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

Version 3.1

ABSTRACT

This paper examines how databases can be integrated into the Grid. At the time of writing, most existing and proposed Grid applications are file-based. Consequently, there has been little work on how databases can be made available on the Grid for access by distributed applications. 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 currently very limited, and so the paper goes on to propose a service-based architecture to meet the requirements. In this architecture, database systems are wrapped within a Grid-enabled service interface that simplifies the task of building applications to 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 proposes a framework for federating database servers over the Grid, in which service federation middleware connects to the service interfaces of the set of database systems to be federated, and creates a "Virtual database system". This presents the same service interface to applications as do the individual, un-federated, database systems, so avoiding the need for applications to interface directly to a set of database systems and implement the federation functionality within the application. While the paper focuses on federating database systems, it also argues that the service-based approach will simplify the task of integrating databases with file-based data on the Grid, where that is required.

1 INTRODUCTION

This paper examines how databases can be integrated into the Grid [Foster and Kesselman 1999]. At time of writing, there is a dearth of Grid applications that use databases to store scientific data – almost all existing applications use files. Consequently, little thought has been given 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 could 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 services 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 this 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. This paper includes a proposal that attempts to resolve this conflict.

This diversity between different types 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 or organisations who will often have made local, independent decisions about both the best database paradigm and design for their data. 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 general middleware support for federating Grid-enabled databases.

Yet another level of heterogeneity also 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 such as XML, and relatively unstructured data such as papers, are valuable sources of information in many fields. Further, this type of data will usually be held in files, rather than a database, and even if it is stored in a database then it may be held as an unstructured Binary Large Object (BLOB). Therefore, in some applications there will be a requirement to federate these types of data with structured, database data.

Existing database management systems do not offer 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 in this paper, this approach does have it limitations as there are some desirable attributes of Grid-enabled databases that cannot be added in this way: they would 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 top of existing technology. In the longer term, if the Grid becomes commercially important, then it is to be hoped that database vendors will themselves provide support for Grid integration in their own products.

This paper addresses three main questions: what are the requirements of Grid-enabled databases? how far does existing Grid middleware 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 (the second half of 2001).

A detailed breakdown of 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 mechanisms 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 querying.

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 IN 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 range of uses of databases in the Grid include:

Metadata. This is information about information, and is important as it adds context to the scientific data made accessible on the Grid, increasing the confidence people have in the data they use, and making it possible to locate data without knowing exactly where it is stored, or its physical name. It appears that relational DBS are now already the main method for storing metadata in the Grid, because they can support metadata querying.

Provenance data. This provides information on the source and subsequent 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.

Resource inventories. To maintain an inventory of the location and characteristics of Grid resources. E.g. hardware, software, networking, applications, physical storage. This is essential for optimising and scheduling the use of resources.

Rule base. To define and maintain rules for determining optimal access paths to data, or to Grid resources (e.g. processing, storage) in terms of performance, cost, cost effectiveness etc. It can be used to optimise resource scheduling.

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.

Accounting. Grid accounting middleware may choose to store usage information in a set of databases. This information can be used for charging, but also for analysing Grid performance, and as the basis for capacity planning.

4 GRID DATABASE REQUIREMENTS

In this section we discuss the database requirements of Grid applications. The most important requirement is that if database data is to be accessible in Grid applications, then a DBS must support the relevant, existing and emerging, Grid standards; for example the Grid Security Infrastructure. It is equally important that those designing these standards should bear in mind the requirements of databases, with the aim of allowing Grid applications that access databases to be constructed in as straightforward a manner as possible. A typical application may consist of a computation that queries one or more databases and carries out further analysis on the retrieved data. If the compute and database systems all conform to the same standards for security, accounting, performance monitoring and scheduling etc. then the task of building such applications will be greatly simplified.

We now look beyond the basic requirement for Grid conformance, and consider whether Grid applications will place any requirements on the DBMS themselves. We see 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 machine resources used to store and process that data. The requirements of applications will vary, but they include:

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

We now consider whether Grid-enabled databases will have requirements beyond those found in existing systems. The Grid is intended to support distributed, large-scale sharing of information. The likely characteristics of such systems will, we believe, generate a 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 [Shiers 1998]. 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. A number of existing DBMS have been designed to meet very high scalability requirements by exploiting parallelism. However, the commercial environments for which they were designed do not have such extreme scalability requirements as will be found in some Grid applications. For this reason it may be that, for some very high performance applications, datasets will be partitioned and stored in files, accessible over the Grid, while databases are used to hold the Metadata that allows users to locate the parts of the datasets they require.

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, and usage restricted. For example, in many current e-commerce applications, databases are hidden behind a Web interface that only supports limited types of access. Further, typical commercial 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. The closest the commercial world comes to such highly unpredictable profiles is when data warehouses are used for analysis. However, even here this usually results in a set of small accesses to the database. Providing the less restrictive, ad-hoc access to databases required by e-science raises the additional problem of DBMS resource management. Current DBMS offer little support for controlling the sharing of their finite resources (CPU time, disk IOs and main memory cache usage). Therefore, if they were exposed in an open Grid environment, little could be done to prevent deliberate or accidental denial of service attacks.

Metadata-driven access. It is already generally recognised that Metadata is very important for Grid applications. Currently, the use of metadata in Grid applications is 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* that is analogous to the *Semantic Web* [Berners-Lee, Hendler et al. 2001]. Metadata-based access to data has important implications for integrating databases into the Grid because it promotes a two-step access to data. In step one, a metadata search 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 set of DBS. 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. The issues and problems of achieving this are discussed in Section 7.

Multiple Database Federation. One of the aims of the Grid is to promote the open publication of scientific data. If this 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 information on a single entity to gain a more complete picture, and to aggregate similar information about different entities. Achieving this will require support for integrating data from multiple DBS, for example through distributed query and transaction facilities. The middleware that supports this must meet the high scalability requirements described above. Indeed, when combining large quantities of data, the requirements on the middleware may, in some cases, exceed the requirements on the individual DBS. As was the case for Metadata-driven access, the design of 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 problem of the semantic integration of multiple databases, which has been the subject of much attention over the past decades, especially in the area of query access [Sheth and Larson 1990]. This problem increases with the degree of heterogeneity of the set of databases being federated, though the use of ontologies and metadata can lessen the problem. 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 from multiple, distributed repositories. In this paper we focus on federating structured data held in databases, but as is stated in the introduction, some applications will also need to combine structured, semi-structured and relatively unstructured data.

In summary, there are a set of requirements that must be met in order to support the construction of Grid applications that access databases. As we have shown, some of these requirements go beyond what is common for other types of applications. However, Grid applications also require the functions and properties provided by current DBMS. These are expensive to develop, and so reinventing the wheel 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 has to be achieved by wrapping the DBMS externally. It should be possible to meet almost all the requirements given above in this way (a method of achieving this is proposed in Sections 6 and 7). One exception is in trying to meet the requirement to provide better levels of resource control in order to avoid denial of service problems in open Grid-connected databases. Addressing this problem satisfactorily would require changes to the internal DBMS design.

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 emerging de facto Grid standards and middleware support databases. We limit ourselves to the two that are currently viewed as being the most important: Globus and the Storage Request Broker.

5.1 GLOBUS

Globus is the leading Grid middleware, but is focused on data held in files, rather than databases [Foster and Kesselman 1999]. Its main mechanism for accessing data is GridFTP, a high performance version of the file transfer tool, which has been integrated with the Grid Security Infrastructure (GSI). GridFTP has been integrated with one database – Objectivity – but only so as to allow the physical files in which Objectivity stores objects to be copied.

Despite the focus on files, Globus still has something to offer to databases. It is clearly vital that the integration of databases into the Grid exploits the GSI. This could provide 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 earlier). Some of the other Globus services 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 a remote application.

5.2 THE STORAGE REQUEST BROKER

The Storage Request Broker (SRB) was designed to provide applications with uniform access to distributed storage resources [SRB 2000]. Its main focus is file-based data. As well as the standard operations that are offered by distributed file systems, the SRB also offers some additional features, including:

- A Metadata Server (MCAT) that holds information on the data, users and resources managed by the SRB. It can also be used to hold application-specific metadata. However, a limitation is that there appears to be no general mechanism for connecting MCATs into a hierarchy (for example to allow the scalable federation of servers).

- A logical naming scheme for datasets. The mapping from the logic name to the physical file is done automatically when a dataset is accessed.

- Automatic replica creation and maintenance. When a replicated dataset is accessed, either the client, or the system selects which replica is to be used.

- A ticketing system that gives some control over access to datasets.

- A federation facility that allows a set of SRB servers to offer a single interface to clients.

- Authentication can be through the Grid Security Infrastructure.

Integration with GridFTP is currently under development. This will allow files held by a SRB server to be transferred using GridFTP.

As well as files, the SRB can manage data stored as Binary Large Objects in a DBMS.

The SRB is free to academic researchers, but a commercial version is currently being produced, and companies will have to pay to use it.

A later addition to the SRB was support for relational database tables. The documentation on this is limited, and is in a *readme* file within the source code distribution, rather than the main user manual. At a recent presentation on the SRB [Kremenek 2001] it was clear that the database interface was no longer supported, and so could not be considered an option for integrating databases into the Grid. However, for completeness, we now briefly discuss its functionality.

The SRB provided the ability to insert data into a database table, and to query tables. However, it did not meet all the requirements listed in the previous Section:

- There was no Transaction interface

- The client application interface was limited when compared to existing database interfaces. For example, there was no support for cursors in the client: the result of a query was returned as a block of data, with the structure identified by extra characters (e.g. linefeeds), or XML tags. It was left to the programmer to parse and process it.

- Prepared Statements were not supported.

- There was no support for integrating data from a set of SRBs, except that an MCAT can contain metadata information on the contents of a set of SRBs.

Another concern was that the documentation stated that each database could only have one user account.

There are no published performance results for the SRB (for file or database access). One specific performance concern is the fact that every client connection to the SRB server requires its own copy of the server process. This will cause performance problems for servers that need to support a large number of simultaneously attached clients, which is why DBMS avoid doing this - multiple threads, rather than processes, are preferable.

Therefore, it is clear that the SRB, like Globus, does not solve the problem of integrating databases into the Grid. However, it still provides an option for making file-based data available to applications.

6 INTEGRATING DATABASES INTO THE GRID

As neither current Grid software, nor existing DBMS are able to fully support the integration of databases into Grid applications, this Section and the next examine how this might be achieved.

An obvious basic starting point would be a Grid-enabled version of JDBC/ODBC or equivalent. This would allow a client to connect to a remote DBS on the Grid and execute queries within transactions. The JDBC/ODBC programming interface makes it relatively straightforward to generate queries and examine the results. It also supports stored procedures. Grid-enabling JDBC/ODBC would require integration with the Globus Security Infrastructure (GSI), so removing the need for the application to connect to each database with a different username and password - the single sign-on approach of the GSI would be used instead (this approach has recently been taken by the Spitfire project [Hoschek and McCance 2001]). This facility would be particularly useful if the application was accessing data from one of a set of databases, as directed by the result of a metadata query.

GridFTP is designed to move files around the Grid at high speed. There is therefore the opportunity to provide an option to use GridFTP to copy the results of a query back from the DBS to the client, in a single file. This would reduce communication costs for large result sets, at the expense of requiring sufficiently large buffer space on the DBS.

In other circumstances, for example where a significant computation needs to be performed on each element of the result set, a facility for streaming the results from the DBS to the compute server could be more efficient than waiting until all the results were produced and then copying them back to the client in one transfer. Globus does not provide a stream interface (thought this would be a useful addition to the basic Grid infrastructure) but at least one existing Grid application [Benger, Hege et al. 2000] uses the streaming facilities provided by the HDF5 package [Ross, Nurmi et al. 2001].

A JDBC/ODBC API could also be the basis for querying object DBS, using, for example, OQL as specified by the ODMG [Cattell 1997]. However, providing navigational access to objects in the database would require more effort.

The JDBC/ODBC approach is based on providing a programming interface that is limited to the core set of functions that almost all relational databases support: e.g. queries and transactions. By staying within this core set, programmers gain portability for their applications: a program that interfaces to a database in one vendor's DBMS through JDBC/ODBC can often be modified so that it can access an identically structured database held in another vendor's DBMS by editing the one line that specifies the DBS to which the program should connect. The caveat "often" is necessary as SQL is not fully standardised between implementations; however, the most commonly used constructs are common across DBMS.

The core set of functionality offered by JDBC/ODBC does not include a number of the operations listed in Section 4 as being required for fully integrating databases into Grid applications. Even where DBMS offer them, their functionality may vary from system to system. For these operations, one approach is therefore to rely on the programmer to find out what facilities are offered by the DBMS they wish to use, and write bespoke code to interface to them. However, the major disadvantage of this is the resulting lack of portability of applications across different DBMS, something that is important if two-step, Metadata-driven access to data on the Grid is to be supported.

We believe that a better approach would be to use a service-driven design. Each DBS would offer a set of services covering the areas described 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. This would be done by adding code to map the operation interface offered by a service 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 relation databases there are variations). One advantage of the service driven approach 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. This would give application builders the information required to exploit whatever facilities were available. It would also still allow the JDBC/ODBC type of access to databases from programs, as the implementation of the client interface would be through connections to the services offered by the target, Grid-enabled database system.

Figure 1 shows the service-based approach, 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). This view of wrapper code allowing a DBS to connect to the Grid is appropriate when we consider connecting an existing DBS to the Grid. However, an alternative, and equally valid approach would be to view the service interface as being an intimate part of a Grid-enabled DBS. Further, in the figure, the DBMS is shown contained within the DBS and distinct from the Grid service interface. However, in the future, if the commercial importance of the Grid increases, then DBMS may natively offer this type of Grid-enabled service interface.

Each of the proposed services shown in Figure 1 is now discussed:

Metadata. This service provides access to metadata about the DBS and the set of services that it offers to Grid applications. Examples include the logical and physical name and the DBMS type. The service description metadata would be used by application builders, and tools wanting to know how to interface to the DBS.

Query. Query languages differ across different DBMS, though the core of SQL is standard across most relational DBMS. Some Grid applications would benefit from methods of result delivery that current DBMS do not provide, e.g. the bulk transfer of results with GridFTP; serial streaming of results to a compute server for further processing; and, parallel streaming of results from a parallel DBMS to a set of parallel nodes for further processing. To provide input to scheduling decisions, and enable efficient distributed query processing across multiple DBS, an operation that provides an estimate of the cost of executing a query (without actually running the query) could be provided.

Transaction. These operations would support not only transactions involving only one DBS, but also to allow it to participate in application-wide distributed transactions, where the DBS supports it. Transaction managers could then be used to control distributed transaction execution. There are a variety of types of transactions that are supported by DBMS (for example, some but not all support nested transactions), and so some degree of heterogeneity between DBS is inevitable.

Bulk Loading. Support for the bulk loading of data, over the Grid into the database will be important in some systems. Integration with GridFTP for high–performance bulk loading of large datasets would be beneficial.

Notification. The ability for clients to register an interest in a set of data would be very powerful for some applications. 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 Grid event service and/or a Grid workflow system that determines what action is taken. For example, a user may be directly informed by e-mail, or an analysis computation may be run automatically.

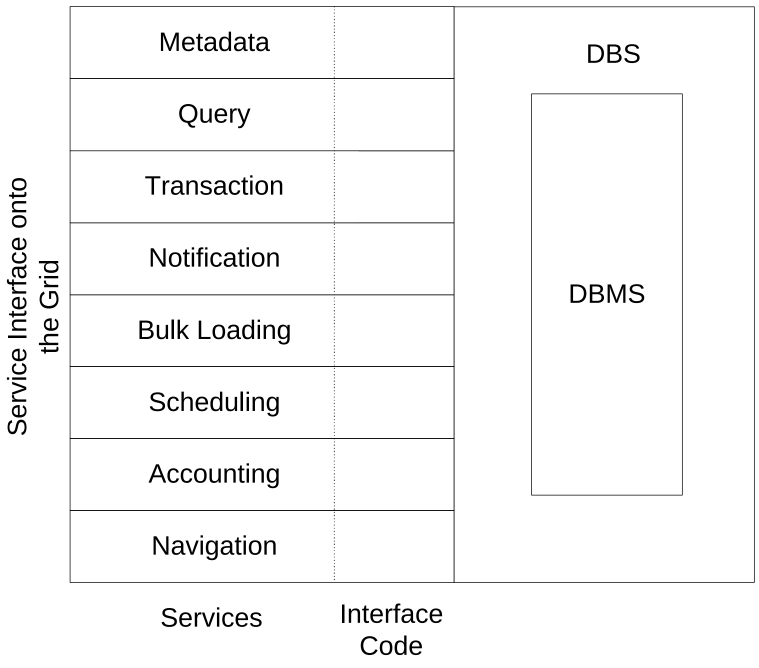

Figure 1. A Database System with a Grid-enabled Service Interface

Scheduling. This would allow users to schedule the use of the DBS. It should support the emerging Grid scheduling service, 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 be pre-allocated. As providing exclusive access to a DBS is unrealistic, 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, but cannot be implement outside the DBMS.

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.

Navigation. This service supports object databases that allow clients to directly access objects in the database as if they were ordinary programming language objects.

We do not claim that this list of services is definitive. It is based on our experience of building systems that utilise databases, but a more detailed analysis of the services required by potential Grid applications is needed. Even then, the need for new services may emerge as more experience is gained with Grid applications.

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. This could then be 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 all 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. In this section we consider how this might be achieved. 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 shown in Figure 2. However, this forces application writers to solve federation problems within the application itself. We believe that this would lead to great application complexity, and duplication of effort and so 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 (though it could 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 DBS that the virtual DBS is federating, in order to compute the result of the service call. This is shown in Figure 3. Because the Virtual DBS has an identical service interface to the "real" DBS, then it is possible for Virtual DBS to federate the services of both "real" DBS, and other virtual DBS.

Two different scenarios for the creation of a Virtual DBS are:

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.

2) A user wishes to work on some data that meets some criteria, e.g. data on Bacillus subtilis 168. A Metadata query would be used to locate the datasets meeting the criteria. These would then be federated to create a Virtual DBS that could then be used in the work. At the end of the work session, the Virtual DBS might 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 dropped.

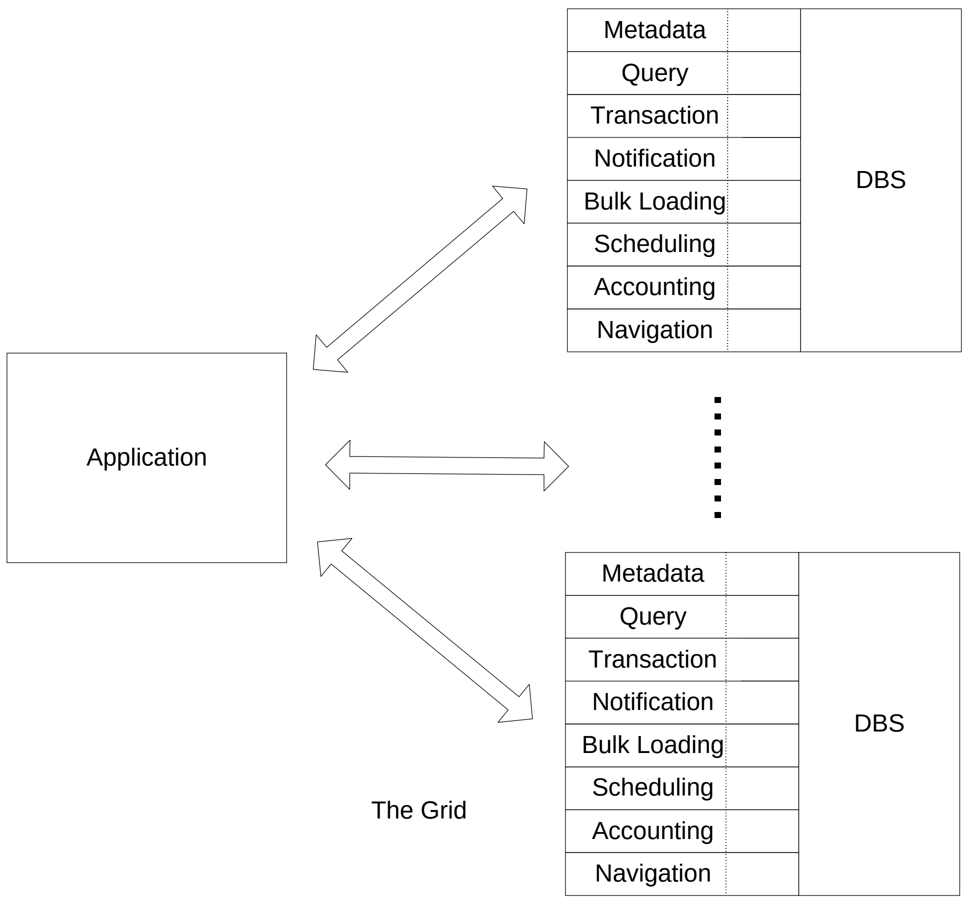

Figure 2.A Grid application interfacing directly to a set of DBS

How can the Virtual DBS be created? The ideal would be to have a tool that takes a set of DBS, and creates the Virtual DBS automatically. 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 create short-lived, Virtual DBS as in the second scenario above. With this in mind, we now consider the issues in federating services.

The service-based approach proposed in Section 6 assists in this 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. Integrating each of the different services proposed in Section 6 raises different issues as will now be described:

Query. Ideally this would present to the user a single integrated schema for the virtual DBS, and accept queries against it. A compiler 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 this distributed query middleware. Grid applications do however introduce new requirements that these do not meet, including the potential need for very high performance – distributed queries across large datasets may require huge joins that would benefit from parallelisation across large compute resources. The main problem occurs if it is necessary to integrate heterogeneous DBS in a Virtual DBMS. For example integrating a set of relational DBS, or a set of object DBS may be possible, but combining the two in one virtual DBMS raises major issues. As described earlier, this is the subject of on-going research in the database community, and the results of this work are likely to be of great benefit to Grid applications.

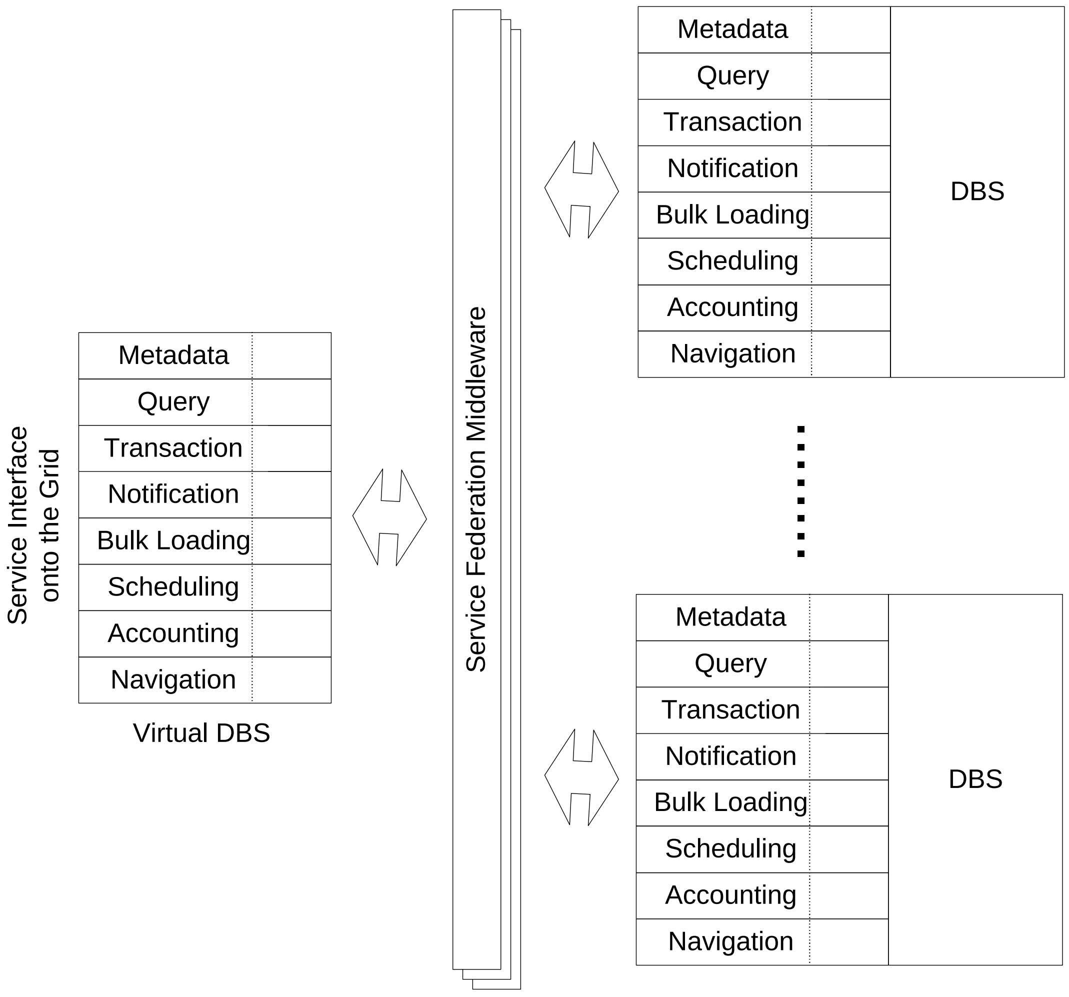

Figure 3. A Virtual Database System on the Grid

Transaction. Distributed transaction systems are well understood and widely available, e.g. the OMG Open Transaction Service. Again, the problems occur when heterogeneous DBS are combined, e.g. if an attempt is made to integrate a DBS that does not expose a transactional interface with some that do, or if nested transactions are required, but one DBS does not support them.

Bulk Loading. This should not be difficult to implement through 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 effect 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 would ideally be done using a Grid-enabled event and/or workflow service.

Metadata. This would describe the set of services offered by the Virtual DBS.

Scheduling. This would provide a common scheduling interface to the virtual DBS. When distributed scheduling middleware is available for the Grid, the implementation of a federated service should be relatively straightforward (though, as described above, there is a major problem is in controlling scheduling on 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.

Navigation. For object databases that support language bindings, this service would allow a client program to bind to, and access objects from any object DBS included in the virtual DBS.

Another useful service that could be provided at this level is Replication, which is already provided for file-based data by the Storage Request Broker. Middleware could create a replica, and then keep it in step with the original. Applications would be able to select the nearest, fastest or cheapest copy of data. The concurrency control regime that is used when there is an update could be selected by the application, as there is a trade-off to be made between consistency and cost, and different applications have different consistency requirements. Some database vendors already implement this type of service.

If each DBS has been wrapped within the service interface proposed in the previous section in order to minimise heterogeneity, then it is possible to envisage how a tool could attempt to create a Virtual DBS automatically. For each service, it would query the metadata service of each "real" DBS in order to determine the supported functionality. Knowing what integration middleware was available for that 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 options would be to not use the service, or to write new federation middleware to effect the integration, if that is possible.

The complexity of the service federation middleware shown in Figure 3 will vary from service to service, and will, in general, increase as the degree of heterogeneity of the services being federated increases; federating the query service for two databases built from identical DBMS will be much easier than federating the query service of two DBMS from different paradigms. In some cases, federation will be impossible, and so the application will have to interact separately with each of the constituent DBS that it would like to federate into a single Virtual DBS. 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 a need for other types of federation. When this occurs, then the hope would be 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. Having a standard framework for federation middleware (all middleware takes a set of standard service interfaces as "inputs" and presents a single, standard federated service as "output") also means that such middleware is easier to write, and more generic than is currently the case where each DBMS has its own, different interfaces. Further, this approach should maximise the range of both DBMS and applications that can utilise each instance of the federation middleware.

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. 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 services and database services.

8 CONCLUSIONS

We have identified a set of requirements for Grid databases, and shown that the existing Grid middleware does not meet them. However, this does not mean that existing Grid middleware can or should be ignored. For example, the Grid Security Infrastructure needs to be supported by all Grid databases so as to allow seamless access by clients, while GridFTP could be used both for bulk loading, and the transfer of query results.

The paper proposed a set of services that should be offered by a Grid-integrated DBS. This service-based approach will simplify the tasks of making databases available on the Grid, and 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 exiting Grid standards and services (e.g. GSI & GridFTP), and to introduce a level of 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. Work is also required to integrate with emerging Grid event and workflow services. Finally, *Scheduling* is the most problematic as to do this properly requires a level of resource management not found in existing DBMS.

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. However, to minimise development, every attempt should be made to adopt and adapt existing middleware, for example OMG distributed transaction services.

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.

9 ACKNOWLEDGEMENTS

This paper was commissioned by the UK e-Science Core Programme. I would like to thank the following for their constructive comments on an earlier version of this paper: Dave Pearson, Pete Lee, Tony Storey, Anne Trefethen, Alex Gray, Malcolm Atkinson, Vijay Dialani, Jim Gray, Clive Page, Guy Rixon, Norman Paton, Ian Foster, David Skillicorn and Tony Hey.

10 REFERENCES

Benger, W., H.-C. Hege, et al. (2000). Efficient Distributed File I/O for Visualization in Grid Environments. Simulation and Visualization on the Grid. B. Engquist, Springer. **13**.

Berners-Lee, T., J. Hendler, et al. (2001). "The Semantic Web." Scientific American **2001**(May).

Cattell, R. (1997). Object Database Standard: ODMG 2.0, Morgan Kaufmann.

Foster, I. and C. Kesselman, Eds. (1999). The Grid: Blueprint for a New Computing Infrastructure. San Francisco, Morgan Kaufmann Publishers Inc.

Hoschek, W. and G. McCance (2001). Grid Enabled Relational Database Middleware. Global Grid Forum, Frascati, Italy.

Kremenek, G. (2001). Lecture on The Storage Request Broker. Rutherford Appleton Laboratory, Oxford, UK, 27th September 2001.

Ross, R., D. Nurmi, et al. (2001). A Case Study in Application I/O on Linux Clusters. Supercomputing 2001, Denver, ACM.

Sheth, A. P. and J. A. Larson (1990). "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases." ACM Computing Surveys **22**(3): 183-236.

Shiers, J. (1998). Building a Multi-Petabyte Database: The RD45 Project at CERN. Object Databases in Practice. A. B. Chaudhri, Prentice Hall: 164-176.

SRB (2000). Storage Request Broker Documentation v1.1.8, http://www.npaci.edu/DICE/SRB/CurrentSRB/SRB.htm.