2009 15th International Conference on Parallel and Distributed Systems

Experiences Running OGSA-DQP Queries Against a Heterogeneous Distributed Scientific Database

Helen X Xiang
Computer Science, University of Hertfordshire, UK
h.xiang@herts.ac.uk

Abstract

This paper explores the use of emerging Grid technologies for the manipulation of a scientific dataset with complex schema. We focus on the potential of a well-known Grid middleware—OGSA-DQP—for querying such datasets. In particular, we investigate running OGSA-DQP queries against SQL Server and Oracle database to access data in a complex schema of a real scientific database—the Sloan Digital Sky Survey (SDSS) database. We used two common databases which support complex schemas and very large datasets: SQL Server and Oracle. The paper briefly provides the background information on the SDSS database. It then examines the running of a few OGSA-DQP queries and the query plans. Various join queries run successfully through OGSA-DQP after we optimized the implementation of some of the OGSA-DQP operations.

1. Introduction

Started in 2000, the *Sloan Digital Sky Survey* (SDSS) project has gathered and produced more than 60 terabytes of data (inducing images and catalogues data). It has built a detailed digital map of the visible stars and galaxies of the night sky [7]. The SDSS data produced by is summarised in a multi-terabyte relational database containing photometric objects and spectroscopic information— this is available online via the SkyServer (http://skyserver.sdss.org).

Our recent papers [2, 3, 4, 5] described how we created an experimental distributed SDSS DR5 database with Grid middleware which was based on Open Grid Services Architecture—Distributed Query Processing (OGSA-DQP) [1]. Please refer to [6] for details of the OGSA-DQP operators.

This experiment exposes some limitations of the technologies. In particular, this paper will focus on running OGSA-DQP queries against the a heterogeneous distributed SDSS database. We describe the modifications and improvements we made for running the OGSA-DQP queries against a complex scientific database schema. This is a follow-up paper of [6].

2 A Distributed SDSS Database

For the propose of understanding the OGSA-DQP queries in this paper, this section describes how we distributed a reduced SDSS database called `MyBestDR5` among three different hosts machines located in different university buildings. Oracle is running on two of the hosts, while Microsoft SQL Server is running on the other.

First, the original SDSS `MyBestDR5` database's `PhotoObjAll` schema was split into three parts using `objID`. We then deployed the migrated SDSS Oracle schema on the two Oracle databases (on hosts 1 and 2) and extracted the relevant data files from the orignal `MyBestDR5` database in SQL Server, with `objID` partitioning conditions. We next injected the appropriate data files to databases on hosts 1 and 2 used the SQL*Loader,. The third partition (on host 3) was formed by a cut down version of `MyBestDR5` database after the appropriate records were removed. We now have a heterogeneous distributed SDSS `MyBestDR5` database over hosts 1, 2, and 3.

We then exposed the distributed SDSS `MyBestDR5` database using the OGSA-DAI middleware and configure it with the OGSA-DQP toolkit before running the distributed queries.

We installed an OGSA-DAI instance on hosts 1, 2 and 3 and deployed OGSA-DAI data service them with the following full URLs:

```
http://host1:8080/wsrf/services/
       ogsadai1/DataServiceBUCK
http://host2:8080/wsrf/services/
       ogsadai2/DataServiceIcg
http://host3:8080/wsrf/services/ogsadai3/
       DataServiceAce
```

The SDSS `MyBestDR5` database is now distributed among three hosts—with heterogeneous DBMSs: two Or-

1521-9097/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPADS.2009.142

acle and one SQL Server—with the distributed SDSS data resources exposed via the OGSA-DAI middleware.

Finally, we installed the OGSA-DQP 3.2 (Tech Preview) toolkit on the distributed sites for querying the distributed SDSS `MyBestDR5` database. We also deployed an OGSA-DQP evaluator service On each host.

The WSDL of the deployed OGSA-DQP evaluators can be viewed via the following URLs:

```
http://host1:8081/dqp-evaluator/
       services/QueryEvaluationService
http://host2:8081/dqp-evaluator/
       services/QueryEvaluationService
http://host3:9080/dqp-evaluator/services/
       QueryEvaluationService
```

An OGSA-DQP coordinator can be installed on a separate host or on one of the three distributed SDSS hosts on the OGSA-DAI Data Services deployed earlier on. The client application is now ready to interact with the OGSA-DQP coordinator service to create an OGSA-DQP coordinator instance for executing queries.

The architecture of the distributed SDSS `MyBestDR5` database system is illustrated in Figure 1. We are now ready to run some OGSA-DQP queries against this distributed database.

3 Running OGSA-DQP Queries

We already examined the OGSA-DQP query plan for a simple local query, and the different SOAP interactions between DQP components in [6]. This section discusses running more complicated queries through OGSA-DQP.

We could now run DQP queries against a table on local databases and return one or more rows. In [6] we successfully ran the query:

```
SELECT SKYVERSION
FROM PHOTOOBJALL
WHERE OBJID=587726014001315907
```
(1)

through OGSA-DQP, against an SDSS Oracle database or SDSS SQL Server database.

Next we wanted to try OGSA-DQP queries that can return many rows. We tried to run the following query on the Oracle database on the first host:

```
SELECT SKYVERSION
FROM PHOTOOBJALL
```
(2)

If successful, this query should return $68,118$ rows containing column value "1". Initially the query failed to complete. On investigation this proved to be caused by a timeout in the OGSA-DAI codes, which was resolved by modifying OGSA-DQP client class.

We discovered we had to increase the memory on both DQP coordinator and OGSA-DQP evaluator to 256MB and 750MB respectively for running query 2. But the query still failed with a `uk.org.ogsadai.client.toolkit.activity.delivery.StreamDataException` exception. The client returned this message after 11 minutes and 23 seconds. But on the server, the query went on running for 3-4 hours until we killed the Tomcats.

We added some debugging lines in the OGSA-DAI source code and reran query 2. We proceeeded to investigate the suspected timeout problem that was preventing larger queries running. After looking into the code for OGSA-DAI and Axis we eventually discovered that this timeout could be disabled by modifying the OGSA-DQP `Client` class. The solution is to call the method:

```
setTimeout(0)
```

on the `DataService` object.

This allowed us for the first time to run queries against the distributed database returning some thousands of rows. The query 2 through OGSA-DQP now worked against the Oracle database on the first host, and returned the correct result—$68,118$ rows of "1"s. However, it took a very long time: 452 minutes and 52 seconds in total. We needed to optimise this if we were to run much larger queries against the full SDSS database.

Figure 2 shows the query plan for query 2. The query plan is very similar to the one in [6]. The only real difference is in a predicate in the `TABLE_SCAN` node (the `WHERE` clause). This predicate is not shown in the figure. The "SQL" blob in Figure 2 is not really a feature of the query plan. It just helps explain how the plan is executed, as we will now discuss.

We had already noticed from the SOAP messages captured in [6] that the `TABLE_SCAN` operation was loading *all columns* of the selected rows of the target table. This might not be a problem if the target tables are small with few columns. But it is a big problem for larger databases with wide tables, and can easily be a performance bottleneck. Most of the tables in SDSS schema are large with many rows and many columns. In particular the table `PHOTOOBJALL` is a very wide table with 446 columns. The standard OGSA-DQP `TABLE_SCAN` operation loads all of the 446 columns of the `PHOTOOBJALL` table even if only one is column required. The single column actually required in our query is projected out in the `APPLY` operation. Therefore the sample query returns $68,118$ rows of "1" (column `SKYVERSION`). But the standard OGSA-DQP first loads $68,118 \times 446 = 30,380,628$ column values. Hence the poor query performance on large queries.

To make meaningfully large scan-type queries practical, OGSA-DQP would need to be optimized to avoid read-

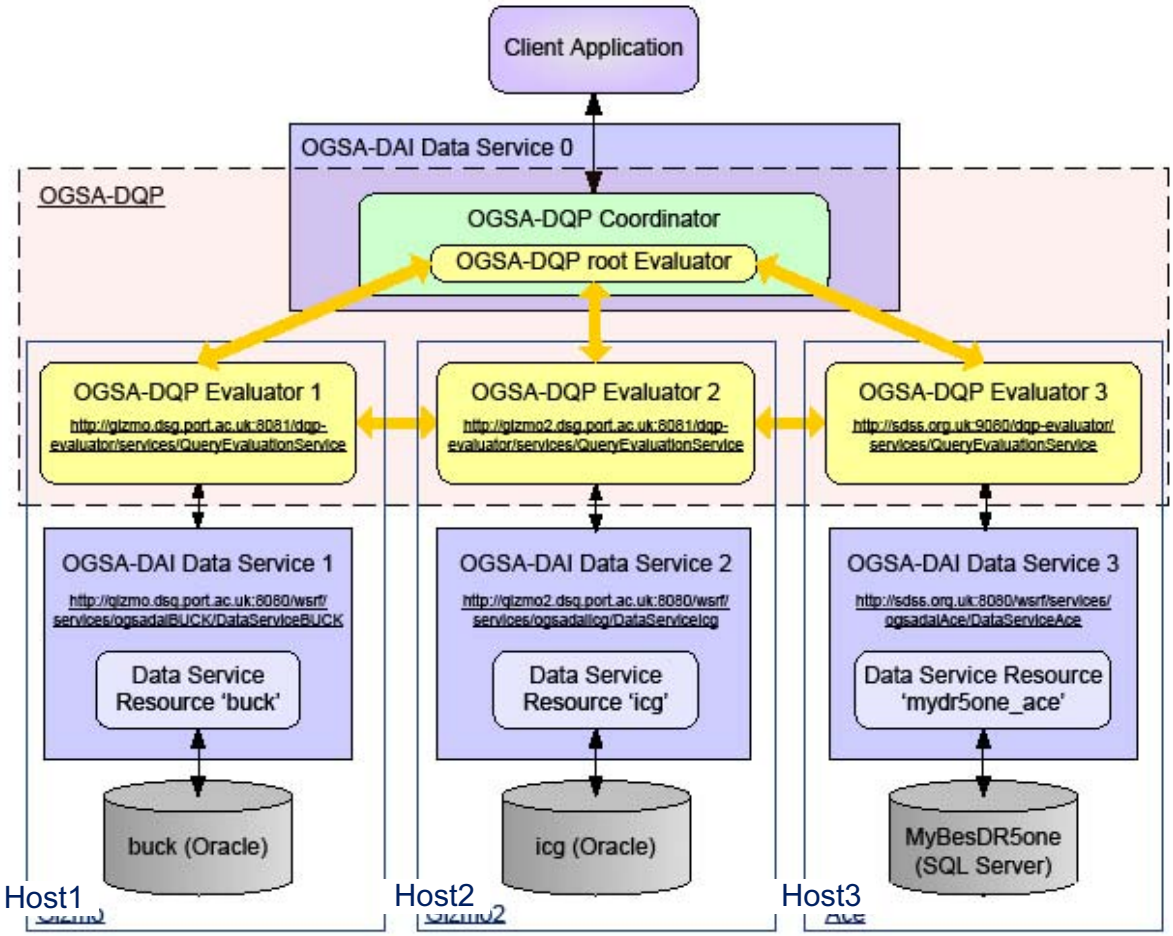

Figure 1. Distributing the SDSS `MyBestDR5` database among three hosts

ing *all* columns of tables. The ideal approach would be to change the query compiler to combine the `APPLY` and `TABLE_SCAN` into a new kind of node that issued an SQL query to the data resource for just the columns needed. But this would involve changes to several modules of OGSA-DQP. Instead we devised an optimization of project/scan-type queries that left the query plan unchanged, but worked by changing the way the evaluator's `TableScanOp` class and `ReduceOp` class process the query plan.

With the implementation of the above optimization, the query performance is dramatically improved for the simple (medium sized) queries. The sample query 2 took only 2 minutes 37 seconds instead of 452 minutes 52 seconds—173 times faster[1]. As explained above, we keep the same query plan, but the evaluator interpets it differently. The optimized query plan in Figure 3 is actually identical to the one in Figure 2, but the changed "SQL" blob now shows that the query sent to the data resource incorporates the projection in the `APPLY` operation as well as the `TABLE_SCAN` operation.

We then tried it on another query for larger number of results with more complicated data type:

```
SELECT OBJID
FROM NEIGHBORS
```
(3)

If everything goes well when this query is run against the first partition of the database, it should return 355,214 rows of `OBJID`.

Unfortunately, there is a problem with client memory exhaustion when returning large number of rows. After examining the log files, we found the OGSA-DQP client program is buffering all results into one huge string before printing them. This might be OK if the results set are small—for example lots of "1" as the previous query returning `SKYVERSION` column. However, column `OBJID` has a bigger data type than column `SKYVERSION`, and trying to buffer 355,214 of data like "587726014001315907" no doubt leads to the `java.lang.OutOfMemoryError` on the DQP client.

[1]Time for running a query similar to 2 against SDSS `MyBestDR5` without using OGSA-DQP was 7 seconds.

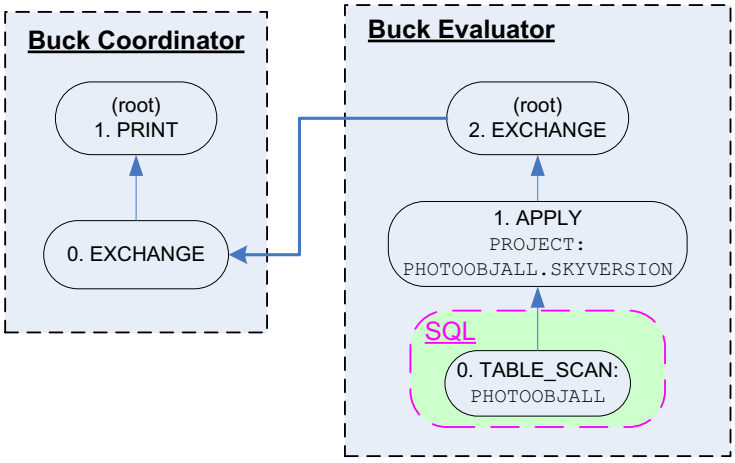

Figure 2. Query plan for query 2 running on the first host

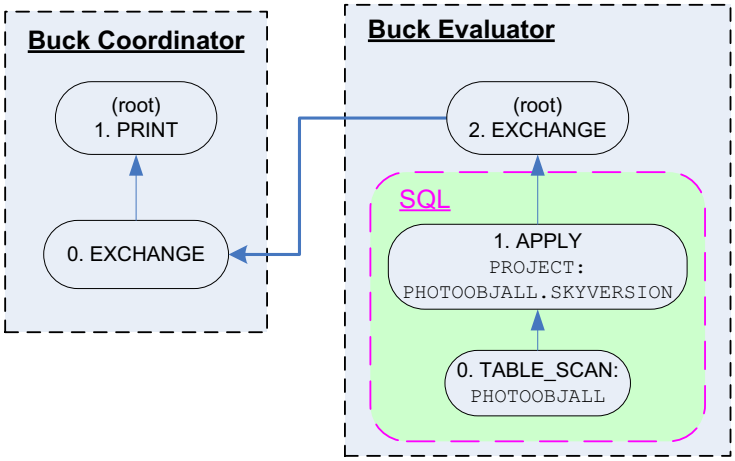

Figure 3. Optimized query plan for query 2 running against the databse on the first host

The client memory exhaustion was fixed by changing the DQP `Client` class to avoid using the method `getResults()`, so that the DQP client no longer buffers all results before printing.

After this modification, we successfully ran a number of OGSA-DQP queries against the Oracle databases, including local joins that involve two or three tables. One example is a sample query that joins two tables and returns $355,214$ rows from the database on the first host (total run time: 21 minutes 46 seconds):

```
SELECT PHOTOOBJALL.OBJID, PHOTOOBJALL.RA,
PHOTOOBJALL.DEC, NEIGHBORS.NEIGHBOROBJID,
NEIGHBORS.DISTANCE
FROM PHOTOOBJALL, NEIGHBORS
WHERE PHOTOOBJALL.OBJID = NEIGHBORS.OBJID
```
(4)

Another example is a query that joins three tables and returns 327 rows from the database on the first host (total run time: 3 minutes 55 seconds):

```
SELECT PHOTOOBJALL.OBJID
FROM FIRST, PHOTOOBJALL, NEIGHBORS
WHERE PHOTOOBJALL.OBJID = NEIGHBORS.OBJID
  AND PHOTOOBJALL.OBJID = FIRST.OBJID
```
(5)

Times for running similar queries against SDSS `MyBestDR5` without using OGSA-DQP were only 54 seconds and 2 seconds respectively.

4. Conclusions

This paper focussed on query execution in a well-known (experimental) Grid-based middleware—OGSA-DQP. It examined the OGASA-DQP operator and the running of some queries through OGSA-DQP. We looked at the OGSA-DQP query plan for some queries and made some adjustment when running the quires.

This paper discussed some problems running OGSA-DQP queries on an Oracle database, in particular queries returning many rows. The memory of both OGSA-DQP evaluator and coordinator was increased. We modified the OGSA-DQP `Client` class and disabled a client time out.

We also optimized the implementation of the OGSA-DQP `TABLE_SCAN` and `APPLY` operations to load only required columns into memory, instead of all the columns in the target tables. This optimisation greatly improved the OGSA-DQP queries against large tables. Finally we fixed a OGSA-DQP Client memory exhaustion problem by avoiding buffering all results before printing.

After the above improvements, various join queries ran successfully through OGSA-DQP. In future publications, we will describe how we have run more complex queries on a distributed version of full SDSS databases to further investigate the OGSA-DQP distributed query performance.

References

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

[2] 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.

[3] H. X. Xiang. Experiences acquiring and distributing a large scientific database. In *Second International Conference on Future Generation Communication and Networking Symposia*, volume 2, pages 14–19, Washington, DC, USA, December 2008. IEEE Computer Society.

[4] H. X. Xiang. A grid-based distributed database solution for large astronomy datasets. In *International Conference on Computer Science and Software Engineering*, volume 3, pages 66–69, Washington, DC, USA, December 2008. IEEE Computer Society.

[5] H. X. Xiang. Supporting complex scientific database schemas in a grid middleware. In *International Conference on Advanced Information Networking and Applications*, volume 0, pages 937–944, Bradford, UK, May 2009. IEEE Computer Society.

[6] H. X. Xiang. Using grid middleware to query a heterogeneous distributed version of the sdss database. In *The Fifteenth International Conference on Parallel and Distributed Systems*, Shenzhen, China, December 2009. IEEE Computer Society.

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