DATABASES AND THE GRID

Paul Watson
Department of Computing Science, University of Newcastle, Newcastle-upon-Tyne, NE1 7RU, UK
e-mail: Paul.Watson@newcastle.ac.uk

Telephone: +44 191 222 7653

Fax: +44 191 222 8232

Keywords: Databases, Grid, Federation

Summary

This paper examines how databases can be integrated into the Grid. Almost all early Grid applications are file-based, and so, to date, there has been relatively little effort applied to integrating databases into the Grid. However, if the Grid is to support a wider range of applications, both scientific and otherwise, then database integration into the Grid will become important. Therefore, this paper investigates the requirements of Grid-enabled databases and considers how these requirements are met by existing Grid middleware. This shows that support is very limited. The paper therefore goes on to propose a service-based architecture, and identifies the key service functionalities needed to meet the requirements. In this architecture, database systems are wrapped within a Grid-enabled service interface that simplifies the task of building applications that access their contents. The ability to federate data from multiple databases is likely to be a very powerful facility for Grid users wishing to collate and analyse information distributed over the Grid. The paper describes how the service-based framework assists in federating database servers over the Grid by supporting the creation of virtual databases that present the same service interface to applications as do the individual, unfederated, database systems.

1 INTRODUCTION

This paper examines how databases can be integrated into the Grid [1]. Almost all early Grid applications are file-based, and so, to date, there has been relatively little effort applied to integrating databases into the Grid. However, if the Grid is to support a wider range of applications, both scientific and otherwise, then database integration into the Grid will become important. For example many applications in the life and earth sciences, and many business applications are heavily dependent on databases.

The core of this paper considers how databases can be integrated into the Grid so that applications can access data from them. It is not possible to achieve this just by adopting or adapting the existing Grid components that handle files, as databases offer a much richer set of operations (for example queries and transactions), and there is much greater heterogeneity between different database management systems than there is between different filesystems. Not only are there major differences between database paradigms (e.g. object and relational), but even within one paradigm different database products (e.g. Oracle and DB2) vary in their functionality and interfaces. This diversity makes it more difficult to design a single solution for integrating databases into the Grid, but the alternative of requiring every database to be integrated into the Grid in a bespoke fashion would result in much wasted effort. Managing the tension between the desire to support the full functionality of different database paradigms, while also trying to produce common solutions to reduce effort, is key to designing ways of integrating databases into the Grid.

The diversity of database systems also has other important implications. One of the main hopes for the Grid is that it will encourage the publication of scientific data in a more open manner than is currently the case. If this occurs then it is likely that some of the greatest advances will be made by combining data from separate, distributed sources to produce new results. The data that applications wish to combine will have been created by a set of different researchers who will often have made local, independent decisions about the best database paradigm and design for their data. This heterogeneity presents problems when data is to be combined. If each application has to include its own, bespoke solutions to federating information then similar solutions will be re-invented in different applications, and effort wasted. Therefore, it is important to provide generic middleware support for federating Grid-enabled databases.

Yet another level of heterogeneity needs to be considered. While this paper focuses on the integration of structured data into the Grid (e.g. data held in relational and object databases), there will be the need to build applications that also access and federate other forms of data. For example, semi-structured data (e.g. XML), and relatively unstructured data (e.g. scientific papers), are valuable sources of information in many fields. Further, this type of data will often be held in files, rather than a database. Therefore, in some applications there will be a requirement to federate these types of data with structured data from databases.

There are therefore two main dimensions of complexity in the problem of integrating databases into the Grid: implementation differences between server products within a database paradigm, and the variety of database paradigms. The requirement for database federation effectively creates a problem space whose complexity is abstractly the product of these two dimensions. This paper includes a proposal for a framework for reducing the overall complexity.

Unsurprisingly, existing database management systems do not currently support Grid integration. They are however the result of many hundreds of person-years of effort that allows them to provide a wide range of functionality, valuable programming interfaces and tools, and important properties such as security, performance and dependability. As these attributes will be required by Grid applications, we strongly believe that building new Grid-enabled database

management systems from scratch is both unrealistic and a waste of effort. Instead we must consider how to integrate existing database management systems into the Grid. As is described later, this approach does have its limitations, as there are some desirable attributes of Grid-enabled databases that cannot be added in this way and need to be integrated in the underlying database management system itself. However, these are not so important as to invalidate the basic approach of building on existing technology.

The danger with this approach comes if a purely short-term view is taken. If we restrict ourselves to considering only how existing databases servers can be integrated with existing Grid middleware then we may loose sight of longer-term opportunities for more powerful connectivity. Therefore, we have tried to identify both the limitations of what can be achieved in the short-term solely by integrating existing components, and cases where developments to the Grid middleware and database server components themselves will produce longer-term benefits. An important aspect of this will occur naturally if the Grid becomes commercially important, as the database vendors will then wish to provide "out-of-the-box" support for Grid integration, by supporting the emerging Grid standards. Similarly, it is vital that those designing standards for Grid middleware take into account the requirements for database integration. Together, these converging developments would reduce the amount of "glue" code required to integrate databases into the Grid.

This paper addresses three main questions: what are the requirements of Grid-enabled databases? how far do existing Grid middleware and database servers go towards meeting these requirements? how might the requirements be more fully met? In order to answer the second question, we surveyed current Grid middleware. The Grid is evolving rapidly, and so the survey should be seen as a snapshot of the state of the Grid as it was at the time of writing. In addressing the third question, we focus on describing a framework for integrating databases into the Grid, identifying the key functionalities, and referencing relevant work. We do not make specific proposals at the interface level in this paper – this work is being done in other projects described later.

The structure of the rest of the paper is as follows. Section 2 defines terminology, and then Section 3 briefly lists the possible range of uses of databases in the Grid. Section 4 considers the requirements of Grid-connected databases, and Section 5 gives an overview of the support for database integration into the Grid offered by current Grid middleware. As this is very limited indeed, we go on to examine how the requirements of Section 4 might be met. This leads us to propose a framework for allowing databases to be fully integrated into the Grid, both individually (Section 6) and in federations (Section 7). We end by drawing conclusions (Section 8).

2 TERMINOLOGY

In this section we briefly introduce the terminology that will be used through the paper.

A *database* is a collection of related data. A *database management system* (DBMS) is responsible for the storage and management of one or more databases. Examples of DBMS are Oracle 9i, DB2, Objectivity and MySQL. A DBMS will support a particular database *paradigm*, for example relational, object-relational or object. A *Database System* (DBS) is created, using a DBMS, to manage a specific database. The DBS includes any associated application software.

Many Grid applications will need to utilise more than one DBS. An application can access a set of DBS individually, but the consequence is that any integration that is required (e.g. of query results or transactions) must be implemented in the application. To reduce the effort required to achieve this, *federated* databases use a layer of middleware running on top of autonomous databases, to present applications with some degree of integration. This can include integration of schemas and query capability.

DBS and DBMS offer a set of *services* that are used to manage and access the data. These include query and transaction services. A service provides a set of related *operations*.

3 THE RANGE OF USES OF DATABASES ON THE GRID

As well as the storage and retrieval of the data itself, databases are suited to a variety of roles within the Grid, and its applications. Examples of the potential range of uses of databases in the Grid include:

Metadata. This is data about data, and is important as it adds context to the data, aiding its identification, location, and interpretation. Key metadata includes the name and location of the data source, the structure of the data held within it, data item names and descriptions. There is however no hard division between data and metadata – one application's metadata may be another's data. For example, an application may combine data from a set of databases with metadata about their locations in order to identify centres of expertise in a particular category of data (e.g. a specific gene). Metadata will be of vital importance if applications are to be able to discover and automatically interpret data from large numbers of autonomously managed databases. When a database is "published" on the Grid some of the metadata will be installed into a catalogue (or catalogues) that can be searched by applications looking for relevant data. These searches will return a set of links to databases whose further metadata (not all the metadata may be stored in catalogues) and data can then accessed by the application. The adoption of standards for metadata will be a key to allowing data on the Gird to be discovered successfully. Standardisation efforts such as *Dublin Core* [2], along with more generic technologies and techniques such as *rdf* [3] and ontologies, will be as important for the Grid as they are expected to become to the *Semantic Web* [4]. Further information on the metadata requirements of early Grid applications is given in [5].

Provenance. This is a type of metadata that provides information on the history of data. It includes information on the data's creation, source, owner, what processing has taken place (including software versions), what analyses it has been used in, what result sets have been produced from it, and the level of confidence in the quality of information. An example would be a pharmaceutical company using provenance data to determine what analyses have been run on some experimental data, or how a piece of derived data was generated.

Knowledge repositories. To maintain information on all aspects of research. This could, for example, extend provenance by linking research projects to data, research reports and publications.

Project repositories. To maintain all information about specific projects: a sub-set of this information would be accessible by all researchers through the Knowledge repository. Ideally, knowledge and project repositories can be used to link data, information and knowledge, e.g. raw data → result sets → observations → models and simulations → observations → inferences → papers.

In all these examples, some form of data is "published" so that it can be accessed by Grid applications. There will also be Grid components that use databases internally, without directly exposing their contents to external Grid applications. An example would be a performance monitoring package that used a database internally to store information. In these cases, Grid-integration of the database is not a requirement and so does not fall within the scope of this paper.

4 THE DATABASE REQUIREMENTS OF GRID APPLICATIONS

A typical Grid application, of the sort in which this paper is concerned, may consist of a computation that queries one or more databases and carries out further analysis on the retrieved data. Therefore, database access should be seen as being only one part of a wider, distributed application. Consequently, if databases are to be successfully integrated into Grid applications, there are two sets of requirements that must be met: firstly, those that are generic across all components of Grid applications and allow databases to be "first class components" within these applications, and secondly those that are specific to databases and allow database functionality to be exploited by Grid applications. These two categories of requirements are considered in turn in this section.

If computational and database components are to be seamlessly combined to create distributed applications then a set of agreed standards will have to be defined and met by all components. While it is too early in the lifetime of the Grid to state categorically what all the areas of standardisation will be, work on existing middleware systems (e.g. CORBA), and emerging work within the Global Grid Forum, suggests that security [6], accounting [7], performance monitoring [8] and scheduling [9] will be important. It is not clear that database integration imposes any additional requirements in the areas of accounting, performance monitoring and scheduling, though it does raise implementation issues that are discussed in Section 6. However, security is an important issue, and is now considered.

An investigation into the security requirements of early data-oriented Grid applications [5] shows the need for great flexibility in access control. A data owner must be able to grant and revoke access permissions to other users, or delegate this authority to a trusted third party. It must be possible to specify all combinations of access restrictions (e.g. read, write, insert, delete), and also to have fine-grained control over the granularity of the data against which they can be specified (e.g. columns, sets of rows). Users with access rights must themselves be able to delegate access rights to other users, or an application. Further, they must be able to restrict the rights they wish to delegate to a subset of the rights they themselves hold. For example, a user with read and write permission to a dataset may wish to write and distribute an application that has only read access to the data. Role-based access, in which access control is based on user role as well as named individuals, will be important for Grid applications that support collaborative working. The user who performs a role may change over time, and a set of users may adopt the same role concurrently. Therefore, when a user or application accesses a database they must be able to specify the role that they wish to adopt. All these requirements can be met "internally" by existing database server products. However, they must also be supported by any Grid-wide security system if it is to be possible to write Grid applications all of whose components exist within a single unified security framework.

Some Grid applications will have extreme performance requirements. In an application that performs CPU-intensive analysis on a huge amount of data accessed by a complex query from a DBS, then achieving high performance may require utilising high performance servers to support the query execution (e.g. a parallel database server) and the computation (e.g. a powerful compute server such as a parallel machine, or cluster of workstations). However, this may still not produce high performance, unless the communication between the query and analysis components is optimised. Different communication strategies will be appropriate in different circumstances. If all the query results are required before analysis can begin then it may be best to transfer all the results efficiently in a single block from the database server to the compute server. Alternatively, if a significant computation needs to be performed on each element of the result set, then it is likely to be more efficient to stream the results from the DBS to the compute server as they are produced. When streaming, it is important to optimise communication by sending data in blocks, rather than as individual items, and to use flow control to ensure that the consumer is not

swamped with data. The designers of parallel database servers have built-up considerable experience in designing these communications mechanisms, and this knowledge can be exploited for the Grid [10] [11] [12].

If the Grid can meets these requirements by offering communications mechanisms ranging from fast large file transfer to streaming with flow control then how should the most efficient mechanism be selected for a given application run? Internally, DBMS make decisions on how best to execute a query through the use of cost models that are based on estimates of the costs of the operations used within queries, data sizes and access costs. If distributed applications that include database access are to be efficiently mapped onto Grid resources then this type of cost information needs to be made available by the DBMS to an application planning and scheduling tools, and not just used internally. Armed with this information a planning tool could estimate not only the most efficient communication mechanism to use for data flows between components, but also decide what network and computational resources should be acquired for the application. This will be particularly important where a user is paying for the resources that the application consumes: if high-performance platforms and networks are underutilised then money is wasted, while a low-cost, low-performance component that is a bottleneck may result in the user's performance requirements not being met.

If cost information was made available by Grid-enabled databases then this would enable a potentially very powerful approach to writing and planning distributed Grid applications that access databases. Some query languages allow user-defined operation calls in queries, and this can allow many applications that combine database access and computation to be written as a single query (or if not then at least parts of them may written in this way). The Object Database Management Group (ODMG) Object Query Language (OQL) is an example of one such query language [13]. A compiler and optimiser could then take the query and estimate how best to execute it over the Grid, making decisions about how to map and schedule the components of such queries onto the Grid, and the best ways of communicating data between them. To plan such queries efficiently requires estimates of the cost of operation calls. Mechanisms are therefore required for these to be provided by users, or for predictions to be based on measurements collected at run-time from previous calls (so reinforcing the importance of performance monitoring for Grid applications). The results of work on compiling and executing OQL queries on parallel object database servers can fruitfully be applied to the Grid [14] [12].

We now move beyond considering the requirements that are placed on all Grid middleware by the need to support databases, and consider the requirements that Grid applications will place on the DBMS themselves. Firstly, there appears to be no reason why Grid applications will not require at least the same functionality, tools and properties as other types of database applications. Consequently, the range of facilities already offered by existing DBMS will be required. These support both the management of data, and the management of the computational resources used to store and process that data. Specific facilities include:

- query and update facilities
- programming interface
- indexing
- high availability
- recovery
- replication
- concurrency control
- transactions
- bulk loading
- manageability
- archiving
- security

- versioning
- evolution
- uniform access to data and schema
- integrity constraints
- change notification (e.g. triggers)

Many person-years of effort have been spent embedding this functionality into existing DBMS, and so, realistically, integrating databases into the Grid must involve building on existing DBMS, rather than on developing completely new, Grid-enabled DBMS from scratch. In the short-term, this may place limitations on the degree of integration that is possible (an example is highlighted in Section 6), but in the longer-term, there is the possibility that the commercial success of the Grid will remove these limitations by encouraging DBMS producers to provide built-in support for emerging Grid standards.

We now consider whether Grid-enabled databases will have requirements beyond those typically found in existing systems. The Grid is intended to support the wide-scale sharing of large quantities of information. The likely characteristics of such systems may be expected to generate the following set of requirements that Grid-enabled databases will have to meet:

Scalability. Grid applications can have extremely demanding performance and capacity requirements. There are already proposals to store Petabytes of data, at rates of up to 1TeraByte per hour, in Grid-accessible databases [15]. Low response times for complex queries will also be required by applications that wish to retrieve subsets of data for further processing. Another strain on performance will be generated by databases that are accessed by large numbers of clients, and so will need to support high access throughput. Popular, Grid-enabled information repositories will fall into this category.

Handling unpredictable usage. The main aim of the Grid is to simplify and promote the sharing of resources, including data. Some of the science that will utilise data on the Grid will be explorative and curiosity driven. Therefore it will be difficult to predict in advance the types of accesses that will be made to Grid-accessible databases. This differs from most existing database applications in which types of access can be predicted. For example, many current e-commerce applications "hide" a database behind a Web interface that only supports limited types of access. Further, typical commercial "line-of-business" applications generate a very large number of small queries from a large number of users, whereas science applications may generate a relatively small number of large queries, with much greater variation in time and resource usage. In the commercial world, data warehouses may run unpredictable workloads, but the computing resources they use are deliberately kept independent of the resources running the "line-of-business" applications from which the data is derived. Providing open, ad-hoc access to scientific databases therefore raises the additional problem of DBMS resource management. Current DBMS offer little support for controlling the sharing of their finite resources (CPU, disk IOs and main memory cache usage). If they were exposed in an open Grid environment, little could be done to prevent deliberate or accidental denial of service attacks. For example we want to be able to support a scientist who has an insight that running a particular complex query on a remote, Grid-enabled database could generate exciting new results. However, we do not want the execution of that query to prevent all other scientists from accessing the database for several hours.

Metadata-driven access. It is already generally recognised that Metadata will be very important for Grid applications. Currently, the use of metadata in Grid applications tends to be relatively simple – it is mainly for mapping the logical names for datasets into the physical locations where they can be accessed. However, as the Grid expands into new application areas such as the life sciences, more sophisticated metadata systems and tools will be required. The result is likely to be a *Semantic Grid* [16] that is analogous to the *Semantic Web* [4]. The use of metadata to locate data

has important implications for integrating databases into the Grid because it promotes a two-step access to data. In step one, a search of metadata catalogues is used to locate the databases containing the data required by the application. That data is then accessed in the second step. A consequence of two-step access is that the application writer does not know the specific DBS that will be accessed in the second step. Therefore the application must be general enough to connect and interface to any of the possible DBS returned in step one. This is straightforward if all are built from the same DBMS, and so offer the same interfaces to the application, but more difficult if these interfaces are heterogeneous. Therefore, if it is to be successful, the two-step approach requires that all DBS should, as far as possible, provide a standard interface. It also requires that all data is held in a common format, or that the metadata that describes the data is sufficient to allow applications to understand the formats and interpret the data. The issues and problems of achieving this are discussed in Section 6.

Multiple Database Federation. One of the aims of the Grid is to promote the open publication of scientific data. A recent study of the requirements of some early Grid applications concluded that "The prospect exists for literally billions of data resources and petabytes of data being accessible in a Grid environment" [5]. If this prospect is realised then it is expected that many of the advances to flow from the Grid will come from applications that can combine information from multiple data sets. This will allow researchers to combine different types of information on a single entity to gain a more complete picture, and to aggregate the same types of information about different entities. Achieving this will require support for integrating data from multiple DBS, for example through distributed query and transaction facilities. This has been an active research area for several decades, and needs to be addressed on multiple levels. As was the case for Metadata-driven access, the design of federation middleware will be made much more straightforward if DBS can be accessed through standard interfaces that hide as much of their heterogeneity as possible. However, even if APIs are standardised, this still leaves the higher-level problem of the semantic integration of multiple databases, which has been the subject of much attention over the past decades [17] [18]. In general, the problem complexity increases with the degree of heterogeneity of the set of databases being federated, though the provision of ontologies and metadata can assist. While there is much existing work on federation on which to build, for example in the area of query processing [19, 20], the Grid should give a renewed impetus to research in this area because there will be clear benefits from utilising tools that can combine data over the Grid from multiple, distributed repositories. It is also important that the middleware that supports distributed services across federated databases meets the other Grid requirements. For example distributed queries run across the Grid may process huge amounts of data, and so the performance requirements on the middleware may, in some cases, exceed the requirements on the individual DBS.

In summary, there are a set of requirements that must be met in order to support the construction of Grid applications that access databases. Some are generic across all Grid application components, while others are database specific. It is reasonable to expect that Grid applications will require at least the functionality provided by current DBMS. As these are complex pieces of software, with high development costs, building new, Grid-enabled DBMS from scratch is not an option. Instead, new facilities must be added by enhancing existing DBMS, rather than by replacing them. The most commonly used DBMS are commercial products that are not open-source, and so enhancement will have to be achieved by wrapping the DBMS externally. It should be possible to meet almost all the requirements given above in this way, and methods of achieving this are proposed in Sections 6 and 7. In the longer-term, it is to be hoped that, if the Grid is a commercial success, then database vendors will wish to provide "out-of-the-box" support for Grid integration, by supporting Grid requirements. Ideally, this would be encouraged by the definition of open standards. If this was to occur, then the level of custom wrapping required to integrate a database into the Grid would be considerably reduced.

The remainder of this paper investigates how far current Grid middleware falls short of meeting the above requirements, and then proposes mechanisms for more fully satisfying them.

5 THE GRID AND DATABASES: THE CURRENT STATE

In this section we consider how the current Grid middleware supports database integration. We consider Globus, the leading Grid middleware before looking at previous work on databases in Grids. As the Grid is evolving rapidly, this section should be seen as a snapshot taken at the time of writing.

The dominant middleware used for building computational grids is Globus, which provides a set of services covering grid information, resource management and data management [21]. Information Services allow owners to register their resources in a directory, and provide, in the Monitoring and Discovery Service (MDS) mechanisms through which they can be dynamically discovered by applications looking for suitable resources on which to execute. From MDS, applications can determine the configuration, operational status and loading of both computers and networks. Another service, the Globus Resource Allocation Manager (GRAM) accepts requests to run applications on resources, and manages the process of moving the application to the remote resource, scheduling it and providing the user with a job control interface.

An orthogonal component that runs through all Globus services is the Grid Security Infrastructure (GSI). This addresses the need for secure authentication and communications over open networks. An important feature is the provision of "single-sign on" access to computational and data resources. A single X.509 certificate can be used to authenticate a user to a set of resources, so avoiding the need to sign-on to each resource individually.

The latest version of Globus (2.0) offers a core set of services (called the Globus Data Grid) for file access and management. There is no direct support for database integration and the emphasis is instead on the support for very large files, such as those that might be used to hold huge datasets resulting from scientific experiments. GridFTP is a version of FTP optimised for transferring files efficiently over high-bandwidth wide area networks and it is integrated with the Grid Security Infrastructure. Globus addresses the need to have multiple, possibly partial, copies of large files spread over a set of physical locations by providing support for replica management. The Globus Replica Catalog holds the location of a set of replicas for a logical file, so allowing applications to find the physical location of the portion of a logical file they wish to access. The Globus Replica Management service uses both the Replica Catalogue, and GridFTP to create, maintain and publish the physical replicas of logical files.

There have been recent moves in the Grid community to adopt Web Services [22] as the basis for Grid middleware, through the definition of the Open Grid Services Architecture (OGSA) [23]. This will allow the Grid community to exploit the high levels of investment in Web Service tools and components being developed for commercial computing. The move also reflects the fact that there is a great deal of overlap between the Grid vision of supporting scientific computing by sharing resources, and the commercial vision of enabling Virtual Organisations - companies combining information, resources and processes to build new distributed applications.

Despite lacking direct support for database integration, Globus does have services that can assist in achieving this. The Grid Security Infrastructure could be used as the basis of a system that provides a *single sign-on* capability, removing the need to individually connect to each database with a separate username and password (which would not easily fit into the two-step access method described in Section 4). However, mechanisms for connecting a user or application to the database in a particular role, and for delegating restricted access rights are required, as described in Section 4, but are not currently directly supported by GSI. A recent development - the Community Authorisation Service [24] - does offer restricted delegation, and so may offer a way forward. Other Globus components could also be harnessed in order to support other

aspects of database integration into the Grid. For example, GridFTP could be used both for bulk database loading and, where efficient, for the bulk transfer of query results from a DBS to another component of an application. The MDS and GRAM services can be used to locate and run database federation middleware on appropriate computational resources, as will be discussed in Section 7. In the longer term, the move towards an OGSA service-based architecture for Globus is in line with the proposed framework for integrating databases into the Grid that will be described in Sections 6 and 7.

Having examined Globus, the main generic Grid middleware project, we now describe two existing projects that include work on Grids and databases.

Spritfire [25], a European Data Grid project, has developed an infrastructure that allows a client to query a relational database over GSI-enabled HTTP(S). An XML based protocol is used to represent the query, and its result. The system supports role-based security: a client can specify the role they wish to adopt for a query execution, and a mapping table in the server checks that they are authorised to take on this role.

The Storage Request Broker (SRB) is middleware that provides uniform access to datasets on a wide range of different types of storage devices [26] that can include filesystems and archival resources. The SRB's definition of dataset is "stream-of-bytes", and so the primary focus is on files and collections of files, rather than on the structured data held in databases that is the focus of this paper. The SRB provides location transparency through the use of a Metadata catalogue (MCAT) that allows access to datasets by logical names, or other metadata attributes. SRB installations can be federated, such that any SRB in the federation can accept client calls and forward them to the appropriate SRB. The SRB also provides replica management for datasets, providing fault tolerance by redirecting client requests to a replica if the primary storage system is unavailable. The SRB supports a variety of authentication protocols for clients accessing data, including the GSI. While the focus of the SRB is on file-based data, it does offer some limited capabilities for accessing data held in databases. Datasets can be held as LOBs (Large OBjects) in databases (in which case they are basically treated as files stored in the database), or an SQL query can be registered as an object with the SRB – the query is executed whenever the object is retrieved.

In summary, while there are aspects of existing Grid middleware that can contribute to integrating databases into the Grid, very few of the requirements of Section 5 have yet been met, and so we now move on to describe a service framework within which they could be achieved, and identify the key functionalities of the required services.

6 INTEGRATING DATABASES INTO THE GRID

In this section we describe a framework for integrating databases into Grid applications and identify the main functionality that must be provided if the requirements of Section 4 are to be met. We do not make specific proposals at the interface level in this paper; many different interfaces could be designed to meet the requirements within the proposed framework, though we hope that work within the Global Grid Forum will lead to the definition of interface standards.

The proposed framework is service-based. Each Grid-enabled DBS would offer a set of services covering the areas identified in the requirements given in Section 4. Where possible, individual operations offered by these services would be standardized to increase portability, and reduce the effort required to build applications that interact with multiple DBS. Standardisation would be done by adding wrapper code to map the service operation interface to the vendor specific interface beneath. However, it is impossible to standardise all services: for example different database paradigms support different types of query languages, and these cannot all be

reconciled into a standard query language (even within the set of relation databases there are variations). One advantage of a service-based framework is however that each DBS made available on the Grid can provide a metadata service that gives information on the range of services and operations that it supports. For example, a DBS may describe itself as offering a query service that supports SQL-92. This service-metadata would give application builders the information required to exploit whatever facilities were available, and is an important pre-requisite to the proposal for database federation given in Section 7.

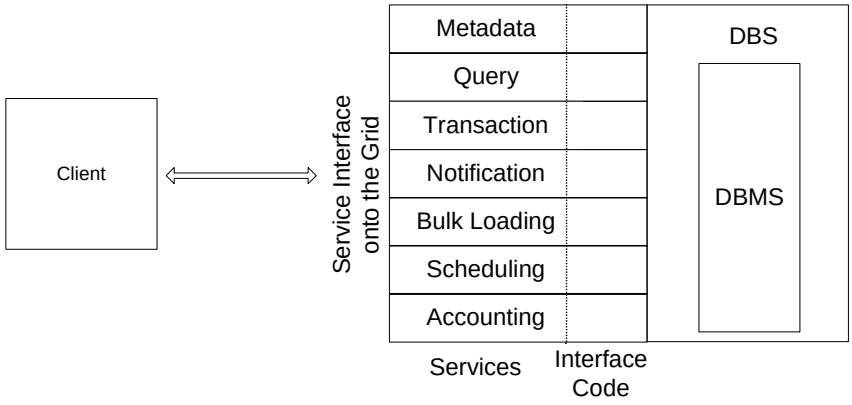

Figure 1. A Database System with a Grid-enabled Service Interface

Figure 1 shows the service-based framework, with a service wrapper placed between the Grid and the DBS (we deliberately refer to DBS here rather than DBMS, as the owner of the database can choose which services to make available on the Grid, and who is allowed to access them). Initially, the service wrappers will have to be custom produced, but, in the future, if the commercial importance of the Grid increases, and standards are defined, then it is to be hoped that DBMS vendors will offer Grid-enabled service interfaces as an integral part of their products.

We now discuss each of the services shown in Figure 1:

Metadata. This service provides access to technical metadata about the DBS and the set of services that it offers to Grid applications. Examples include the logical and physical name of the DBS and its contents, ownership, version numbers, the database schema and information on how the data can be accessed. The service description metadata would, for each service, describe exactly what functionality is offered. This would be used by Grid application builders, and tools that need to know how to interface to the DBS. It is particularly important for applications that are dynamically constructed – the two-step access to data described in Section 4 means that the databases that are to take part in an application are now known until some preliminary processing of metadata has taken place. Each run of such applications may result in the need to access a different set of databases, and so mechanisms are required to dynamically construct interfaces to those DBS – if they are not all able to offer completely standard interfaces, then the metadata can be accessed to determine their functionality and interfaces, so that they can be dynamically incorporated into the application.

Query. Query languages differ across different DBMS, though the core of SQL is standard across most relational DBMS. It is therefore important that the service metadata defines the type and level of query language that is supported. To provide input to scheduling decisions, and enable the efficient planning of distributed Grid applications, an operation that provides an

estimate of the cost of executing a query is highly desirable. As described in the requirements section, the query service should also be able to exploit a variety of communications mechanisms in order to transfer results over the Grid, including streaming (with associated flow control) and transfer as a single block of data. Finally, it is important that the results of a query can be delivered to an arbitrary destination, rather than just to the sender of the query. This allows the creation of distributed systems with complex communications structures, rather than just simple client-server request-response.

Transaction. These operations would support transactions involving only a single DBS (e.g. operations to Begin, Commit and Rollback transactions), and also allow a DBS to participate in application-wide distributed transactions, where the DBS supports it. There are a variety of types of transactions that are supported by DBMS (for example, some but not all support nested transactions), and so a degree of heterogeneity between DBS is inevitable. In the longer term, there may also be a need for loosely co-ordinated, long-running transactions between multiple enterprises, and so support for alternative protocols (e.g. the Business Transaction Protocol - BTP [27]) may become important. Given the variety of support that could be offered by a transaction service, the service-description metadata must make clear what is available at this DBS.

Bulk Loading. Support for the bulk loading of data over the Grid into the database will be important in some systems. For large amounts of data, the service should be able to exploit Grid communication protocols that are optimised for the transfer of large datasets (e.g. GridFTP).

Notification. This would allow clients to register some interest in a set of data, and receive a message when a change occurred. Supporting this function requires both a mechanism that allows the client to specify exactly what it is interested in (e.g. additions, updates, deletions, perhaps further filtered by a query) and a method for notifying the client of a change. Implementing this service is made much simpler if the underlying DBMS provides native support, e.g. through triggers. When a notification is generated, it would be fed into a generic Grid event service to determine what action is to be taken. For example, a user may be directly informed by e-mail, or an analysis computation may be automatically run on new data.

Scheduling. This would allow users to schedule the use of the DBS. It should support the emerging Grid scheduling service [9], for example allowing a DBS and a supercomputer to be co-scheduled, so that large datasets retrieved from the DBS can be processed by the supercomputer. Bandwidth on the network connecting them might also need to be pre-allocated. As providing exclusive access to a DBS is impractical, mechanisms are needed to dedicate sufficient resources (disks, CPUs, memory, network) to a particular task. This requires the DBS to provide resource pre-allocation and management, something that is not well supported by existing DBMS, and cannot be implemented by wrapping the DBMS and controlling the resources at the operating system level. This is because DBMS, like most efficiently designed servers, run as a set of processes that are shared among all the users, and the management of sharing is not visible or controllable at the operating system process level.

Accounting. The DBS must be able to provide the necessary information for whatever accounting and payment scheme emerges for the Grid. This service would monitor performance against agreed service levels, and enable users to be charged for resource usage. The data collected would also provide valuable input for application capacity planning, and for optimising the usage of Grid resources. As with scheduling, as a DBS is a shared server it is important that accounting is done in terms of the individual users (or groups) use of the DBS, and not just aggregated across all users.

We do not claim that this list of services is definitive. It is based on our experience of building systems that utilise databases, but the need for new services may emerge as more

experience is gained with Grid applications. It is also the case that specific types of DBMS may require other services – for example, a navigation service may be required for ODBMS.

There is no separate security service. This is because access control is needed on all the operations of all services, so DBS owners can choose what each user (or group of users) is allowed to do.

The above services are all at a relatively low level and are very generic: they take no account of the meaning of the stored data. Higher-level, semantics-based services will also be required for Grid applications, and these will sit above, and utilise, the lower-level services described in this paper. For example, the need for a generic Provenance service might be identified, implemented once and used by a variety of applications. It may, for example offer operations to locate data with a particular provenance, or identify the provenance of data returned by a query. Identifying these higher-level services, and designing them to be as general as possible, will be important for avoiding duplicated effort in the construction of Grid applications.

7 FEDERATING DATABASE SYSTEMS ACROSS THE GRID

Section 4 stressed the importance of being able to combine data from multiple DBS. The ability to generate new results by combining data from a set of distributed resources is one of the most exciting opportunities that the Grid will offer. In this section we consider how the service-based framework introduced in Section 6 can help to achieve this.

One option is for a Grid application to interface directly to the service interfaces of each of the set of DBS whose data it wishes to access. This approach is illustrated in Figure 2. However, this forces application writers to solve federation problems within the application itself. This would lead to great application complexity, and duplication of effort.

To overcome these problems we propose an alternative, in which Grid-enabled middleware is used to produce a single, federated "virtual database system" to which the application interfaces. Given the service-based approach proposed in Section 6, federating a set of DBS reduces to federating each of the individual services (query, transaction etc.). This creates a Virtual DBS, which has exactly the same service interface as the DBS described in the previous section but does not actually store any data (advanced versions could however be designed to cache data in order to increase performance). Instead, calls made to the Virtual DBS services are handled by service federation middleware that interacts with the service interfaces of the individual DBS that are being federated, in order to compute the result of the service call. This approach is shown in Figure 3. Because the Virtual DBS has an identical service interface to the "real" DBS, then it is possible for a Virtual DBS to federate the services of both "real" DBS, and other Virtual DBS.

Two different scenarios can be envisaged for the creation of a Virtual DBS:

1) A user decides to create a Virtual DBS that combines data and services from a specific set of DBS that they wish to work with. These may, for example, be well known as the standard authorities in their field.

2) A user wishes to find and work with data on a subject of their interest, but they do not know where it is located, e.g. a biologist may want information on Bacillus subtilis 168. A Metadata query would be used to locate appropriate datasets. These would then be federated to create a Virtual DBS that could then be queried. At the end of the work session, the Virtual DBS could be saved for future use. For example, the notification service might be configured to inform

the user of interesting new data. Alternatively, the virtual DBS might not be required after the current session is ended, and so could be destroyed.

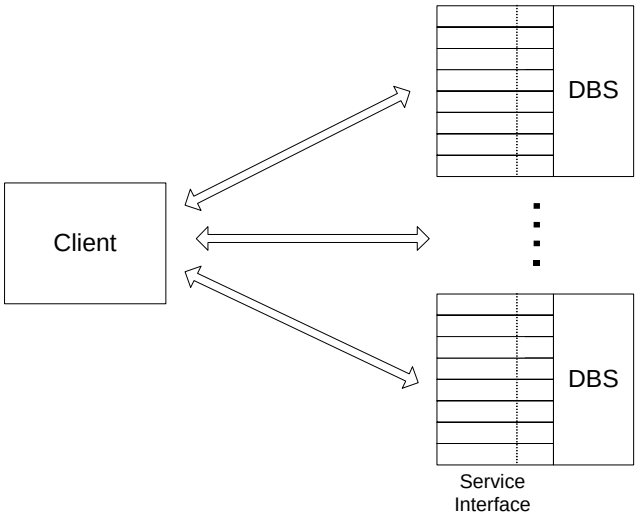

Figure 2.A Grid application interfacing directly to a set of DBS

How can the Virtual DBS be created? The ideal situation would be for a tool to take a set of DBS and automatically create the Virtual DBS. At the other end of the scale, a set of bespoke programs could be written to implement each service of the Virtual DBS. Obviously, the former is preferable, especially if we wish to dynamically create Virtual DBS as in the second scenario above. Bearing this in mind, we now consider the issues in federating services.

The service-based approach proposed in Section 6 assists in the process of federating services, by encouraging standardisation. However, it will not be possible to fully standardise all services, and it is the resulting heterogeneity that causes problems. A tool could attempt to create a Virtual DBS automatically as follows. For each service, the tool would query the metadata service of each of the DBS being federated in order to determine their functionality and interface. Knowing the integration middleware that was available for the service, and the requirements that this middleware had for the underlying services, the tool would determine the options for federation. If there were more than one option then one would be selected (possibly taking into account application or user preferences). If no options were available then the application or user would be informed that no integration of this service was possible. In this case, the user would either not be able to use the service, or would need to write new federation middleware to effect the integration, if that were possible.

Integrating each of the services proposed in Section 6 raises specific issues that are now described:

Query. Ideally this would present to the user a single integrated schema for the virtual DBS, and accept queries against it. A compiler and optimiser would determine how to split up the query across the set of DBS, and then combine the results of these sub-queries. The major relational DBMS products already offer "Star" tools that implement distributed query middleware. Grid applications do however introduce new requirements, in particular the need for conformance with Grid standards, and the ability to query across dynamically changing sets of databases. The service-based approach to Grid-enabling databases simplifies the design of federation middleware as it promotes the standardisation of interfaces, but, as was stated in the requirements section, it does not address the higher-level problem of the semantic integration of

multiple databases, which has been the subject of much attention over the past decades [17]. It is to be hoped that the challenges of Grid applications give a further impetus to research in this area within the database community, as the results of this work are likely to be of great benefit to those building data-intensive Grid applications. Previous work on distributed query processing for parallel systems [10, 11] is very relevant to the Grid, which has a potential need for very high performance – distributed queries across large datasets may require huge joins that would benefit from parallelisation. The nature of the Grid does however offer some interesting new opportunities for distributed query processing [12]. Once a query has been compiled, Grid resources could be acquired on demand for running the distributed query execution middleware. The choice of resources could be made on the basis of the response time, and price requirements of the user. For example, if a join operator was the bottleneck in a query, and performance was important, then multiple compute nodes could be acquired and utilised to run that part of the query in parallel. If the user was charged for time on the compute nodes, then a trade-off between price and performance would need to be made. Further, because query optimisers can only estimate the cost of a query before it is run, queries sometimes take much longer than expected, perhaps because a filter or join in the middle of a query has produced more data than expected. An option here for Grid-based distributed query execution is to monitor the performance at run-time and acquire more resources dynamically in order to meet the performance requirements of the user. There has been previous work in this area of dynamic query adaptation [28], and the Grid, which offers the ability to acquire and discard resources dynamically, could exploit this.

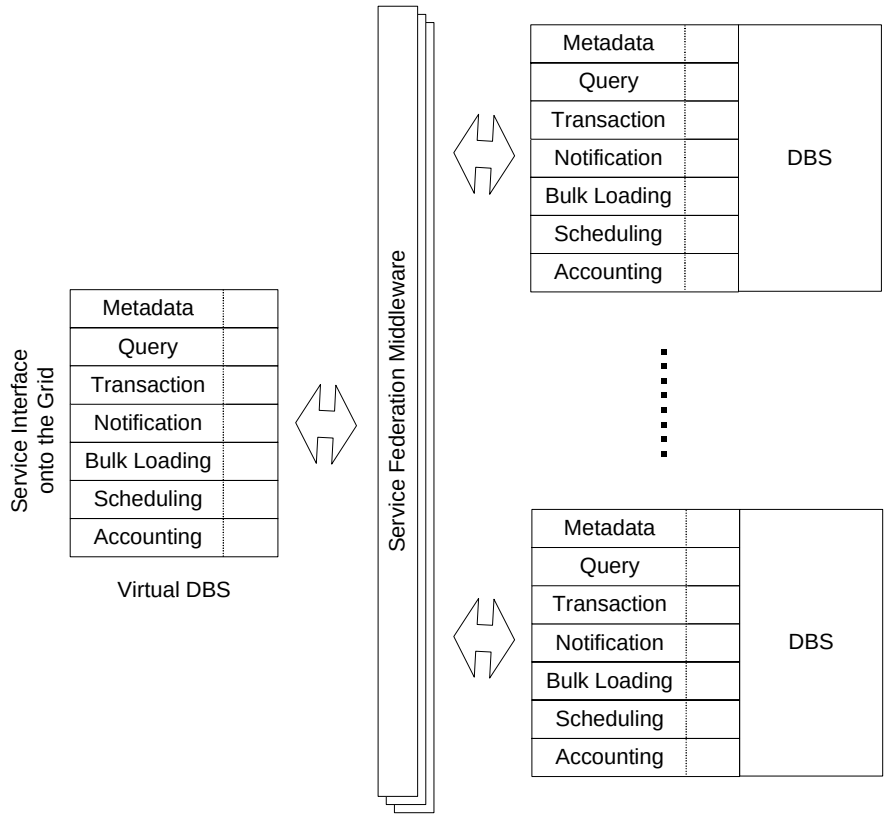

Figure 3. A Virtual Database System on the Grid

Transaction. The basic transaction service described in Section 6 already supports the creation of distributed transactions across multiple databases.

Bulk Loading. This could be implemented by middleware that takes a load file, splits it into separate files for each DBS and uses the bulk load service of each individual DBMS to carry out the loading.

Notification. A client would register an interest in the virtual DBS. Middleware would manage the distribution of the notification operations: registration, filtering and notification, across the DBS. This should ideally be done using a generic Grid-enabled event service so that a database specific federation solution is not required.

Metadata. This would be a combination of the metadata services of the federated databases, and would describe the set of services offered by the Virtual DBS. At the semantic, data-description level (e.g. providing a unified view of the combined schema) the problems are as described above for the query service.

Scheduling. This would provide a common scheduling interface for the virtual DBS. When generic, distributed scheduling middleware is available for the Grid, the implementation of a federated service should be relatively straightforward (though, as described in Section 6, there is a major problem in controlling scheduling within the individual DBMS).

Accounting. This would provide a combined accounting service for the whole virtual DBS. As a Grid accounting service will have to support distributed components, the implementation of this service should be straightforward once that Grid accounting middleware is available.

As has been seen, the complexity of the service federation middleware will vary from service to service, and will, in general, increase as the degree of heterogeneity of the services being federated increases. However, we believe that the service-based approach to federating services provides a framework for the incremental development of a suite of federation middleware, by more than one supplier. Initially, it would be sensible to focus on the most commonly required forms of service federation. One obvious candidate is query integration across relational DBMS. However, over time, applications would discover the need for other types of federation. When this occurs, then the aim is that the solution would be embodied in service federation middleware that fits into the proposed framework described above, rather than it being buried in the application specific code. The former approach has the distinct advantage of allowing the federation software to be re-used by other Grid applications. Each integration middleware component could be registered in a catalogue that would be consulted by tools attempting to integrate database services. The process of writing integration components would also be simplified by each taking a set of standard service interfaces as "inputs" and presenting a single, standard federated service as "output". This also means that layers of federation can be created, with virtual databases taking other virtual databases as inputs.

While the focus in this document is on federating structured data held in databases, Section 4 also identified the requirement to federate this type of data with file-based, semi-structured and relatively unstructured data [18]. The framework for federation described above can also be used to support this, through the use of special federation middleware. To enable any meaningful forms of federation, at least some basic services would have to be provided for the file-based data, for example a simple query service. This could be achieved by either the owner, or the consumer of the data, providing a query wrapper that would be accessed by the middleware. With this in place, service federation middleware could be written that interfaces to, and federates both file and database services, for structured and less-structured data. The adoption of a service-based framework does not however provide any solutions for the semantic integration of data and metadata. That remains an open research area.

8 CONCLUSIONS

This paper identified a set of requirements for Grid databases, and showed that existing Grid middleware does not meet them. However, some of it, especially the Globus middleware, can be utilised as lower level services on which database integration middleware can be built.

The paper proposed a set of services that should be offered by a Grid-integrated database system. This service-based approach will simplify the task of writing applications that access databases over the Grid. The services themselves vary considerably in the degree of complexity required to implement them. Some - *Transactions, Query and Bulk Loading* - already exist in most current DBMS, but work is needed both to integrate them into the emerging Grid standards and services, and to introduce a level of interface standardisation where appropriate. Another, *Accounting*, should be relatively straightforward to implement once a Grid-wide accounting framework is in place. The effort required to develop a *Notification* service will depend on whether or not the underlying DBMS provides native support for it, and work is also required to integrate emerging Grid event and workflow services. Finally, *Scheduling* is the most problematic as to do this properly requires a level of resource management that is not found in existing DBMS, and this functionality cannot be added externally.

The paper showed how the service-based approach to making databases available on the Grid could be used to structure and simplify the task of writing applications that need to combine information from more than one database system. This is achieved by federating services to produce a single Virtual DBS with which the application interacts. The effort required to federate services will vary depending on the type of service, and the degree of heterogeneity of the DBS being federated.

The service-based approach advocated in this paper is independent of any particular implementation technology. However, the recent moves in Grid development towards a service based middleware for the Grid through the Open Grid Services Architecture (OGSA) [23] will clearly simplify the process of building distributed applications that access databases through service interfaces. While this paper has focussed on describing a framework for integrating databases into the Grid and identifying required functionalities without making proposals at the detailed interface level, the UK core e-Science programme's OGSA - Database Access and Integration project (OGSA-DAI) is currently designing and building wrappers for relational databases and XML repositories so that they offer Grid-enabled services conforming to the OGSA framework. In conjunction with this, the Polar* project is researching into parallel query processing on the Grid [12].

To conclude, we believe that if the Grid is to become a generic platform, able to support a wide range of scientific and commercial applications, then the ability to publish and access databases on the Grid will be of great importance. Consequently, it is vitally important that, at this early stage in the Grid's development, database requirements are taken into account when Grid standards are defined, and middleware is designed. In the short term, integrating databases into Grid applications will involve wrapping existing DBMS in a Grid-enabled service interface. However, if the Grid becomes a commercial success then it is to be hoped that the DBMS vendors will Grid-enable their own products by adopting emerging Grid standards.

9 ACKNOWLEDGEMENTS

This paper was commissioned by the UK e-Science Core Programme. Work to design and implement versions of the services described in the paper is being carried out in the OGSA-DAI project (a collaboration between IBM Hursley, Oracle UK, EPCC and the Universities of

Manchester and Newcastle) funded by that programme, and within the EPSRC funded Polar* project (at the Universities of Manchester and Newcastle). I would like to thank the other members of the UK Grid Database Taskforce (Malcolm Atkinson, Vijay Dialani, Norman Paton, Dave Pearson and Tony Storey) for many interesting discussions held over the last year. I am also grateful to the following for their constructive comments on earlier versions of this paper: Ian Foster, Alex Gray, Jim Gray, Tony Hey, Inderpal Narang, Pete Lee, Clive Page, Guy Rixon, David Skillicorn, Anne Trefethen, and the anonymous referees.

10 REFERENCES

1. Foster I., Kesselman C., eds. *The Grid: Blueprint for a New Computing Infrastructure*. 1999, Morgan Kaufmann Publishers Inc.: San Francisco.
2. Hillmann D., *Using Dublin Core. Dublin Core Metadata Initiative*. 2001.
3. Lassila O., Swick R., *Resource Description Framework (RDF) Model and Syntax Specification. W3C, www.w3c.org.REC-rdf-syntax-19990222*. 1999.
4. Berners-Lee T., Hendler J., Lassila O., *The Semantic Web*. Scientific American, 2001. 2001(May).
5. Pearson D., *Data Requirements for The Grid: Scoping Study Report. UK Grid Database Taskforce*. 2002.
6. Global Grid Forum, *Global Grid Forum Security Working Group, www.gridforum.org*. 2002.
7. Global Grid Forum, *Accounting Models Research Group, www.gridforum.org*. 2002.
8. Tierney B., Aydt R., Gunter D., Smith W., Taylor V., Wolski R., Swany M., *A Grid Monitoring Architecture. Global Grid Forum. GWD-Perf-16-2*. 2002.
9. Global Grid Forum, *Scheduling and Resource Management Area, www.gridforum.org*. 2002.
10. Graefe G., *Encapsulation of Parallelism in the Volcano Query Processing System*. in *SIGMOD Conference*. 1990. Atlantic City, NJ, USA: ACM Press.
11. Smith J., Sampaio S.d.F.M., Watson P., Paton N.W., *Polar: An Architecture for a Parallel ODMG Compliant Object Database*. in *ACM CIKM International Conference on Information and Knowledge Management*. 2000. McLean, VA, USA: ACM.
12. Smith J., Gounaris A., Watson P., Paton N.W., Fernandes A.A.A., Sakellariou R., *Distributed Query Processing on the Grid. submitted to the 3rd International Workshop on Grid Computing (GRID 2002)*. 2002. Baltimore, USA.
13. Cattell R., *Object Database Standard: ODMG 2.0*. 1997: Morgan Kaufmann.
14. Smith J., Watson P., Sampaio S.d.F.M., Paton N.W., *Speeding up Navigational Requests in a Parallel Object Database System*. in *EuroPar 2002*. 2002. Paderborn, Germany.
15. Shiers J., *Building a Multi-Petabyte Database: The RD45 Project at CERN*, in *Object Databases in Practice*, M.E.S. Loomis and A.B. Chaudhri, Editors. 1998, Prentice Hall. p. 164-176.
16. De Roure D., Jennings N., Shadbolt N., *Research Agenda for the Semantic Grid: A Future e-Science Infrastructure. UK National e-Science Centre.UKeS-2002-02*. 2001.
17. Sheth A.P., Larson J.A., *Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases*. ACM Computing Surveys, 1990. 22(3): p. 183-236.
18. Lahiri T., Abiteboul S., Widom J., *Ozone: Integrating Structured and Semistructured Data*. in *Seventh Int. Workshop on Database Programming Lanaguages*. 1999. Kinloch Rannoch, Scotland.
19. Yu C.T., Chang C.C., *Distributed Query Processing*. ACM Computing Surveys, 1984. 16(4): p. 399-433.
20. Suciu D., *Distributed Query Evaluation on Semistructured Data*. ACM Transactions on Database Systems, 2002. 27(1): p. 1-62.
21. Foster I., Kesselman C., Tuecke S., *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International Journal of Supercomputer Applications, 2001. 15(3).
22. Graham S., Simeonov S., Boubez T., Daniels G., Davis D., Nakamura Y., Neyama R., *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI*. 2001: Sams. 450.

23. Foster I., Kesselman C., Nick J., Tuecke S., *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. The Globus Project, www.globus.org*. 2002.
24. Pearlman L., Welch V., Foster I., Kesselman C., *A Community Authorization Service for Group Communication. Submitted to the 3rd Intl. Workshop on Policies for Distributed Systems and Networks*. 2001.
25. Hoschek W., McCance G., *Grid Enabled Relational Database Middleware*. in *Global Grid Forum*. 2001. Frascati, Italy.
26. Rajasekar A., Wan M., Moore R., *MySRB & SRB - Components of a Data Grid*. in *11th International Symposium on High Performance Distributed Computing (HPDC-11)*. 2002. Edinburgh.
27. OASIS Committee, *Business Transaction Protocol Version 1.0*. 2002.
28. Gounaris A., Paton N.W., Fernandes A.A., Sakellariou R., *Adaptive Query Processing: A Survey*. in *British National Conference on Databases (BNCOD)*. 2002. Sheffield, UK: Springer.