacity, processing power. A data structure is adopted to describe the entry of AIT. The data structure should be maintained in memory so that they can be accessed with little overhead. The data structure takes 32 bytes per entry.

For a physical domain which has 1000 GOS devices, it takes only

1000*32 = 32000 bytes

to track the whole physical domain.

Compared with the size of main memory, the storage capacity of AIT is negligible.

Table 4.1: Adjacent Information Table (AIT)

Field	Size	Key Value
AIT ID	4 bytes	Alphanumeric
IP address	32-bits	Alphanumeric
Storage capacity	8 bytes (in MB)	Floating points
Processing power	8 bytes (in MHz)	Floating points

Algorithm#1: When a new GOS device wants to join the domain:

Step1: Send "JOIN" (the device sends a one hop probing multicast "JOIN" request of its coming and the corresponding local resource information such as the processing power, storage capacity, and waits for the response of other online GOS devices).

Step2: Receive "JOIN" request (the online GOS devices which receive the "JOIN" request add the oncoming device's information to their AIT).

Step3: Send ack "ACCEPT" (online GOS devices send an "ACCEPT" ack back to the probing GOS device using unicast)

Step4: Create AIT (the oncoming device constructs its own AIT in terms of the ack messages.)

Algorithm#2: Leaving of a GOS device

DSSM is based on stable and trusted GOS devices; the leasing of a GOS device is normally caused by maintenance, upgrade and other reasons.

Step1: Send "LEAVE" (multicast leave message in the domain)

Step2: Receive "LEAVE" (once the online GOS devices receive the message, the devices delete the leaving device's information)

Step3: Update AIT

Step4: Repeat Step2 (if message is not received then the device is assumed to have failed and other reminder GOS devices delete the device from their AIT).

All GOS devices in the physical domain at the bottom level are organized as a scalable storage system and assigned with a multicast IP address to provide a single entry point to the storage space. In order to avoid unnecessary network traffic to those GOS devices and switches where the request is not intended due to the multicast, any two GOS devices are at most one hop away. The formation algorithm of physical domain allows GOS devices to join or leave the running GOS network automatically. All GOS devices in the domain are self-discovery and self-organizing as an autonomy system. Each GOS device maintains an Adjacent Information Table (AIT) which keeps a list of all active GOS devices with their related resource information. Each entry of the AIT represents the information of one GOS device. The entry consists of IP address, remainder storage capacity, and processing power. A data structure is adopted to describe the entry of AIT. The data structures should be maintained in memory so that they can be accessed with little overhead. Therefore the storage capacity occupied by the data structures is important and should be kept as small as possible. All GOS devices are equal in functions except the GOS agents. Each GOS device has a coarse grain metadata description of the overall resource information of the physical domain in terms of AIT, which is very

helpful for a candidate GOS device to take over the role of the GOS agent when the agent leaves its domain.

4.2.2 Algorithm Formulation at Upper Level

Multiple physical domains can be formed simultaneously, and the domain formation procedure may continue iteratively. Eventually, all GOS devices will become affiliated with domains. Although the domains can be self-organized and self-administered independently, multiple domains may choose to operate cooperatively. The domain agents are selected to play this role.

During the process of the physical domain formation at bottom level, a processing power list of all GOS devices in the domain is formed and recorded on the AIT. Because a domain agent has to communicate with other domain agents to coordinate and manage all GOS devices within the domain, it consumes more computing resources than the ordinary domain members. The GOS device which has the highest processing power within a domain is selected as a domain agent. If the GOS agent fails or leaves the domain, the second GOS device on the processing power list automatically takes over the role. The isolated GOS devices are identified as default domain agents even if there is only one device in a domain.

The selection procedure of a GOS agent is straightforward. The first GOS device of a domain is regarded as an agent by default. The second GOS device which wants to join the domain compares its processing power against the first one. The GOS device which has higher processing power will be reselected as an agent. If the second GOS device has the same processing power as that of the first one, the agent will be kept unchanged. By analogy, we can have a unique GOS agent in a physical domain eventually. Once the domain agents are determined, now the problem is how to take advantage of the agents to coordinate multiple physical domains through GOS network efficiently. The domain agents do not use flat flooding or multicast to probe other agents due to the communication traffic and security reasons. A GOS service which maintains a list of all domain agents within the DSSM is running on the Internet. If a new domain agent wants to join the DSSM, firstly, the agent sends a request to contact the running GOS service

which has a public address. Secondly, according to the agent list, the GOS service redirects the request to the running domain agents which are close to the requester. All domain agents who are equal in functions communicate with each other to construct a virtual agent domain in terms of the AIT.

Algorithm#3: Select agent from particular physical domain

Step1: Select MAX[PP] (select highest processing power agent within a domain)

Step2: Repeat Step1 (if fail to select highest processing power agent)

Step3: Compare agent to another GOS

Step4: Repeat Step1 (highest processing power will always reselect as an agent)

Step5: In case of same processing power unchanged.

4.3 Storage Resource Monitoring and Discovering (SRMD)

The resources in Grid are often characterized by the dynamic, heterogeneous and distributed features, therefore the description, discovery, and monitoring of resources are challenging problems. Efficient resource discovery is a crucial problem in the large-scale and geographically distributed Grid systems. The main requirements include:

- Low performance overhead;
- Fast response to query;
- Ability to locate the service provider with good Quality of Service (QoS);
- Self-organizing capability to deal with dynamic joining and leaving of resources without centralized control [13].

Grid users are interacting with SRMD. Within the grid environment, the storage users can achieve the following functions [8]:-

- Obtain a certificate from the certificate authority.
- Establish the connection through Grid services
- With the help of SRMD, search and discovery appropriate storage service
- Access the corresponding storage resource transparently.

4.3.1 Structure of SRMD

1:http:/192.168.16.10:8080/srmd/gridusers/services/storservice

2:http:/192.168.16.12:8080/srmd/gridusers/services/mgtservice

3:http:/192.168.16.20:8080/srmd/gridusers/services/secservice

4:http:/192.168.16.10:8080/srmd/gridusers/services/comservice

Figure 4.2: SRMD Structure

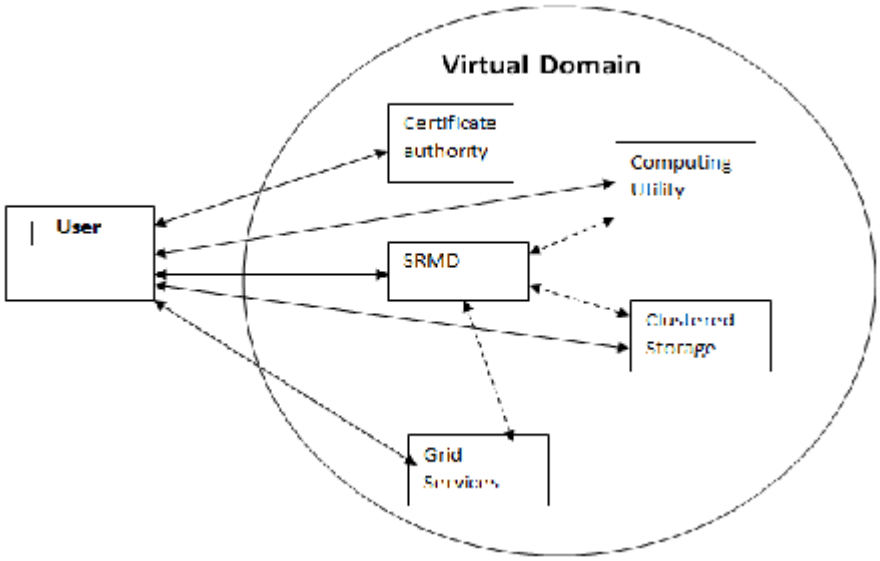

Figure 4.3: Storage Resource Monitoring and Discovery

4.3.2 Some methods for service invocation

Subscribe() –create a new account with the required storage capability

Unsubscribe() – release the storage capacity

Lst_dir() – display directory/path information

Upload() – transmit data to the remote storage resource

Download() – get data from the remote storage capacity

Delete() – remove data from storage capacity

Check() – check the properties such as storage capacity, life cycle of the storage service.

4.3.3 Properties of Storage Service

Storage capacity

- Username
- Password
- Path
- Total size
- Used size

Storage lifecycle

- Start_Time
- End_Time
- Used_Time

4.3.4 Grid service based storage resource discovery

The resource discovery in DSSM is based on the unstructured approach but different with that. Figure 4.4 depicts the resource discovery process in DSSM. The whole system consists of three GOS domains, three domain agents are cooperating with each other to provide index service (in the dashed rectangle). We only focus on the location phase after we gain an entry point into the GOS service community. The entry point can be obtained by out-of-band method or other approaches.

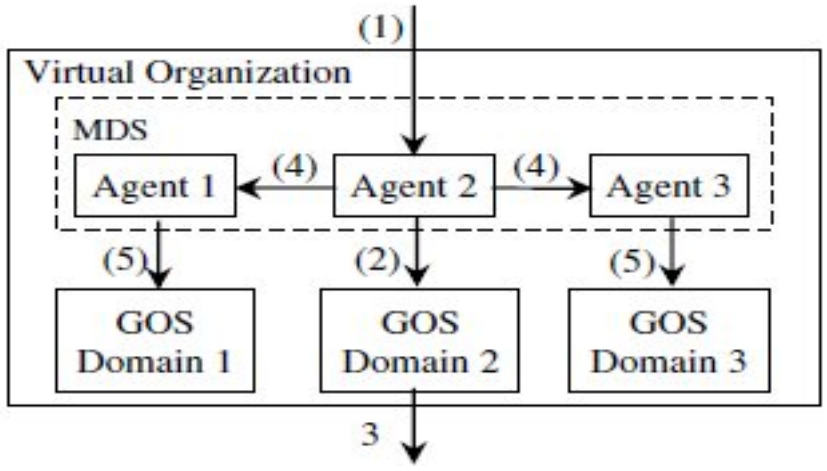

Figure 4.4: Resource Discovery of DSSM in GT4 Environment [29]

The major steps of resource discovery are labeled with the sequence number as defined in the following descriptions.

(1) A request initiated by a user is redirected from the GOS service published on the Internet to the domain agent 2.

(2) Once the request is received, the domain agent 2 searches its AIT since the AIT has the resources information of the whole physical GOS domain 2.

(3) If satisfied resources are found in the domain, the domain agent 2 will notify the corresponding GOS devices which have the required resources to communicate with the user directly (e.g. call GridFTP to transfer data) (The domain agent has the ability to pool resources in the domain to match the user's resource requirements).

(4) Otherwise, the agent 2 forwards the request to its neighbor agent 1 and agent 3 using unicast according to its AIT.

(5) The agent 1 and agent 3 will simultaneously repeat the operations as agent 2 did in step (2) and step (3) to achieve inter-domain parallelism.

For the traditional unstructured P2P searching method, the corresponding device sends a query message to its neighbors, which in turn forward it to their own neighbors. If a device possesses the requested resource, it sends a query hit message that will follow the same path back to the requesting device [29]. The method used in DSSM is more efficient and scalable than the traditional one, because it avoids flooding, eliminates centralized or hierarchically architecture, and the GOS device which has satisfied resources can communicate with users directly to avoid expensive store-and-forward data copying between GOS devices, GOS agents and users.

4.3.5 Wrapping Storage Resources into Grid Service

Building a large-scale system by composing of hundreds or even thousands of geographically distributed resources poses challenges to manage the resources efficiently. Service is becoming a basic application pattern of Grid because the service can be considered as a basic building block of an infinite resource pool which provides good scalability through its inherent parallelism, and facilitates simple incremental resource expansion (to add resources, one just adds services). Grid users can stack simple modular

service piece by piece as demand grows. A Grid service is a stateful Web Service with an associated lifetime which provides a set of interfaces through which Grid users may interact. The latest Open Grid Service Architecture (OGSA) [21] defines a set of protocols and standards for creating, naming, and discovering persistent and transient Grid service instances. Grid service must provide their users with the ability to access and manipulate state. As the basis of OGSA, Web Services Resource Framework (WSRF) [22] is a family of six Web Services specifications that build on SOAP [23], WSDL [24], and WS-Addressing [25] to define the WS-Resource approach for modeling and managing state in a Web service context. The WS-Resource consisted of a resource document and a corresponding Web service to implement a stateful service. Because Web service is stateless, the resource document employs XML [26] schema to capture state information for a WS-Resource due to the portability and ease in machine processing. The Web service can check and alter states contained in the resource document.

The DSSM divides GOS devices into multiple domains which are self-organized in a P2P manner according to the geographical location. All domains cooperate with each other through their domain agents. Based on the WSRF, each domain agent in the DSSM calculates the available storage resources in the domain in terms of its AIT, and wraps the storage resources as a Grid service. If one domain agent cannot provide satisfied resources, multiple domain agents can cooperate to provide storage resources (e.g. one domain can borrow storage resources from another domain through the communication of the domain agents). This approach introduces another level of flexibility on the resource utilization while taking into account of the overall efficiency.

The storage resource oriented Grid service consists of four parts that are implemented by four classes: A storage service factory, a storage service instance, a storage resource home, and the storage resources. When dealing with multiple resources, the WSRF specifications recommend employing the factory/instance pattern that is a well-known design pattern in software design, and especially in object oriented languages. The factory/instance pattern adopts one service in charge of creating the resources (the service factory) and another one to actually access the information contained in the resources (the

service instance). Because the GOS devices offer resources for hundreds of thousands of Grid users with different requirements, the factory/instance pattern is employed in the storage service. The storage resource home is responsible for registering and updating the MDS when it is necessary (e.g. when a user subscribes to some storage resources, the information of the remained storage resources in the SSP has to be updated in MDS.). The reader is referred to [27] for a comprehensive understanding of the storage resource oriented Grid service.

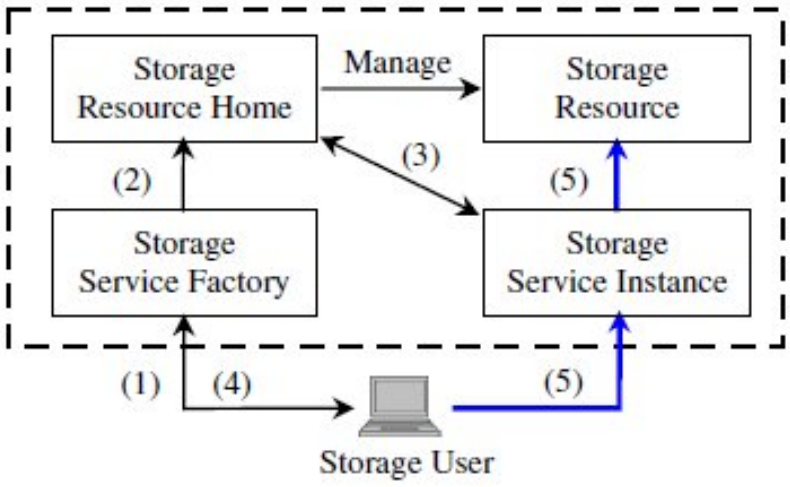

Figure 4.5: Components of a Storage Resource Oriented Grid Service [31].

The WS-Resource is adopted in the storage oriented Grid service to describe and access any state of the storage resources. This method can solve the multi attribute range queries. The WSRF lifetime management is employed to describe the resources that are destroyable via web services interfaces. Each time a new storage service instance is created, two properties are assigned, namely storage capacity and lifecycle. The users set the initial values when they submit a request for storage resources with their requirements. The states of these two properties are changed frequently due to the user's operation on the storage resource (e.g. when a user transfers some data to the subscribed storage resource, the state of the storage capacity has to be changed correspondingly). A newly created WS-Resource is uniquely identified by a WS-Addressing EPR [28] that is used to interact with the resource and distinguish between different storage service instances for different users. The storage users can employ the EPR to identify the

corresponding storage resources and perform methods to change the state of the properties.

4.4 Method of Dynamic Data Storage Management

A method of dynamic data storage and access, comprising:
(a) Allowing a user to input data of a record having an input length;

(b) Comparing the input length to a stored length of a first record segment of the record at a first location;

(c) Allocating a second record segment of the record at a second location different from the first location and having a length equal to a storage space difference which is a difference in storage space between the input and stored lengths when the input length is greater than the stored length;

(d) Linking the first record segment of the record to the second record segment of the record by storing an address of the second record segment of the record in the first record segment of the record when the input length is greater than the stored length;

(e) Storing the input data of the record in the first and second record segments of the record when the input length is greater than the stored length;

(f) Reducing a size of the first record segment of the record when the stored length is greater than the input length;

(g) Indicating the storage space difference is available for data storage when the stored length is greater than the input length;

(h) Allowing the user to indicate a record segment of the record is to be deleted and deleting the record segment;

(i) Indicating the deleted record segment is available for data storage when the record segment is deleted;

(j) Searching for file vacant space having a space length;

(k) Combining the first and second record segments of the record and into a contiguous combined single record segment and storing the combined record segments as the single record segment in the vacant space when the combined first and second record segments have a combined length shorter than or equal to the space length of the vacant space;

(l) Storing a record number of the record and the vacant space address in a record cross reference index table when the first and second record segments are combined and storing the record number and the first location in said table when the first and second record segments are not combined;

(m) Accessing the record comprising the single record segment using the vacant space address of said table when the first and second record segments are combined; and

(n) Accessing the record comprising the first and second record segments using the first location of said table and the address of the second record segment stored in the first record segment when the first and second record segments are not combined.

Chapter 5
Implementation and Experimental Evaluation

We constructed a proof-of-concept prototype with three different Thapar University's networks, one High End Computer Lab (HECL) Network emulator, Software Engineering Lab (SEL) Network and several client machines of PG Hostel, J Hostel's networks. All components were connected through a 1000/100M adaptive switch. Table-1 shows the system configurations of the prototype. The WAN emulator forwards packet at line rate and has user-settable delay and drop probability. In order to simulate a distributed WAN environment, all traffic among domains passed through the WAN emulator.

5.1 Dynamicity and reliability evaluation

The dynamicity and reliability of the DSSM architecture were measured at two stages. At the first stage, we set the IP address of three GOS devices within one network segment to denote a single physical domain. To illustrate the dynamic scalability and reliability of the architecture, we first configured one GOS device in HECL Network, and then the other two GOS devices joined the domain SEL Network, PG Hostel Network respectively, finally, the three GOS devices continuously left and joined the domain.

Table-5.1: System Configurations of the Prototype

	GOS Device	**WAN Emulator and Clients**
CPU	Intel Xeon 2.8GHZ	Intel Pentium IV 2.66 GHZ
Memory	3GB	512MB
NIC	Two Broadcom 100/1000M	Intel(R) PRO/100M
Disks	Six IBM FRU 32P0730	Seagate ST340014A
OS	Red Hat(Kernel 2.4.21)	Red Hat (Kernel 2.4.21)/MS Windows 7

We repeated the above process for more than 50 times, it did not cause any problems in our experiment. Because all GOS devices in the domain have the same processing power, the first GOS device of the domain is selected as an agent by default even it has no other GOS agents to cooperate with. At the second stage, to simulate two physical domains (HECL Network, SEL Network), we configured two network segments by setting the IP address of the three GOS devices. The first domain consisted of two GOS devices, and the second domain had only one GOS device. The communication traffic between two domains passed through a WAN emulator. The DSSM architecture was formed successfully by the following steps:

Step1: First GOS device in HECL Network was started and regarded as an agent by default.

Step2: The second GOS device joined the HECL Network. The GOS device which has higher processing power should be selected as an agent. In our test, the GOS devices had the same processing power. Therefore, the first GOS device played the role of an agent

Step3: Two GOS device in the HECL Network created their AITs in terms of multicast communications in the domain.

Step4: The GOS agent of HECL Network wrapped the storage resources of the two GOS devices into Grid services, and then put its address information on the list of the GOS service which has a public address.

Step5: The single GOS device in SEL Network acted as a domain agent, wrapped its local storage resources into Grid service, and registered its address in the public GOS service.

Step6: The agents of HECL Network and SEL Network constructed their AITs in terms of the request and acknowledgement, respectively. According to the AIT, a virtual agent domain consisting of two agents will be constructed by the communication among the domain agents. The formation and decomposition of DSSM were repeated more than 50 times continuously without incurring any problems. The DSSM is based on stable and trusted GOS devices that are less dynamic than pure P2P nodes. The leaving or joining of

GOS devices or GOS agents are normally caused by maintenance, upgrade and other reasons. It is unlikely to happen frequently. We do believe the above test can validate the dynamicity and reliability.

5.2 The impact of dynamicity on the resource discovery

We expect that the dynamicity of DSSM has a negligible performance impact on the resource discovery, because when a GOS agent leaves a domain, another candidate GOS device should immediately take over the role. Figure 5.1 shows the average response time of a static and a dynamic DSSM, respectively. It illustrates that the dynamicity of DSSM does have performance impact on the resource discovery. The reason is that there is only one GOS device which plays the role of a GOS agent in the SEL Network. If this GOS agent leaves DSSM, no candidate in the domain can take over the query traffic which goes to the agent, which results in a heavy traffic to the GOS agent of HECL Network. With the increase in number of users, the situation becomes worse, which incurs a long queue time and leads to a high response time. It validates that the query traffic can be shared by multiple GOS agents, and the traffic which goes to each GOS agent decreases with the growth in number of GOS agents.

5.3 Bandwidth evaluation

This report focuses on a dynamic and scalable storage management architecture which may involve hundreds or even thousands of distributed GOS devices. The goal of this work is to support long-distance and bulk data access of large-scale and complex Grid applications. Grid service combines the Web service and WSRF to provide a service based Grid environment that enables heterogeneous environments to be integrated and reconciled to accomplish complex tasks. The performance of the implemented prototype was measured by system bandwidth that is the amount of data divided by the data transfer time. The data transfer time was defined as the difference between the end time and start time of a data transfer excluding the process of resource discovery. A set of files appropriate range were transferred over the emulated WAN to measure the bandwidth.

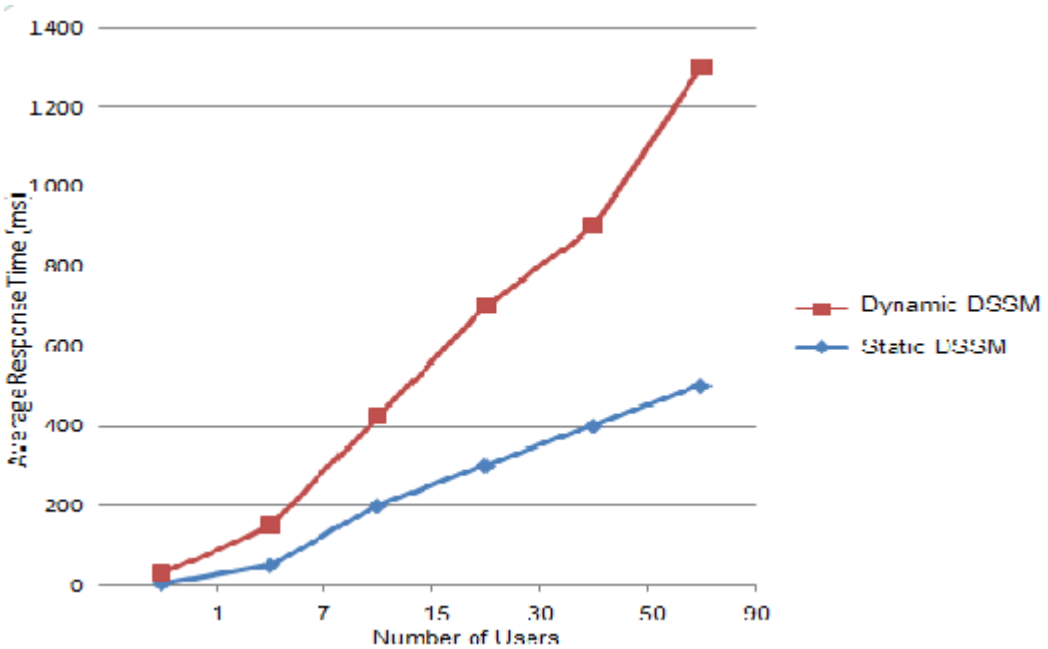

Figure 5.1: Average Response Time of the Static and Dynamic DSSM.

The file with specified size is called workload in the following discussion. All the performances reported in this report are based on the average of 600 measurements. The parallel data stream transfer of multiple streams of the prototype was investigated since this is a key feature of GridFTP. Figure 5.2b shows that as the size of the workloads increases, the bandwidth grows at the same time. The system bandwidth can be improved by increasing the number of parallel streams of GridFTP. In our experiment, the performance curves with different workloads are very similar. The bandwidth increases with the growth of parallel stream number, but there are some exceptions. It is interesting to observe that bandwidth of workload reaches the maximal value and then starts to decrease slowly.

Parallel data transfer requires that the data is partitioned across multiple parallel streams at sender side (scatter) and the striped data is combined at receiver side (gather). The data scatter/gather incurs some overhead penalty. To justify the penalty, the performance gain achieved by parallel transfer has to be greater than the overhead involved in data scatter/gather. For a certain size of workloads, with the increase of the parallel stream number, the overhead produced by scatter/gather grows.

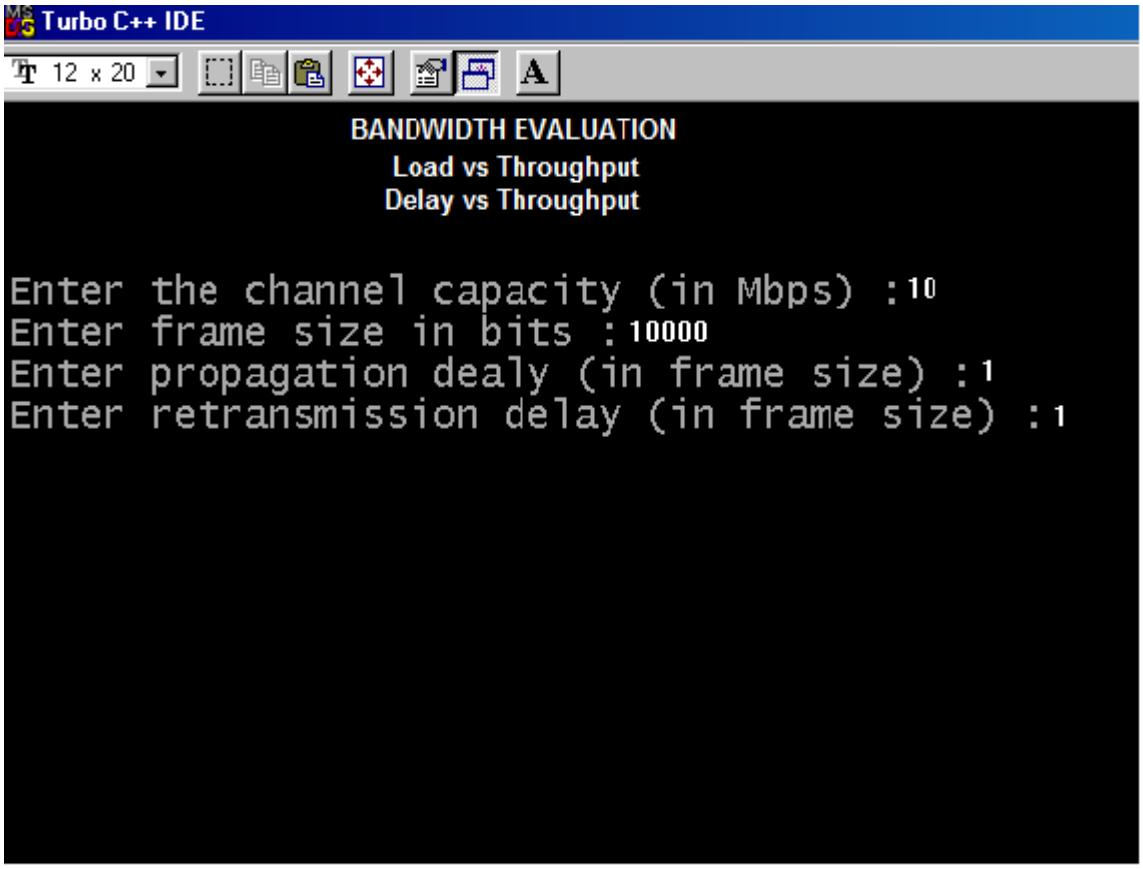

Figure 5.2a: Bandwidth Evaluation (Input Data Stream)

There are a fixed number of parallel streams which can strike a balance between the overhead involved in data scatter/gather and the benefit gained by parallel transfer. For a fixed parallel stream number, with the growth of load size, compared with the performance gain obtained by parallel transfer, the overhead of scatter/gather is relatively reduced.

We conclude that parallel stream data transfer works better on large files than small files. In order to investigate the impact of the background query requests on the system bandwidth, we adopted two client machines to submit query requests continuously to simulate a real working environment, while another client machine employed GridFTP to transfer data.

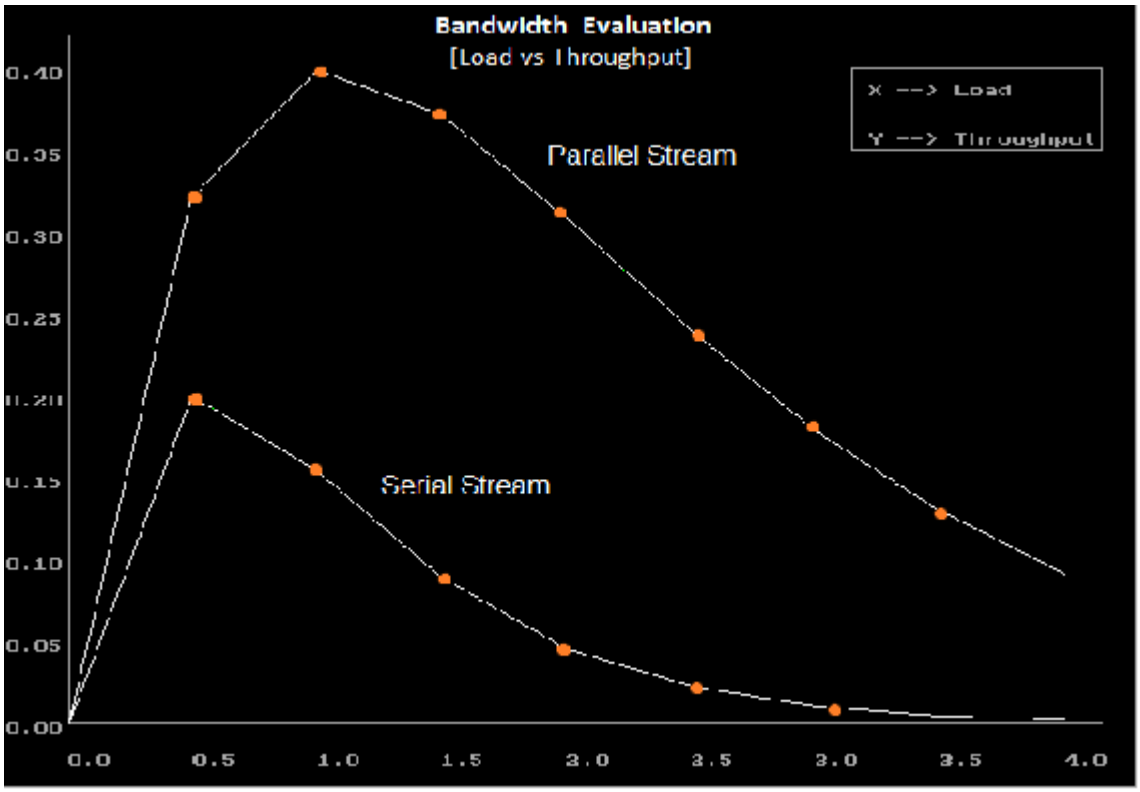

Figure 5.2b: Bandwidth Evaluation (Load vs. Throughput)

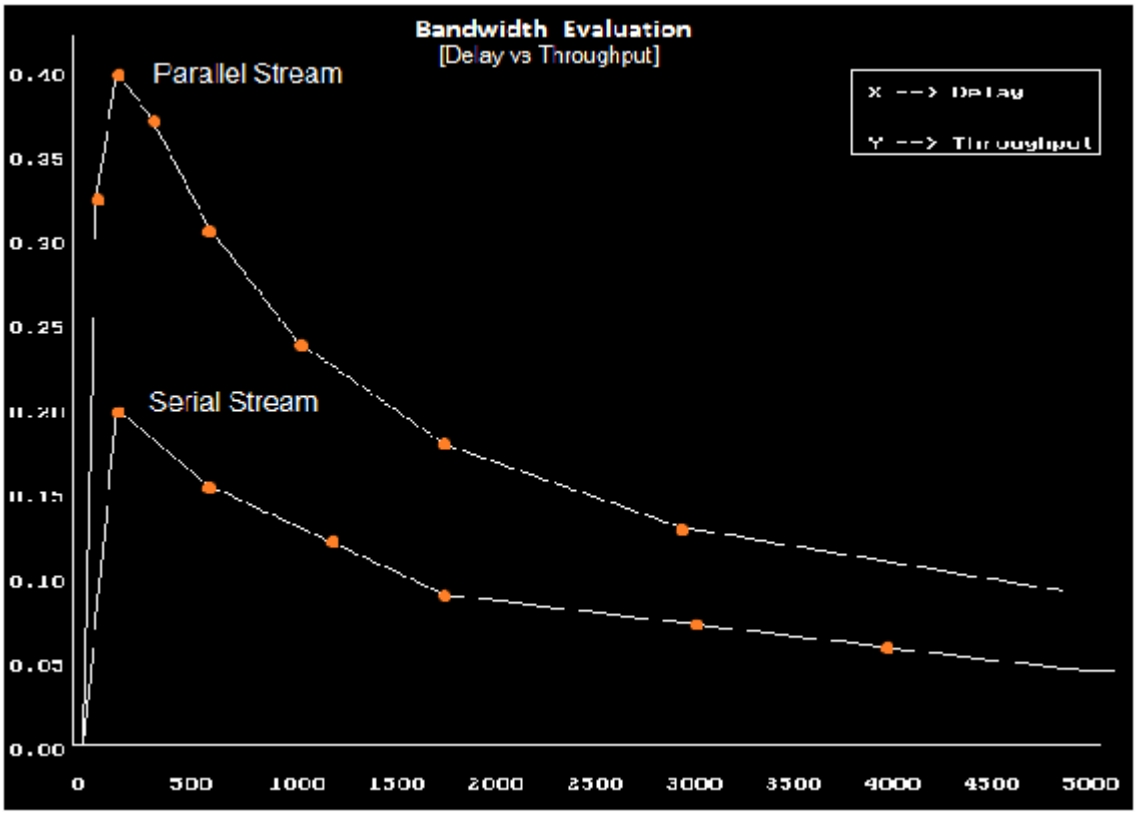

Figure 5.2c: Bandwidth Evaluation (Delay vs. Throughput)

We observed negligible bandwidth variation. The reason is that the query requests are processed by the GOS agents, while the data is transferred between the GOS device and the user directly. The traffic does not interfere with each other if the network is not a bottleneck.

5.4 Implementation Layout and Print Screen

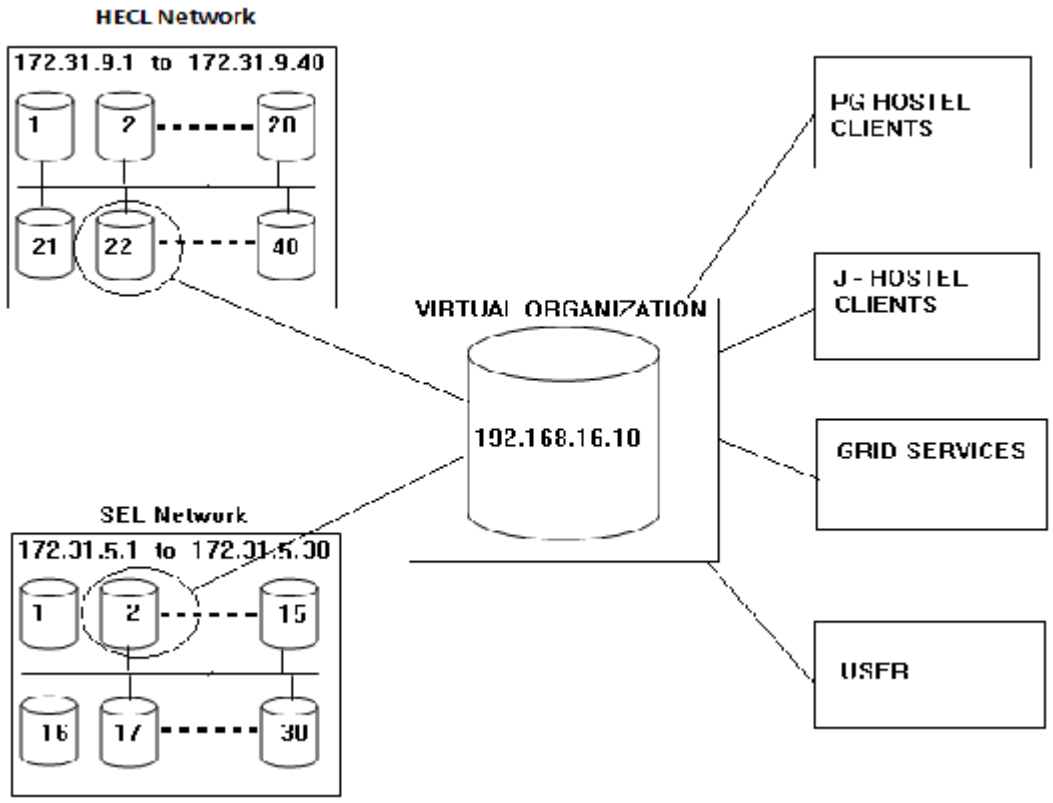

Figure 5.3: Implementation Layout of DSSM

There are following steps to used DSSM Application:

Step1: Enter IP Address of DSSM Server to any user who wants to become a member of Grid (shown in figure 5.4a).

Step2: After established the connection users have to enter the User ID or User Name of own machine (shown in figure 5.4b).

Step3: Server receives the message by user and gives response to user and so on for all other users who want to become a DSSM Member

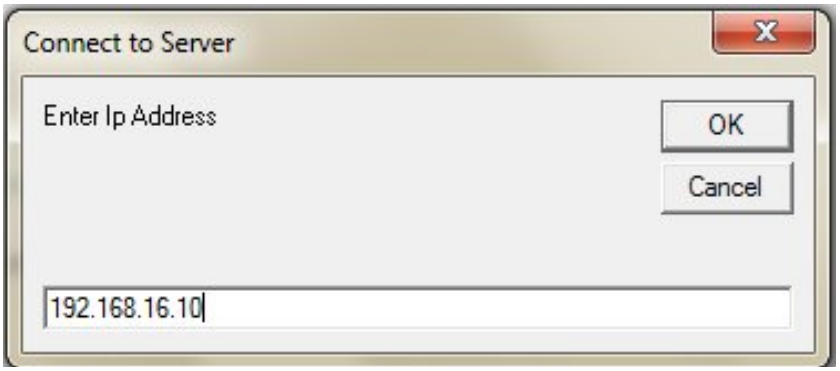

Figure 5.4a: Connect to DSSM Server

Figure 5.4b: Login to DSSM Server

Figure 5.4c: Receive message from user

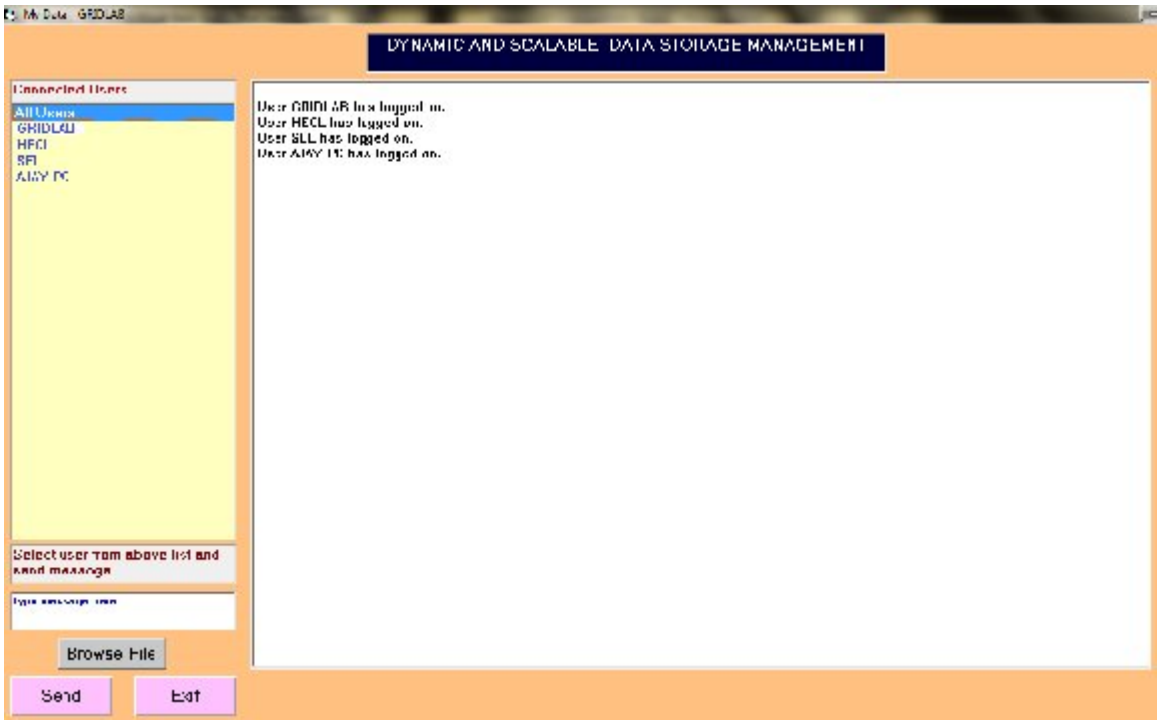

Figure 5.5a: Virtual Storage Management (Connected Users List)

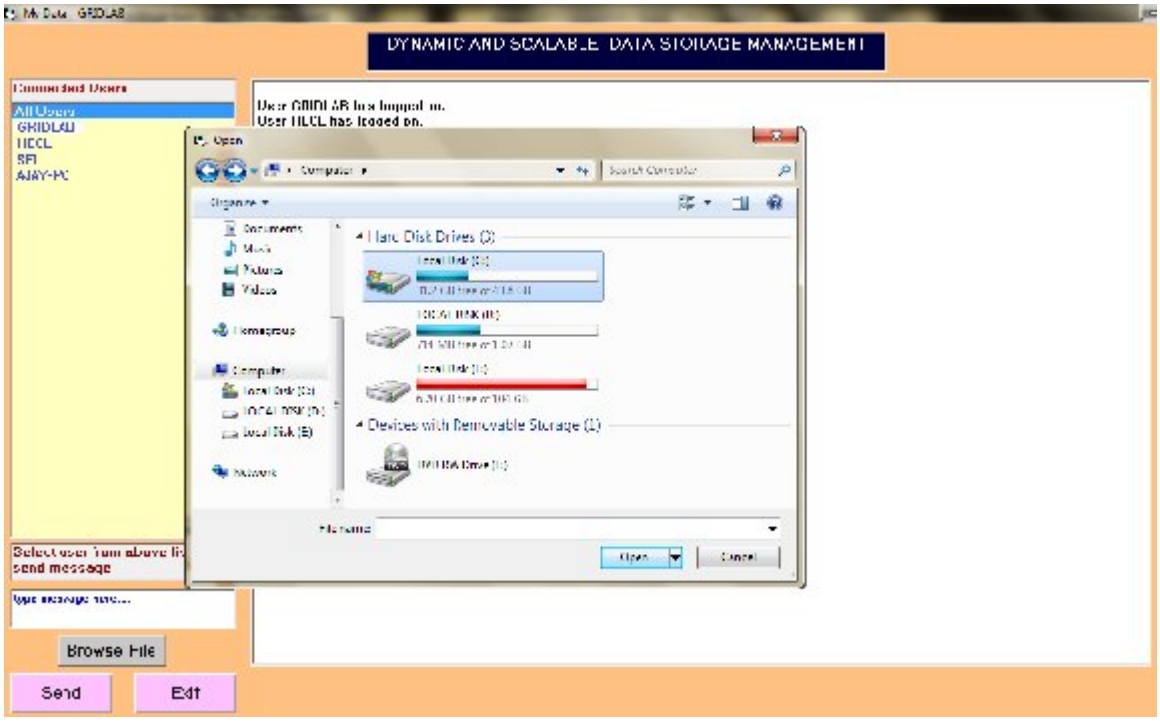

Figure 5.5b: Virtual Storage Management (Browse and Send Files)

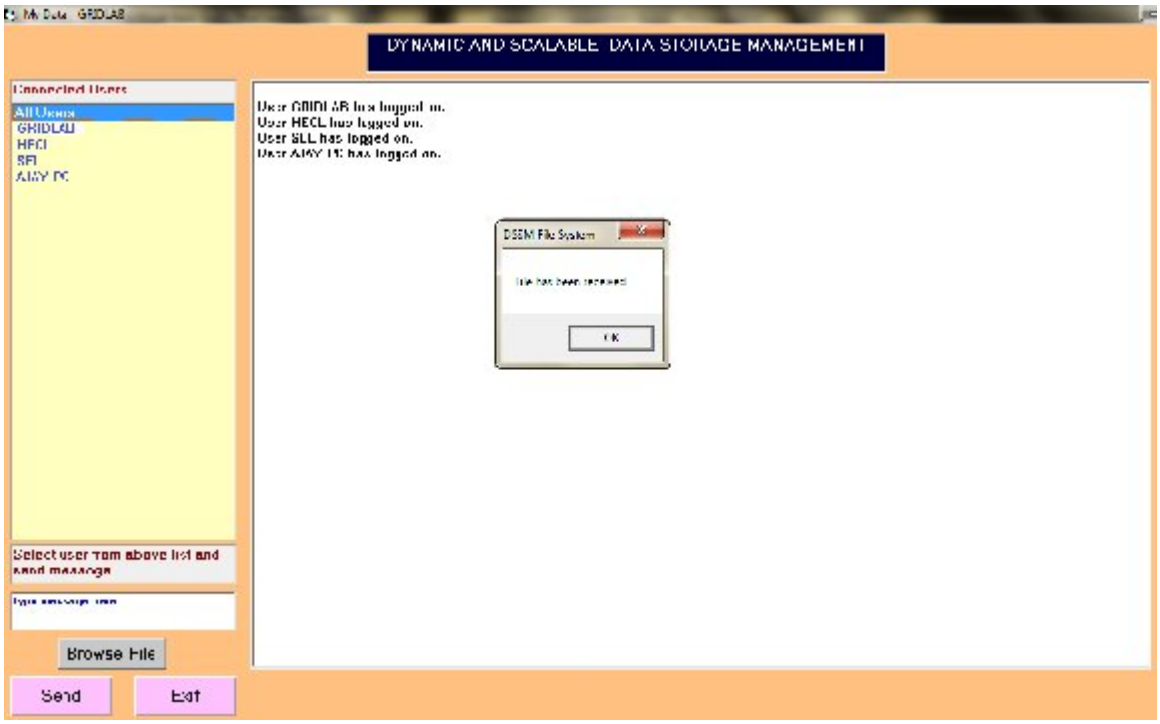

Figure 5.5c: Virtual Storage Management (Receive Message)

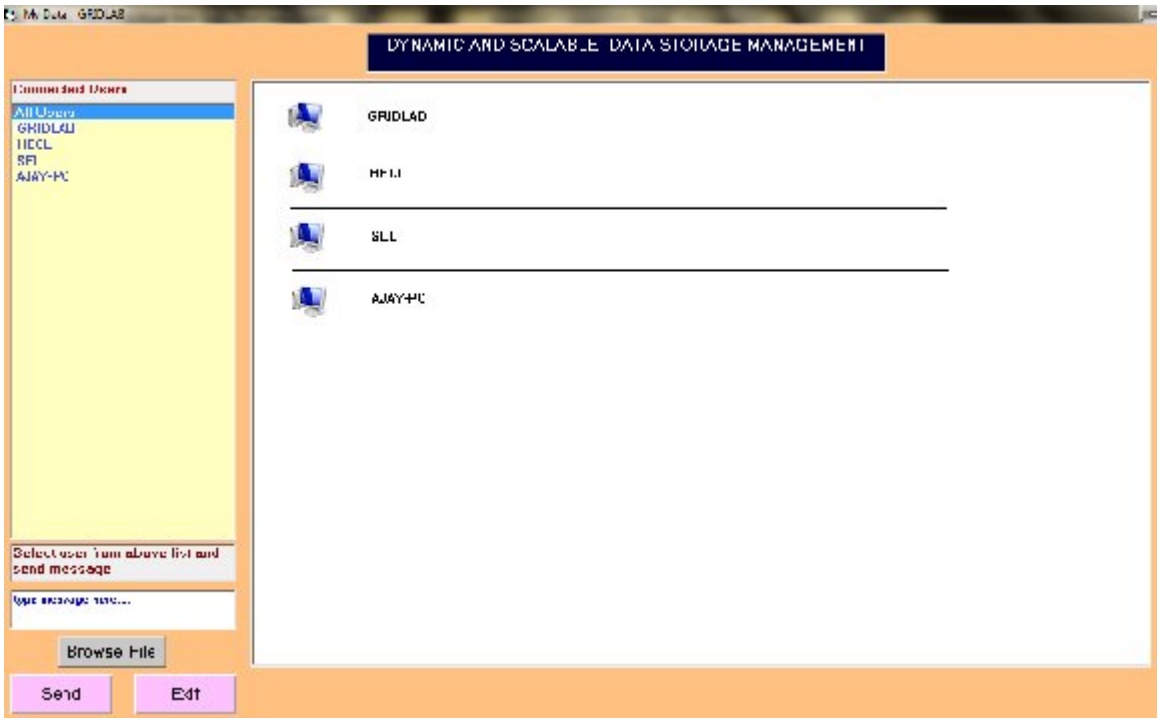

Figure 5.5d: Virtual Storage Management (Storage Files)

In summary, the prototype and experimental evaluation demonstrate that the DSSM architecture proposed in this report can organize the distributed GOS devices into a dynamic, scalable, and reliable storage pool to satisfy the storage capacity and bandwidth requirements posed by complex Grid applications. The system could be expanded to a large-scale with an evaluation of a large-scale experiment.

Chapter 6
Conclusions and Future Scopes

6.1 Conclusions

In this thesis report, based on an existing Grid environment, we propose and designe a DSSM architecture which organizes GOS devices into different domains to enhance the data access locality in terms of geographical and distributed areas. P2P model is employed to endow the architecture with dynamic scalability and reliability. Storage resources are wrapped into Grid services which are employed as basic building blocks of an infinite storage pool.

Storage expansion is achieved by simply adding services. The DSSM architecture avoids the hierarchical or centralized approaches of traditional Grid architecture, eliminates the flat flooding of unstructured P2P system, and provides a dynamic storage pool in Grid environment. The proof-of-concept prototype gives useful insights into the architecture behavior of DSSM architecture.

6.2 Future Scope

The DSSM can be regarded as a virtual two-tiered hierarchy, but it replaces a few root nodes of the traditional hierarchy with many GOS agents which can be expanded to a large-scale, thus avoiding the single point of failure and the potential performance bottleneck. The DSSM architecture does incur some additional overhead which results in performance penalty on the GOS agents in three scenarios.

The first, if hundreds of thousands query requests go to a particular domain agent simultaneously, the agent could become a potential system bottleneck. The reason is that all query traffic going to a domain has to pass through the domain agent. The second, an intelligent domain agent is able to pool the storage resources and keep the load balance among the GOS devices in the domain. It takes some additional computing overhead to do this job. The third one is the worst scenario. When a GOS agent is serving data access to users, a large number of query requests going to the agent will heavily overload the

agent. This scenario should be avoided or alleviated. However, the I/O traffic taken over by each agent will be decreased with the increase in number of domain agents. The probability that most of the I/O traffic goes to a particular domain can be decreased, if the hot data is arranged properly. A dynamic and transparent data replication mechanism which automatically places the data replicas across domains where they are needed is able to alleviate the performance impact on the GOS agents, it is also crucial to the overall performance. The first, it improves the aggregate bandwidth by providing simultaneous data access to multiple data copies. The second, it reduces the latency between the data provider and data consumer by copying the data near the data consumer, because the latency is incurred not only by the protocol stack of Grid service, but also by the distance of the communication [30].

References

[1] Wong Han Min, Wang Yonghong Wilson, Yeo Heng Ngi, et al., Dynamic storage resource management framework for the Grid, in: Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies, 2005, pp. 286–293.

[2] Viktors Berstis, Fundamentals of Grid Computing, IBM Redbooks Paper 2002.

[3] Storage Resource Broker, <http://www.sdsc.edu/srb/index.php/Main_Page>.

[4] A. Rajasekar, M. Wan, R. Moore, G. Kremenek, T. Guptil, Data Grids, collections, and Grid bricks, in: Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03), 2003, pp. 2–9.

[5] Michael Di Stefano, Distributed Data Management for Grid Computing, ISBN 0-471-68719-7, 2005.

[6] Distributed Parallel Storage System Project, <http://wwwdidc.lbl.gov/DPSS/>.

[7] Atsushi Manabe,Kohki Ishikawa, Yoshihiko Itoh, et al., A data Grid testbed environment in Gigabit WAN with HPSS, Computing in High Energy and Nuclear Physics (CHEP03), 2003.

[8] DataCutter, <http://www.cs.umd.edu/projects/hpsl/ResearchAreas/DataCutter.htm>.

[9] D. Talia and P. Trunfio, Towards a synergy between P2P and Grids, *IEEE Internet Computing* **7** (4) (2003), pp. 94–96.

[10] A. Rowstron, P. Druschel, Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01), 2001, pp. 188–201.

[11] John Kubiatowicz, David Bindel, Yan Chen, et al., OceanStore: an architecture for global-scale persistent storage, in: Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), 2000, pp. 190–201.

[12] S. Basu, S. Banerjee, P. Sharma, Sung-Ju Lee, NodeWiz: Peer-to-peer resource discovery for Grids, in: Proceedings of the IEEE International Symposium on Domain Computing and the Grid (CCGrid 2005), vol. 1, 2005, pp. 213–220.

[13] Cheng Zhu, Zhong Liu, Wei Ming Zhang, Weidong Xiao, Dongsheng Yang, Analysis on greedy-search based service location in P2P service Grid, in: Proceedings of the Peer-to-Peer Computing, 2003, pp. 110–117.

[14] Frederik W. Jansen, Erik Reinhard, Data locality in parallel rendering, in: Proceedings of the 2nd Eurographics Workshop on Parallel Graphics and Visualisation, 1998, pp. 1–15.

[15] Aart J.C. Bik, Reshaping access patterns for improving data locality, in: Proceedings of the 6th Workshop on Compilers for Parallel Computers, 1996, pp. 229–310.

[16] Mar'ia E. G'omez, Vicente SAITonja, Characterizing temporal locality in I/O workload, in: Proceedings of the 2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, 2002.

[17] B. Yang, H. Garcia-Molina, Improving search in peer-to-peer networks, in: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria, July 2002, pp. 5–14.

[18] Kai Hwang, Hai Jin and Roy S.C. Ho, Orthogonal striping and mirroring in distributed RAID for I/O-centric cluster computing, *IEEE Transanctions on Parallel Distribution System* **13** (1) (2002), pp. 26–44. View Record in Scopus | Cited By in Scopus (24)

[19] A. Ephremides, J.E. Wieselthier, D.J. Baker, A design concept for reliable mobile radio networks with frequency hopping signaling, in: Proceedings of IEEE, vol. 75, 1987, pp. 56–73.

[20] A.K. Parekh, Selecting routers in ad-hoc wireless networks, in: Proceedings of the SBT/IEEE International Telecommunications Symposium (ITS1994), 1994, pp. 420–424.

[21] The Open Grid Services Architecture, <http://www.globus.org/ogsa/>.

[22] Web Services Resource Framework, <http://www.globus.org/wsrf/>.

[23] Simple Object Access Protocol, <http://www.w3.org/TR/soap/>.

[24] Web Services Description Language, <http://www.w3.org/TR/wsdl>.

[25] WS-Addressing, <http://msdn2.microsoft.com/enus/library/ms951225.aspx>.

[26] Extensible Markup Language, <http://www.w3.org/XML/>.

[27] Yuhui Deng and Frank Wang, A heterogeneous storage Grid enabled by Grid service. ACM SIGOPS operating systems review, *Special Issue: File and Storage Systems* 41 (1) (2007)

[28] WS-Addressing, <http://msdn2.microsoft.com/enus/library/ms951225.aspx>.

[29] Carlo Mastroianni, Domenico Talia, Oreste Verta, A P2P approach for membership management and resource discovery in Grids, in: Proceedings of the Internationa Conference on Information Technology (ITCC05), 2005, pp. 168–174.

[30] Yuhui Deng and Frank Wang, Challenges and opportuAITies of service enabled storage Grid, *ACM SIGOPS Operating Systems Review* 41 (4) (2007), pp. 79–82.

[31] Dynamic and scalable storage management architecture for Grid Oriented Storage devices by Yuhui Deng , Frank Wang, Na Helian, Sining Wu, Chenhan Liao

[32] Arie Shoshani, Alex Sim, Junmin Gu Storage Resource Managers: Middleware Components for Grid StorageLawrence Berkeley National Laboratory Berkeley, California 94720

[33] Yuhui Deng,Deconstructing Network Attached Storage systems, aEMC Research China, Beijing 100084, PR China -2008

List of Papers Published/Communicated

Papers communicated in International Conference:

Conference: IEEE International Conference, Jaypee University, Solan, (H.P.)

Paper Title: Dynamic and Scalable Storage Management in Grid Environment (Dr. Seema Bawa, Ajay Kumar)

Abstract:

Grid applications typically deal with large amounts of data. In traditional approach high-performance dedicated servers are used for data storage and data replication. As computational grid focuses on the usage of idle resources from interconnected machines, a large amount of unused storage space can be used when the machines are idle or underutilized. This allows opportunistic grids to share not only the computational cycles, but also the storage space. In this paper new mechanism for Dynamic and Scalable Storage Management (DSSM) in grid environments is presented. The storage can be transparently accessed from any grid machine, allowing easy data sharing among grid users and applications. We developed the concept of virtual ids that, allows the creation of virtual spaces. Also, we design algorithms at bottom and upper level for standardization dynamic and scalable storage management, along with higher bandwidths.

Papers communicated in International Journal:

Journal: Journal of Computing, August 2010

Paper Title: Efficient Idleness Data Storage Management in Grid Environments (Dr. Seema Bawa, Ajay Kumar)

Abstract:

To enhance performance, reliability, and availability of data storage systems are scheduled with low priority and served during idle times. At this situation, idleness becomes a valuable resource that needs to be efficiently managed. A common approach in system design is to be network conserving by "idle waiting", that is delay the scheduling of background jobs to avoid slowing down upcoming foreground tasks. The storage system is better utilized without compromising foreground performance. Our analysis shows that if idle times have low variability, then idle waiting is not necessary. Only if idle times are highly variable does idle waiting become necessary to minimize the impact of background activity on foreground performance. If there is burstiness in idle intervals, then it is possible to predict accurately the length of incoming idle intervals and use this information to serve more background jobs without affecting foreground performance.