

# **Grid Computing - Issues in Data grids and Solutions**

A proposal submitted to the University of Cincinnati in  
partial fulfillment of the degree of Doctor of Philosophy

by

**Sudhindra Rao**

Bachelor of Engineering(University of Pune)

Master of Technology(IIT Bombay)

Master of Science(University of Cincinnati)

Operating Systems and Computer Architecture Lab

Department of Electrical and Computer Engineering and Computer Science

University of Cincinnati

*Chair : Dr. Yiming Hu*

Director, OSCAR Lab

31 March 2006

## **Abstract**

Grid computing in general comes from high-performance computing, super computing and later cluster computing where several processors or work stations are connected via a high-speed interconnect in order to compute a mutual program. Originally, the cluster was meant to span a local area network but then it was also extended to the wide area. A Grid itself is supposed to connect computing resources over the wide area network. The Grid research field can further be divided into two large sub-domains: Computational Grid and Data Grid. Whereas a Computational Grid is a natural extension of the former cluster computer where large computing tasks have to be computed at distributed computing resources, a Data Grid deals with the efficient management, placement and replication of large amounts of data. However, once data are in place, computational tasks can be run on the Grid using the provided data.

Data grids act as intermediaries between the compute grids and the data ( real world information). Design of these data grids is critical as data and not computation is the heart of business or scientific application. Data grids are still maturing to provide complex services and ensuring data availability over the internet. Here we discuss these data grids and argue that traditional computation focused design of systems is passe. We look afresh at the data grid design and try to provide solutions for improving them.

# Contents

<b>1</b>	<b>Grid Computing</b>	<b>4</b>
1.1	Compute Grids . . . . .	5
1.2	Data Grids . . . . .	7
<b>2</b>	<b>Data Grids</b>	<b>9</b>
2.1	Data grid Architecture . . . . .	10
2.2	Core Data Grid Services . . . . .	11
<b>3</b>	<b>Data Management in Grid Computing</b>	<b>12</b>
3.1	Motivations for a Data Grid Architecture . . . . .	12
3.2	Types(Levels of QoS requirements) for data grids . . . . .	13
3.3	Features of a Data Grid . . . . .	14
<b>4</b>	<b>Related Work</b>	<b>15</b>
4.1	Grid File System . . . . .	15
4.2	Grid NFS . . . . .	16
4.3	Secure File System . . . . .	16
<b>5</b>	<b>Technologies in Data management</b>	<b>19</b>
5.1	JavaSpaces . . . . .	19
5.1.1	Key Features of Javaspaces . . . . .	20
5.2	Distributed memory . . . . .	21
5.3	Global Replication of data - Oceanstore . . . . .	21

5.4	Grid Fabric - from Integrasoft . . . . .	22
<b>6</b>	<b>Issues in Data Management</b>	<b>23</b>
6.1	Data Delivery . . . . .	23
6.2	Datacentric grids - a need . . . . .	24
<b>7</b>	<b>Research directions in data management</b>	<b>26</b>
7.1	Data Grid as an extended DBMS . . . . .	26
7.2	WebServices and Data Grids . . . . .	30
7.2.1	Grid and SOA . . . . .	30
7.3	Web 2.0 and Semantic Data Grid . . . . .	31
<b>8</b>	<b>Research Plan</b>	<b>34</b>
8.1	Datagrid with persistence . . . . .	34
8.2	Using tags as indexes . . . . .	37
8.3	Data centric design of data grids . . . . .	38
8.4	Simulation . . . . .	40
8.5	Data access patterns and traces . . . . .	41
8.6	Issues/Roadblocks in completing the research . . . . .	42
8.7	Expected Results . . . . .	42
8.8	Conclusion . . . . .	42

# List of Figures

1.1	Distributed Computing Evolution . . . . .	5
1.2	Grid computing can solve problems . . . . .	6
1.3	Applications for Computational Grids . . . . .	7
2.1	Data grid supporting compute grids . . . . .	10
3.1	Application Complexity and QoS requirements . . . . .	14
7.1	Data grids and Relational Databases . . . . .	27
7.2	Web Services and data grids . . . . .	31
7.3	Web 2.0 . . . . .	33
8.1	Data regions as Databases . . . . .	36
8.2	Data regions acting as Persistence mechanism . . . . .	37

# Chapter 1

## Grid Computing

The proliferation and acceptance of personal computing has reduced the cost of computing drastically. Moore's law still holds true and the processor speeds and transistor packing is still on the rise. The understanding about performance has improved and engineers now understand that faster clock and more memory does not necessarily mean better performance. The ready availability of PCs and faster networks have made distributed computing feasible.

Large scale distributed computing has become cheaper and accessible. Large number of personal computers with equal amount processing power and memory are available. Most of the time these powerful machines are idle. Also these machines when networked together build a manageable network of computers ( called clusters). Hooking up a number of machines just multiplies the processing power. Instead of building monstrous super-computer which can only do speciality tasks ( or they are only economical when they perform such speciality tasks) researchers are moving towards building clusters of general purpose machines and tuning their applications to run on these clusters.

One such effort in Virginia Tech under Dr. Varadarajan [1] has proved to be path breaking where millions of dollars were saved by building a linux cluster instead of using an expensive super computer. Google the largest internet company today uses linux clusters to store its vast webcrawled data. Not only is it very cheap to build these clusters but also it is equally maintainable. Google does not need to call a super-computer scientist if something breaks down. The distributed computing paradigm has changed the way scientific applications are being built.

Figure 1.1 shows the progress of networked computing to distributed computing. It indicates the



complex business problems.

Business are growing and becoming distributed geographically for various reasons. Grid computing is becoming more relevant in such a scenario. Grid computing allows to solve difficult business problems inexpensively. It allows geographic and physical distribution of computational resources - thus providing redundancy. This also meant the frequently used, regional resources could be co-located. Slowly but certainly it is becoming apparent that computation power may soon become omnipresent like telephony and electricity. The ultimate goal would be to make it more reliable, always available, cheap, easy to fix and plug-n-play like these other technologies.

Figure 1.2 shows the driving forces that make grid computing possible. Also as technology makes progress these forces only justify grid computing as a better model for large scale systems.

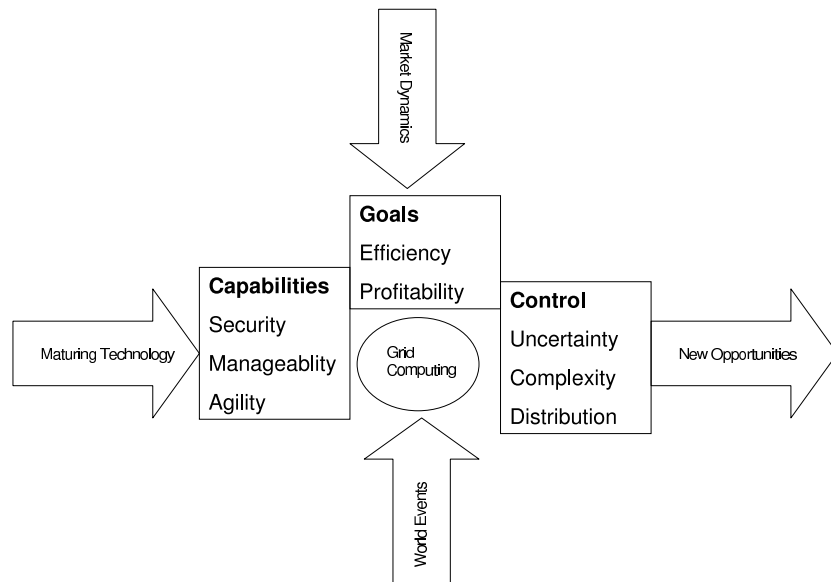


Figure 1.2: Grid computing can solve problems

Salient features of Grid computing can be listed as :

- Used to tap underutilized computing resources to do useful work
- Create Compute farms to scale gracefully for large scale problems
- Share resources

- Use like a single machine - very easy to maintain

Figure 1.3 depicts the nature of applications that can be suitable to be run on computational grids. Increasingly, a lot of businesses find grid computing a suitable solution for their applications.

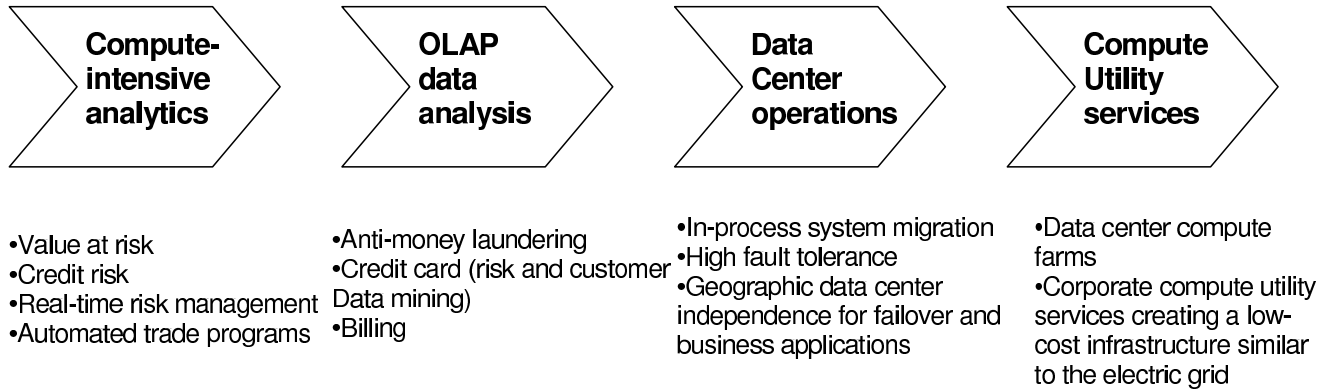


Figure 1.3: Applications for Computational Grids

The research at Argonne National Laboratory [2] kicked off interest in compute grids. Details of different designs of computational grids can be found in [3], [4] and [5]. Simulation tools for studying compute grids are available at [6] and [7]. The book [3] is an excellent research reference on compute grids and provides details on grid architecture, applications and research challenges.

## 1.2 Data Grids

In domains as diverse as global climate change, high energy physics, and computational genomics, the volume of interesting data is already measured in terabytes and will soon total petabytes [8]. The need for Data Grids stems from the fact that scientific applications like data analysis in High Energy Physics (HEP), climate modelling or earth observation are very data intensive and a large community of researchers all around the globe wants to have fast access to the data. Another factor that is promoting the growth of large scale distributed systems is the geographical distribution of resources, people and organisations. The communities that need to access and analyze this data (often using sophisticated and computationally expensive techniques) are often large and are almost always geographically distributed, as are the computing and storage resources that these communities rely upon to store and analyze their

data. This combination of large dataset size, geographic distribution of users and resources, and computationally intensive analysis results in complex and stringent performance demands that are not satisfied by any existing data management infrastructure. A large scientific collaboration may generate many queries, each involving access to—or supercomputer-class computations on—gigabytes or terabytes of data. Efficient and reliable execution of these queries may require careful management of terabyte caches, gigabit/s data transfer over wide area networks, coscheduling of data transfers and supercomputer computation, accurate performance estimations to guide the selection of dataset replicas, and other advanced techniques that collectively maximize use of scarce storage, networking, and computing resources. Effective management of data becomes very important in these scenarios. Data grids were developed as an intermediary to bridge this gap. Data grids by design are built to help manage vast amount of data effectively. We will discuss the features of data grids and design challenges in this research. We further motivate the focus on design of data grids and reason attention to intricacies in them.

# Chapter 2

## Data Grids

Early network computers were dependent on powerful machines called as servers for their needs of data or computation. There was a direct connection between the client and the server. As the workload increased, i.e. when more clients connected to a server, the server was burdened with more requests and eventually would become a bottleneck. Even the world wide web ( or the internet) used this client/server model. The limitations of this model became obvious. The Peer-to-Peer technology tried to change this model by adding the possibility of treating every client as a server too.

Similar problem was faced by early grid computing models. In this model a number of machines established a one-to-one communication channel with the data server. Now the server had to deal with these funnel of requests. This was not scalable as compute nodes were located all over the network and the data delivery became difficult. A data grid layer was added to this compute grid model. The data grid layer helps manage data required by compute grid - it caches, replicates, delivers, moves data according to the needs of the application. The data grid also helps manage data connections with not one but many data servers geographically distributed over the network. Figure 2.1 shows how the introduction of data grids helps manage compute grids as well as balance the load on the data centers.

In the following section we discuss data grid architecture, its principles, services and challenges.

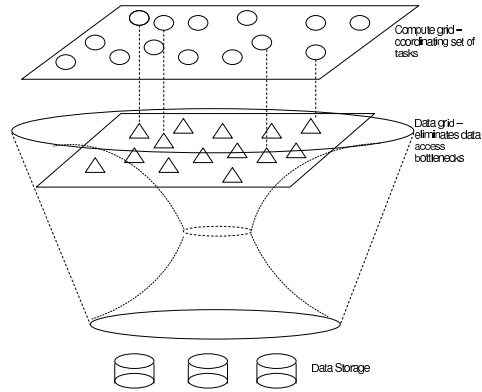


Figure 2.1: Data grid supporting compute grids

## 2.1 Data grid Architecture

The following four principles drive the design of data grid architecture. These principles derive from the fact that data grid applications must frequently operate in wide area, multi-institutional, heterogeneous environments, in which we cannot typically assume spatial or temporal uniformity of behavior or policy.

- *Mechanism neutrality* : The data grid architecture is designed to be as independent as possible of the low-level mechanisms used to store data, store metadata, transfer data, and so forth.
- *Policy neutrality* : As far as possible, design decisions with significant performance implications are exposed to the user. Thus, while data movement and replica cataloging are provided as basic operations, replication policies are implemented via higher-level procedures, for which defaults are provided but that can easily be substituted with application-specific code.
- *Compatibility with Grid infrastructure* : The structure of the data grid architecture is such that more specialized data grid tools are compatible with lower-level Grid mechanisms. This approach also simplifies the implementation of strategies that integrate, for example, storage and computation.
- *Uniformity of information infrastructure* : As in the underlying compute grid, uniform and convenient access to information about resource structure and state is emphasized as a means of enabling runtime adaptation to system conditions. In practice, this means that use the same data model and

interface to access the data grid's metadata, replica, and instance catalogs be used as in case of the underlying Grid information infrastructure.

## 2.2 Core Data Grid Services

Core services that are required to be offered by any data grid are listed in [8] as follows :

- **Storage Systems and the Grid Storage API** In a Grid environment, data may be stored in different locations and on different devices with different characteristics. As discussed above, mechanism neutrality implies that applications should not need to be aware of the specific low-level mechanisms required to access data at a particular location. Instead, applications should be presented with a uniform view of data and with uniform mechanisms for accessing that data. Together, these define the data access service.

A storage system holds data, which may actually be stored in a file system, database, or other system. A storage system will associate with each of the file instances (we also call it a data atom) that it contains a set of properties, including a name and attributes such as its size and access restrictions. The name assigned to a file instance by a particular storage system is arbitrary and has meaning only to that storage system. In many storage systems, a name will be a hierarchical directory path.

- **Grid Storage API** The behavior of a storage system as seen by a data grid user is defined by the data grid storage API, which defines a variety of operations on storage systems and file instances.
- **MetaData Service** The second set of basic machinery that is required is concerned with the management of information about the data grid itself, including information about file instances, the contents of file instances, and the various storage systems contained in the data grid. We refer to this information as metadata. The metadata service provides a means for publishing and accessing this metadata.

With these principles defined and core data grid defined we further discuss data management as related to grid computing.

# Chapter 3

## Data Management in Grid Computing

### 3.1 Motivations for a Data Grid Architecture

Data grids take away the burden of data management from the compute grids. Some motivations to design data grids comes from the changing requirements of computational problems.

Research in [9] details various aspects of grid computing and throws light on the importance of data grids. Some of the basic requirements to build data grids mentioned here are

- Coordinating compute and data plane - including streamlining access to data centers, data synchronization, managing data connections, etc.
- Task and resource management - managing scheduled tasks so as to facilitate faster computation with minimum blocking required for data.
- Sharing data distribution and location information - thus facilitating data mobility, caching, scheduling of compute tasks.
- Leveraging data locality with functions such as data affinity - affecting computation and resource management based on data available.

Data grids manage data in chunks termed as data atoms. This name signifies that the chunk of data is always considered as a group and enforces a number of data management mechanisms based on the grouping. Managing this granularity of data imposes some constraints on data grids. Data grids not only

face challenges in the management of data but have to provide the following guarantees about data to the compute plane.

- Dependability - Compute grids should not be required to perform any validation on the data delivered by the data grid.
- Consistency - Data should not change without any computation regurgitation.
- Pervasiveness - Data should be transparent. Data accessed directly at the source and that at the compute layer should not vary in form or content.
- Security - Unauthorized access to data should not be allowed. Data grids have to enforce some kind of authentication/authorization mechanism to prevent malicious use/alteration of data.
- Inexpensive -Data grid should be as inexpensive as possible as data storage and computation will be the areas that require more investment. Data transportation cost should go down with the lowering costs and higher availability of networks.

## 3.2 Types(Levels of QoS requirements) for data grids

Figure 3.1 depicts the levels of QoS requirements of business and research applications. Close examination of the plot shows the trend of applications in terms of their QoS requirements. More and more applications deal with dynamic data and need support for transactions.

There are numerous factors that influence how the interdependencies of application data requirements, data integrity and quality of service(QoS). Significant differences exist in Level 0 and Level 1 QoS requirements. We consider the solutions in a bit more detail to elaborate on the necessity of Level 1 data grids. Features of Level 0 Grids -

- Mainly for distribution of large static data sets - do not address updates, transactions and integration with external systems
- Distributed filesystems and GridFTP satisfied the requirements by providing efficient means to distribute data, providing a protocol for data movement but were unable to resolve management issues like transactions, synchronization, integration or querying of data.

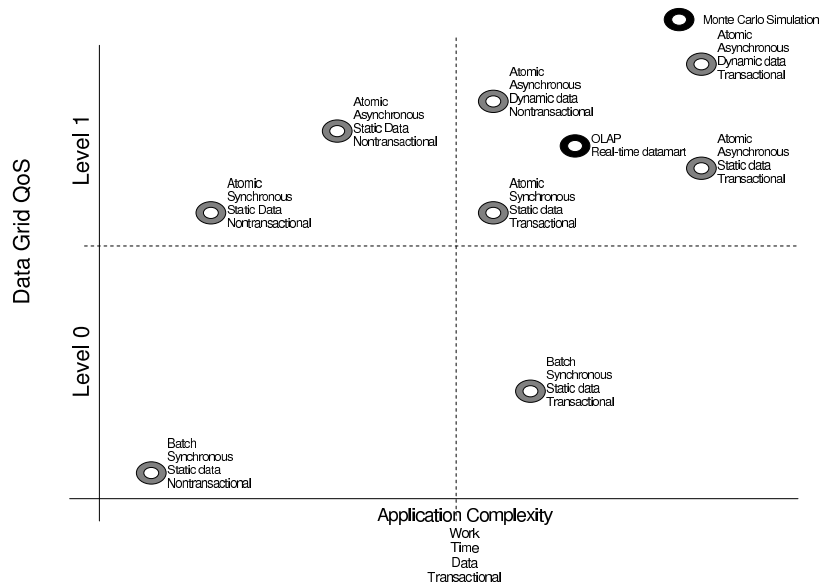


Figure 3.1: Application Complexity and QoS requirements

On the other hand Level 1 data grids can :

- Support datasets that change as frequently as required by the application - hourly to second-to-second.
- Supply methods to access, manage, synchronize and act on transactions.
- Support higher order data structures like tables, arrays, hashes - as against data bytes.

### 3.3 Features of a Data Grid

Finally we can say that the following are the high level features that one expects from a Level 1 data grid

- Data Grid plane and compute grid plane work together
- data grid provides data management and distribution
- Data Regionalization which includes - synchronization, distribution, transactionality, fault tolerance/data availability

# Chapter 4

## Related Work

We discuss some of the related data management as applied to distributed computing and find out how it fits into the requirements imposed on to the grid architecture.

### 4.1 Grid File System

Grid File system as discussed by various researchers is a step towards organizing data on the grid using a file system. Major driving forces for this file system are requirements like

- Human readable naming system to organize grid information (mapping service oriented URIs as collections)
- Location independent logical naming
- Data-intensive applications can execute anywhere in grid
- Data handling system must provide location transparency
- Dynamic provisioning of heterogeneous storage
- Storage space from multiple administrative domains and multiple heterogeneous storage systems
- Logical storage resource identifiers (in spite of the storage virtualization) for QoS and Technology Migration

All said this system is very similar to a file system and has the similar issues related to a file system. Also from moving to one domain to another the data identifier may have to be renamed depending on the naming conventions of the new domain. Establishing a standard naming in this scenario is difficult. Also developing data synchronization mechanisms for a large file system is a challenge. Furthermore this file system does not necessarily provide transactionality and querying that is transparent. Essentially it leads to a Level 0 data grid.

## **4.2 Grid NFS**

GridNFS[10] is a collection of NFS version 4 features and minor versions with supporting daemons, data bases, and tools that enables NFSv4 to integrate with emerging Grid technologies to provide petabyte scale file service for the Grid. NFSv4 is the IETF standard for distributed file systems that is designed for security, extensibility, and high performance. Global name space can be constructed. GridNFS is currently only interested in the file storage protocol for the Data path. Scaling characteristics of a single NFSv4 server is dependent upon the file system being exported. Maximum file size, delivery of data through file system Server hardware configuration, disk configuration, etc The NFSv4 pNFS minor version with file layouts in combination with enterprise NFSv4.0 service has the potential to deliver a global distributed file service scaling to petabytes of data.

## **4.3 Secure File System**

Locating and storing data efficiently has been a challenge on large scale systems. With large scale distribution data security becomes more important. Secure File System(SFS) [11] is one such mechanism which ties together location management and security of data. Although it ties these two things together they are rather loosely coupled. SFS uses that filenames that contain public keys making them self-certifying pathnames. SFS goal of a single global file system requires that it look the same from every client machine in the world. SFS takes a new approach to file system security: it removes key management from the file system entirely. SFS introduces self-certifying path names/file names that effectively contain the appropriate remote servers public key. Because self-certifying pathnames already

specify public keys, SFS needs no separate key management machinery to communicate securely with file servers. Thus, while other file systems have specific policies for assigning file names to encryption keys, SFSs key management policy results from the choice users make of which file names to access in the first place.

SFS further decouples user authentication from the file system through a modular architecture. External programs authenticate users with protocols opaque to the file system software itself. These programs communicate with the file system software through well-defined RPC interfaces. Thus, programmers can easily replace them without touching the core of the file system.

Pushing key management out of the file system lets arbitrary key management policies coexist on the same file system, which in turn makes SFS useful in a wide range of file sharing situations. This paper will describe numerous key management techniques built on top of SFS. Two in particular certification authorities and password authentication both fill important needs. Neither could have been implemented had the other been wired into the file system.

Without mandating any particular approach to key management, SFS itself also provides a great key management infrastructure. Symbolic links assign human-readable names to self-certifying pathnames. Thus, SFSs global namespace functions as a key certification namespace. One can realize many key management schemes using only simple file utilities. Moreover, people can bootstrap one key management mechanism with another. In practice, we have found the ability to combine various key management schemes quite powerful.

SFS assumes that users trust the clients they use for instance, clients must actually run the real SFS software to get its benefits. For most file systems, users must also trust the server to store and return file data correctly (though public, read-only file systems can reside on untrusted servers). Without external information, SFS must obtain file data securely given only a file name. SFS therefore introduces self-certifying pathnames file names that inherently specify all information necessary to communicate securely with remote file servers, namely a network address and a public key.

Every SFS file system is accessible under a pathname of the form `sfs/Location:HostID`. Location tells an SFS client where to look for the file systems server, while HostID tells the client how to certify a secure channel to that server. Location can be either a DNS hostname or an IP address. To achieve secure communication, every SFS server has a public key. HostID is a cryptographic hash of that key and the

servers Location. HostIDs let clients ask servers for their public keys and verify the authenticity of the reply. Knowing the public key of a server lets a client communicate securely with it.

# Chapter 5

## Technologies in Data management

### 5.1 JavaSpaces

JavaSpaces technology [12] and [13] is a high-level coordination tool for gluing processes together into a distributed application. JavaSpaces technology provides a fundamentally different programming model that views an application as a collection of processes cooperating via the flow of objects into and out of one or more spaces. The JavaSpaces technology's shared, persistent object store encourages the use of distributed data structures, and its loosely coupled nature simplifies the development of distributed protocols.

A space is a shared, network-accessible repository for objects. Processes use the repository as a persistent object storage and exchange mechanism; instead of communicating directly, they coordinate by exchanging objects through spaces.

To build space-based applications, we design distributed data structures and distributed protocols that operate over them. A distributed data structure is made up of multiple objects that are stored in one or more spaces. Representing data as a collection of objects in a shared space allows multiple processes to concurrently access and modify the data structure.

Distributed protocols define the way participants in an application share and modify these data structures in a coordinated way. Distributed protocols written using spaces have the advantage of being loosely coupled: because processes interact indirectly through a space (and not directly with other processes),

data senders and receivers aren't required to know each other's identities or even to be active at the same time. Conventional network tools require that all messages be sent to a particular process (who), on a particular machine (where), at a particular time (when). Instead, using a JavaSpaces system, we can write an object into a space with the expectation that someone, somewhere, at some time, will take the object and make use of it according to the distributed protocol. Uncoupling senders and receivers leads to protocols that are simple, flexible, and reliable.

### 5.1.1 Key Features of Javaspaces

- **Spaces are shared:** Spaces are network-accessible "shared memories" that many remote processes can interact with concurrently. The "shared memory" also allows multiple processes to simultaneously build and access distributed data structures, using objects as building blocks.
- **Spaces are persistent:** Spaces provide reliable storage for objects.
- **Spaces are associative:** Objects in a space are located via associative lookup, rather than by memory location or by identifier. Associative lookup provides a simple means of finding the objects you're interested in according to their content.
- **Spaces are transactionally secure:** The JavaSpaces technology provides a transaction model that ensures that an operation on a space is atomic (either the operation is applied, or it isn't). Transactions are supported for single operations on a single space, as well as multiple operations over one or more spaces (either all the operations are applied, or none are).
- **Spaces allow exchange of executable content:** While in the space, objects are just passive data we can't modify them or invoke their methods. However, when we read or take an object from a space, a local copy of the object is created. Like any other local object we can modify its public fields as well as invoke its methods, even if we've never seen an object like it before. This capability gives us a powerful mechanism for extending the behavior of our applications through a space.

## 5.2 Distributed memory

Another architecture for data storage is becoming known - WebOS. WebOS provides access to geographically distributed data- dissemination and computing resources[14]. WebOS needs smart clients, smart proxies and also a smart FileSystem. Applications like Internet Chat, Remote Compute Engine, Wide Area Cooperative Cache and Rent-a-job or rent-a-server are available. It provides resource discovery, persistent storage, process control with security and authentication mechanisms.

Against the WWW which provides access to read only data - most of it is managed and discovered using human interfaces, WebOS has smart infrastructure. Smart Clients can have been developed to support dynamic reconfiguration, redirection of requests based on geographical location. Smart clients also have the ability to handle load balancing, naming, fault tolerance mechanisms. Smart proxies are more manageable since managing one interface effectively manages a family of clients. Smart infrastructure - can deliver/discover what is suitable based on various QoS policies (like next generation IPV6 network). This research effort is not complete and a lot of details are being dealt with.

## 5.3 Global Replication of data - Oceanstore

OceanStore [15, 16] provides a consistent, highly-available, and durable storage utility atop an infrastructure comprised of untrusted servers.

The utility model thus combines the resources from federated systems to provide a quality of service higher than that achievable by any single company.

OceanStore caches data promiscuously; any server may create a local replica of any data object. These local replicas provide faster access and robustness to network partitions. They also reduce network congestion by localizing access traffic.

Any server in the infrastructure may crash, leak information, or become compromised. Promiscuous caching therefore requires redundancy and cryptographic techniques to protect the data from the servers upon which it resides.

OceanStore employs a Byzantine-fault tolerant commit protocol to provide strong consistency across replicas. The OceanStore API also allows applications to weaken their consistency restrictions in ex-

change for higher performance and availability.

A version-based archival storage system provides durability which exceeds today's best by orders of magnitude. OceanStore stores each version of a data object in a permanent, read-only form, which is encoded with an erasure code and spread over hundreds or thousands of servers. A small subset of the encoded fragments are sufficient to reconstruct the archived object; only a global-scale disaster could disable enough machines to destroy the archived object.

## **5.4 Grid Fabric - from Integrasoft**

Integrasoft a company that sells data management software has developed what they call as a Integrasoft Grid Fabric(IGF) [9]. IGF essentially provides effective ways to manage data - both from the user as well as the grid perspective. It provides same levels of service as a relational database system.

IGF's engine creates a distributed memory space that spans the physical nodes of the grid. Various levels of data management services are available within this memory - data regions, transactionality, querying and data affinity. IGF brings data closer to the computation thus offering immediate access to data as if it were like a client server relationship. IGF as a product is different from JavaSpaces in that [9]

- JavaSpaces is a java-only solutions
- Long term persistence of data is defines through policies that the application can set with the data grid.
- Additional distributed data management features like data affinity, integration with compute grid etc are available.

IGF works closely with the compute task scheduler by feeding it with data locality type of information - thus altering the sequence of tasks based on its knowledge of data availability. IGF has been successfully deployed in various business systems.

# Chapter 6

## Issues in Data Management

### 6.1 Data Delivery

Data delivery without strong decoupling between the data grid and the compute grid - Most of the techniques discussed above require that the data grid and the compute grid act very closely with each other to improve the performance of the application. This imposes restrictions on the design of the data grid. Data management in such cases is very much dependent on the way the compute grid is scheduling tasks and fetching data. Thinking about the data grid as an independent machine gives us the freedom of designing it better. Data search in the data grid : Data distribution and location on the data grid plane is a challenge. Locality based data availability also forces that data be transferred over the data grid closer to the requesting compute location. Also dynamic scheduling of jobs makes reuse of delivered data difficult. In this scenario if data can be searched faster can reduce the delay and also the prefetch traffic on the data grid. Data organization and efficient search thus go together.

Traditional implementations of data grids do not necessarily work efficiently - main reasons being :

- Data is tightly coupled to computation.
- Data is cached locally - distribution is haphazard and reuse is minimal
- Data movement is by data pulling - rather than data delivery on demand/forecast
- Are most effective on smaller systems

## 6.2 Datacentric grids - a need

Datasets are getting larger, more distributed and are often times immovable. Most data is only used when it is processed and never before or after. It will be more effective if we could send the program to the data instead of otherwise. Thinking about datacentric way of computation may be the only way to solve the problem of immobility. This datacentric way of looking at computation has some advantages [17, 18]

1. Users can be productive even from a thin client (whereas computational grids assume a fairly thick client)
2. Applications require only thin pipes within the internet
3. Code mobility is essential
4. The format and content of a data repository will often be unknown to an application until it actually starts (locally) accessing it
5. Generic programming will be predominant - the algorithm does not depend on type of data
6. Applications will tend to be standardized (but run on user specific datasets)
7. Applications will often be built from templates, perhaps even expressed using a query language
8. Re-execution of an application on a different or updated dataset will be common - analyzing weather data for a different region, analyzing stock information for a different exchange
9. There will be increased sensitivity about information leakage - most of the data can be sealed.
10. As a result, applications are likely to run inside Virtual Private Grids (VPGs), but will need to access public resources without leaking (meta)information

The killer application for datacentric grids is parallel and distributed data mining. Some examples:

1. Customer data collected at different touchpoints (web site, toll free number, store)
2. Using references in different online documents to measure interest in a topic

3. Analyzing galactic properties from multiple online catalogues
4. Investigating aircraft part failures by examining maintenance history in worldwide facilities

The above argument strengthens the fact that a lot needs to be done to get to ideal data grid design. With enough motivation for research in this area we further look at technologies that will affect and become important in the data grid design.

# Chapter 7

## Research directions in data management

### 7.1 Data Grid as an extended DBMS

Databases evolved as the need was perceived to manage data efficiently and transparently. The idea of storing and retrieving data irrespective of the requesting mechanism. Data retrieval thus became a mundane task and the user community shifted focus to using data rather than worrying about how to write to the disk and writing custom drivers to do that. Maintaining such drivers/applications was not a trivial task either.

Databases solved the problem by providing a complete data management solution for data intensive applications. In similar sense a data grid is intended to provide separation between the user computation and data required to complete the computation. This reduces the burden on the computation engine (a compute grid in this case) and allows it to concentrate on what it does best- computation. A data grid is designed specifically to provide data delivery services which combine tasks like data caching, management, transparency, storage and so on.

Grid-DBMS is a system which automatically, transparently and dynamically reconfigures at run-time components such as Data Resources, according to the Grid state, in order to maintain a desired performance level. It must offer an efficient, robust, intelligent, transparent, uniform access to Grid-Databases[19, 9]. By dynamic reconfiguration we mean:

- Data source relocation (a Data Source can be moved from one node of the grid)

- Data source replication (a Data Source can be replicated on different nodes of the grid)
- Data source fragmentation (a Data Source can be partitioned on different nodes of the grid)

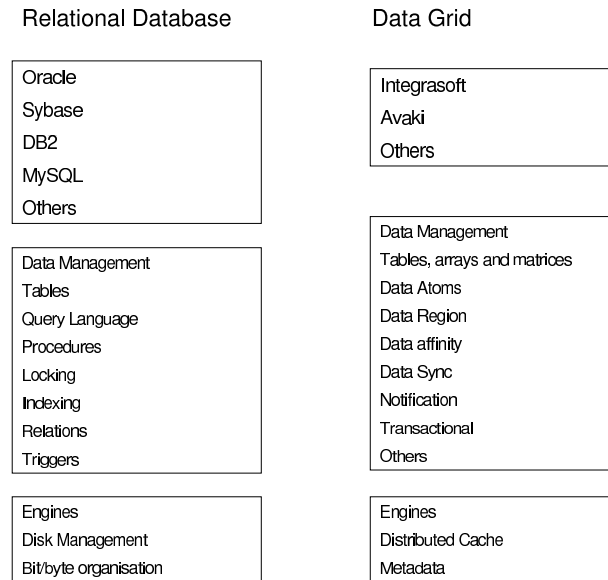


Figure 7.1: Data grids and Relational Databases

The Grid-DBMS must provide the following basic requirements:

- **Security:** data security is a fundamental requirement of a Grid-DBMS that aims at protecting data against unauthorized accesses.
- **Transparency:** it refers to separation of the higher level semantics of a system from low-level implementation issues. There are various possible forms of transparency within a distributed environment, so a Grid-DBMS must hide many implementation details strongly connected with: physical data location: the user must know nothing about the physical location of a database on the grid. This way mechanisms which move (totally or partially) a data source can be entirely transparent to the user (data relocation transparency); network: considering distributed databases we need to properly handle the network and a Grid-DBMS must also hide any details connected with it; data replication: replication of data improves performance, reliability and availability of the entire system. It is worth noting here that the user must not be aware of the existence of multiple copies of the same logical information.

- **Data fragmentation:** fragmentation of data consists of dividing database information into smaller fragments and treating each one of them as a separate unit. This process allows the system to improve global performance, availability and reliability. Moreover, fragmentation increases the level of concurrency and therefore the system throughput.
- **DBMS heterogeneity:** An increasing number of applications interact with not relational databases (e.g., flat files in the bioinformatics field). The Grid-DBMS has to conceal this heterogeneity (different back-end errors, APIs, data types, physical support, etc.) providing a uniform access interface to data sources, that is, performing data virtualization. This way the access mechanism will be independent (transparent) of the actual implementation of the data source.
- **Easiness:** the Grid-DBMS must provide an easy solution for accessing data sources in grid environments. Easiness both from the developer and the administrator point of view must be provided by rich, automatic and powerful instruments (such as libraries, high level configuration tools, console of management, etc.).
- **Robustness:** it represents a fundamental key factor in a distributed environment. Indeed stability is a basic requirement for such components which provide access to and interaction with data sources.
- **Efficiency:** from the performance point of view the Grid-DBMS must provide high throughput, concurrent accesses, fault tolerance, reduced communication overhead, etc. Three factors have strong impact on the performance of a Grid-DBMS:- data localization, that is the ability to store/move data in close proximity to its point of use. This can lead to: reduced query response time (due to data fragmentation) and better exploitation of distributed resources (using for different portion of a database, different CPUs and I/O services); - query parallelism due basically to data distribution (intra-query parallelism) and concurrent accesses to the Grid-DBMS (inter-query parallelism); - high level queries: in grid environments new kind of queries can improve global performance (reducing connection time and/or amount of data transferred) exploiting advanced and efficient data transport protocols (i.e. protocol supporting parallel data streams), compression mechanisms, and so on.
- **Dynamicity:** as we stated in our definition a Grid-DBMS must be dynamic in the sense that ac-

ording to the state of the grid resources, it has to reconfigure its components (data sources) in order to provide high performance, availability and efficiency. So, relocation of data sources, fragmentation (which consequently leads to the fragment allocation problem) and replication of databases are the three basic pillars which can be jointly used by the Grid-DBMS to perform more complex and dynamic data management activities.

- **Intelligence:** A Grid-DBMS must provide some intelligent components (i.e., smart schedulers) in order to carry out the dynamic mechanisms cited before.

In the Grid-DBMS architecture that we envision, we basically need two kinds of schedulers: a data-scheduler, which must perform:

- relocation of data sources;
- replication of the data sources;
- fragment allocation,

and a query scheduler, which has to:

- provide a distributed query optimization engine which must find, in grid environments, the best node (computational resource) on which critical operations (join, semi-join, union, cartesian product, etc.) can be performed;
- choose, from a set of replicated catalogues, the best replica of the dataset to use, in order to maximize throughput, provide load balancing, minimize response time and communication overhead. To support decision making processes (scheduling activities, self-diagnosis tool, etc.), the schedulers have to retrieve both static and dynamic information from a Grid Information Service (local information about the machines, i.e. CPU, memory, disk), Network Information Service (global information about the network, i.e. bandwidth and latency) and the Replica Performance Monitor (statistical information about query response time related to different replica of the same database hosted on different grid nodes).

Exploiting a Grid-DBMS, an Enterprise can :

- reuse the existing physical framework, optimizing the usage of physical resources in terms of storage and computational power. This way, reusing the EG infrastructure (a low-cost solution), buying new large and expensive computing systems (high cost solution) can be avoided;
- improve the performance of the entire system in terms of efficiency, reliability and availability moving, replicating and distributing data sources over the most performant machines of the EG (the Grid-DBMS is able to support changing workload by exploiting dynamic data resource reconfiguration processes);
- transparently access/join data stored in heterogeneous and widespread data sources providing a real data virtualization;
- easily extend the physical framework adding and/or deleting resources without either turning off the system or activating complex reconfiguration processes;
- automatically monitor all the physical resources (EG) and the data sources (Grid-DB), providing self-diagnosis instruments aimed at reducing the human interaction.

## 7.2 WebServices and Data Grids

### 7.2.1 Grid and SOA

Service Oriented Network Architecture(SONA) - Today a significant increase in computing power is driving a fundamental paradigm shift in technical communication. SONA is characterized as an overlay network, of Internet scale. It is designed to take advantage of virtualized hardware and policy based dynamic resource allocation. The implementation of SONA is a nexus of grid computing and Web Services(SOA). Apart from socio-economic drivers the following technological imperatives will push towards SONA.

- Network Computing Power Explosion - will enable a higher degree of intelligence in the network layer. This will lead to a more complex routing and networking model.
- Moore's law and Metcalf's law - Moore's law states that the number of transistors on a chip doubles every 18 months. Metcalfe's law observes that with the number of users the value of network will

increase exponentially.

- Isomorphism to evolution of previous technologies.
- Grid and Web Services as Manifestation of State Transition - Combined with the above laws and network computing power the grid capabilities and network based computation will be closely interrelated.

Figure 7.2 shows how Web Services technology requires data grid as an important member. Also with the grid technology webservice should be simpler and easier to design and implement.

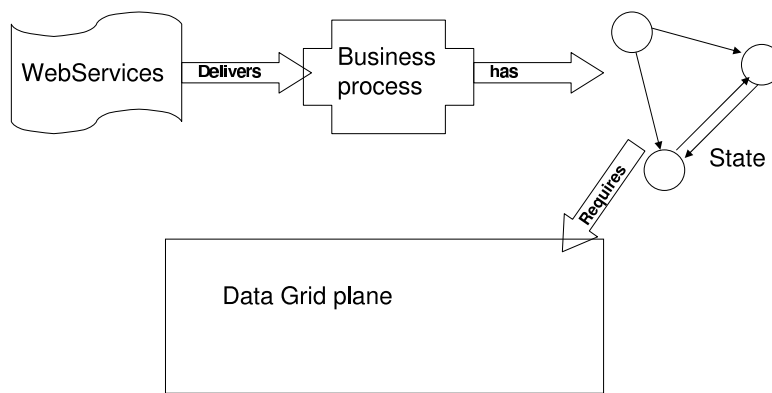


Figure 7.2: Web Services and data grids

Grid computing will thus bring ubiquity of understanding between network computational resources. SOA will move to SONA.

### 7.3 Web 2.0 and Semantic Data Grid

[20] presents an excellent survey of technologies that form the next generation web technology viz. Web 2.0. Many of the web technologies were meant to harness collective intelligence over the web. Hyperlinking is the foundation of the web. As users add new content, and new sites, it is bound in to the structure of the web by other users discovering the content and linking to it. Much as synapses form in the brain, with associations becoming stronger through repetition or intensity, the web of connections grows organically as an output of the collective activity of all web users. Other than the success stories of

Yahoo!, Google, ebay and Amazon new technologies of collaboration have leveraged user participation to succeed.

Some examples include

- Wikipedia, an online encyclopedia based on the unlikely notion that an entry can be added by any web user, and edited by any other, is a radical experiment in trust.
- Sites like del.icio.us and Flickr, two companies that have received a great deal of attention of late, have pioneered a concept that some people call "folksonomy" (in contrast to taxonomy), a style of collaborative categorization of sites using freely chosen keywords, often referred to as tags. Tagging allows for the kind of multiple, overlapping associations that the brain itself uses, rather than rigid categories. In the canonical example, a Flickr photo of a puppy might be tagged both "puppy" and "cute"—allowing for retrieval along natural axes generated user activity Similar pattern can be used to tag weather data with tags like Cincinnati, Coldest day in Eastern US(if found after one such computation).
- "SQL is the new HTML." Database management is a core competency of Web 2.0 companies, so much so that we have sometimes referred to these applications as "infoware" rather than merely software.

In the internet era, one can already see a number of cases where control over the database has led to market control and outsized financial returns. The monopoly on domain name registry initially granted by government fiat to Network Solutions (later purchased by Verisign) was one of the first great moneymakers of the internet. While we've argued that business advantage via controlling software APIs is much more difficult in the age of the internet, control of key data sources is not, especially if those data sources are expensive to create or amenable to increasing returns via network effects. Data is indeed the Intel Inside of these applications, a sole source component in systems whose software infrastructure is largely open source or otherwise co modified. Figure 7.3 shows a picture of what Web 2.0 will look like.

Much as the rise of proprietary software led to the Free Software movement, we expect the rise of proprietary databases to result in a Free Data movement within the next decade. One can see early signs of this countervailing trend in open data projects such as Wikipedia, the Creative Commons, and in software projects like Greasemonkey, which allow users to take control of how data is displayed on their computer.

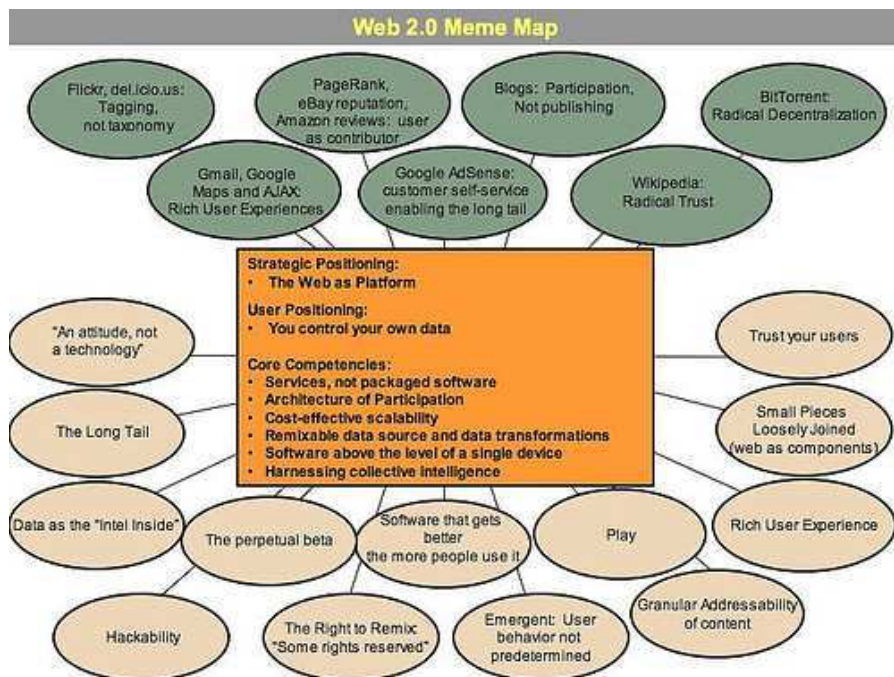


Figure 7.3: Web 2.0

# Chapter 8

## Research Plan

The detailed discussion above provides us motivation to look into data grids as a technology that will change the way computation is perceived. As it is being realized that data is the heart of the system, data grids are receiving the expected interest. We would like to further the research in this direction and would like to study some designs and models of data grids that come to our mind.

The technological improvements discussed in sections 7.3, 5.4 and ?? allow us to improve the way grid computing can evolve. Not only that the requirements mentioned in section 6.2 force us to think of data grids as the being the wheel that drives the distributed computing research.

As discussed above the recent advances in data management make large scale distributed computing possible. We propose to study an architecture of data grid that can be loosely coupled with a compute grid. Our model of data grid closely resembles data grid model as a DBMS. Important differences exist which should make our model more reliable.

We present the problems we propose to look at in our research on data grids in the following sections.

### 8.1 Datagrid with persistence

A data grid as discussed above shows stark similarities with a database management system. A database management system performs similar functions and provides decoupling between data storage and computation. Data search is available by default.

Another layer of data manager is typically used in computing applications with a database - called

a persistence layer. This layer helps the application manage data connections to the database, provides data synchronization without exposing the primitives. Data replication and caching are also available as an integral part of the design.

We claim that a data grid not only performs the function of a data store like a DBMS but also provides other features like replication, synchronization and transactionality. This behaviour closely resembles that of a persistence layer mechanism which is used in application development. Popular designs of persistence layers come in the form of TOPLINK (from Oracle), Hibernate (an open-source software) and many others.

This combination of a database and the persistence layer very closely resemble what a data grid architecture is.

Database persistence mechanisms evolved as an interface mechanism between applications and DBMS. They were designed to move the responsibility of maintaining and managing data transfers away from the application. The application could then be written to focus on the business problem and could delegate the task of data management to the persistent layer. Along with data delivery persistent mechanisms like the now standard TOPLINK provided features like data synchronization, data caching, database connection pooling. Thus the application could be designed without the burden of data management coupled with it.

A grid application faces similar challenges. The datagrid should in this case not only act as a complex DBMS but also provide the services that a persistent mechanism provides. Our research plans to model a data grid as a DBMS with a persistent mechanism. This will separate the design of data storage from data management per se.

Our design for data grid evolves in the following way - Data storage is performed in a seamless way as done in a DBMS as shown in figure 8.1. This includes

- providing simple interface to access data atoms
- indexing data based on various access patterns - use complex indexing techniques as discussed below 7.3.
- data storage and reliability which is transparent to the application.

- fault tolerance - like using RAID disks for data storage.

<b>DBMS</b>		<b>Data Regions</b>
Tables	<b>Schema</b>	Ordered Structure
Triggers	<b>Events</b>	Events
Stored Procedures	<b>Optimizations</b>	Distributed procedures
Intra-table fields	<b>Indexing</b>	Cross-structure
Table/row level	<b>Locking</b>	Data atom level
Table joins	<b>Relation</b>	Data atom
SQL	<b>Query</b>	Programmatic string base
Indexes	<b>Repeated data access</b>	Tags

Figure 8.1: Data regions as Databases

The data persistence layer provides

- data synchronization - by offering primitives to ensure data consistency.
- data replication localized to usage patterns - based on usage the cache maintains a copy of the data until the connection is terminated or it is flushed to a disk to reclaim space.
- data availability - Recently accessed data atom as well as newly requested data are provided using object based interfaces.
- transactionality - by design persistence mechanisms support transactions based on data rows - or joins on multiple tables.
- fault tolerance - replicating data geographically to prevent sudden loss of data.

Figure 8.2 shows how the persistence layer makes the model of a Level 1 data grid complete.

In this research we intend to apply this philosophy to work with a model of data grid that supports these features without being tightly coupled with the compute grid.

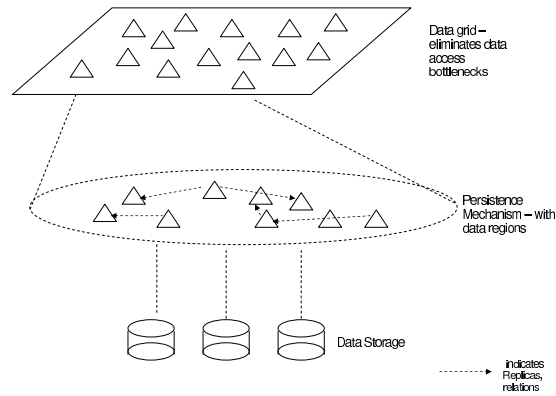


Figure 8.2: Data regions acting as Persistence mechanism

The persistence layer (also termed as an Object/Relational Mapper) is designed to provide separation between business logic and data in an application. Persistence layers should application developers to concentrate on what they do best, develop applications, without having to worry about how their objects will be stored. Furthermore, persistence layers also allow database administrators (DBAs) to do what they do best, administer databases, without having to worry about accidentally introducing bugs into existing applications. With a well-built persistence layer DBAs should be able to move tables, rename tables, rename columns, and reorganize tables without affecting the applications that access them.

Similarly a data grid reduces the burden on compute grids by providing a management mechanism for data movement. Our data grid model acts as a resource manager for data by providing the data grid services. By including the qualities of a persistence layer we hope to enhance the capabilities of the data grid.

## 8.2 Using tags as indexes

Furthermore we would like to study the performance of data search when data tags are used to locate data. Each data atom is tagged after a computation. This tag will be indicative of the features of data and possibly what the salient result of this computation was. Tagging (essentially) indexing should help retrieve that data atom from anywhere in the data grid faster. Also data can be dispersed and grouped based on geography or any other criteria. We would like to study the effect of using these tags on data replication,

synchronization and data search. Imagine a scenario where one weather program calculates the highest temperature for Cincinnati for a day in Summer. Once this calculation has been done that particular data item can be tagged with the tags - Summer, Cincinnati, year, highest temperature. Another program that wants to search for the hottest city in Ohio for summer can look at these tags to determine whether Cincinnati is one of the candidates or not. Tagging will not only reduce computation but also will help easily access data. Another tag that could be added for security which specifies a public key required to access this data. This tag will ensure that no malicious user can force the data to change or even can be prevented from reading sensitive information based on his/her QoS settings.

From the above analysis we see that data management in the primarily read only web environment has made progress to fit to the new user requirements. This is more relevant in the scenario where user data will be read/written and modified on a daily basis. Not only its storage but its security and location become a major issue. We foresee the merger of technologies like SFS 4.3 and tagged indexing as in Web 2.0 to provide complete solution to the data management problems on a data grid. And coupling SFS with tagging (indexing) will provide the benefits of hashing together with indexing in the database. In our research we plan to use the above techniques as applied to the data grid to study the improvements it can provide over plain old caching. Further to this we would like to study the effect of using a version of SFS by using public keys as data atom identifiers. These identifiers will ensure unique keys across the data grid but their scalability in different size of grids. We would like to study the performance of SFS for geographically distributed data.

### **8.3 Data centric design of data grids**

Lastly as mentioned in 6.2 the author [17] elaborates on issues in data management on the grid.

- Large-scale robust data transport coupled with space management
- The space reservation dilemma -incremental allocation?
- Co-scheduling of compute and storage resources
- Identify bottlenecks, automatic replication

- Automated space management and garbage collection
- Space and data objects lifetime mechanisms
- Ensure that important data-objects are not lost

Allocation and authorization issues

- Space management and allocation
- Managed by virtual organizations
- Allocating quotas, enforcing and reporting resource usage
- Authorization management and enforcement
- Data-objects authorization
- Centralized? How to coordinate authorization updates?
- Performance prediction Performance prediction
- Its not only the space availability and the network speed
- Its also the I/O allocation at the storage system
- And also estimating access from Magnetic storage (it is on tape/disk?)

5.4 discusses task scheduling based on data availability. We would like to study task scheduling based on data affinity as well as compute grid restructuring based on data locality. This will help loose coupling the data grid and the compute grid. The data grid and the compute grid exchange status information to understand each others capabilities and requirements. With this information one can imagine that a node in the data grid can force a job to be scheduled based on certain available resources. Now not only does the data grid influence computation but it can also control it to some extent, thus making the system more data centric.

We plan to dig deeper into these issues to evolve the computing to be more data centric. Moving the computation closer to data is one of the options that seems feasible. In the course of our simulations

as discussed in the section below we feel we would be able to resolve some of these issues. Also data delivery traditionally means delivery data on demand. We also would like to consider the effects of changing the computation model based on data availability as well as changing data grid patterns to suit geographical requirements of data. This will improve the traffic distribution and we hope to derive better results from this experiment.

## 8.4 Simulation

The availability of large scale system locally may not be feasible for us. Hence we would depend on the freely available simulation environments to complete this study. The simulation environments discussed below are fairly complete simulators of computational grids. They lack the primitives for mapping a data grid model but as they are extensible we hope to be able to include them.

- **GridSim [7]:**The management resources and scheduling of applications in such a large-scale distributed systems is complex undertaking. In order to prove the effectiveness of resource brokers and associated scheduling algorithms, their performance need to be evaluated under different scenarios such as varying number of resources and users with different requirements. In Grid environment, it is hard and even impossible to perform scheduler performance evaluation in a repeatable and controllable manner as resources and users are distributed across multiple organizations with their own policies. A Java-based discrete-event Grid simulation toolkit called GridSim was developed to overcome this. The toolkit supports modeling and simulation of heterogeneous Grid resources (both time and space-shared), users and application models. It provides primitives for creation of application tasks, mapping of tasks to resources and their management. To demonstrate suitability of the GridSim toolkit, the authors have simulated a Nimrod-G like grid resource broker and evaluated the performance of deadline and budget constrained cost- and time minimization scheduling algorithms.
- **SimGrid [6]:**Simgrid provides a set of core abstractions and functionalities that can be used to easily build simulators for specific application domains and/or computing environment topologies. Simgrid performs event-driven simulation. The most important component of the simulation pro-

cess is the resource modeling. The current implementation assumes that resources have two performance characteristics: latency (time in seconds to access the resource) and service rate (number of work units performed per time unit). Simgrid provides mechanisms to model performance characteristics either as constants or from traces. This means that the latency and service rate of each resource can be modeled by a vector of time-stamped values, or trace. Traces allow the simulation of arbitrary performance fluctuations such as the ones observable for real resources. Fortunately, traces from real resources (e.g. CPUs, network links) are available via various monitoring tools. In essence, traces are used to account for potential background load on resources that are time-shared with other applications/users. This trace-based model is not sufficient to simulate all behaviors observable in real distributed systems. However, this first implementation is a first step in that direction and is already a great improvement over current simulation practice for scheduling algorithms evaluation.

The simulation environments discussed above are open-source and extensible. We plan to build a data grid simulation layer on these that will help us with the research. Also these simulation environments can work with different types of trace data that is available from some scientific research institutions. The data access patterns and computation complexities are representative of various workloads.

## **8.5 Data access patterns and traces**

We have managed to obtain data traces of three months worth of sanitized accounting records for the 128-node iPSC/860 located in the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center [21]. The NAS facility supports industry, academia, and government labs all across the country. The workload on the iPSC/860 is a mix of interactive and batch jobs (development and production) mainly consisting of computational aerospace applications. These will be our reference traces for large scale grid simulations.

The research in [22] discusses and provides traces for cluster computers in two educational institutions. These traces will provide us a starting point to study some standard cluster computing applications. A good analysis of generating and using CPU workload traces is also explained in [22]. The authors here also have access to various data traces.

## 8.6 Issues/Roadblocks in completing the research

Our limited knowledge of compute grids and their resource management policies may lead us along the wrong paths. Difficulties in simulating large scale computation problems still exist with GridSim. Scalability and extensibility of GridSim is still not completely known. We feel that this limited knowledge about the simulation environments can be a significant roadblock in completing this research.

Verifying the results with the real life applications will be difficult since a real compute grid is not accessible. We will be relying on our simulation environments and the related research available around to backup our results. Discrepancies in published research may mean that we have to find ways to validate our own results and reason them.

## 8.7 Expected Results

At the end of this research we hope to answer the following questions

- How difficult is it to move computation closer to the data ?
- What are the effects of using a data grid layer that includes the persistence layer?
- How is the data delivery performance affected by the use of frequently updated tags?
- How scalable is the SFS for data grids which have to manage geographically distributed data?  
How heavy is this mechanism when a number of independent clusters combine to form a grid - over traditional unix file naming systems?
- Can we design and implement a data grid that is completely or loosely coupled with the compute grid - since in most of the cases data grids are tightly coupled with compute grids?

## 8.8 Conclusion

Having motivate the research in data grids we hope to achieve the expected results through simulating various models of data grids. We hope to build a better data grid model that makes computation data centric - thus reinforcing the fact that data is the most important part of any IT system.

# Bibliography

- [1] F. Doswell, M. Abrams, and S. Varadarajan, “The effectiveness of cache coherence implemented on the web,” July 26 2002.
- [2] I. T. Foster, “The anatomy of the grid: Enabling scalable virtual organizations,” in *CCGRID*, pp. 6–7, 2001.
- [3] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, CA: Morgan Kaufmann, 1999.
- [4] “Gridbus.org.” <http://www.gridbus.org/>.
- [5] J. Joseph and C. Fellenstein, *Grid Computing*. NJ: IBM Press, 2005.
- [6] H. Casanova, “Simgrid: A toolkit for the simulation of application scheduling,” 2001.
- [7] M. Murshed, R. Buyya, and D. A. Abramson, “Gridsim: A toolkit for the modeling and simulation of global grids.,” Tech. Rep. 2001/102, 2001.
- [8] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, “The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets,” Mar. 02 1999.
- [9] M. D. Stefano, *Distributed Data Management for Grid Computing*. NJ: John Wiley & Sons, June 2005.
- [10] “Network file system version 4.” <http://www.ietf.org/html.charters/nfsv4-charter.html>.
- [11] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel, “Separating key management from file system security,” in *SOSP*, pp. 124–139, 1999.

- [12] E. Freeman, S. Hupfer, and K. Arnold, *JavaSpaces: Principles, Patterns, and Practice*. Addison-Wesley, 1999.
- [13] Sun Microsystems, “Javaspace service specification.” <http://java.sun.com/products/javaspaces>, Oct. 2000.
- [14] A. Vahdat, T. Anderson, and M. Dahlin, “WebOS: Operating system services for wide-area applications,” in *High Performance Cluster Computing* (R. Buyya, ed.), vol. 2, Programming and Applications, pp. 225–245, Upper Saddle River, NJ: Prentice Hall PTR, 1999. Chap. 11.
- [15] J. Kubiawicz, D. Bindel, Y. Chen, S. E. Czerwinski, P. R. Eaton, D. Geels, R. Gummadi, S. C. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Y. Zhao, “Oceanstore: An architecture for global-scale persistent storage,” in *ASPLOS*, pp. 190–201, 2000.
- [16] D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, and J. Kubiawicz, “Oceanstore: An extremely wide-area storage system,” June 01 2000.
- [17] A. Shoshani, “Data management and transport data management and transport.” <http://www-fp.mcs.anl.gov/nc2004/Presentations/Shoshanilab.pdf>.
- [18] D. Skillicorn, “The case for datacentric grids.”
- [19] G. Aloisio, M. Cafaro, S. Fiore, and M. Mirto, “The grid-DBMS: Towards dynamic data management in grid environments,” in *ITCC (2)*, pp. 199–204, 2005.
- [20] T. O’Reilly, “What is web 2.0 (design patterns and business models for the next generation of software).” <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>.
- [21] B. Nitzberg, “Nas ipsc/860 workload data 4q93.” <http://www.nas.nasa.gov/home.html>.
- [22] P. Dinda and D. O’Hallaron, “Realistic cpu workloads through host load trace playback,” in *5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers, Rochester, NT, May, 2000*, 2000.