Query Optimization over a Heterogeneously Distributed Scientific Database

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

Abstract—This paper briefly discusses about the concepts about distributed databases management system and distributed query processing. It then talks about the need of distributing large Scientific datasets such as the Sloan Digital Sky Survey. The main focus of this paper is to investigate fully distributed queries involving cross-joins across a heterogeneously distributed scientific database.

I. INTRODUCTION

The concepts of Distributed Database Management System (DDBMS) and Distributed Query Processing (DQP) are very much related but they have distinct meanings. A DDBMS uses two or more DBMSs to distribute data in a distributed system that appears to an application or user as a single database. A DQP system is used to distribute application tasks among different databases and different computers in a network. DDBMS technology is mainly about distributing data. DQP technology is mainly about distributing computing processes. As well as being used to modify the global database, the implementation of DQP is used in distributed database applications to access both local and remote data in a DDBMS.

A distributed database is normally composed of a set of data with different parts stored in separated DBMSs that operate on different hardware systems with independent computer systems. Each participating computer system has its own right in controlling its own local applications, while taking part in the global transactions and applications of the DDBMS as a whole. A homogenous DDBMS uses the same DBMS at all the distributed sites, whereas a heterogeneous DDBMS employs two or more DBMSs. The global transactions in a DDBMS often involve data from multiple distributed sites. However, such characteristics of a DDBMS should be hidden from the end users. In other words, the users should not be aware of the distribution and heterogeneity of the data. Please see [11] for an in-depth review of distributed databases systems technologies.

The Sloan Digital Sky Survey (SDSS) is the first wide-are survey to use electronic light detectors, collecting accurate images and building a very detailed digital map of the visible stars and galaxies in the night sky [17], [18]. Currently, the SDSS-III collaboration is making the map of the Milky Way and searching for extrasolar planets as well as trying to solve the mystery of dark energy [3]. At the point of writing this paper, there have been nine major data releases from the SDSS project. The size of the SDSS database has increased considerably in size with each new release. With the latest release (DR9), the data produced by the survey is summarised in an sixty-terabyte relational database containing photometric objects and spectroscopic information with 14,555 square degrees of the sky coverage. The SDSS data is available to the scientists and the general public via the SkyServer (http://skyserver.sdss.org), the Science Archive Server (SAS) DR9 (http://dr9.sdss3.org) or various mirror sites. The SDSS telescope is collecting data from the sky every night. Terabytes of raw data has been gathered every year since the SDSS project started in April 2000. Even though storage is getting cheaper, hosting such a data repository on a single site will not only become expensive, but may eventually be a performance bottleneck too. The growth of datasets such as the SDSS requires new scientific methods to organise the rapidly growing volume of data. In the long term, we suppose distributing the SDSS database over multiple sites may be the most feasible way to house the data.

In our past papers we described how we created a heterogeneous distributed version of the SDSS database [9], [10], [12], [14] and experimented with data integration using union queries [16] via the Grid middleware. This is based on OGSA-DQP Distributed Query Processing [19] and the OGSA-DAI middleware[1]. We used OGSA-DAI and OGSA-DQP to integrate the data across different sites—forming a logical distributed database system. Global distributed queries can be processed over this logical database system [11].

Our last publication [15] began to discuss distributed query processes involve joins within a single database. In this paper, we are going to look at fully distributed join queries, involving cross-joins.

Please refer to [10], [12] for the details of how we distributed the SDSS MyBestDR5 database system among hosts within the University of Portsmouth and between the universities of Manchester and Portsmouth.

See [14] for the details of the OGSA-DQP system and its architecture. This paper follows on from our earlier publications [8], [9], [10], [12], [13], [14], [15], [16].

II. DISTRIBUTED JOIN QUERIES

This paper discusses running more complicated distributed queries through OGSA-DQP. The distributed join queries we will execute through OGSA-DQP typically involve local joins and cross joins. Local joins join tables in the same database while cross joins join tables from different databases. For
example, suppose we have an original query like this:

\[
\text{SELECT } <\text{select-list}> \text{ FROM } tableA, tableB \text{ WHERE } <\text{predicate}>
\]

To run a query equivalent to 1 on a database distributed on two hosts the query needs to be broken down into local joins and cross joins. In OGSA-DQP the local joins would be like this:

\[
\text{SELECT } <\text{select-list}> \text{ FROM } host1\_tableA, host1\_tableB \text{ WHERE } <\text{predicate}>
\]

\[
\text{SELECT } <\text{select-list}> \text{ FROM } host2\_tableA, host2\_tableB \text{ WHERE } <\text{predicate}>
\]

and the cross joins would be like this:

\[
\text{SELECT } <\text{select-list}> \text{ FROM } host1\_tableA, host2\_tableB \text{ WHERE } <\text{predicate}>
\]

\[
\text{SELECT } <\text{select-list}> \text{ FROM } host2\_tableA, host1\_tableB \text{ WHERE } <\text{predicate}>
\]

Here \(host1\) and \(host2\) are the identifiers for the two data resources. The above four queries would then be united with the UNION operations of OGSA-DQP.

Most query examples in the previous publications use OBJID in the WHERE clauses. These queries are relative easy to execute because we used OBJID to partition the SDSS database. The OBJID is a unique SDSS identifier composed from the several columns including skyVersion, rerun, run, camcol, field, and obj (see Table I).

Stepping back to real life, the majority of the SDSS users are not interested in searching the database by OBJID. They are interested in measured properties of objects. Therefore, we are going to test a couple of astronomical queries from the SkyServer Sample Queries [6].

Two different types of distributed queries will be considered here:

1) Data exists in all distributed sites, for example query 4.
2) Some of the data only exists on one of the distributed sites but not the others, for example query 5.

The Galaxies blended with stars query from SkyServer looks like this:

\[
\text{SELECT Galaxy.ObjID, Galaxy.u, Galaxy.g, Galaxy.r, Galaxy.i, Galaxy.z} \\
\text{FROM Galaxy, Star} \\
\text{WHERE Galaxy.parentID > 0} \\
\text{AND Galaxy.parentID = Star.parentID} \\
\text{-- object was deblended...} \\
\text{AND Galaxy.parentID = Star.parentID} \\
\text{-- and the star has the same parent.}
\]

Query 4 returns 26,978 rows on the MyBestDR5 database, taking about 4 seconds.

The Stars in specific fields query from SkyServer looks like this:

\[
\text{SELECT Star.psfMag_g, Star.run, Star.camCol, Star.rerun, Star.field} \\
\text{WHERE Star.fieldID = Field.fieldID} \\
\text{AND Star.psfMag_g < 20} \\
\text{AND Field.pspStatus = 2}
\]

Query 5 returns 91 rows on the MyBestDR5 database, taking about 1 second. Schema object Star has data at all our distributed sites, but Schema object Field only has data in the partition on Ace—not in any of our SDSS Oracle partitions at Portsmouth or Manchester.

Database MyBestDR5 is distributed among three databases at three different hosts: buck on Gizmo, icg on Gizmo2 and MyBestDR5one on Ace. To run query 4 and query 5 on the distributed MyBestDR5 database through OGSA-DQP, we need to break each query into three local joins and six cross joins, and integrate those join results using the UNION ALL operations. This leads to the very lengthy queries that are displayed in Figures 1 and 2.

We tested the OGSA-DQP query 4 on Ace against the distributed MyBestDR5 database. The individual queries of local joins and cross joins through OGSA-DQP went well, and the times taken were between 25 seconds and 1 minute 59 seconds. But the union query of Figure 1 appeared to just hang, and never finished.

We examined the DQP evaluator’s log file on Ace. It contained the following exception message:

java.net.SocketTimeoutException: Read timed out

With more careful investigation we figured out the reason for the failure: the OGSA-DQP EXCHANGE operation inside the evaluator was timing out after about ten minutes. We saw a similar problem previously with the standard implementation of OGSA-DQP client. In that case, as described in an earlier publication [14], the client was waiting for the query results from the coordinator and it also timed out after ten minutes. Here, the process involves three different DQP evaluators at three hosts: Ace, Gizmo and Gizmo2. One evaluator was waiting on another evaluator for the UNION ALL operations—this waiting timed out after ten minutes.

The EXCHANGE operation timeout problem was solved by modifying the OGSA-DQP evaluator’s TransportHandler class, adding the following code:

```java
{StubtransportPort}.setTimeout(0);
```

After this modification, the union query of Figure 1 runs through OGSA-DQP successfully and returns the correct result sets. A final run only took three minutes in total\(^1\) (the time for executing the original SDSS query 4 was about 4 seconds.)

We should note that both schema objects in the FROM clause of query 4 have data entries on all distributed SDSS

\(^1\)We do not have a good explanation for the final run time being less than the previously observed timeout period.
Fig. 1. OGSA-DQP version of query 4 for distributed MyBestDR5 database.

Fig. 2. OGSA-DQP version of query 5 for distributed MyBestDR5 database.
sites. Query 4 uses column parentID in the WHERE clause. We know from the Sky Survey schema browser [5] that parentID is either the pointer to an object’s parent object if it is deblended, or its bright object detection, or nothing, in which case it is set to 0. In other words, an object’s parentID is another object’s objID or 0. Due to the way the partitioning is done on our distributed SDSS database ([11] describes this in more details), stars and galaxies with the same parent (if they have one) will all come from the same objID range. That is why the result data of query 4 comes from the same partitions. Therefore, the six cross joins queries of the OGSA-DQP query of Figure 1 actually return nothing. The final result of the nine-query union in Figure 1 is in fact from the three local join queries. In other words, all the result data is from three local joins that took place within the individual SDSS sites (please refer to [15] for more details on local joins).

So this is not an ideal test, but actually the evaluator is doing just as much work on the cross joins as it would if they produced non-zero results.

From the data origin point of view, query 5 is different from query 4. There are two schema objects in the FROM clause of query 5: Star and Field. All the data of the Field table is located in the SQL Server database on Ace and none on the Oracle databases on Gizmo and Gizmo2. However, data of the view Star is distributed across three sites: Ace, Gizmo and Gizmo2.

For the OGSA-DQP query of Figure 2, no result will come back from local joins at buck and icg, but there are possible results from local join at MyBestDR5one and the cross joins between different databases.

The individual local joins and cross joins in the query of Figure 2 all went well through OGSA-DQP. The total run time took between 4 and 14 seconds. However, the OGSA-DQP query 5 failed when run from Ace over the distributed MyBestDR5 database—again we had problem with the result sets union.

To break down the problem, we first tried to union the result set from the cross join from Gizmo to Ace with the result set from the local join on Ace. This leads us to query 6:

```sql
(SELECT MyBestDR5one_Star.psfMag_g, MyBestDR5one_Star.run, MyBestDR5one_Star.camcol, MyBestDR5one_Star.rerun, MyBestDR5one_Star_field)
FROM MyBestDR5one_Star, MyBestDR5one_Field
WHERE MyBestDR5one_Star_fieldID = MyBestDR5one_FIELDID
AND MyBestDR5one_Star.psfMag_g < 20
AND MyBestDR5one_Field.pspStatus = 2)
UNION ALL
(SELECT buck_STAR.PSFmag_g, buck_STAR.run, buck_STAR.camcol, buck_STAR.rerun, buck_STAR.FIELD)
FROM buck_STAR, MyBestDR5one_Field
WHERE buck_STAR.FIELDID = MyBestDR5one_FIELDID
AND buck_STAR.PSFmag_g < 20
AND MyBestDR5one_Field.pspStatus = 2)
```

Query 6 through OGSA-DQP likewise appears to hang. Its query plan in Figure 3 indicates the UNION operation was allocated to the evaluator on Gizmo. This is not a very effective plan because the local join result from MyBestDR5one on Ace would have to be transferred to Gizmo for the UNION operation. The final result sets from UNION operation would then be transferred back to the root evaluator (inside the coordinator) on Ace.

We enabled the debugging and examined the evaluator’s log files on both Gizmo and Ace. We turned on different debugging messages and re-ran query 6 a few more times. Interestingly, the query failed in different ways on different occasions.

There are two streams of immediate data from Ace evaluator to Gizmo evaluator as the two red data lines shown in Figure 3. In data stream A, EXCHANGE operation 2 is shipping the result sets from the APPLY operation 1 at Ace to Gizmo for the HASH JOIN operation 3 (this is the Gizmo-Ace cross join query). At the same time in data stream B, EXCHANGE operation 9 is shipping the result sets to Gizmo for UNION operation 6 from Ace’s APPLY operation 8 (for the Ace local join). It seemed possible the evaluator on Gizmo was confused, with some kind of sharing of data variables.

We decided to try another similar query. To simplify data processing and avoid unnecessary data transfer between Ace and Gizmo, the UNION operation should be done on the evaluator on Ace. The decision on placement of the UNION operation may be to do with the order of the sub-queries in the union query. In query 6 the Gizmo-Ace cross join query on the left UNION ALL operator and the Ace local join query is on the right. In query 7, we switched the sub-queries positions

<table>
<thead>
<tr>
<th>Bits</th>
<th>Length (# of bits)</th>
<th>Mask</th>
<th>Assignment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0x8000000000000000</td>
<td>empty</td>
<td>unassigned</td>
</tr>
<tr>
<td>1-4</td>
<td>4</td>
<td>0x7800000000000000</td>
<td>skyVersion</td>
<td>resolved sky version (0=TARGET, 1=BEST, 2-15=RUNS)</td>
</tr>
<tr>
<td>5-15</td>
<td>11</td>
<td>0x07FF000000000000</td>
<td>rerun</td>
<td>number of pipeline rerun</td>
</tr>
<tr>
<td>16-31</td>
<td>16</td>
<td>0x0000FFFF00000000</td>
<td>run</td>
<td>run number</td>
</tr>
<tr>
<td>32-34</td>
<td>3</td>
<td>0x000000000E000000</td>
<td>camcol</td>
<td>camera column (1-6)</td>
</tr>
<tr>
<td>35</td>
<td>1</td>
<td>0x0000000010000000</td>
<td>firstField</td>
<td>is this the first field in segment?</td>
</tr>
<tr>
<td>36-47</td>
<td>12</td>
<td>0x0000000000FFFF0000</td>
<td>field</td>
<td>field number within run</td>
</tr>
<tr>
<td>48-63</td>
<td>16</td>
<td>0x0000000000000000</td>
<td>object</td>
<td>object number within field</td>
</tr>
</tbody>
</table>

TABLE I
like this:

```sql
(SELECT buck_STAR.PSFMAG_G, buck_STAR.RUN, buck_STAR.CAMCOL,
    buck_STAR.FIELDID)
FROM buck_STAR, MyBestDR5one_Field
WHERE buck_STAR.FIELDID = MyBestDR5one_Field.FIELDID
AND buck_STAR.FIELDID = 2
UNION ALL
(SELECT buck_STAR.PSFMAG_G, buck_STAR.RUN, buck_STAR.CAMCOL,
    buck_STAR.FIELDID)
FROM buck_STAR, MyBestDR5one_Field
WHERE buck_STAR.FIELDID = MyBestDR5one_Field.FIELDID
AND buck_STAR.FIELDID = 2
```

The new query plan in Figure 4 looks more efficient. This is consistent with our guess that DQP optimiser chooses the query placed before the UNION ALL operator as the “leading query”, and will try to perform the UNION ALL operation on the related “leading” evaluator. The data from the query placed after the UNION ALL operator will be shipped to the “leading” database for the UNION operation. The final result sets will then be transferred to the coordinator site where the DQP query was issued (this is Ace in our examples).

However, query 7 failed in a similar way, despite a better query plan.

The evaluator log files in both cases appeared to show problems occurring inside org.apache.axis.message.addressing. This package is part of the Apache implementation of WS-Addressing, used by the Web Services Resource Framework (WSRF). The problem arises in a query plan where the evaluator issues two concurrent SQL queries to the local data service resource. It was eventually attributed to a race condition occurring in the Addressing handlers invoked in the client (evaluator) side of the request. Two instances of these handlers concurrently call out into the getUUIDGen() method of the UUIDGenFactory class in Apache Axis SOAP implementation (Axis) 1.2RC2. This method in turn calls the newInstance() of AxisProperties. The race condition seems to occur somewhere inside this core Axis class2. Making the newInstance() method into a synchronized method resolved the problem.

After fixing this problem in Axis, query 6 ran successfully through OGSA-DQP, taking 19 seconds. The original 9-union query of Figure 2 also ran successfully and only took 28 seconds in total (the time for executing the original SDSS query 5 was about 1 second.)

The non-determinism we observed is exactly the behaviour expected if some global variable is unintentionally shared between two threads, leading to classic non-determinism in a concurrent program. We did not have time to trace the exact location of the problem, which could be deep inside the AxisProperties class. It could either be a bug in Axis, or it could be that the original Axis method was never meant to be called more than once concurrently, and the WS-Addressing handlers were wrong to use it like this.

---

2The non-determinism we observed is exactly the behaviour expected if some global variable is unintentionally shared between two threads, leading to classic non-determinism in a concurrent program. We did not have time to trace the exact location of the problem, which could be deep inside the AxisProperties class. It could either be a bug in Axis, or it could be that the original Axis method was never meant to be called more than once concurrently, and the WS-Addressing handlers were wrong to use it like this.
SUMMARY AND FUTURE WORK

This paper tested a couple of astronomical queries against the distributed MyBestDR5 database. The SQL distributed queries were broken into an OGSA-DQP format of local joins and cross joins, integrated with UNION ALL operator. We intervened in the TransportHandler class of the DQP evaluator to address a timeout issue. We also resolved a race condition by synchronizing the newInstance() method of the AxisProperties class in Axis. Finally, we successfully used OGSA-DQP to run the queries that involve local joins and cross joins against a Heterogeneously distributed gigabyte-database—MyBestDR5. We will move on to running similar queries against a geographically distributed terabyte database next.

REFERENCES


