Vijayshankar Raman, IBM Almaden Research Center
Inderpal Narang, IBM Almaden Research Center
Chris Crone, IBM Silicon Valley Lab
Laura Haas, IBM Silicon Valley Lab
Susan Malaika, IBM Silicon Valley Lab
Tina Mukai, IBM Silicon Valley Lab
Dan Wolfson, IBM Silicon Valley Lab
Chaitan Baru, San Diego Supercomputer Center

Services for Data Access and Data Processing on Grids

Status of This Memo

This memo provides information to the Grid community concerning data management. Distribution is unlimited.

Copyright Notice

Abstract

An increasing number of grid applications manage data at very large scale, of both size and distribution. In this paper we discuss data access and data processing services for such applications, in the context of a grid. The complexity of data management on a grid arises from the scale, dynamism, autonomy, and distribution of data sources. The main argument of this paper is that these complexities should be made transparent to grid applications, through a layer of virtualization services. We start by discussing the various dimensions of transparent data access and processing, and illustrate their benefits in the context of a specific application. We then present a layer of *grid data virtualization services* that provide such transparency and enable ease of data access and processing. These services support federated access to distributed data, dynamic discovery of data sources by content, dynamic migration of data for workload balancing, parallel data processing, and collaboration. We describe both our long-term vision for these services and a concrete proposal for what is achievable in the near term. We also discuss some support that grid data sources can provide to enable efficient virtualization.

Contents

Vijayshankar Raman, IBM Almaden Research Center
Inderpal Narang, IBM Almaden Research Center
Chris Crone, IBM Silicon Valley Lab
Laura Haas, IBM Silicon Valley Lab
Susan Malaika, IBM Silicon Valley Lab
Tina Mukai, IBM Silicon Valley Lab
Dan Wolfson, IBM Silicon Valley Lab
Chaitan Baru, San Diego Supercomputer Center

DAIS Working Group   February 9, 2003

1. Introduction

Grid computing began with an emphasis on compute-intensive tasks like prime number finding, key cracking, and so on. These applications benefit from massive parallelism for their computation needs, but are not data intensive; the data that they operate on does not scale in proportion to the computation they perform. In recent years, this focus has shifted to more data-intensive applications, where significant processing is done on very large amounts of data. These applications fall into two categories, based on how they use the grid:

For Collaboration: In these applications, data is generated at a large number of sites, and needs to be accessed and processed in a unified fashion. For example, the Digital Radiology project at the University of Pennsylvania proposes to share mammogram information across 2000 hospitals in the U.S.A. [i]. e-Diamond is a similar project in the U.K. In the CERN CMS project scientists all over the world will analyze data generated from particle accelerators and share the results of their analysis with other scientists [ii]. There are many proposals for "World Wide Experiments" in astronomy, earth-observation, life sciences, etc., where expensive experimental equipment is installed at a few places and the experiment is observed and controlled by scientists across the world [iii,iv].

For Scalability: In other applications, a large amount of data needs to be processed, but the data is generated from only a few sites. The task is highly data parallel, in that each part of the data can be processed largely independent of the other parts, with little communication. So the task can be easily parallelized by spreading data across grid nodes. Many data analysis workloads often exhibit such data parallelism. Another example is a Web server running an Online Transaction Processing (OLTP) workload. Web applications often have bursty loads that can saturate at the underlying database server. These peak loads can be dynamically offloaded to "mid-tier" caches at the Web Application Server [v].

The goal of this paper is to investigate grid services for meeting the data access and data processing needs of such applications. We start by discussing these needs.

1.1   Data Access and Data Processing on Grids

As discussed in [vi,vii], the fundamental value proposition of a grid is virtualization, or transparent access to distributed compute resources. For a data-intensive application to derive value from a grid, this virtualization needs to include distributed data sources as well. Two sets of transparencies are relevant to data. First is *transparencies for data access* that present a unified view of grid data sources to applications accessing data. Second is *transparencies for data processing* that present a unified view of grid compute resources to applications processing data.

1.1.1   Transparencies for Data Access
Heterogeneity Transparency

An application accessing a data source should be independent of the actual implementation of that source, so that both can evolve independent of each other. To achieve this independence, the application should program to a logical schema (view) that is separate from the implementation. In the conceptual relational data model, the implementation comprises the physical data organization and the layout of data across tables. However, in practice data sources exhibit many other vagaries of implementation as well, such as the data model used (e.g. relational DBMS vs. LDAP directory) and the specific software product used.

Name Transparency

A common principle in distributed systems is that application be transparent to data location. With grid sources, applications should not even *specify* data objects explicitly. Instead, data access should be via *logical domains*, qualified by *constraints on attributes* of the desired object. For example, in the digital radiology project, a doctor may want to find records of all patients in a specific age group, having a specific symptom. The doctor does not know which hospitals have such patients, and his query

mentions only "Patients", a logical domain spanning 2000 hospitals. We propose that the query processor use a separate *discovery service* to map from logical domains to data sources.

Name transparency automatically subsumes the traditional notions of *location* and *replication* transparency. Since the application does not even specify sources explicitly, the sources can migrate or be cached or replicated without changing the applications. As part of the data access request, applications can specify properties like proximity, staleness, and cost, and suitable replicas should be chosen automatically. Note that these properties are simply another kind of constraint on the data object, along with the content-based constraints discussed above.

Ownership & Costing Transparency

 If grids are successful in the long term, they will evolve to span organizational boundaries, and will involve multiple autonomous data sources. As far as possible, applications should be spared from separately negotiating for access to individual sources, whether in terms of access authorization, or in terms of resource usage costs.

1.1.2   Transparencies for Data Processing
Parallelism Transparency

An application processing data on a grid should automatically get the benefits of parallel execution over grid nodes. The application should only have to specify the dependencies among the tasks it needs to execute, in a declarative workflow language. A workflow coordination service should automatically orchestrate this workflow in a parallel fashion, taking care of data movement, node failures, etc. In cases where the processing consists of traditional data management tasks like Online Transaction or Analytic Processing (OLTP or OLAP), the parallel DBMS should automatically expand (shrink) by adding (removing) nodes in response to workload fluctuations.

Distribution Transparency

Applications should be able to maintain distributed data in a unified fashion. This maintenance involves several tasks, such as ensuring consistency and data integrity, auditing access, taking backups, and so on.

1.1.3   Performance Tradeoffs in Virtualization
Virtualization almost always involves loss of performance. Since many applications may use a grid primarily for high performance, all these transparencies should be discardable. Virtualized access should be the default but not the only behavior. An application that wants high performance should be able to directly access the underlying sources, e.g., in order to apply optimizations specific to a particular data format.

1.2   An Example

To understand the power of such transparent access, consider a worldwide grid of hospital information systems, containing patient records such as hospital visits, medication history, doctor reports, x-rays, symptoms history, genetic information, etc. Such a grid could enable a variety of useful tasks. We outline a few examples below:

Personal Health Digest: A patient forms an integrated view of his medical records scattered across various hospitals. An insurance agency could use such views to estimate policy risks and decide on premiums.

Progress Comparison: A patient compares her progress on particular symptoms with that of other similar patients, to evaluate her doctor and her hospital.

Biohazard Detection: A public health agency periodically mines patient records from various hospitals in a geographical area, to detect biohazards, e.g., it could check if all hospitals within 10 kms of a certain river have experienced a spike in a particular symptom, to detect river water contamination.

Computer Aided Diagnostics: A doctor compares a given patient's symptoms with that of other patients around the world, to diagnose diseases. This can be especially helpful for diseases that are uncommon in that region and therefore unfamiliar to the doctor.

Direct Marketing: A pharmaceutical company tries to access names and addresses of patients taking a certain medication, so that it can send them a coupon for its alternative drug. An authorization service automatically checks if the patients have "opted in" to this service, and denies access to records of those who have not.

Pharmacogenetics: A researcher studies patients with common characteristics to study the efficacy of various treatments on classes of people. The analysis is compute-intensive, but the data and computation are dynamically distributed among multiple nodes on a grid.

The challenge in performing these tasks is that hospital information systems are distributed, heterogeneous, and autonomously administered. Patient information is independently entered at different hospitals, which bear responsibility for the security and privacy of this data. The goal of grid data services is to tackle these challenges and present a unified view of this data, so as to enable tasks such as the above. We will use this hospital grid as a running example throughout this paper.

1.3   Overview of the Paper

The level of transparency described above is far beyond the capabilities of current data management technologies. Nevertheless, we think that this is the promise of the grid, and is worth pursuing as an end goal. In the rest of the paper we present an evolutionary approach towards this goal.

We start the paper with a discussion of current data management technologies, the levels of transparency they provide, and their limitations when applied to the grid context (Section 2).

We then present a layer of *grid data virtualization services* to be developed over the long term (Section 3). This layer lies between grid applications and grid data sources, and masks the distributed, heterogeneous, and autonomous nature of grid data sources to grid applications. The Open Grid Services Architecture (OGSA) [6] proposes a layered stack of services for transparent access to compute resources. The data virtualization services we propose fit mainly in the data management layer, but interact closely with other OGSA services.

After discussing this long-term goal we make a concrete proposal for virtualization services to be implemented in the near term (Section 4). We also discuss support that the grid data sources can provide for effective virtualization. We briefly describe some related grid projects in Section 5 and conclude in Section 6.

2.   Current Data Management Technologies and Limitations

Data management technologies such as Database Management Systems (DBMSs) and file systems have been studied for several decades, and are well established. Grids introduce new challenges like scale, heterogeneity, distribution, and autonomy. However, current data management software already address these challenges to some extent, especially heterogeneity and distribution transparency. We now discuss this support in detail.

2.1   File Systems and Single Site DBMSs

The lowest level data access interface that most applications deal with is the file open/read/write/close interface. Distributed file systems enhance this interface with some location transparency, mapping logical file names onto file locations. However, all further data processing must be explicitly programmed by applications.

 DBMSs have long provided a higher level interface to manage structured data. This interface virtualizes the details of the physical data organization, and provides applications with a high level, *declarative* query language: *SQL.* Applications only need to specify *what* task they want to perform,

and do not have to program *how* the task is to be performed; the DBMS optimizer automatically searches among several possible implementations and chooses the best one. Besides declarative access, DBMSs also provide several business-critical features such as transactional updates, data integrity, reliability, availability, high performance, concurrency, parallelism, and replication.

Traditionally, DBMSs have focused on structured data, laid out in tabular form. In recent years, they have been extended to provide support for non-tabular data, including large objects, abstract data types, user-defined functions, and so on [viii,ix]. XML [x] has now become popular not only as a markup language for data exchange, but also as a data format for semi-structured data. Therefore DBMSs have been adapted to store XML data, either shredded into relational rows (without the markup), or stored intact in the tables (with the markup). A new query language for XML called XQuery [xi] is under draft design in the W3C. Prototype implementations for XQuery are gradually becoming available, including implementations that query over XML views of relational data [xii].

2.1.1   Integrated Access to External Data

A recent advance in DBMS technology has been to allow data stored in the database to be linked with data stored external to the database [xiii, 16]. The most common use of this is to integrate file data with database metadata. This coupling allows an application to run sophisticated queries against the DBMS to determine files of interest, *without slowing down the subsequent accesses to the linked files* [xiv] (Figure 1).
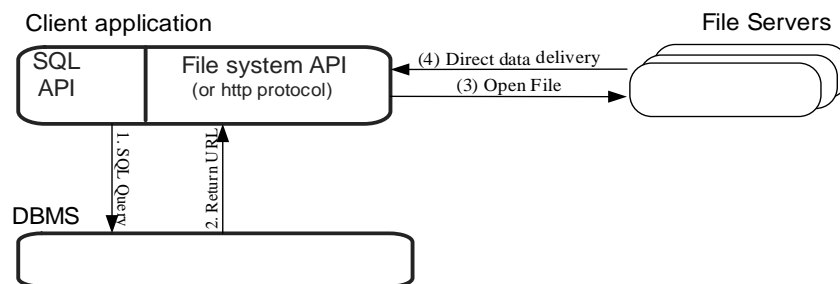

Figure 1: Steps involved in integrated file and database access

Individual columns in a table are declared as *DATALINK* columns [xv] to indicate that they store links to externally stored files. The value stored in a DATALINK column is a URL, which is provided along with values for other columns when a row is inserted. A DATALINK column can be annotated with access rules for read or update operations to the linked files, based either on database authorization or on file system ACLs. For the database authorization option, the DBMS may augment the URL with the access-token information, which can be validated by a servlet or an intermediary on the fileserver, before the file can be accessed. Thereby access policies to both data and metadata can be set and controlled at a single point [13]. A DBMS can also coordinate the access to files and database metadata in other ways, such as referential integrity enforcement, unified replication, and coordinated backup/recovery. Since these deal with virtualization, we expand on them in Section 3.

The net effect of DATALINKs is a simple federation of structured DBMS data with semi-structured and unstructured file data. The DBMS provides name and location transparency for files access, because an application can specify predicates on file and other metadata, which the DBMS can evaluate and map onto a file handle.

Another form of integration between files and relational data is the ability of DBMSs to index XML data held in files, making it possible to search and locate XML file content speedily through SQL interfaces [xvi].

2.1.2   Data Access Interfaces

There are many programmatic interfaces to access SQL DBMSs, with JDBC (Java Database Connectivity) and ODBC being the most widely used [xvii]. They provide support for a variety of database operations, including SQL query specification and execution, transaction processing, etc. The

popularity of Web Services as a method for programs to communicate with one another has introduced another interface to relational DBMS. It is now possible for Web Service clients to issue SQL requests and to invoke database "stored procedures" (*e.g.,* [xviii]).

ADO.NET is an application level interface to Microsoft's OLE DB data sources [xix]. It enables applications to query databases using SQL, access information in a file store over the Internet, access email systems, save data from a database into an XML file, and perform transactional database operations. ADO also supports a disconnected mode of operation, where clients can work on cached copies of prior query results without a connection to the data source.

Recent work has pointed out some limitations of the aforementioned interfaces for long-running tasks, especially for result delivery; JDBC and ODBC deliver query results in a synchronous, cursor-based fashion, whereas long running applications instead prefer asynchronous result delivery, optionally to third party sites [xx, xxi]. But these interfaces have still further limitations with respect to virtualization.

JDBC and ODBC provide some heterogeneity transparency, handling SQL DBMSs and a few other simple data types (like comma-separated-value files). However, they do not handle many kinds of data sources well, especially ones without database-like query and transaction interfaces. We give examples of such data sources below in Section 2.2.

Support for other kinds of transparency (name, ownership and costing) is even more limited in these current data access interfaces.

2.2   Federated DBMS

Distributed data is the reality in most modern enterprises, even without Grid computing [xxii, 23]. Competition, evolving technology, mergers, acquisitions, geographic distribution, and the inevitable decentralization of growth all contribute to create a diversity of sites and data formats in which critical data is stored and managed. Yet, it is only by combining the information from these systems that an enterprise can realize the full value of the data they contain. Hence, the data management community has developed *federated database technology*, which provides a unified access to diverse and distributed data [xxiii, xxiv, xxv, xxvi].

In a federated architecture, a *federated DBMS* serves as a middleware providing transparent access to a number of heterogeneous, distributed data sources. The federated DBMS provides two kinds of virtualizations to users:

- Heterogeneity transparency, via the masking of the data formats at each source, the hardware and software they run on, how data is accessed at each source (via what programming interface or language), and even about how the data stored in these sources is modeled and managed.

- Distribution transparency, via the masking of the distributed nature of the sources and the network communication needed to access them.

Instead, a federated system looks to the application developer like a regular DBMS. A user can run queries to access data from multiple sources, joining and restricting, aggregating and analyzing it at will, with the full power of a DBMS query language like SQL or XQuery. A user can also update the data, if they have the right permissions at the sources. Yet unlike JDBC, ODBC, or ADO, the data sources in a federated system need not be DBMSs at all, but in fact could be anything ranging from sensors to flat files to application programs to XML, and so on.

2.2.1   Query Execution
A typical federated system is shown in Figure 2. Applications can use any supported interface (including ODBC, JDBC, or a Web service client) to interact with the federated DBMS. The federated DBMS communicates with the data sources by means of software modules called *wrappers* [26]. When an application submits a query to the federated system, the federated DBMS identifies the relevant data sources and develops a query execution plan for obtaining the requested data. The plan typically breaks

the original query into fragments that represent work to be delegated to individual data sources, as well as additional processing to be performed by the federated DBMS to further filter, aggregate or merge the data [13, 15]. The ability of the federated DBMS to further process data received from sources allows applications to take advantage of the full power of the query language, even if some of the information they request comes from data sources with little or no native query processing capability, such as simple text files.
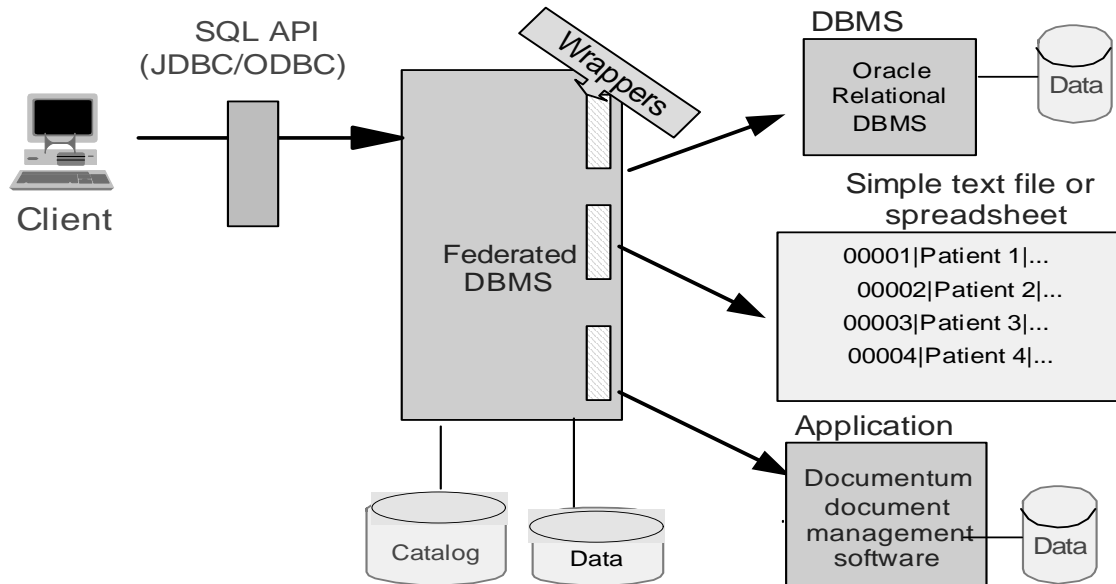

Figure 2: Federated DBMS Architecture

The federated DBMS has a local data store to cache query results for further processing and refinement, if desired, as well as to provide temporary storage for partial results during query processing.

2.2.2   Data Access Mechanisms

A wrapper is a piece of code, packaged as a shared library that can be loaded dynamically by the federated server when needed. Often, a single wrapper is capable of accessing several data sources, as long as they share a common or similar Application Programming Interface (API). The process of using a wrapper to access a data source begins with registration in a system catalog. This is the means by which a wrapper is defined to the federated server and configured to provide access to selected collections of data managed by a particular data source.

Increasingly, Web services mechanisms are being used to access data sources, and data access APIs are being enhanced to operate over Web Services interfaces. Web Service interfaces can make it possible to access data diverse data sources without knowing the underlying implementation, which is an ideal characteristic for the grid. There is also work on integrating the Web Service access interface into a wrapper itself [xxvii].

Figure 2 shows a sample federated system architecture in which the federated DBMS accesses diverse data sources that are shown on the right, which are a traditional DBMS such as Oracle, a simple text file or spreadsheet, and a specialized application, such as the Documentum document management software. A single federated query could perform a join between disease data in Oracle, patient data in a simple text file, and medical journals on diseases in a Documentum data source.

To summarize, a federated DBMS today provides good heterogeneity transparency and some distribution transparency. A key limitation is that applications have to explicitly specify the data sources in a federated query. This means that addition of new data sources involves changing the application, typically a very expensive task. Each data source must also be explicitly registered to the federated DBMS, along with its wrapper. There is not much ownership transparency or location transparency because the application has to be aware of individual sources.

2.3   Security, Accounting, and Billing

DBMSs currently handle security, accounting, and billing on a per data source basis. Data security breaks down into three distinct tasks:

- **Authentication** - Current DBMS systems offer various "sign-on" mechanisms for identifying the user. The simplest of these mechanisms rely on trust; the DBMS assumes that the application above the DBMS authenticates the user. A slightly more robust authentication mechanism allows the application to provide a userid and password (either as clear text, or encrypted). Some DBMSs also support more sophisticated schemes like Kerberos, which use encrypted authentication tickets.

- **Authorization:** Once a user is authenticated, a DBMS must determine what data they may access, and what tasks (query, update, create table, etc.) they may perform. SQL DBMSs can maintain such permissions internally, using GRANT and REVOKE statements. Permissions can be specified on arbitrary subsets of the database defined by *views*. Authorization can be granted at the level of a single user or a group. Authorization can also be managed by an external security service. For example, DB2 for z/OS can use the z/OS SecureWay Security Server (formally known as RACF) or other third party security services (e.g. ACF/2 or TOP SECRET). An advantage of external security services is that they can easily control access to multiple DBMS systems, thus reducing the overhead of managing multiple DBMSs.

- **Auditing:** Most DBMS systems support auditing of access to data to verify that authorizations are being enforced correctly. Audit records can log who accessed the data, and what they did (e.g. User A read 3 rows of data, and updated 5 rows of data in table XYZ), and allow the detection of unauthorized access or attempted access. Most systems allow some degree of tailoring of the events that are audited.

The main limitation of security support in current systems lies in security across multiple sources. Authentication, authorization, and auditing are typically enforced and managed separately for each data source, even for the same user. This results in a loss of ownership transparency because applications have to be separately handle security with each data source.

2.3.1   Accounting and Billing

It is desirable to have a DBMS provide integrated accounting facilities with tools to perform analysis of this data, and help to automate the billing of end users. Usually the actual charge-back and billing process is handled by a separate, often third party, piece of software that takes into account, not only the DBMS usage, but also things like OS, Network, and Storage resource usage.

2.4   Maintenance of Distributed Data

Besides supporting query and update operations, DBMSs also provide various other facilities to manage data, such as backup and recovery, replication, data integrity enforcement, etc. Grid applications often require such management facilities, across multiple data sources. For example, the Encyclopedia of Life project at the San Diego Supercomputer Center [xxviii] has an analysis pipeline that reads data from multiple biological databases, in order to produce putative assignments of biological function for all recognizable proteins in the currently know genomes. The resulting assignments thus have a "referential integrity" relationship with the data in the input databases. Input data could change due to subsequent experimentation and scientific developments, and this needs to related back to the quality of the assignments.
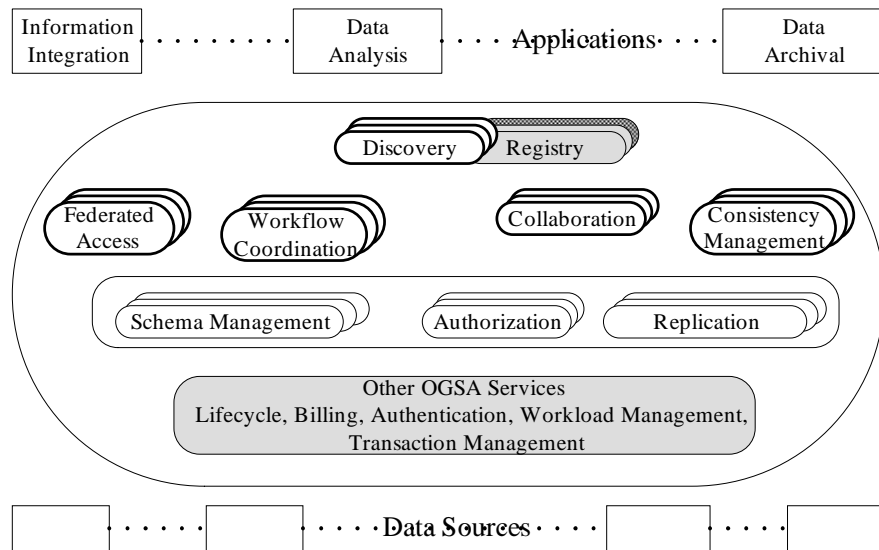
Figure 3: Grid Data Services. Shaded services are not data-specific and are taken from OGSA

3. Grid Data Virtualization Services

Figure 3 shows our data service architecture, with a layer of virtualization services between grid applications and data sources. These services virtualize various aspects of the grid, and make it appear as a single entity to the end-user applications. We believe that most applications will access data through this virtualization layer. For improved performance, many virtualization services could support "pass-thru" modes whereby an application can bypass much of the overhead of virtualization, at some penalty in functionality. Applications could also cache the result of prior virtualization service requests and bypass them on subsequent accesses. For example, an application could cache the result of a prior call to a discovery service, and directly access the underlying grid data source.

In the short term, we propose that even if data sources are autonomous and dynamic, these virtualization services be statically setup and administered. In the long run, we envisage that these virtualization services will evolve into a *networked collection of autonomous entities, with dynamic federation*. They will independently choose the data sources that they virtualize (subject to security and access control constraints), and the level of service they provide based on application demand[1].

Our virtualization architecture is made up of core data virtualization services – discovery, federated access, `workflow` coordination, consistency management, and collaboration – that virtualize various aspects of the grid. These services in turn make use of some auxiliary services: authorization, schema management, and replication. All these services interact closely with a few OGSA services. We have also discussed and reconciled these virtualization services with the overall data grid architecture of the Globus project [51, 52].

Table 1 summarizes the role played by each data service in virtualization. We now discuss each service in turn, starting with the core virtualization services (Section 3.1), then the auxiliary services (Section 3.2), and then their interaction with OGSA services (Section 3.3).

---

[1] For instance, in our hospital grid example, various Health Maintenance Organizations (HMOs) and hospital groups could start virtualization services to cater to particular sets of customers whom they charge for access.

Table 1: Data Virtualization Services and Auxiliary Services

Virtualization Services	Transparencies				
	Heterogeneity	Distribution	Name	Ownership and Costing	Parallelism
Core Services					
Discovery			Maps logical domain + predicates onto actual data source or service Chooses the best location and replica for a query	Optimizes for source-independent metrics like time or cost	
Federated Access	Allows access independent of data format and implementation	Provides unified access to distributed data sources	Provides some location transparency for file access		
Consistency Management		Maintains consistency of distributed data			
Collaboration		Maintains consistency of data updated in multiple places			
Workflow Coordination					Automatically parallelizes request
Auxiliary Services					
Authorization				Single sign-on	
Replication/Caching			Migrates data on demand to right place		
Schema Management	Maps between data formats				

3.1   Core Virtualization Services

3.1.1   Discovery Service

A discovery service virtualizes the name and location of data on the grid, and forms the basic data virtualization service. Other virtualization services build on top of this name transparency to offer other kinds of transparency. As mentioned before, applications specify data sources in terms of logical domains qualified by predicates, rather than giving exact locations. When a virtualization service receives a request involving such a logical domain, it uses a discovery service to map this domain onto actual data source locations (OGSA Grid Service Handles (GSHs) [5]). In our hospital grid example, a query to a Federated Access Service can be "*Find records for patient with specific symptom and in specific age* group." A discovery service is needed to map the logical domain "patient" and the predicates on age and symptom, into locations of data sources (hospitals) that could contain relevant records.

In addition to tracking data locations, discovery services also track the locations of other virtualization services, such as Federated Access Service, that support a particular logical data domain. When there are multiple instances of the same virtualization service, grid applications can use a discovery service to choose one instance based on criteria such as speed, cost, etc.

Of course, there could be multiple instances of discovery services themselves. In the near term, one can implement the discovery service as a single centralized service, and let data sources publish their content to this service. In future, it needs to be made distributed for greater scalability, availability, and autonomy. Each grid application or virtualization services will then access a single discovery service, which in turn automatically spreads the request to other discovery services if needed.

Data can be published and updated at a discovery service through two mechanisms:

Advertising: A data source could explicitly publish its schema and content to one or more discovery services if it believes that the data is of value to many users. It can also publish its capabilities for querying the content (Section 4) to the discovery services. Likewise, a virtualization service could publish the sources it virtualizes. Such explicit publishing is akin to advertising.

Crawling/Gathering: Instead of the data sources advertising their content, a discovery service could periodically crawl one or more data sources or virtualization services, to find out their content and capabilities.

3.1.2   Federated Access Service (FAS)

An enhanced version of a Federated DBMS (Section 2.2) is vital to virtualizing the distributed nature of grid sources. The main value of such a Federated Access service (FAS) is to provide applications with the virtual image of a single database, which can be queried and updated as a unified whole. Since the data is not explicitly moved to a "warehouse", the application always accesses the most current version of the data.

The limitations regarding location transparency (recall Section 2.2) are circumvented when Federated Access Service is combined with the Discovery Service. The application specifies its queries in terms of logical domains and predicates; the discovery service maps these onto relevant sources. Thus, the combination of FAS and Discovery Service provides applications with heterogeneity, distribution, and location transparency.

Negotiation, Ownership and Costing Transparency, and Source Independent Performance Metrics

As mentioned in Section 2.2, there is little or no ownership and costing transparency with respect to the following:

- data source access costs
- data source access permissions
- data source resource usage limits

The way to avoid this burden on the application is through a process of *negotiation* by the FAS. As part of its optimization process, the FAS validates access permissions, estimates resource usage of the query across all sources, and negotiates access costs on behalf of the application [22].

Transformation: The FAS's SQL querying functionality can directly be used to transform data between multiple source formats, for data integration. Such SQL transformation can be made quite efficient since the FAS can optimize the transformations. SQL allows mainly tabular kinds of transformations, but there has been work on extending this functionality [12, xxix].

3.1.3   Workflow Coordination Service

As mentioned in the introduction, a large number of data-intensive grid applications use grids for scalability of performance, rather that for integrating data sources. These applications typically take the form of complex workflows of transformation and data analysis operations, running over large numbers of discrete objects. Since these operations are often data parallel grids can be used to scale up these applications on demand. However, it would be wasteful for each application to develop a separate way of parallelizing its workflow. Instead, we propose that applications only specify their workflow in

terms of inter-operation dependencies. A Workflow Coordination Services will automatically spread these operations across grid nodes, taking responsibility for moving and caching data and functions, recovering from node failures, and so on. In addition, the Workflow Coordination Service can also keep track of the lineage of operations performed on each data object, for provenance management and for avoiding redundant work.

3.1.4   Consistency Management Service

Grid applications often distribute their data across multiple sites, so we need a service that keeps the different pieces consistent with one another.

The simplest form of such consistency is *referential integrity* [xxx]. For example, if data at one site A refers to data at another site B, an application may want to ensure that the data at B is not deleted while the reference at A remains. As we have discussed, DBMSs can provide such referential integrity between data in files, and database records that refer to these files. In addition, some DBMSs can also backup and restore database and file data in a unified manner [14]. A weaker notion of consistency is useful in the Replica Location Service, to maintain integrity between various replica directories [50].

Some grid datasets may also need more sophisticated integrity constraints. These constraints could also vary on an application-specific (user community-specific) basis, if different applications have different (even conflicting) views of the integrity constraints and relationship among distributed pieces of data. In addition to relationships among data objects, it will be important to maintain relationships among data and computational routines as well. For example, if a particular data set is the result of executing a particular program with a given set of inputs, the data set may not be very useful without the program or the inputs.

3.1.5   Collaboration Service

Many data-intensive grid applications involve sharing of data between users at different sites. To propagate updates to all users and to resolve conflicts, we need a collaboration service. The prime role of this service is to virtualize independent, distributed data updates; when a user asks the collaboration service to "checkout" a copy of a data object, she wants the latest version, irrespective of who, where, or when it has been updated (of course, the collaboration service can also maintain history and allow users to ask for prior versions). Clearly, the collaboration service must rely on the grid data sources to maintain version information; we discuss this support in Section 4.2

3.2   Auxiliary Services

3.2.1   Authorization Service

The OGSA is developing a single-sign on mechanism for users and applications to authenticate themselves on a grid. However, as discussed earlier, data sources have their own specialized models for access authorization. We believe that it will be helpful to have an authorization service that maps between the grid authentication and data source specific authorization schemes, much like the third party security packages we discussed in Section 2.3.

3.2.2   Schema Management service

Many current grid applications store their data in fairly standardized and uniform schemas, even across sites. However in the long run there is likely to be schematic heterogeneity, as more and more data sources become available via a grid.

Transforming and integrating data between heterogeneous data formats is a huge problem facing many enterprises today [xxxi]. The problem is not in executing the transformation, but rather in formulating it. Indeed, there is an entire cottage industry of vendors developing Extract/Transform/Load (ETL) tools for data integration [xxxii].

Simple heterogeneities can be resolved through the Federated Access Service itself, as we discussed in 3.1.2. For resolving more complex heterogeneities, we propose that there be a separate Schema Management service on the grid. This service *will not function automatically*. Rather the maintainer of this service will have to manually use ETL tools to develop transformations between various popular

data formats (these are called "mapping metadata" in [xxxiii]), and then expose these transformations as a service that other applications and virtualization services can invoke[2].

Another use of transformation is for data archival. An emerging approach is to transform a given data set into an XML data object with a well-defined schema, and associate appropriate (standardized) metadata with every such object (e.g. see the Archival Information Package defined in the Open Archival Information System (OAIS) [xxxiv]).

3.2.3   Replication Service

Replication Services are responsible for maintaining replicas and caches of data on a grid, in order to utilize the aggregate compute resources available. Consider an application that initially accesses data at a single site. If this site becomes a performance or availability bottleneck, the grid workload manager on that site [6] dynamically invokes a replication and caching service to copy subsets of the data to other lightly loaded sites. The application can now improve its performance by accessing these subset replicas rather than the original. The Federated Access Service can in turn make accesses to these distributed replicas transparent to the application. Effectively the distributed compute resources have been virtualized to the data-intensive application.

A replication and caching service is responsible for initially replicating a dataset in whole or as a subset, and automatically maintaining these replicas in the face of updates to the original. A key requirement is that this operation be completely automatic and independent of the application data format. The invoker of the service should only have to specify the subset to replicate, and the latency (staleness) that can be tolerated.

A replication service can also be used to gather data from multiple sources, rather than to distribute data to multiple targets. In fact, such gathering is necessary for data warehousing, as we shall see in Section 3.4.

3.3   Interaction with other OGSA services

The data virtualization services described above interact very closely with the grid services being defined by OGSA [5], to facilitate heterogeneity, ownership, and distribution transparency. We give some of the main interactions below.

- **Registry:** The OGSA defines registries as places to store various kinds of information about grid resources. The Discovery Service can use these registries to store two kinds of information. The first is mappings from logical domains and predicates to locations of relevant data sources (there may be multiple data sources with identical data, due to replication). This information will be used to discover sources of interest for federated queries, and to exploit replicas whenever available. The second is the query and update capabilities of various data sources. This is quite important for federated query processing, because the FAS needs to know capabilities of various data sources in order to choose a query plan to access these sources.

- **Authentication Services:** Ensuring the security of data will often involve support for multiple authentication mechanisms [xxxv], because local and grid-based security will need to interact due to issues such as legacy security infrastructure, or local security requirements. A grid security infrastructure will need to support many security scenarios [xxxvi]. In order to handle the complexities of grid data management, additional services such as Credential/Identity propagation, Identity Mapping, Single Sign-on, Message level security, Transport level security (e.g. SSL), Message integrity, and Key/Certificate Management will be required to interact in a seamless, well-architected manner to ensure data security and integrity.

---

[2] On each transformation request, the Schema Management service has to decide whether the data should be shipped to the Schema Management service or the transformation should be shipped to the source. This must be made based on the size of data involved, network transfer costs, and the extent to which the transformation function is proprietary and not exposable to the service invoker.

- **Accounting/Billing Services**: As discussed, the FAS must negotiate accounting and billing for a query or transaction with all the various data sources involved in the query or transaction. Although this need is well understood [xxxvii], grid design for accounting and billing are still in the early stages. Some work is being done on establishing service level agreements for billing within IBM [xxxviii], and the ASP Industry consortium has proposed the Common Billing Interface Definition. The grid data access community needs to be involved as these services evolve, and be among the first users when they are available.

- **Notification Service**: Grid virtualization services can use the grid notification service to know about various changes occurring at the data sources, such as:

  - **Schema changes**: This can be important for federated query processing. For example, schema changes of a foreign source may require re-evaluation of a query fragment of a federated query plan. Schema change notification is also important for replication if the target is to be kept consistent with the source.

  - **State changes**: Failures at a data source may have an impact on the processing of many virtualization services. For example, a Federated Access Service needs to know about failure of a query source so that it can adapt gracefully to return partial results involving the remaining sources. A Replication service would use such notifications to know if the source or target of replication have failed, and take corrective action to restart the replication at the failed end.

3.4   Usage Scenario Explaining above Services and their Interaction

We now return to the hospital grid example of the introduction, to illustrate how these data virtualization services can be used and how they interact with each other. We consider each of the processing tasks in turn:

3.4.1   Federated Access Tasks

Figure 4 shows the interaction between different grid data services for performing federated queries against multiple grid data sources. A query is submitted to a Federated Access Service, without specifying the data sources explicitly. The Federated Access Service contacts a Discovery Service to find data sources (hospitals) with relevant data, contacts the Authorization Service to get appropriate authorization tokens, and then queries them.
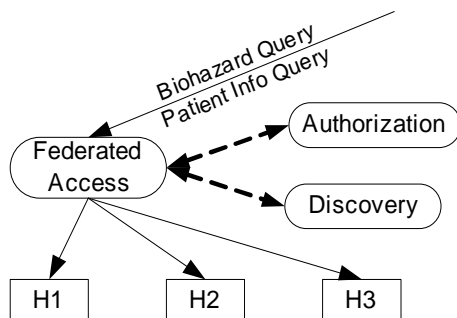

Figure 4: Executing a Federated Query (control flow in dashed lines)
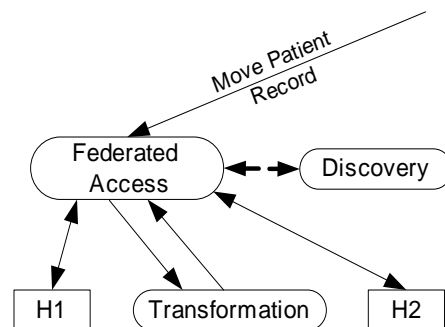

Figure 5: Moving a patient's record between hospitals (control flow in dashed lines)

Two of the tasks in our hospital scenario (Section 1) fall into this category.

- **Personal Health Digest:** The query specifies a patient identifier (e.g. SSN or name) and queries all data sources to get results of lab tests that were requested by any of the patient's dermatologists. The result is a 3-way join of the union of relevant patient records, the union of relevant doctor records, and the union of all lab records.

- **Biohazard**: The query specifies a river, a distance from the river, and a symptom. The Discovery Service finds relevant data sources (in this case, hospitals satisfying the predicate "distance from river < X"). The result is a union of records from these hospitals, qualified by the symptom and distance predicate

3.4.2   Data Movement Tasks

One of the main tasks in our hospital scenario that involves data movement is the Patient Relocation task; when a patient relocates, he wants to move his records to the hospital in his new location. The patient may move between hospitals that store their records in different formats, involving complex transformations[3].

Figure 5 shows the service interaction needed to move data across grid data sources. The movement task is submitted to the Federated Access Service that transactionally (a) reads data from the source, (b) transforms it according to target's schema (or may invoke Schema Management service), (c) inserts it into the target, and d) deletes it from the source.

3.4.3   Federated Data Analysis and Mining Tasks

We now consider data analysis and mining tasks. Unlike federated queries, these are complex and long running, and *need not run on the most current data*. The Medical Research, Computer Aided Diagnostics, and Progress Comparison tasks fall into this category (the biohazard detection task does not, because it is run very frequently and needs access to current data for early detection).
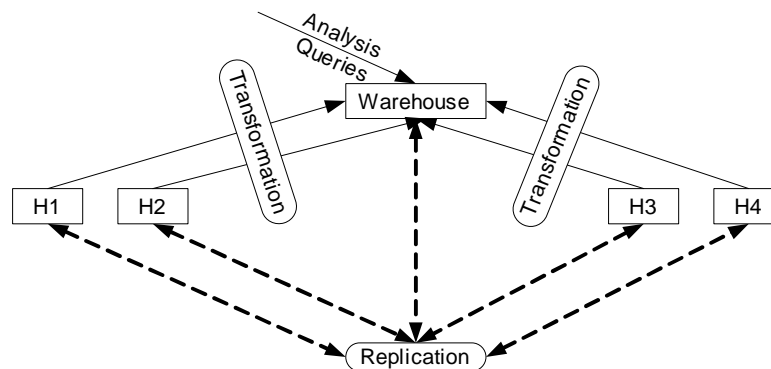

Figure 6: Interaction between virtualization services for warehousing (control flow in dashed lines)

Such analysis tasks are typically not run against the data sources themselves, but instead against warehouses. Figure 6 shows how patient records from hospitals can be aggregated at warehouses using the replication service. Transformation is crucial during warehousing for two reasons: (a) to unify data formats and schemas across multiple hospitals, and (b) to anonymize patient records. Transformation is *not needed* in federated querying (Figure 4), because the Federated Access itself transforms data (via query language operations) before integration. Notice that the data never flows to the replication service, but instead flows directly from hospital to warehouse.

---

[3] The Health Level 7 [3] standard notation for patient and hospital exchange information is being gradually redefined as an XML schema. This XML standard can be adopted as the default data exchange format for such an application; it simplifies the transformation task and also allows sophisticated SQL/XML and XQuery queries to be run directly on the data being moved.

Figure 7 shows a data analysis query run against the warehouse. Since the query is long running and compute intensive, the warehouse gets overloaded. The grid workload manager [6,29] at the warehouse node asks the discovery service for lightly loaded (or additional) nodes, and then invokes the replication service to dynamically partition warehouse data to these. The choice of the columns to partition on can be made by the partition advisor of the warehouse DBMS [xxxix]. The analysis query is automatically run against these replicas through the Federated Access Service.
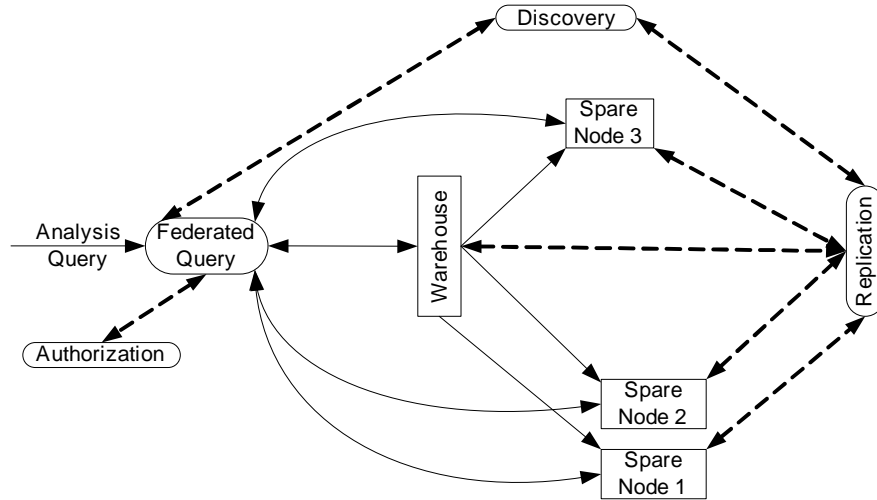
Figure 7: Querying against a warehouse, with dynamic replication (control flow in dashed lines)

4. Proposal for Stage 1 Grid Data Services

The previous sections introduced a grand vision of data virtualization services for grids. We now lay out a plan for virtualization services to be built in the short term (Section 4.1). These *Stage 1 Data Virtualization Services* also need support from the underlying grid data sources, and we discuss these requirements in Section 4.2.

4.1   Stage 1 Virtualization Services

4.1.1   Schema Management and Discovery Service

Most of the data access and processing scenarios we have discussed so far rely critically on data source discovery. For Stage 1 we propose to implement the discovery service as a centralized service that maps pairs of logical domain and query predicate to data sources containing data from that domain and satisfying that predicate[4]. Data sources explicitly advertise their content to this discovery service, and other virtualization services invoke it to find data sources.

We believe that, at least in Stage 1, each grid application will have a separate discovery service. For example, our hospital grid could have a discovery service containing information about hospital data sources alone.

This discovery service also serves as a natural place for schema management. Different data sources may have different ways of describing the same data (schema and content). When sources register

---

[4] Naturally, the number of such predicate mappings will be limited by the storage at the discovery service. Our current plan is to only maintain information for broad partitions of the data domain, than for every possible predicate. For example, in our hospital grid, a mapping could be that "patients with age < 18" get directed to a set of children's hospitals.

with this service, it can map the source schemas into a unified domain schema, possibly using ontologies for domain-specific mappings. Application queries can then be specified against this domain schema, and source schemas can evolve independently.

In the long run, if a large number of data sources access the same discovery service, it could become a performance and availability bottleneck. Moreover, it can be a hindrance for autonomous data sources to explicitly publish data to the discovery service. We plan to generalize the discovery service to a network of peer-to-peer discovery services, extending recent work on indexing in peer-to-peer systems.

4.1.2   Federated Access Service

Federated and distributed querying has been studied in the database community for a long time now, and there have been several research [e.g. 22, 23] and commercial projects [e.g., xl, xli] in this area. Therefore, a basic level of federated technology is quite mature, and merits adoption by grid applications today.

As discussed in Section 2 however, this technology's current focus is primarily on heterogeneity and distribution transparency. For Stage 1 we propose that the Federated Access Service use a DBMS, with the following extensions.

Integrated access to file + DB content

Most grid applications want to manage data stored in files along with structured metadata. In Stage 1, the FAS can use employ a DBMS (this need not be a federated DBMS) to store this metadata, running queries against the database to get URLs, which are then accessed, through regular file system interfaces. This provides location transparency for file access, since the access can be through arbitrary SQL queries that involve predicates on the metadata. The main advantage of declaring columns as URLs is that the application will get integrated access control and integrated replication of database and external data. (see the Appendix for details). The DATALINK [15] data type can also provide referential integrity for URL. If a DBMS does not support the DATALINK data type natively, it can support URLs without referential integrity enforcement, as user-defined types.

To provide heterogeneity, distribution, and location transparency, the FAS must be use a federated DBMS to query and update data. We believe that a few simple improvements to this technology can be made in Stage 1, which will prove of high value to applications.

Dynamic data source discovery

Currently federated DBMSs do not provide location transparency because data sources are explicitly specified in the query. As a first step, the federated DBMS could accept logical domains (for example, these could replace table names in the FROM clause of an SQL query), and invoke the Schema Management and Discovery service to map the domain name and query predicate to actual data source names. Although the Stage 1 discovery service provides only limited functionality, but still allows applications to be independent of data sources. As the discovery service improves, the FAS will benefit automatically.

An important pre-requisite for dynamic source discovery is dynamic loading of the wrappers used to access data sources. Currently wrappers are registered with the federated DBMS in a static fashion, in its catalog. Loading wrappers on demand, from the source or the discovery service, will also improve source autonomy.

In the long term, the FAS can be extended in many ways. One is to perform negotiation with the data sources, as discussed in Section 3.1.2. Another is to gracefully degrade to provide partial results in the event of source failures.

4.1.3   Data Transport and Replication Service

The role of this service is to transport and maintain copies of a data source at one or more targets. These copies could be of subsets of a data source, and could be out-of-date with the data source; such copies are like caches. The copying could be done only once (in which case this is pure

transportation), or at periodic intervals. Many DBMSs have replication products today. However, these are designed for specific DBMSs, and have very complex setup procedures. Our plan is to develop a generic grid replication service that is *independent* of the data source. We plan to support two kinds of replication:

- **Query replication:** Periodically run a query against the replication source and copy results to the target

- **Change replication:** Continually monitor changes occurring at the source and propagate them to the target.

The former allows the target to be a transformed version of the source, and is easier to implement. With a FAS, query replication can join, aggregate or reshape data from several heterogeneous data sources and apply the result to the target. The problem is that it can be wasteful to run the same query at frequent intervals; change replication needs to only propagate the changes that have happened since the last replication.

The main challenges in grid replication are as follows:

- Data Source Independence: To replicate arbitrary grid data sources in a generic manner.

- Consistency: To maintain the replicas in a consistent fashion, in the face of updates to the source.

- Reliability: To restore the replication service to a consistent state after hardware or software crashes at the source, target, or communication networks.

- Flow Control: To provide flow control when the source and the replicas run at varying speeds.

- Quality of Service (QOS): To allow replication to be invoked from application programs, specifying only application-level parameters. For instance, an application may state its desired QOS, and the frequency at which the data must be replicated. However, it should not have to specify the amount of memory or buffer space for replication, or the network bandwidth to be used; the system should automatically allocate and manage these resources.

Our goal for stage 1 is to develop replication as a J2EE container [xlii] that automatically provides the above functionality to arbitrary grid data sources. The replication source needs to provide only a notification interface that exposes changes happening at the source, and the replication target needs to only provide an interface to accept and apply changes sent from the target. All other functionality is handled by the replication service. We intend to support replication of files and metadata in a unified manner by storing URLs in the database (see the appendix for details).

For the long-term, our main goal is to provide quality of service in replication. For example, the application may have a maximum tolerance for replica staleness (e.g. no more that five minutes out-of-date with source). We also want to develop a *cache advisor* that dynamically decides which subsets of each data source are worth replicating, and where, based on the workload of accesses to each source.

4.1.4   Authorization service

For stage 1, we think that the most important aspect of security will be mapping the user credential derived from GSI into data source specific authorization ids. For example, in a DBMS, the grid GSI needs to be mapped to user names and passwords so that database permissions can be accessed. The main problem here is lack of ownership transparency, because the database authorization has to be explicitly mapped from the GSI credentials for each source. In future we can build an authorization service as in Section 3.2.1, to which data sources can publish their access permissions. The FAS can then directly access this authorization service to map credentials.

4.2   Data Source Requirements for Virtualization

Having presented a proposal for virtualization services, we turn our attention to the interface to Grid data sources. We discuss various functionalities that the grid data sources could provide to enable efficient virtualization.

4.2.1   Context Management

Grid data sources are by their very nature stateful services. Their state involves not only the underlying data, but also the context of ongoing queries and updates. The context is passed in along with every request to the data source, and is used to link these requests together. This context needs to be maintained at multiple levels:

- **Session Context**: This includes information about all current connections, such as user name, authentication tokens, billing information, etc. It is tied closely with OGSA's context management service, because the grid application's requested level of resources and quality of service are determined in an end-to-end fashion, across all data sources involved in the task.

- **Transaction Context**: This context maintains properties of the transaction into which the query or update falls. These properties include consistency levels, save point information, distributed transaction state, etc.

- **Command Context**: This contains state that is specific to a single command – query or update. It includes not only the properties of the command itself, but also state associated with the command execution, such as delivery options and delivery handles for the command result, and command checkpoint information (if any).

Many of these contexts are already supported in existing interfaces like JDBC. For Stage 1, we think that the main extensions needed in context handling are:

- Explicit support for a hierarchy of contexts (session, transaction, command), which is coordinated with OGSA's context management service.

- Explicit support for result delivery options and result delivery handles.

4.2.2   Data Access

Access Cost Estimation and Negotiation: As discussed in Section 3.1.2, the FAS needs to negotiate with data sources based on source-independent performance metrics like response time, throughput, or cost. As a first step towards such negotiated access, the data sources can provide two kinds of information for each access request:

- Statistical properties of the requested data, such as cardinality and data distribution

- Access Cost, in terms of response time or throughput

This would allow the FAS to optimize the overall access plan to minimize the total access cost across all the sources in an access request. DBMSs have a long history of cardinality and cost estimation; providing this support might be harder for other data sources like file systems. Another aspect of negotiation is allocation of resources (at the data sources) for executing the request. Unlike in traditional federated DBMSs, the data sources are autonomous and their system properties might have changed between optimization time and run time. So it will be beneficial if the data source supports negotiation of resource allocation decisions at run time.

Versioning: As discussed before, collaboration is an important style of data sharing in grid applications. Therefore it will be beneficial if grid data sources support a *version number* as part of every query (the default being to get the latest version).

Result delivery: Many traditional data sources, including virtually all DBMSs, are designed for synchronous result delivery. On the grid, queries can be long running, and return large amounts of data. It has been pointed out that data sources should support asynchronous result delivery, and result delivery to nodes other than the query requestor [20, 21]. Therefore we suggest that access requests

contain both a description of the work to be performed, e.g. a query, and information about how the results are to be formulated, and delivered. For instance, if the expected result of a query is large, the requestor can ask that results be broken apart into several messages, and delivered asynchronously and in a pipelined fashion, possibly to another node.

If a guaranteed delivery protocol such as MQSeries [xliii] is employed, then the requestor, service provider, and result consumer need not operate concurrently. In a widely distributed environment this can have significant benefits. Applications may start, initiate a request, and then shutdown. At a later time the application could start up again, and process any waiting results. Similarly, the database server need not be immediately reachable by the requestor. An additional advantage of this decoupling is that the database service provider may choose to process a request immediately, forward the request to another service provider, or to delay the processing of the request until a more convenient time.

4.2.3   Data Source Notification

Update notification of a schema is required for FAS and data replication. In addition, replication requires notification of data updates, auditing service requires notification of data accesses (as an option due to its overheads). There is no standard way of requesting notifications in current data source interfaces like JDBC. So we suggest that the grid data source interface contain a function that can be invoked to access a variety of events happening at the data source. These events could include:

- Schema updates (for FAS and replication service)

- Data updates (for replication service)

- Data accesses (for auditing service)

The format of the update delta or audit log record for the notification is entirely up to the data source. The only constraint is that in the case of replication, both the source and target must understand the delta format (likewise for auditing, the auditing service must understand the audit record format). This could be facilitated by using a self-describing format for the delta (say an XML format), though the actual schema would still have to be standardized in the application domain.

5. Related Grid Projects

Most current grid applications use files to manage data. Directory services that need more sophisticated access are typically implemented with LDAP. As applications become more data-intensive, the grid community has taken greater interest in data management.

The Relational Grid Information Systems project is investigating the pros and cons of implementing grid directory services with relational DBMSs [xliv]. The SDSC Storage Resource Broker (SRB) is a distributed, attribute-based file system [xlv].The SRB employs a DBMS to store all its metadata and user-defined attributes. It supports GSI-based security and the ability to invoke remote data filtering operations via a proxy mechanism. It can manage a variety of digital entities including files, directories, SQL command strings, tables in databases, and services. These entities are registered into a logical name space and all operations are performed relative to the logical name space. The SRB manages the mapping from the logical name space to the physical name space that describes where the digital entity is actually stored. [34] studies the use of SRB on a grid, and its benefits and limitations.

There are many efforts to investigate the use of DBMSs from grid applications [34]. The projects closest in spirit to this work are the Work Package 2 of the European Data Grid project [xlvi], and the OGSA-DAI project [xlvii].

The Work Package 2 of the European Data Grid project has in its charter some virtualization services for the grid, including data discovery, replica management, and workflow optimization. The SpitFire project proposes a grid service that will mediate between a DBMS and a grid client, converting HTTP requests made by the client into JDBC requests to the DBMS, and mapping tabular results from the DBMS into an XML output to the client [xlviii].

The main difference with our design is that we have an explicit suite of virtualization services providing a wide range of transparencies. These services virtualize all data sources, including file systems, DBMSs, and even application programs, using three key technologies: federated DBMSs, a powerful discovery service and a generic data replication service. The federated DBMS allows integrated access and management of distributed and heterogeneous data sources. The discovery service provides name transparency. The replication service is used to distribute data across the grid to aggregate compute resources, and to consolidate data from multiple data sources for warehousing.

The OGSA-DAI project is drafting a general grid service interface for accessing grid data sources, including relational and XML databases through query languages like SQL, XPath, and XQuery [49]. We anticipate that the Grid Data Service interface being developed by this project will be able to interact well with the virtualization services proposed in this paper.

6. Conclusions

Many modern applications involve large data sizes and wide distribution. These applications are well suited for running on a grid, but need technologies for efficient information integration and for efficient utilization of the compute resources. Data management technologies like DBMSs and file systems are quite mature. However, grids introduce new challenges like large scale, wide distribution, and source autonomy.

To allow grid applications to develop without worrying about these complexities, data access and processing should be virtualized. In this paper we have identified a set of transparencies that are fundamental to data virtualization: heterogeneity transparency, location transparency, distribution transparency, ownership & costing transparency, and replication transparency.

Current data management technologies have some important limitations in providing these transparencies, especially location, ownership, costing, and replication. We have proposed a suite of data virtualization services to enable these transparencies. These virtualization services lie between grid applications and grid data sources, and interact closely with other OGSA services.

There are a few other aspects of virtualization that we have not considered in this paper, because we feel they are complex issues meriting separate treatment. First, grid applications might want end-to-end guarantees on quality of service. Supporting this involves several changes to data services, including quality of service guarantees from the underlying data sources themselves. Second, grids need a mechanism to control invocation of autonomous grid services. We think that microeconomic mechanisms might be appropriate, though this needs further investigation.

In this paper we have described both the long term goals of data virtualization services and what is achievable in the short term. We would like to have wider discussion of both these goals in GGF and the rest of the grid community, especially to prioritize the implementation of these services. We have also identified some functionality that grid data sources can provide in order to enable virtualization. We hope that these, along with other proposals on grid data source interfaces, will stimulate discussion at GGF and form the basis for a standardized grid data source interface.

7. Security Considerations
See Section 2.3.

Author Information

Vijayshankar Raman, Inderpal Narang
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA

rshankar@almaden.ibm.com

Chris Crone, Laura Haas, Susan Malaika,
Tina Mukai, Dan Wolfson
IBM Silicon Valley Lab
555 Bailey Avenue,
San Jose, CA 95141, USA

Chaitan Baru
San Diego Supercomputer Center
9500 Gilman Drive,
La Jolla, CA 92093, USA

Intellectual Property Statement

Full Copyright Notice

References

i. National Digital Mammography Archive. http://nscp01.physics.upenn.edu/ndma/
ii. Koen Holtman. CMS Data Grid System Overview and Requirements. CMS Note 2001/037. Available at
    http://kholtman.home.cern.ch/kholtman/cmsreqsweb/cmsreqs.html

iii. J. Gray, D. Slutz, A. Szalay, A. Thakar, P. Kuntz, and C.Stoughton. Data Mining the SDSS SkyServer Database. MSR TR 2002-1, Microsoft Research, 2002. Available at http://www.research.microsoft.com/~gray/.

iv. Telescience for Advanced Tomography Applications. NPACI Alpha Project. http://www.npaci.edu/Alpha/

v. Qiong Luo, Sailesh Krishnamurthy, C. Mohan, Hamid Pirahesh, Honguk Woo, Bruce G. Lindsay, and Jeffrey F. Naughton: Middle-tier Database Caching for e-Business. In *Proceedings of the ACM SIGMOD International Conference on Management of Data,* 2002.

vi. Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steve Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Manuscript, January 2002. Available at http://www.globus.org/research/papers.html

vii. Ian Foster, Carl Kesselman, and Steve Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In *International Journal of . Supercomputer Applications*, 15(3), 2001.

viii. Michael J. Carey, Donald D. Chamberlin, Srinivasa Narayanan, Bennet Vance, Doug Doole, Serge Rielau, Richard Swagerman, and Nelson Mendonça Mattos. O-O, What Have They Done to DB2? In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1999.

ix. Michael Stonebraker and Dorothy Moore. Object-Relational DBMSs: The Next Great Wave. Morgan Kaufmann 1996

x. Extensible Markup Language (XML) 1.0. W3C Recommendation. Available at http://www.w3.org/TR/REC-xml.

xi. XQuery 1.0: An XML Query Language. W3C Draft Document. Available at http://www.w3.org/TR/xquery/

xii. Catalina Fan, John Funderburk, Hou-in Lam, Jerry Kiernan, Eugene Shekita, and Jayvel Shanmugasundaram. XTABLES: Bridging Relational Technology and XML. IBM DB2 Developer Domain http://www.ibm.com/software/data/pubs/

xiii. Information technology -- Database languages -- SQL -- Part 9: Management of External Data (SQL/MED). ISO/IEC 9075-9:2000. International Organization for Standardization, 2000.

xiv. Rodolphe Michel. Data Links: Managing Files Using DB2. IBM Redbook. http://www.redbooks.ibm.com

xv. Jim Melton, Jan-Eike Michels, Vanja Josifovski, Krishna Kulkarni, Peter Schwarz, and Kathy Zeidenstein. SQL and Management of External Data. SIGMOD Record, 30(1), 2001.

xvi. XML Extender Administration and Programming. SC27-1234-00. http://www.ibm.com/software/data/db2/extenders/xmlext/

xvii. JDBC Data Access API 3.0 Specification. Available at http://java.sun.com/products/jdbc/

xviii. Web Services Object Runtime Framework. http://www.ibm.com/software/data/webservices/

xix. Microsoft ActiveX Data Objects. http://www.microsoft.com/data/ado/default.htm

xx. Wolfgang Hoschek and Gavin McCance. Grid Enabled Relational Database Middleware. Technical report, Glasgow   University, GLAS-PPE/2001-11.

xxi. Dave Pearson, Data Requirements for The Grid: Scoping Study Report. Global Grid Forum (GGF) 4, 2001.

xxii. Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A Wide-Area Distributed Database System. *VLDB Journal 5(1):* 48-63(1996)

xxiii. Michael J. Carey, Laura M. Haas, Peter M. Schwarz, Manish Arya, William F. Cody, Ronald Fagin, Myron Flickner, Allen Luniewski, Wayne Niblack, Dragutin Petkovic, Joachim Thomas II, John H. Williams, and Edward L. Wimmers. Towards heterogeneous multimedia information systems. In *Proceedings of the International Workshop on Research Issues in Data Engineering*, 1995.

xxiv. Amit P. Sheth and James A. Larson, Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys, 22(3), 1990.

xxv. Donald Kossman. The state of the art in distributed query processing. ACM Computing Surveys, 32(4), 2000.

xxvi. V. Josifovski, P. Schwarz, L. Haas and E. Lin. Garlic: A New Flavor of Federated Query Processing for DB2. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2002.

xxvii. IBM DB2 Life Sciences Data Connect, Planning, Installation, and Configuration Guide, Version 8, GC27-1235-00

xxviii. The Encyclopedia of Life project, http://eol.sdsc.edu.

29. Enterprise Workload Management. http://www.ibm.com/servers/eserver/introducing/eliza/

xxix. Laks V.S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. SchemaSQL – A Language for Interoperability in Relational Multi-Database Systems. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1996.

xxx. Raghu Ramakrishnan and Johannes Gehrke. Database Management Systems, McGraw Hill, 1998.

xxxi. Sunita Sarawagi (editor). Special Issue on Data Cleaning. Bulletin of the IEEE Technical Committee on Data Engineering, 23(4), 2000.

xxxii. Data Extracting, Transforming, and Loading (ETL) Tools. http://www.dwinfocenter.org/clean.html

xxxiii. Paul Watson, Databases and the Grid, 7th December 2001, version 2.

xxxiv. Brian Lovoie. Meeting the challenges of digital preservation: The OAIS reference model. OCLC Newsletter, January/February 2000.

xxxv. R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. A National-Scale Authentication Infrastructure. *IEEE Computer*, 33(12):60-66, 2000. Available at http://www.globus.org/research/papers.html#GSI1

xxxvi. Security in a Web Services World: A Proposed Architecture and Roadmap. Joint IBM and Microsoft Whitepaper, 2002. Available at http://www.ibm.com/developerworks/library/ws-secure/

xxxvii. Rajkumar Buyya, David Abramson, Jonathan Giddy, An Economy Driven Resource Management Architecture for Global Computational Power Grids , *The 2000 International Conference on Parallel and Distributed Processing Techniques and Application*, 2000. Available at http://www-unix.globus.org/mail_archive/discuss/2001/Archive/pdf00000.pdf

xxxviii. Juliana Silva da Cunha, Fábio Q. B. da Silva, Gérman Goldszmidt, and Karen Appleby-Hougham, "SALMON - an Architecture to Define, Store, Monitoring and Billing ISLAs in a Server Farm," Second Latin American Network Operations and Management Symposium, 2001.

xxxix. Jun Rao, Chun Zhang, Guy Lohman, Nimrod Megiddo. Automating Physical Database Design in a Parallel Database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data,* 2002.

xl. P. Gupta and E. T. Lin. Datajoiner: A practical approach to multi-database access. In *Proc. of the Intl. IEEE Conf. on Parallel and Distributed Information Systems*, 1994

xli. Oracle Transparent Gateways. http://otn.oracle.com/products/gateways/gateways_fov.html

xlii. J2EE 1.3 Specification. JSR 58, http://www.jcp.org/jsr/detail/58.jsp

xliii. Dan Wolfson. Using MQSeries from DB2 Applications. IBM DB2 Developer Domain http://www.ibm.com/software/data/pubs/

xliv. P. Dinda and B. Plale. A Unified Relational Approach to Grid Information Services. Grid Forum Informational Draft GWD-GIS-012-1

xlv. Arcot Rajasekar, Michael Wan and Reagan Moore. MySRB & SRB - Components of a Data Grid. In *International Symposium on High Performance Distributed Computing (HPDC)*, 2002.

xlvi. Work Package 2 of European Data Grid Project. http://grid-data-management.web.cern.ch/grid-data-management/Index.html

xlvii. Norman Paton, Malcolm Atkinson, Vijay Dialani, Dave Pearson, Tony Storey and Paul Watson. Database Access and Integration Services on the Grid. UK e-Science Programme Technical Report Series Number UKeS-2002-03, National e-Science Centre, UK.

xlviii. Wolfgang Hoschek and Gavin McCance. Grid Enabled Relational Database Middleware. Technical report, Glasgow University, GLAS-PPE/2001-11.

49. OGSA Database Access and Integration project. http://umbriel.dcs.gla.ac.uk/NeSC/general/projects/OGSA_DAI

50. Carl Kesselman. Personal Communication, 2002.

51. Ann Chervenak. Personal Communication, 2002.

52. Ann Chervenak. Data Management: The Globus Perspective. GlobusWorld, 2003.