2009 International Conference on Advanced Information Networking and Applications

Supporting Complex Scientific Database Schemas in a Grid Middleware

Helen X Xiang
Computer Science, University of Hertfordshire, UK*
helen.x.xiang@gmail.com

Abstract

The volume of digital scientific data has increased considerably with advancing technologies of computing devices and scientific instruments. We are exploring the use of emerging Grid technologies for the management and manipulation of very large distributed scientific datasets. Taking as an example a terabyte-size scientific database with complex database schema, this paper focuses on the potential of a well-known Grid middleware—OGSA-DQP—for distributing such datasets. In particular, we investigate and extend the data type support in this system to handle a complex schema of a real scientific database—the Sloan Digital Sky Survey database.

1. Introduction

The *Sloan Digital Sky Survey* (SDSS) is a project that has built a very detailed digital map of the visible stars and galaxies in the night sky [23]. The data produced by the survey is summarised to a multi-terabyte relational database containing photometric objects and spectroscopic information. The SDSS database is available to the scientists and the public via the SkyServer (http://skyserver.sdss.org) or various mirror sites (including one we set up in the University of Portsmouth).

In recent papers [19, 20, 21] we described a how we created an experimental distributed version of the SDSS DR5 database, using Grid middleware. This is based on OGSA-DQP (Open Grid Services Architecture—Distributed Query Processing), developed by the University of Manchester and the University of Newcastle upon Tyne [10]. OGSA-DQP is based on the OGSA-DAI middleware, developed by the Edinburgh Parallel Computing Centre (EPCC) and the UK National e-Science Centre (NeSC) [9].

The original SDSS database was implemented in Microsoft SQL Server. We created a copy of SDSS DR5 SQL Server database in the University Portsmouth. A large subset of this database (table objects and data) was then migrated to Oracle database and transferred to a Linux server in Manchester *National Grid Service* (NGS). The rest of this database stays in a Windows server in Portsmouth. We used OGSA-DAI and OGSA-DQP to integrate the data across these two sites—forming a logical distributed database system. Global distributed queries can be processed over this logical database system [22].

This experiment exposes some limitations of the emerging Grid technologies. In particular, this paper will focus on the extension of data type support in OGSA-DQP that we added to support a realistic large scientific database.

This paper begins by describing the OGSA-DQP system and its architecture. In particular, we analyse the OGSA-DQP workflow and the interactions among its components. We also explain the data type support and the OGSA-DQP flow of type information. The paper then describes how we extend the data type support to support large schemas in Oracle and SQL Server.

2. The OGSA-DQP System

Extending the multiple-data source functions of OGSA-DAI, the DQP system is a service based distributed query processor for planning, scheduling and executing distributed queries in parallel [1, 2, 4, 7]. OGSA-DQP evaluates against distributed data sources that are exposed by OGSA-DAI data service resources. OGSA-DQP aims to provide homogeneous access to heterogeneous data sources over OGSA-DAI middleware. It enables transparent distribution and parallelism among Grid data services, and supports Grid abstractions for on-demand resource allocation. The first release of OGSA-DQP was in September 2003, and the version 3.2 *Tech Preview* was the latest revision at the time when our research was undertaken. OGSA-DQP 3.2 Tech Preview is based on OGSA-DAI WSRF/WSI 2.2. The final version of OGSA-DQP 3.2, based on OGSA-DAI 3, was released as this paper was being written. We expect most of our conclusions would carry over to the new release.

OGSA-DQP currently supports MySQL and has been

*Previous address: Institute of Cosmology and Gravitation, University of Portsmouth.

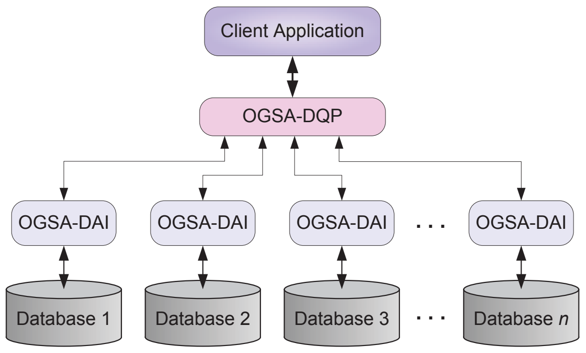

Figure 1. Mediator-Wrapper Vision of OGSA-DQP

test in myGrid project in simple bioinformatics databases [3].

2.1 The OGSA-DQP architecture

Adopting the mediator-wrapper approach [18], the OGSA-DQP prototype uses OGSA-DAI middleware as the wrappers and DQP as the mediator. In Figure 1, databases are wrapped by OGSA-DAI data services.

There are two different types of services in OGSA-DQP: the *Grid Distributed Query Service* (GDQS) and the *Query Evaluation Service* (QES). The GDQS—also known as the *DQP coordinator*—is responsible for interacting with the client applications, and parsing and scheduling distributed query executions. The QES—also known as the *DQP evaluator*—is responsible for the actual query execution such as joins.

The DQP coordinator is implemented as an OGSA-DAI data service component for extracting the database schema information via the data service resources which exposes the underlying databases. The database schema, metadata and resource computational information are held in XML format, and this information helps the query compiler in the DQP coordinator to make decisions when parsing the distributed queries, generating and optimising the query execution plans. Once a query plan is generated by the query compiler, the DQP coordinator communicates with the DQP evaluators to run the query and ensures the query is executed according to its (query) plans. Like any other OGSA-DAI services, the DQP coordinator can be discovered and invoked from the command line or via the client software.

2.2 OGSA-DQP Workflow

Figure 2 presents the interactions inside the DQP Query Compiler. The distributed query departs from the client first

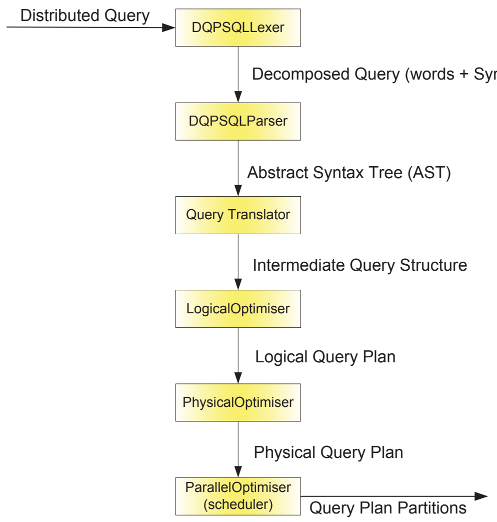

Figure 2. Interactions inside the DQP Query Compiler

to the *Lexical Analyser*, which breaks it into words and symbols before passing it to the *Query Parser*. It is parsed by the Query Parser to produce an *abstract syntax tree* (AST). Then the *Query Translator* converts the AST to an intermediate representation of the query structure. This is fed into the *Logical Optimiser* and *Physical Optimiser*.

The next three phases build the query plan. A query plan is a tree of operations such as `TABLE SCAN` and `JOIN` operations. The Logical Optimiser analyses `SELECT` and other statements and produces a logical query plan, which consists a series of joins of intermediate-results tuples from the `TABLE SCAN` operations. An important function of the Logical Optimiser is making estimates of the table cardinalities of the intermediate results. It places the smallest cardinality of the row set on the left-hand side of each `JOIN operation`[1]. The evaluators support several `JOIN` implementations, such as `HASH JOIN` and `HASH LOOP JOIN` operations. It is the Physical Optimiser's job to analyse the logical query plan and make decisions on what type of `JOIN` operations to use in different query partitions.

After the logical and physical optimisation, the *Parallel Optimiser* (also known as the *scheduler*) generates the query plan for multiple partitions. The query plan partitions are then sent to the appropriate DQP evaluator for execution. According to the query plan partition, the DQP evaluator then contacts the data source via the OGSA-DAI data services to obtain the required data. The intermediate data can flow from one evaluator to the other and the final result

[1]The cardinality information comes from the physical metadata in the database itself.

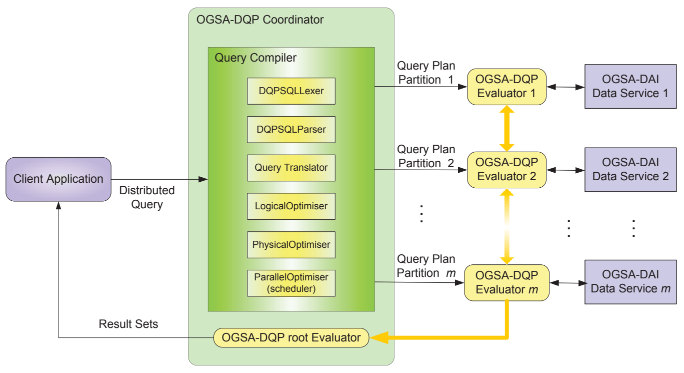

Figure 3. OGSA-DQP Workflow

sets are sent to the client via the root DQP evaluator.

Figure 3 summarizes the typical interactions among the OGSA-DQP components.

2.3 Data Type Support

Before extending the data type support we need to identify the various type systems used in the software. Information here is based on the source code of the OGSA-DQP 3.2 Tech Preview.

There are many different type systems involved. These include schema types, which get mapped to Java types that appear in the tuple types used by query compiler and query plan. These get mapped to types that appear in the tuple types used by the evaluator. These eventually get mapped to types appearing in the final WebRowSet. Figure 4 shows the data type flow in between different OGSA-DAI and OGSA-DQP interactions. Table 1 summarises the OGSA-DQP data type flow in eight steps.

The first two steps happen during the construction of a `DQPDataResource`. Type information originally comes from the schema metadata obtained from the database through OGS-DAI and ultimately JDBC (e.g. `bigint`). In the second step, this is converted to a "Java type"—a subclass of `java.lang.Class`—by the method `Types.mapSQLTypeToDQPType()` in the DQP coordinator. This is the form of type information used by the query compiler (e.g. `Long.class`).

The third step of type conversion happens in the final stages of query compilation when the `PartitionWriter` class uses method `Types.getEvaluatorType()` to convert a "Java type" to an "evaluator type"—one of the type name strings enumerated in the class `Evaluator`. This is the type as it appears in the XML query plan (e.g. `"long"`).

The fourth step happens when the XML query plan reaches the evaluator and is processed by the class `ObjectBuilder`. The `BuildOutputTupleType()`

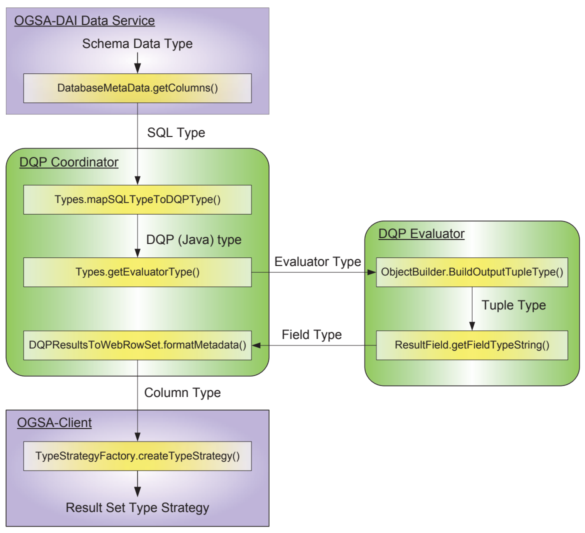
OGSA-DAI Data Service
Schema Data Type
DatabaseMetaData.getColumns()
SQL Type
DQP Coordinator
Types.mapSQLTypeToDQPType()
DQP (Java) type
Types.getEvaluatorType()
DQPResultsToWebRowSet.formatMetadata()
Evaluator Type
Field Type
DQP Evaluator
ObjectBuilder.BuildOutputTupleType()
Tuple Type
ResultField.getFieldTypeString()
Column Type
OGSA-Client
TypeStrategyFactory.createTypeStrategy()
Result Set Type Strategy

Figure 4. OGSA-DQP Data Type Flow

method in this class converts the type in the query plan to an integer type code enumerated in the class `TypeDefs` (e.g. `TypeDefs.INT_KIND` = 15).

The fifth step happens when the root evaluator has finished and is ready to return the final results. It does this by executing a `PRINT` operator represented internally by the class `PrintOp`. This uses the method `ResultField.getFieldTypeString()` to convert the type code to a string (e.g. `"int"`).

This string is stored in a document that is passed to the `putData()` of `QueryExecutionProcessor`. This document is in turn passed to the `convertBlock()` method of `DQPResultsToWebRowSet`. This class executes the sixth step by calling its `formatMetadata()` to write elements in the WebRowSet containing (an insignificant type name, e.g. `"INTEGER"`, and) an integer type code enumerated in the class `java.sql.Types` (e.g. Types.INTEGER = 4).

In the final step, when the Web row set reaches the client, `WebRowSetToResultSet` will pass an instance of a subclass of `AbstractResultSet` to the `parse()` method of `WebRowSetParser`. This will call the `putMetaData()` method of `AbstractResultSet`. Here, `TypeStrategyFactory` instantiates a `TypeStrategy` for reading this column (e.g. `IntegerStrategy`).

Table 2 illustrates the supported databases data type of the current standard OGSA-DQP release and the detailed data type mapping during DQP processing[2].

[2]The greyed dateType="data" and timestamp="datetime" Type definitions are not supported by the DQP evaluator in the current release.

Table 1. OGSA-DQP Flow of Type Information

Stage	Convertor			Domain	Context
	Component	Class	Method		
1	JDBC	DatabaseMetaData	getColumns	values of TYPE_NAME column of result	OGSA-DAI ExtractDatabaseSchemaActivity
2	OGSA-DQP coordinator	Types	mapSQLTypeToDQPType	java.lang.Class objects	initialise() method DQPDataResource
3	OGSA-DQP coordinator	Types	getEvaluatorType	fields of Evaluator class in coordinator	convertTupleType() method of PartionWriter
4	OGSA-DQP evaluator	ObjectBuilder	BuildOutputTupleType	fields of TypeDefs class in evaluator	run() method of QueryExecutionEngine
5	OGSA-DQP evaluator	ResultField	getFieldTypeString	one of {"int", "char", "boolean", "string", "float", "double", "var", "date", "datetime", "unknown"}	Next() method of PrintOp
6	OGSA-DQP coordinator	DQPResultsToWebRowSet	formatMetadata	fields of java.sql.Types	putData() method of QueryExecutionProcessor
7	OGSA-DAI	TypeStrategyFactory	createTypeStrategy	instance of TypeStrategy	getResultSet() method of WebRowSetToResultSet
8	OGSA-DAI	TypeStrategy subclass	get*Type*	Java types	get*Type*() method of AbstractResultType

3. A Large Database Schema in a Distributed Environment

The SDSS is a project that has built a very detailed digital map of the visible stars and galaxies in the night sky. The SDSS is the first wide-area survey to use electronic light detectors, producing images substantially more sensitive and accurate than earlier surveys, which relied on photographic plates. It is one of the most ambitious astronomical surveys ever attempted. A 2.5 meters SDSS telescope at Apache Point Observatory near Sunspot, New Mexico, in the United States is dedicated full-time to the SDSS, and collects images almost every night (depending on the weather). Once the survey is finished, the SDSS data will map a quarter of the whole sky in detail. The data is made available electronically to the scientific community and the general public, both as images and as digital catalogues of all the objects discovered. With the release of DR6 in June 2007, data for approximately 287 million objects in the sky has been made available so far [11, 15].

So far there have been six major public data releases from the SDSS project. In the four years since the first data release (DR1) up to DR6, data volume has grown from 2.34 terabytes of image data and 0.46 Terabytes of catalogue data to 10.00 and 4.00 terabytes respectively. Hosting a data repository of such a large size on a single site is becoming costly and there may be a performance bottleneck occurred sooner or later. We think the growth of large databases requires new scientific methods to organise the rapidly growing volume of data. In the long term, distributing large datasets such as the SDSS over multiple sites is seen as the only feasible way to house the data [19].

The SDSS database was only deployed in a Microsoft SQL Server 2000 instance. The SDSS DR5 database contains 85 user tables, 44 user views, 164 stored procedures and 173 user-defined functions, which enable advanced data searches and query optimisation. Indexing is used automatically in the SQL query optimiser to speed up the performance of queries.

We wanted to distribute the SDSS dataset to multiple sites, and investigate the possibility of running distributed queries using Grid technologies such as OGSA-DQP. Most of Grid sites run on Linux platform. Since Oracle database supports all known platforms including Linux, we migrated the SDSS DR5 database (table schema and data) from SQL Server to Oracle. To construct the distributed SDSS database system, we partitioned the SDSS database and distributed among different sites with either Oracle 10g or SQL Server 2000 installed.

We used OGSA-DAI middleware and OGSA-DQP toolkit to deploy our distributed SDSS database system. In Figure 5, databases are wrapped by OGSA-DAI data services (or other Web services that can be invoke for data analysing processing). OGSA-DQP involves several Grid services for distributed queries compiling, scheduling and executing.

The client application issues a distributed query to the DQP coordinator. The coordinator then compiles the query

Table 2. The supported Data Types flow details in the current release of OGSA-DQP

SQL Type (Stage 1)	DQP Type (Stage 2)	Evaluator Type (Stage 3)	Tuple Type (Stage 4)	Field Type (Stage 5)	Column Type (Stage 6)	Type Strategy (Stage 7)
varchar						
text	String.class	strType="string"	STR_KIND=16	"string"	VARCHAR =12	VarCharStrategy
char						
longvarchar						
real						
double	Double.class					
float		doubleType ="double"	DOUBLE_KIND =25	"double"	DOUBLE=8	DoubleStrategy
numeric	BigDecimal.class					
decimal						
integer						
bigint						
int	Integer.class	intType="int"	INT_KIND=15	"int"	INTEGER=4	IntegerStrategy
smallint						
tinyint						
bit	Boolean.class	boolType ="boolean"	BOOL_KIND=17	"boolean"	BOOLEAN =16	BooleanStrategy
boolean						
date	Date.class	dateType="date"	DATE_KIND =9902	"date"	DATE=91	DateStrategy
time	Timestamp.class	timestampType ="datetime"	DATETIME_KIND =9901	"datetime"	TIMESTAMP =93	TimestampStrategy
timestamp						

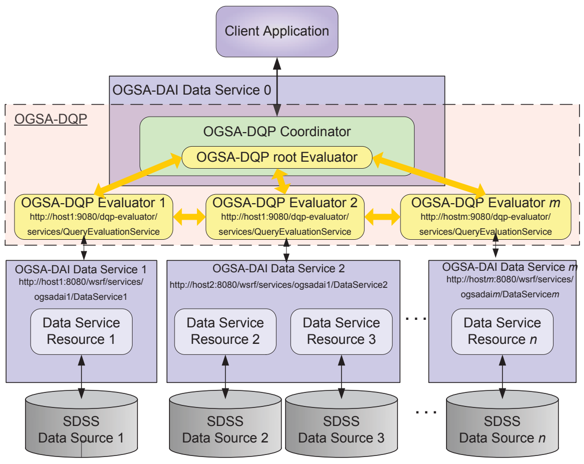

Figure 5. A Grid-Based Distributed Database System

and generates execution plans, which could involve a number of evaluators. The coordinator then invokes its acquired DQP evaluators to execute the partitioned query plan. During the query execution, the evaluators retrieve data from the underlying data sources via the connected data services. The data can flow between evaluators. The processed data is then transferred to the DQP root evaluator of the coordinator. Finally, the DQP coordinator passes the result datasets to the client application.

4. Using OGSA-DQP to Support Large Schemas with Oracle and SQL Server

Our distributed SDSS database system is currently built on Oracle 10*g* (Release 2) and Microsoft SQL Server 2000. This distributed system employs the OGSA-DAI and OGSA-DQP as a middleware to access and integrated the distributed SDSS database.

We deployed the OGSA-DQP toolkit (version 3.2 Tech Preview) and OGSA-DAI WSRF 2.2 on our database servers. After studying the source code, we found that the standard OGSA-DQP software has relatively poor support for databases other than MySQL; in particular the support for mapping data types lacks many features we require.. We think similar problems would probably arise using OGSA-DQP on any other databases with comparable schema complexity.

4.1 Generating the Coordinator Instance

A DQP coordinator factory is created when OGSA-DQP is deployed. The coordinator factory is used to create coordinator instances. The factory assigns the coordinator a resource ID with prefix "`dqpogsadai-`". A client can then make requests to those coordinator instances. This pattern of creating service instances follows WSRF [8] and related grid infrastructures.

To configure a coordinator instance, we need to identify one or more *evaluators* that can be used by the coordinator, and one or more OGSA-DAI data service resources that expose the underlying RDBMSs. The coordinator instance configuration file is in XML format.

When generating a coordinator instance, the database schemas of the RDBMSs wrapped by these OGSA-DAI data service resources must be imported. Loading of database schemas is a crucial part of creating the coordinator instance.

4.2 Type Mapping

Because OGSA-DQP is written in Java, and data types in RDBMSs do not correspond directly to data types in the Java programming language, there needs to be some mechanism for translating data between OGSA-DQP, which uses Java types, and RDBMSs that use SQL types. Section 2.3 explains the OGSA-DQP supported data types and their mapping details. Data types mapped by OGSA-DQP have been influenced by the original target MySQL test databases. Compared to those OGSA-DQP test cases, the SDSS database is far more complex, with much larger data volumes.

As exposed by the OGSA-DAI data service resource (which gets its information using JDBC), the SQL Server SDSS database schema is a $7,333$-line XML file. The Oracle SDSS database schema is a $74,503$-line XML file. The Oracle version of SDSS schema is a lot larger mainly because it includes many system tables—unfortunately neither OGSA-DAI or JDBC provide a simple way of excluding system tables from schema metadata. Compared with the standard OGSA-DQP, many additional SQL data types are needed to import these large SDSS database schema.

We learned from the database migration in [22] (about data type mapping between Oracle and Microsoft SQL Server) that there are considerable variations between the SQL types supported by different RDBMS products. Even when different RDBMSs support SQL types with the same semantics, they may give those types different names. For example, both Oracle and SQL Server support an SQL data type for storing large binary values, but SQL Server labels this `image` and Oracle labels it `BLOB`.

Because our distributed SDSS database is based on Oracle and SQL Server RDBMSs, we added data type mapping support for these RDBMSs to OGSA-DQP. This was done by modifying the `Types` class in the OGSA-DQP coordinator.

To support the Microsoft SQL Server version of the SDSS database, we added several new data type mappings (and associated evaluator types) including:

`bigint`	$\mapsto$	`Long.class`
`nvarchar`	$\mapsto$	`String.class`
`sysname`	$\mapsto$	`String.class`
`binary`	$\mapsto$	`byte[].class`
`datetime`	$\mapsto$	`byte[].class`
`image`	$\mapsto$	`byte[].class`
`varbinary`	$\mapsto$	`byte[].class`

To support the Oracle version of the SDSS schema we additionally added:

`varchar2`	$\mapsto$	`String.class`
`raw`	$\mapsto$	`byte[]`
`blob`	$\mapsto$	`Blob.class`
`clob`	$\mapsto$	`Clob.class`
`number`	$\mapsto$	`BigDecimal.class`
`rowid`	$\mapsto$	`oracle.sql.ROWID.class`
`long`	$\mapsto$	`String.class`

The Oracle SDSS schema includes many system tables as well as the user tables, and these include many different customised types. Examples of include `anydata` and `EXF$INDEXOPER`, but there are very many more of these and it is impractical to enumerate them all. In the end we just mapped all unrecognized types to `String.class`.

The data type mappings above are mostly chosen by referring to the *Sun Microsystems* online document *Mapping SQL and Java Types* [14] and the book *JDBC API Tutorial and Reference: Universal Data Access for the Java 2 Platform* [17]. However, mapping the Oracle data type `number` to `BigDecimal.class` proved to be problematic later on—please refer to Section 4.3 for more details on this.

4.3 Running A Query

When we ran the following query using the OGSA-DQP client, the results were correct except that in the test query submitted to a SDSS Oracle database through OGSA-DQP, the `OBJID` column of the result was truncated. The result for `OBJID` is `5.8772601400131597E17`. The correct value is $587,726,014,001,315,907$.

```
SELECT MATCH, ID, OBJID
FROM FIRST
WHERE OBJID=587726014001315907
```

The Oracle data type of `OBJID` column is `number`, which we had originally mapped to `BigDecimal.class`, following the standard JDBC documentation [14, 17]. Referencing our Table 2, we find `BigDecimal.class` is mapped by OGSA-DQP to an evaluator type of `Double`, which corresponds to an IEEE 754 double precision floating point number. According to the IEEE Standard 754 for Binary Floating-Point Arithmetic [6], double precision for representing floating point values has a mantissa of 54-bits:

$$2^{54} = 18,014,398,509,481,984$$

However, the OBJID values of the SDSS DR5 database range between $587,722,951,693,303,809$ and $588,848,901,777,785,938$—this explains why the value got truncated.

To fix this problem, we changed the mapping of the Oracle data type number to Long.class by modifying the method Types.mapSQLTypeToDQPType() in the OGSA-DQP coordinator. This is acceptable because in our translation of the original SQL Server SDSS schema to Oracle we only converted integer types (bigint, bit, int, smallint, tinyint) to Oracle number type; floating point types (float, real) were always converted to float. (Please refer to [22] for more details.)

When we reran the OGDSA-DQP query, we now got an SQL exception:

```
java.sql.SQLException: Cannot convert
            587726014001315907 to int
```

We soon found that when the result set contains a column of type long (or equivalently BIGINT), the metadata in the WebRowSet that comes back to the client has the column marked as INTEGER. In the client OGSA-DAI class AbstractResultSet the column OBJID is then treated as integer instead of long.

From inspection of the source code of OGSA-DQP we learned that the types embedded in the WebRowSet were controlled by inputs to a PRINT operator in the evaluator.

The easiest way to fix the problem was by modifying the method formatMetadata() in the DQP coordinator class DQPResultsToWebRowSet so that it later maps *all* items with field type int to Types.BIGINT rather than Types.INTEGER (see Figure 4, Tables 1 and 2). After the corrections, the DQP query submitted to Oracle returns the correct value for OBJID.

5. Discussion

Section 2.3 of this paper summarised the OGSA-DQP data type flow in eight stages. We spent a lot of time working out the data type mapping between different stages. OGSA-DQP is dealing with too many different type systems, and this caused various difficulties when customizing types to support our requirements. We believe there must be scope for simplification of the steps involved in the data type flow of OGSA-DQP.

On the other hand, the current release of OGSA-DQP claims to support Microsoft SQL Server but in fact it does not include many of the common SQL Server data types. We suggest support for more standard data types should be added to OGSA-DQP.

The architecture and software of the OGSA-DQP system seems to have developed out of the earlier systems Polar and Polar*. Polar [13] was developed as a parallel (not necessarily distributed) *object* database. The later Polar* system [12] took Polar as a foundation but moved to the Globus platform, to create a distributed object database for the Grid (using MPICH-G for the exchange operator). These earlier systems were implemented in C++, and the authors of [12] say:

> Polar* adopts the model and query language of the ODMG object database standard [5]. As such, all resource wrappers must return data using structures that are consistent with the ODMG model. Queries are written using the OGDMG standard query language, OQL.

In turn, OGSA-DQP seems to have been developed from Polar* by moving from a C++ evaluator—with direct use of Globus and MPI [16]—to a Java evaluator, using OGSA-DAI and SOAP. Up until release 3.1, OGSA-DQP was still using the old Polar* distributed query compiler (written in C++), and it was still using OQL as the input query language (rather than SQL). Only in the most recent release of OGSA-DQP was the query translator converted to Java, and the input language changed to SQL.

So the components of the OGSA-DQP system seem to have had a complicated development, changing through different languages. We think this may help explain the multiple type systems in the current software. In other words, we think they may be a legacy from older versions of the system.

If a similar system was designed assuming a pure Java and SQL implementation from the start, we believe it might only need about two type systems: JDBC's set of SQL types, and some form of "Java type" for internal use in the query compiler and evaluator.

6. Conclusions

This paper focussed on data type support in a well-known (experimental) Grid-based middleware for querying distributed databases—OGSA-DQP. We applied this middleware to a large-scale scientific database with a complex schema (SDSS). We described the various data type systems involved in the OGSA-DQP data type mapping process.

When we came to apply OGSA-DQP to our large scientific database, we found that many required data types were not supported. We enlarged the data type support in OGSA-DQP to parse the SDSS database schema exposed by the OGSA-DAI data service resource based on Microsoft SQL Server and Oracle database instances. Data types mapped by the standard OGSA-DQP have apparently been influenced by the original simple MySQL test databases. The SDSS database is a lot larger and with more complex data types. In order to parse the SDSS database schema, we modified the OGSA-DQP Types class and added improved data type support.

We went on to improve the data type mapping to avoid certain data being truncated during queries.

An additional problem not discussed in detail above occurred in `UNION ALL` queries integrating results of queries from different RDBMSs implementation. We needed to alter the data type mapping in the `Types` class to make sure data of the same schema object from Microsoft SQL Server and Oracle is mapped to the same data types in OGSA-DQP. After modifying the OGSA-DQP coordinator source code, we were able to successfully ran union queries across multiple machines with integrated results.

With these problems relating to data types resolved, we went on to perform a series of queries of increasing complexity across our distributed database. See [22] for details of other modifications made to OGSA-DQP to support large scale queries.

Based on our experiences with this realistic use case, Section 5 contained some general observations about the the type systems utilized in OGSA-DQP. We believe that there is scope for considerable simplification, and such a simplification could make future versions of this middleware more useful in practise. Meanwhile, we have adapted the existing OGSA-DQP software to provide support for additional data types, particularly those common in Oracle and Microsoft SQL Server databases.

References

[1] M. Alpdemir, A. Gounaris, A. Mukherjee, D. Fitzgerald, N. Paton, P. Watson, R. Sakellariou, A. Fernandes, and J. Smith. Experience on performance evaluation with OGSA-DQP. In *Fourth UK e-Science All Hands Meeting*, 2005.

[2] M. N. Alpdemir, A. Mukherjee, A. Gounaris, A. A. A. Fernandes, N. W. Paton, and P. Watson. An experience report on designing and building OGSA-DQP: A service based distributed query processor for the Grid.

[3] M. N. Alpdemir, A. Mukherjee, A. Gounaris, N. W. Paton, A. A. A. Fernandes, R. Sakellariou, P. Watson, and P. Li. Using OGSA-DQP to support scientific applications for the Grid. In P. Herrero, M. S. Prez, and V. Robles, editors, *SAG*, volume 3458 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2004.

[4] M. N. Alpdemir, A. Mukherjee, A. Gounaris, N. W. Paton, P. Watson, A. A. A. Fernandes, and D. J. Fitzgerald. OGSA-DQP: A service for distributed querying on the Grid. In E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Bhm, and E. Ferrari, editors, *EDBT*, volume 2992 of *Lecture Notes in Computer Science*, pages 858–861. Springer, 2004.

[5] R. Cattell et al. *The Object Database Standard: ODMG 3.0*.

[6] IEEE 754: Standard for binary floating-point arithmetic. http://grouper.ieee.org/groups/754/.

[7] A. Mukherjee and P. Watson. Adding dynamism to OGSA-DQP: Incorporating the DynaSOAr framework in distributed query processing. Technical Report 979, Newcastle University, School of Computing Science, Aug 2006.

[8] OASIS. Web Services Resource Framework (WSRF). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[9] The OGSA-DAI project home page. http://www.ogsdai.org.

[10] The OGSA-DQP project. http://www.ogsadai.org/about/ogsa-dqp.

[11] The SDSS Data Release 6 (DR6). http://www.sdss.org/dr6/start/aboutdr6.html.

[12] J. Smith, A. Gounaris, P. Watson, N. Paton, A. Fernandes, and R. Sakellariou. Distributed query processing on the grid. In *Third Workshop on Grid Computing (GRID2002)*, 2002.

[13] J. Smith, P. Watson, S. de F. Mendes Sampaio, and N. W. Paton. Polar: An architecture for a parallel ODMG compliant object database. In *CIKM*, pages 352–359, 2000.

[14] Sun Microsystems, Inc. Mapping SQL and Java types. http://java.sun.com/j2se/1.5.0/docs/guide/jdbc/getstart/mapping.html.

[15] A. S. Szalay, J. Gray, A. Thakar, P. Z. Kunszt, T. Malik, J. Raddick, C. Stoughton, and J. vandenBerg. The SDSS SkyServer—public access to the Sloan Digital Sky Server data. In *SIGMOD Conference*, pages 570–581, 2002.

[16] D. W. Walker and J. J. Dongarra. MPI: a standard Message Passing Interface. *Supercomputer*, 12(1):56–68, 1996.

[17] S. White, M. Fisher, R. Cattell, G. Hamilton, and M. Hapner. *JDBC API Tutorial and Reference: Universal Data Access for the Java 2 Platform (2nd Edition)*. Pearson Education, June 1999.

[18] G. Wiederhold. Mediators in the architecture of future information systems. pages 185–196, 1998.

[19] H. Xiang, M. Baker, and R. Nichol. Experiences mirroring and distributing the Sloan Digital Sky Survey. In *Fifth International Conference on Grid and Cooperative Computing Workshops (GCC 2006), Changsha, China*, pages 518–521. IEEE Computer Society, October 2006. http://doi.ieeecomputersociety.org/10.1109/GCCW.2006.38.

[20] H. X. Xiang. Experiences acquiring and distributing a large scientific database. submitted to 2008 International Conference on Database Theory and Application, Hainan Island, China, 2008.

[21] H. X. Xiang. Experiences acquiring and distributing a large scientific database. submitted to 2008 International Conference on Computer Science and Software Engineering, Wuhan, China, 2008.

[22] H. X. Xiang. *A Grid-based Distributed Database Solution for Large Astronomy Datasets*. PhD thesis, Portsmouth, UK, February 2008.

[23] D. G. York et al. The Sloan Digital Sky Survey: Technical summary. *Astronomical Journal*, 120:1579–1587, 2000. http://www.sdss.org.