Optimising query execution time in LHCb Bookkeeping System using partition pruning and Partition-Wise joins

Zoltan Mathe, Philippe Charpentier
CERN, European Organization for Nuclear Research, Switzerland
E-mail: zoltan.mathe@cern.ch, philippe.charpentier@cern.ch

Abstract. The LHCb experiment produces a huge amount of data which has associated metadata such as run number, data taking condition (detector status when the data was taken), simulation condition, etc. The data are stored in files, replicated on the Computing Grid around the world. The LHCb Bookkeeping System provides methods for retrieving datasets based on their metadata. The metadata is stored in a hybrid database model, which is a mixture of Relational and Hierarchical database models and is based on the Oracle Relational Database Management System (RDBMS). The database access has to be reliable and fast. In order to achieve a high timing performance, the tables are partitioned and the queries are executed in parallel. When we store large amounts of data the partition pruning is essential for database performance, because it reduces the amount of data retrieved from the disk and optimises the resource utilisation. This research presented here is focusing on the extended composite partitioning strategy such as range-hash partition, partition pruning and usage of the Partition-Wise joins. The system has to serve thousands of queries per minute, the performance and capability of the system is measured when the above performance optimization techniques are used.

1. Introduction

In High Energy Physics (HEP) data intensive applications produce and analyse large amounts of datasets, which are distributed over millions of files and replicated on the Computing Grid around the world. In order to process these datasets efficiently metadata (Metadata is information that describes the data) or descriptive information about the datasets needs to be managed. Provenance information describes the way of how the datasets are produced. In order to support the data intensive applications a metadata catalogue is essential.

The LHCb Bookkeeping System [1] is a metadata and provenance catalogue used to record information about the datasets such as creation, transformation, type and quality of the data, and it allows querying for data, based on these descriptive attributes. One of the key components of the system is the LHCb Bookkeeping Database (BKDB), based on Oracle 11g and designed to store the conditions relative to jobs, files and their metadata, as well as their provenance information in a Relational Model.

In section 2 we present some database optimization techniques, which are followed by a short overview of the BKDB in section 3. In section 4 we present the performance of the database
using sequential and parallel queries and also comparing the performances of parallel queries when 1 or 2 database instances are used.

2. Techniques used for better database performance

Different optimization techniques are available in Oracle. One of the common techniques is the partitioning, that allows decomposing big tables into small manageable pieces, called partitions. A set of one or more columns of a certain table determines the partition for each row called partition key. The following partitioning methods are available: Range, List, Hash and Composite partitioning [2]. The Range, List and Hash partitioning methods determine how the data is partitioned based on the value of the partition key. The Composite partitioning is a combination of the previous methods. In our cases most of the data is historical, stored in big tables. Consequently, the most common type of partitioning is the Range partitioning, often based upon dates. However, the table access must be evaluated in order to join two big tables efficiently. If the partition key is not in the WHERE clauses, then partitioning of the tables does not improve the query performance. For example the jobs table is monthly partitioned on the creation time column. Most of the time, the job execution time, CPU, memory usage, etc. metrics are more interesting than the creation time. Consequently, partitioning the jobs table using the creation time as partition key does not change the database performance.

2.1. Partition Pruning

Partition pruning is used when the tables are already partitioned. The Oracle optimiser analyzes the FROM and WHERE clauses of the SQL statement in order to eliminate the partitions which are not relevant to the statement.

2.2. Parallelism

Parallelism plays an important role in a data warehouse when the tables are partitioned. It allows the database to be used efficiently and reduces the execution time of the queries. Parallel execution in Oracle is based on the principles of a Query Coordinator (QC) and parallel execution (PX) server processes. The Oracle optimizer decides the execution plan of a certain query. QC decides if a statement should execute in parallel or not according to the execution plan. In case of parallel execution, QC decides the degree of parallelism (The degree of parallelism is a metric which indicates the number of processors which can be used to execute a query simultaneously by a computer). After that QC distributes the work to the PX servers.

2.3. Partition-Wise joins

Partition-Wise joins is a query optimization technique, which reduces the query response time by minimising the data read from the cache/memory, because it only uses the relevant table partitions. It improves the CPU and memory usage.

Partition-Wise joins can be full or partial. A Full Partition-Wise join only performs when the tables are partitioned on their join keys. Partition-Wise joins can be performed parallel using the Producer/Consumer model.

3. Overview of BKDB

The database schema is divided into two parts: Warehouse schema and View. The Warehouse schema consists of various tables containing all data which ever entered to the database, while the Views contain data aggregated from the Warehouse schema. These views are used to improve the performance of those queries that are executed very often. The three main tables of BKDB are the following: Transformations (31 thousand rows, 1MB size) stores metadata of the productions. Each production contains a set of jobs which have the same configuration and
processing phases. Jobs (69 million rows, 13 GB size) stores metadata of the jobs which belongs to a certain production. Files (154 million rows, 27 GB size) stores metadata of the files which are produced/processed by jobs. These tables are partitioned and each one contains indexes which are partitioned as well.

3.1. Current partition schema
The jobs and files tables are Range partitioned on the jobId column. The main drawback of this strategy is that if the jobId is not in the FROM and WHERE clauses, we would benefit from the partition pruning only when the jobs and files tables are joined together.

3.2. New partition schema
In the new partition scheme we adopted the Range and Composite partitioning methods. We have partitioned only the largest tables: jobs, files and transformations. The jobs table is partitioned using Range-Range composite partitioning as shown in figure 1. According to the partition key of the jobs table each partition contains jobs which are in a certain production range. In addition, each partition has a sub-partition and contains only jobs with JobIds which are in a given interval. Figure 2 shows the partitioned schema of the files table. We applied Range-List composite partitioning method to partition the files table. The partition key of the files table is the JobId and the sub partition key is the replica flag which can be Yes or No. The Transformations table is Range partitioned on the Production column.

4. Tests and Results
In order to test the performance of the system we used the same query in our tests and we stopped the database applications to make sure the tests was unaffected by other database activities.

Listing 1. The query used by our tests to measure the database performance

```
SELECT COUNT(*), SUM(f.eventstat), SUM(f.filesize)
FROM job j, simulationconditions sim, transformations prod, files f
WHERE prod.simid=sim.simid and prod.production=j.production and sim.simdescription='Beam3500GeV–Oct2010–MagDown–Nu2,5' and prod.configurationid=2031 and f.jobid=j.jobid and f.gotreplica='Yes' and f.visibilityflag='Y';
```
According to the query tree, which is shown in figure 3, the queries are executed in the following order: the database joins the simulations conditions table to the transformations table. The result of this join will be joined to the jobs table, which will be joined to the files table. The query shown in listing 1 is executed serially or in parallel by four client processes (each client is a python thread which executes the query parallel using cx_Oracle python module) for a duration of one hour. We use a 2-node cluster, which has shared disk architecture in which all disks are accessible from first and second cluster node. Each node had 2 x Quad Core Intel E5630 @ 2.53 CPU, 48 GB of RAM and 36 SATA 7200 RPM 2TB disks configured with Oracle Automatic Storage Management (ASM) as RAID10. We measured the Response Time and number of queries in the client side using 1 and 2 database instances. Each database instance runs in a separate node. We used Maximum Likelihood Estimation (MLE) to estimate the response time of the system.

4.1. Test 1
The query introduced in listing 1 is executed by four client processes sequentially during 1 hour using the current and new partition schema. Figure 4 shows the result of this experiment. The estimated response time of the current system is 69.82 seconds and it is decreased to 54.47 seconds using the new partition schema. The number of executed queries in the current system is 250, which is increased to 524 using the new partition schema. The increase the throughput...
and decrease of the response time is attributable to the Full Partition-Wise join and the Partition Pruning.

4.2. Test 2
We executed the same query parallel by four client processes during 1 hour. We compared the response time and the number of executed queries using alternatively 1 or 2 database instances. In addition we measured the speed up of the parallel queries. The query does not scale to more than 8 cores, as shown in figure 5. We assume that it is attributed to the available physical cores (a single database instance has 8 physical cores, and 16 logical cores as the hyper-threaded is enabled) or the query can not be parallelised to more than 8 cores.

Figure 6. Histogram of 1361 executed queries

Figure 7. Histogram of 2041 executed queries

According to the figures 6 and 7, the number of executed queries increases by executing the query parallel. Consequently, the number of executed queries increases and the response time decreases by adding more nodes to the system.

5. Conclusion
The presented optimization techniques can be used to decrease the response time and increase the throughput of a database system. According to the results the composite partitioning is very useful when the size of the data is large and the queries are complex. Parallelism plays an important role using the CPU efficiently as well as improving the performance of the database.

References
[1] Mathe Z 2012 Ph.D. thesis University College Dublin URL http://cds.cern.ch/record/1491175
[2] Oracle 2011 Oracle Database Concepts, 11g Release 1 (Oracle Corporation) URL http://docs.oracle.com/cd/B28359_01/server.111/b28318.pdf
[3] Myung I J 2003 Journal of Mathematical Psychology 47 90 URL http://people.physics.anu.edu.au/~tas110/Teaching/Lectures/L3/Material/Myung03.pdf
[4] Oracle 2011 Database Parallel Execution Fundamentals (Oracle Corporation) URL http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-parallel-execution-fundamentals-133639.pdf
[5] Oracle 2012 Storage Administrator’s Guide 11g Release 1 (11.1) (Oracle Corporation) URL http://docs.oracle.com/cd/B28359_01/server.111/b31107.pdf