OGSA-WebDB: An OGSA-Based System for Bringing Web Databases into the Grid

Said Mirza Pahlevi and Isao Kojima
National Institute of Advanced Industrial Science and Technology (AIST), Grid Technology Research Center, Tsukuba, Ibaraki 305-8568, Japan {mirza, kojima}@ni.aist.go.jp

Abstract: An OGSA-based system, OGSA-WebDB, is proposed, which enables grid applications to query web databases using a standard database query language (SQL). OGSA-WebDB consists of two main components: proxy databases and a mediator. Proxy databases represent the desired web databases while the mediator acts as an interface between the proxy databases and the web databases. The mediator accepts an SQL query from local applications and transforms the query into one or more Boolean conditions that are then sent to the target web databases. The mediator processes the SQL query in parallel, taking into account the characteristics of the web databases. Experimental results revealed that the query processing time is so small that it can be ignored for timing considerations. The system has been fully implemented on top of Globus Toolkit and OGSA-DAI software components.

Key words: Web databases, grid technology, grid security architecture
Reviewed and accepted: 31 Mar. 2004

1. Introduction

Grid computing technology [23, 25] has been widely used in many scientific and commercial applications. Many of these applications use database management systems (DBMSs) to store and manage important datasets. There is an urgent, widespread need to interconnect pre-existing, independently operated databases in the grid environments [31, 15]. This has resulted in the formation of the Database Access and Integration Services Working Group (DAIS-WG) [4] in the Global Grid Forum (GGF).

Very often scientists and other users need to access and integrate information from multiple sources in order to obtain the information they desire. This may include accessing information from data sources available online on the web (called web databases). For example, a biologist designing a new drug might wish to access information from a drug patent database [21] or a biomedical literature citation database [11] and integrate it with information from local DNA databases.

In general, many web databases cannot be accessed directly using local database drivers or be queried using a specific database language, such as SQL. Instead, they must be accessed using HTTP GET or POST requests via web search interfaces and queried using keywords combined with Boolean operators. Web database owners usually do not make the metadata of their databases available.

The current implementation of specification [18, 19, 8] assumes that grid applications and grid database components can query databases using database languages, and that the databases can be accessed using specific drivers, such as JDBC. In addition, this specification requires that the database owners cooperate by exposing their database metadata. Clearly, the enforcement of these grid-database requirements will limit access to valuable information on the web by many potential grid users or applications.

One way to enable access to web databases from within a grid is to use a mediator [33] to serve as an interface between these two systems. In order to provide an effective interface, the mediator must satisfy the following grid-specific requirements:

- It must provide an interface between the grid and web databases that allows web databases to appear to satisfy the above grid-database requirements.
- It must comply with grid specifications so that it can be plugged easily into any grid environment.
- It must support the Grid Security Infrastructure (GSI) for authenticating web databases.

So far, many mediator/data integration systems have been proposed as noted in Section 7. However, most of them do not satisfy one or more of the above requirements. Therefore, they cannot be used to mediate between grid and web databases.

In this paper, an OGSA-based system, OGSA-WebDB, is proposed that enables access to web databases and also provides for the integration of web databases from within the grid. The system is designed such that it satisfies the above grid-specific requirements. To accomplish the mediation task, the system uses an adaptive query execution algorithm that takes into account the characteristics of target web databases.

OGSA-WebDB has been fully implemented on top of standard grid software components. Experiments have been conducted to evaluate its effectiveness and efficiency.

The rest of this paper is organized as follows. Section 2 gives an overview of database access in the grid. Section 3 describes the system architecture of OGSA-WebDB. Section 4 explains the query execution algorithm. Section 5 presents experimental results. Section 6 describes the implementation of the prototype system. Section 7 reviews related work. The final section gives our conclusions and some proposals for future work.

2 Database Access in the Grid

The grid data service specification [17] defined by the DAIS-WG is a specification of data access and integration on the grid based on the Open Grid Services Architecture (OGSA) [9]. OGSA-DAI [8] developed by the UK e-Science Programme project is software components that implement the specification and works with the Globus Toolkit (GT) [24]. It defines Grid Data Service (GDS) that connects grid applications with a database.

Fig. 1 outlines how a database can be accessed from within the grid. First, an application searches a Grid Service Registry (GSR) for a Grid Data Service Factory (GDSF) that can create a GDS for the required database (step 1). After receiving a Grid Service Handle (GSH) from the GDSF (step 2), the client asks the GDSF to create a GDS (step 3). The GDSF then creates a GDS (step 4) and returns...
the GSH of the newly created GDS to the client (step 5). Finally, the client uses the GSH to retrieve data from the database (step 6). To access the database, the GDSF and GDS need to know the metadata of the target database. The metadata includes the database schema, database driver, and database language(s) supported. The application also needs to know the database schema to formulate a query in the supported database language.

3. The Proposed System

3.1. Overview

In this paper, we assume that search interfaces for web databases support at least a simple conjunctive Boolean query format (i.e., keywords connected with AND operators) and optionally, may also support a field search. Search interfaces may also require authentication before a user/application can access the data. We assume that the query and authentication data will be sent via HTTP requests. Access will only be made to a relational DBMS and data requests will only use an SQL SELECT statement. We chose a relational DBMS because the technology is already mature and most of the data of interest is currently stored using this system.

Our goal was to construct a grid/web database mediation system that satisfies the grid-specific requirements. To do this, we built a system that consists of two main components: proxy databases and a mediator. Proxy databases are databases that are already on the grid (i.e., they can be accessed by grid applications). They are used to interface to web databases. Each web database is represented by a proxy relation within a proxy database. Therefore, any grid application wishing to access a web database will access its proxy relation instead.

The mediator bridges the gap between the proxy databases and the desired web databases. It accepts an SQL query (sent by a grid application) and transforms the query into one or more Boolean conditions sent to the target web databases. The query processing capabilities of the search interfaces for the web databases are very limited. Therefore, the mediator transforms only the part of the query that can be processed by the search interfaces. It leaves the remaining part to be processed by a DBMS that manages the proxy relations for the web databases. When the results are returned from the web databases, the mediator inserts the results into the appropriate proxy relations of web databases. Then, the grid applications can access the results using the proxy relations.

By using proxy databases OGSA-WebDB allows seamless access to web databases from within the Grid. This also allows OGSA-WebDB to make use of the query processing capability of the existing relational DBMS. The grid single sign-on mechanism is conserved by storing authentication information in management relations maintained in the proxy databases.

Details of the management relations are given in the next section.

3.2. System Architecture

Fig. 2 shows the system architecture and procedures involved in the web database access mechanism. When an SQL query is received (step 1), a GDS invokes the mediator and passes the query and a username to the mediator (step 2). The username is the name that will be used to log into the DBMS that maintains the proxy databases. The SQL analyzer module in the mediator accepts the SQL query and the username and extracts the proxy relation and column/attribute names from the query. The extracted information and the username are then passed to the database connection module (step 3). This module then uses them to obtain the necessary information for querying the web databases. The query information is retrieved from the management relations stored in a proxy database (step 4). In particular, the database connection module does the following.

1. It determines target web databases and retrieves names of their wrapper classes.
2. It retrieves the authentication information for web databases that require user authentication.
3. It maps column names in the SQL query into appropriate (field) names used to query web databases.

The following are the management relations and their schemata. Note that a ‘#’ in front of a column name indicates that the column is a primary key column, while a ‘*’ indicates that the column is a foreign key column.

- **User(#uid,localName,description):** this relation manages the usernames used to log into the proxy database system. **localName** contains the usernames.
- **Authentication(uid,extName,passwd,*wid):** this relation manages the authentication data. **extName** and **passwd** contain usernames and passwords used to log into web databases that require user authentication. **wid** and **uid** is a foreign key column containing web database and local user identifiers, respectively.
- **Resource(#wid, relName, url, wrapper, login):** this relation manages information about web databases. **relName** and **url** contain names of proxy relations and URLs of web databases, respectively. **wrapper** contains names of accessible wrapper classes that wrapped the web databases, **login** contains ‘y’/’n’ flags indicating whether a web database requires authentication.
- **Fieldmapping(#wid, localColumn, extField):** this relation maps column names of proxy relations into field names in the field search of web databases. **localColumn** and **extField** contain the column and field names, respectively.

Data in the management relations is maintained by a system administrator using the proxy database management module.

For example, to include a new web database, the administrator inserts an entry into the **Resource** relation specifying the wrapper and the proxy relation name of the web database. If the web database requires user authentication, the authentication data for (related) users in the **User** relation is added into the **Authentication** relation. The management module is also used by the administrator to install OGSA-WebDB into a grid environment. This module is a grid client, so it uses a GDS to accomplish its tasks.

Authentication mapping is necessary because a user might have different login names for different web databases that each require authentication and we need to keep a single sign-on for the grid in our architecture. Our system implementation, which is described in Section 6, uses the OGSA-DAI software components [8]. OGSA-DAI...
maps X509 credentials supplied by a client of a GDS to a username and the password for a specific database to which the GDS provides access. OGSAWebDB accepts the username from the GDS and then maps it into usernames and passwords that are used to log into a web database.

The database connection module passes the retrieved query information into the Internet connection module (step 5). This module extracts from the SQL query the query conditions that can be processed by the search interfaces of the target web databases. Then it passes these conditions into the wrappers of the target web databases. If the web database requires user authentication, the authentication data are then passed to the wrappers. Details of the SQL query processing are given in the next section.

A wrapper extracts the desired data from HTML pages returned by a web database in response to a query. These data are transformed into an XML document, which is then passed to the database connection module (step 6). When it receives the XML document, the database connection module inserts the query result into the appropriate proxy relations (step 9).

Finally, after all of the proxy relations (included in the SQL query) have been updated with the data from the web databases, the GDS sends the SQL query to the proxy database (step 10). The DBMS that maintains the proxy database processes the query and forwards the result to the GDS (Step 11). Note that step 10 can be performed only after the proxy relations of the target web databases have been updated with the necessary data (i.e., after steps 2–9 have been completed). The updated data are removed once they have been successfully accessed by the GDS.

4. SQL Query Processing
In this paper, we deal with SQL queries with the following WHERE clause: WHERE JC AND C. JC is a conjunction of join conditions, “jc1 AND ... AND jcq” (q ≥ 0), and C is a condition defined below:
• It is a conjunction of conditions “cc1 AND ... AND ccq” (q ≥ 1) where cc is an SQL query condition “r.col op value”, col is a column name of relation r and op is an SQL operator, r could be a proxy relation or an ordinary local relation.
• If conditions for r exist, where r is a proxy relation, at least one of the conditions is in the form “r.col LIKE %kw%” or “r.col = kw”. kw is a keyword, “LIKE” is a pattern matching operator and the percent sign “%” is a wild card to match any possible character. For easy reference, we denote this condition keyword-based condition.

The main goal of the query processing is to retrieve data from web databases and local relations that matches conditions in the WHERE clause in a fast and efficient manner. To do this, we need to transform the conditions into a form that is acceptable by the target web databases. However, it is not a trivial task because query processing capabilities of the web database search interfaces are very limited. For examples, they cannot handle join queries and some of them such as DBLP [20] cannot process range queries such as ‘year > 2000’.

Our approach is to process an SQL query in two passes. The first pass fills all of the proxy relations included in the WHERE clause by retrieving data from their respective web databases. For a proxy relation that has conditions in the WHERE clause, the data are retrieved by using these conditions. For a proxy relation that has no conditions, the retrieval conditions are created using data from a local relation or a proxy relation with conditions based on a join condition that join the two relations (the semijoin operation). The (newly) created conditions are then used to retrieve data from the respective web database. The second pass retrieves the data from the updated proxy relations and the local relations included in the WHERE clause that satisfy the rest/all of the conditions in the clause. This pass is accomplished using the query processing capabilities of the DBMS that maintains the relations.

4.1. First Pass Query Execution
Fig. 3 summarizes an algorithm used in this pass. It assumes that the mediator supports the following queries and operators.

1. Selection query denoted as $\sigma_{\phi}(r, X, col)$. This query retrieves a set of data items that satisfies condition $\phi$ from relation r. If r is a proxy relation, then the items are retrieved from the respective web database using keyword-based conditions only (i.e., non keyword-based conditions in the retrieve condition are ignored).

2. Semijoin query denoted as $\sigma_{\phi}(r, X, col)$. This query retrieves a set of items from relation r using values in item set X that corresponds to column $col$ of r. If r is a proxy relation, then the items are retrieved from the respective web database.

3. Parallel operator denoted as $\parallel_{\phi_{1}, \ldots, \phi_{m}}(X)$. This operator executes queries $q_{i},...q_{m}$ in parallel and puts the returned item sets into $X_{1},...X_{m}$.

4. Insert operator denoted as $\text{insertOp}(r, X)$. This operator inserts item set X into relation r.

The algorithm starts by constructing selection queries using conditions in C (lines 1-3), and then invokes $\text{exec}$ method while passing the queries (line 5). The selection queries include queries that retrieve data from web databases to fill their proxy relations. $\text{exec}$ executes the passed queries in parallel and puts the returned item sets into variables Xs to be further processed (line 6). Each time a query execution has been completed, the returned item set is immediately used to construct semijoin queries to fill other proxy relations (lines 7-19). This first-come-first-served basis procedure will

Figure 3. Query execution algorithm

Figure 4. A condition that joins four relations

Note that the query may also include relations other than proxy relations (i.e., local relations).
significantly shorten the overall response time and be the basis to decide which relation is used to fill the remaining unfilled (proxy) relations. For each retrieved item set \( X \), if it is retrieved from a web database, then the item set is inserted into the database’s proxy relation (lines 9-11). \( \text{EXEC} \) then gets all join conditions in \( JC \) such that the conditions join \( r_2 \) with other relations in \( R \), and puts them into \( \text{tup} \) (line 12). All relations and join conditions included in \( \text{tup} \) are then removed from \( R \) and \( JC \) (lines 14 and 15) because they will be processed in the next lines. Since \( r_2 \) might be joined with more than one relations, join conditions in \( \text{tup} \) are further grouped based on relations which \( r_2 \) is joined with (line 16). The algorithm then constructs a semijoin query for each join condition in the join condition group (lines 17 and 18), and selects a semijoin query with a minimum execution cost (line 19).

The execution cost is a function of a time period to complete the semijoin query (i.e., query response time). The query response time is estimated with the number of distinct values in the joined column. More specifically, it is expected that the response time is proportional to the number of the distinct values. That is, the larger the number of the distinct values, the longer the response time will be.

Finally, the created semijoin queries are executed in parallel by recursively calling method \( \text{EXEC} \) (line 20). The algorithm stops if \( \text{tup} \) is empty (line 13).

An Example

Fig. 4 shows a condition that joins four proxy relations. A join condition \( (jc) \) is shown by a solid line connecting two columns \( (col) \) and an SQL condition \( (c) \) on a column is shown by a circle connected to the column. Note that \( c_1 \), \( c_2 \), and \( c_3 \) are keyword-based conditions, while \( c_4 \) is not.

is executed. Suppose that the execution of \( pi \) has first completed. The algorithm then constructs semijoin queries from data items of \( r_2 \) to \( r_3 \) and \( r_4 \). It creates \( jc_2 \) and \( jc_3 \) and groups the join conditions based on a relation that joined with \( r_2 \). A semijoin query for \( r_2 \) is created based on the first element (i.e., \( (jc_2, \{jc_2\}) \)). Similarly, a semijoin query for \( r_4 \) is created based on the second element (i.e., \( (jc_3, \{jc_3\}) \)). However, since there are two join conditions, we have to select one of the conditions to construct the queries. Suppose that the cost for creating semijoin query based on \( c_3 \) is smaller than that based on \( c_2 \). Thus, the algorithm creates semijoin query: \( \sigma_{jc_3} \) for \( r_4 \). Finally, the semijoin queries are executed in parallel with \( \text{paraOp} \) operator.

4.2. Second Pass Query Execution

This pass is done by a DBMS that maintains proxy databases. It accepts an SQL query from the GDS and processes the query against the local relations and (the already filled) proxy relations. In particular, this pass performs a field projection, join operations, selections of data items based on keyword-based and non keyword-based conditions, and other SQL statement constructs such as aggregation functions, \( \text{GROUP BY} \) and \( \text{ORDER BY} \) clauses. Note that, non keyword-based conditions on proxy relations that are ignored in the first pass are accomplished in this pass.

5. Experimental Evaluation

This section evaluates query processing time and parallel query execution of OGSA-WebDB. The query processing time is the time needed to accomplish the first pass query execution. It consists of the mediator processing time and the web database retrieval time. The former is the time needed to analyze an SQL query, look up management relations and insert the data into the proxy relations. The latter is the time needed to retrieve the data from the web databases. Since the second pass is done by a DBMS, which usually has a very fast query processing time, we have chosen to ignore the query execution time of the second pass. The experiments were performed in the daytime on a Windows based PC with 1.5 Ghz Pentium processor and 1 GB RAM.

Fig. 5 shows the query processing time of SQL queries querying various web databases: DBLP [20], the Collection of Computer Science Bibliographies (CSB) [3], CiteSeer [12], ChemFinder [2], Drugs@FDA [5], and Protein Data Bank (PDB) [10]. Five simple SQL queries (i.e., queries without join operations) were constructed for each web database and the average of the query processing times was taken. As shown in the figure, web database retrieval times depend heavily on the web database sources, while mediator processing times are almost the same for all database sources. As the mediator processing time is very fast (only tens of milliseconds) compared to the retrieval time (which is several seconds), we have chosen to ignore it.

\[ \text{query execution time of the second pass} = \text{query processing time} - \text{mediator processing time} \]

This may happen because the response time is usually different for each web database.
Fig. 6 shows the query processing time for various numbers of retrieved data items for CSB 5. For each retrieved item number, an SQL query is executed five times and the average of the query processing times is taken. The figure shows that as the number of retrieved items increases, the retrieval time also increases, but the mediator processing time remains almost the same. Based on two experimental results, it can be concluded that the mediator processing time is not affected by the web database sources and the number of retrieved data items. Therefore, since this processing time is so small, we have chosen to ignore it.

Fig. 7 shows web retrieval time using parallel and serial query execution. The result of the mediator processing time is almost the same with that obtained in the previous experiments so we omitted it. There are four SQL queries: q1=q2 and three web databases: DBLP, CiteSeer, and CSB used in the experiment q3=q4 and q5 are SQL queries that join two proxy relations (web databases) while q6 joins three proxy relations. The queries are constructed such that the mediator will perform semijoin queries to accomplish the query execution. This is done by placing a keyword-based condition on a column of a relation and joining the relation with the others on another column. For example, q1 joins DBLP and CSB on their title columns and places a keyword-based condition on the year column of DBLP (e.g., “dblp.year = 2000”). We execute each query five times and take the average of the query processing time. For each query execution, we use a different keyword for the keyword-based condition. From the figure, it is clear that parallel operation can significantly reduce the retrieval time. The reduction becomes larger as the number of semijoin queries increases (i.e., from joining two relations to three relations). This comes from the fact that semijoin queries against several web databases can be executed in parallel and so too a set of selection queries (against a web database) which is constructed from a semijoin query. This result shows the advantage of using the parallel operator in the algorithm.

6. System Implementation

We have fully implemented OGSA-WebDB using Java. Currently, the system runs on top of GT 3.0.2 [6] and OGSA-DAI 3.1 [8]. It provides access to several scientific, drug, patent, and bibliography web databases. The service factories of the web databases are deployed in a Tomcat servlet container [1], and the proxy databases are stored in an open source MySQL database [7]. The web database wrappers were written using Compaq’s Web Language (WEL) [13] and Republica’s Xfitch Wrapper [14].

Fig. 8 shows a snapshot of a grid client. The client sends an SQL query that joins data from two web databases (Drugs@FDA [5] and ChemFinder [2]) and a local mydruglist relation. The query retrieves drug names containing active ingredients, where the names of the ingredients contain string “pri”, from Drugs@FDA. Then it obtains the brand name and the manufacturer of the drugs from mydruglist and the Chemical Abstract Service (CAS) number of the drugs from ChemFinder. It should be noted that ChemFinder is a web database that requires authentication, and that OGSA-WebDB handles the authentication process automatically.

Fig. 9 shows a snapshot of the proxy database management tool that installs a new wrapped web database into the system. This tool also helps an administrator to install OGSA-WebDB in a grid environment by updating some grid configuration files. It is implemented as a grid client so it can work in a grid shared environment.

7. Related Work

The web and database communities have been investigating how to integrate and access data across multiple/heterogeneous data sources for a long time. Fundamentally, there are two approaches to accessing and integrating data sources: the materialized warehousing approach and the virtual approach [22]. In the materialized approach [36, 35], data from multiple sources are loaded into a central warehouse/database. Then, queries are constructed and processed against the warehouse. In the virtual approach, a query is constructed against an integrated view of the data sources (i.e., mediated schema) that consist of virtual relations designed for a particular data integration application. Since data remain in the data sources (i.e., the data are not loaded into a central storage), the query must be reformulated or decomposed into subqueries sent to the remote sources. This approach is appropriate for sources whose data change frequently and for autonomous data sources. For these reasons, research on integrating web databases and conventional databases has focused on a virtual approach [26, 29, 28]. This approach is also used by systems that integrate data sources with more specific data, such as biomedical data [16, 34, 27].

Our approach is close to the virtual approach. However, most of the systems using a virtual approach are not gridware systems; they use a proprietary SQL, such as SQL combined with a graph structure/object oriented concept [26, 36], description logic-based languages [29], or collection programming language (CPL) [16, 34, 27] and they are not aware of the GSI. Another aspect that differentiates our approach from existing systems is that it makes use of the query processing capabilities of a DBMS located inside the grid. This greatly reduces the query processing time of the mediator.

Distributed query processing has also been executed on the grid. OGSA-DQP [30] is a distributed query processing system that runs on the grid and integrates various database systems. As our system enables grid applications to access web databases, OGSA-DQP may also include web databases in its environment using our system.

8. Conclusions and Future Work

The number of data sources available online on the web continues to increase rapidly. Many of these data sources store scientific, engineering, medical, and other data that may be very useful for grid applications. OGSA-WebDB, which complies with grid specifications, enables grid applications to access these data sources seamlessly using a standard database query language SQL. To achieve this, it uses proxy databases and a mediator approach.
We are now considering the use of proxy relations as data caches. As soon as the data have been accessed by a GDS, OGSA-WebDB removes the data from the proxy relations. However, instead of removing the data, the stored data could be used as a cache to satisfy subsequent queries. This greatly reduces the query processing time of the mediator.

Another issue is utilizing source descriptions in the SQL query processing. A source description is metadata concerning the query-processing capability of each search interface. Most search interfaces support pattern-matching operators, such as “LIKE” in SQL, but several also support other important operators. For example, search interfaces for patent, scientific journal, and bibliography web databases may also support searches of publication year using equality and inequality operators. Given such search interfaces with richer query processing capabilities, we can transform many more conditions in the WHERE clause into a Boolean condition so that precise data selection can be “pushed” to the first pass earlier.

References
3. CSB. http://liinwww.ira.uka.de/bibliography.
4. Database Access and Integration Services Working Group (DAIS-WG) of the GGF. http://www.ggf.org/6 DATA/dais.htm
5. Drugs@FDA. http://accessdata.fda.gov/scripts/cder/drugsatfda
7. MySQL. http://www.mysql.com/
10. PDB. http://www.rcsb.org/pdb
13. WEBL. http://research.compaq.com/SRC/WebL/
20. DBLP. http://www.informatik.uni-trier.de/ley/db/
21. DOLPHIN. http://www.cp-dolphin.com
30. OGSA-DQP. http://www.ogasadai.org.uk/docs/dqp/