Research on the Advanced Computing Method for Supporting Large Data Quality Assessment and Improvement

He Yang[1, *a], Jiangqi Chen[1], Xiaojia Xiang[2], Heng Liu[3] and Yunpeng Li[4]

[1]Global Energy Interconnection Research Institute, State Grid Corporation of China, Beijing 102200, China;

[2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100089, China.

[3]State Grid Beijing Electric Power Company, Beijing 100000, China.

[4]State Grid Jiangsu Electric Power Company Nantong Power Supply Company, Nantong 226006, China.

[a]yanghe@geiri.sgcc.com.cn,*Corresponding author Email: [a]yanghe@geiri.sgcc.com.cn

Keywords: Cross-database Queries, Big Data Processing, Apache Hive, Data Quality Assessment and Improvement, Task Relevance.

Abstract. To support the high efficient and fast data quality assessment of electrical operation, we need to make optimization of high performance computing technology on computing platform, this paper carry out in-depth research on the performance bottleneck that the data quality evaluation system faces, after the analysis of big data platform on data quality assessment and improvement, we make the design and implementation of easily cross-database queries, which can seamlessly integrate relational data into Hadoop ecosystem, and put forward a kind of optimization model for Hive by considering task relevance.

1. Introduction

In recent years, more and more grid companies have begun to use the MapReduce-based platform for data quality assessment and improvement. It can be noted that most of these companies that make use of emerging technologies are often Internet companies. But in fact, large grid companies are also has a huge demand on relatively low-cost, scalable data analysis platform. Different from Internet companies these grid companies often accumulated a large number of data in relational database. When the amount of data increases, on one hand the cost of expansion of relational database has doubled; on the other hand, program performance and scalability is still facing serious problems. These grid companies will inevitably start using MapReduce and Hive platform for offline data analysis. However, directly using Hadoop ecosystem in the grid companies are also facing many difficulties.

In order to solve these problems, this paper presents a cross-database query system, and make the design and implementation of a Hive optimization model by considering task relevance. Later, this paper is organized as follows, the second section describes the cross-database query language, the third section describes the optimization for the Hive query, and the fourth chapter is the summary.

2. Cross-database Query Model

2.1 Demand Analysis

A lot of legacy data stored in relational databases cannot be fully made use of in the new tide of technology, which forms an isolated information island, and results in a waste of the original investment, now we need a custom Hive, with which we can have direct access to a variety of relational databases, and no longer need to use other additional tools, based on this demand, we designed a cross-database query language.

Copyright © 2017, the Authors. Published by Atlantis Press.
This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

2.2 Design and Implementation

As we know, Hive use ANTLR[7] to compile Hive's syntax files to generate java source files. ANTLR is a tool that can accept language descriptors and generate programs that recognize the sentences produced by those languages. We extend Hive's syntax file according to the rules of ANTLR to recognize the external relational data source. Then we modify the Hive source code and add IO operator to the Hive's original operators to deal with the external data table. IO operator's implementation is based on the open-source Sqoop[8], but is different from Sqoop. At first, Sqoop and Hive have to run separately, which means that it needs additional operations, while IO operator can directly run by running the expansion of HiveQL; Secondly, Sqoop can only support the import and export between RDBMS and Hive (HDFS), not between RDBMS and memory, while IO operator can support the choice of two modes, During importing, you can first store the table into Hive, and then process the table, or you import the table directly into memory instead of Hive, and process it in memory, During exporting, you can first store the table into Hive, and then export the table into RDBMS, or you can export the table directly from memory instead of Hive to RDBMS.

2.3 Usage

First of all we have to define the data source, we can give the data source a legal name, assuming that the name is 'orcl1', and then use the set statement to define the data source information, Here we need three statements to define the user name, password and URL, respectively, Examples are as follows.

```
set orcl1.user=YONGCAI;
set orcl1.password=PASSWORD;
set orcl1.url=jdbc:oracle:thin:@10.68.27.127:1521:;
```

Then we can directly use the defined data source, suppose we have defined the 'orcl1' and 'mysql1', and we would like to query the table 'table1' in the database 'database1' in the 'orcl1' data source and the table 'table2' in the database 'database2' in the 'mysql1' data source. You can see that this is a cross-database query, the query is as follows, eTable is a keyword defined by me which means external table exists.

```
select * from eTable.orcl1.databases1.table1 join eTable.mysql1.database2.table2 on table1.id=table2.id;
```

IO operator supports two modes, in mode 1 we handle the table in memory, during importing the data of table is not imported into the HDFS, Hive directly processes the data in memory, during export we directly store the data in memory to RDBMS, in mode 2 the data will be stored in the HDFS, and then imported to the Hive or exported to RDBMS, in this mode the query statement is unchanged, during importing Hive will automatically import the table into Hive before the query executes, the default mode is mode 2, if you want to enable import and export in memory (mode 1), you can use the following statement in hive.

```
set ETableinMemory=true;
```

3. High Performance Computing Platform

3.1 Principle

This computing platform considers the relativity between tasks during Hive execution and minimizes the number of MapReduce jobs. Specifically, the HQL statement is transformed into a MapReduce physical plan and the shuffle key used for sorting in each shuffle stage of the MapReduce Job in the MapReduce physical plan is obtained; and the associated MapReduce Job in the MapReduce physical plan is merged according to the relevance rules.

3.2 Rules

Rule a: If n (n is positive integer and n is less than the total number of MapReduce jobs in the MapReduce physical plan) MapReduce jobs do not contain a parent-child relationship and the shuffle key is the same, the n MapReduce jobs are mutually related MapReduce Jobs and we can merge the associated MapReduce Jobs determined by the rule a;

Rule b: If the parent MapReduce job's shuffle key is the same as the shuffle key of all its child MapReduce Jobs, then the parent and all its child MapReduce jobs are mutually related MapReduce jobs, and we can merge the associated MapReduce jobs determined by the rule b;

Rule c: If the parent MapReduce job's shuffle key is the same as the shuffle key of one of the child MapReduce jobs in all its child MapReduce jobs, then the parent MapReduce job is related with a child MapReduce job in all of its child MapReduce jobs, and we can merge the associated MapReduce Jobs determined by the rule c;

3.3 Implementation

The associated MapReduce jobs determined by the rule a are merged as follows:

The n MapReduce Job read disks to get the data $\{(k_1^1, v_1^1), (k_2^1, v_2^1) \dots (k_i^1, v_i^1) \dots (k_n^1, v_n^1)\}$ to be processed by the map functions of the n MapReduce Jobs, and then we associate the n shuffle keys of the map functions' output data $\{(k_1^2, v_1^2), (k_2^2, v_2^2) \dots (k_i^2, v_i^2) \dots (k_n^2, v_n^2)\}$ of the n MapReduce Jobs respectively with n tags $\{tag_1, tag_2 \dots tag_i \dots tag_n\}$. The result of merging is $\{(k_1^{2\prime}, v_1^2), (k_2^{2\prime}, v_2^2) \dots (k_i^{2\prime}, v_i^2) \dots (k_n^{2\prime}, v_n^2)\}$, in detail, (k_i^1, v_i^1) is the data to be processed by the mapping function corresponding to the i-th MapReduce Job, k_i^1 is the key of the data to be processed by the map function corresponding to the i-th MapReduce Job, and v_i^1 is value of the key of the data to be processed by the map function corresponding to the i-th MapReduce Job. The data (k_i^2, v_i^2) is the output data of the map function corresponding to the i-th MapReduce Job, k_i^2 is the shuffle key corresponding to the i-th MapReduce Job and the shuffle keys corresponding to the n MapReduce Jobs is the same (equal), v_i^2 is the value of the shuffle key of the output data of the map function corresponding to the i-th MapReduce Job. There is a unique correspondence between the shuffle key k_i^2 and tag_i which is used to mark the corresponding relationship between the shuffle key and the MapReduce job.

The merging result $\{(k_1^{2\prime}, v_1^2), (k_2^{2\prime}, v_2^2) \dots (k_i^{2\prime}, v_i^2) \dots (k_n^{2\prime}, v_n^2)\}$ is put into the shuffle process, and the data source is determined by the mark corresponding to the shuffle key, that is, the data $(k_{\mathrm{i}}^2, v_{\mathrm{i}}^2)$ marked by tag_i is sent to the reduce function corresponding to the i-th MapReduce Job.

The associated MapReduce jobs determined by the rule b are merged as follows:

All child MapReduce jobs of the parent MapReduce job are mutually related MapReduce jobs according to the rule a, so all child MapReduce jobs of the parent MapReduce job are merged according to the rule a at first.

The result of the reduce functions corresponding to all the child MapReduce jobs of the parent MapReduce job is sent directly to the map function of the parent MapReduce job and the map function of the parent MapReduce job directly sends the output of the map function of the parent MapReduce job to the reduce function of the parent MapReduce job.

If in addition to getting data from the child jobs, the parent MapReduce Job also reads data from a table file, the table file is equivalent to the execution result file of MapReduce Job that cannot be merged in rule c, and is handled in the same way as rule c.

The associated MapReduce jobs determined by the rule c are merged as follows:

The child MapReduce jobs that cannot be merged is executed first, and the result is saved to disk.

The parent MapReduce job reads the saved data in the previous step from disk, and the data to be processed by the map functions of the child MapReduce jobs merged with it.

Send the data to be processed by the map functions of the child MapReduce jobs to the map functions of child MapReduce jobs and obtain the output results of the map functions of the child MapReduce jobs. The output result is put into the shuffle process and then sent to the reduce function of child MapReduce jobs, the output results of the reduce functions of the child MapReduce jobs are sent directly to the map function of the parent MapReduce job and the parent MapReduce job directly sends the output result of the map function of the parent MapReduce job to the reduce function of the parent MapReduce job without shuffle.

Send the saved result of the child MapReduce jobs that cannot be merged to the map function of the parent MapReduce Job and obtain the output result of the map function of the parent MapReduce

Job, put the output result into the shuffle process, after which the output result will be sent to the reduce function of the parent MapReduce Job.

3.4 Examples

We use the TPCH Q17 as an example, Transaction Processing Performance Council (TPC) is a non-profit organization founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry. As show in Figure 1, Hive translates the TPCH Q17 into four MapReduce jobs, wherein MapReduce job 1 and MapReduce job 2 has no child tasks, reading the table data from disk, can be directly executed, the child jobs of MapReduce job 3 are MapReduce job 1 and MapReduce job 2, MapReduce job 3 reads the results of MapReduce job 1 and MapReduce job 2 from disk, so MapReduce job 3 has to be executed after the finish of MapReduce job 1 and MapReduce job 2. The child job of MapReduce job 4 are MapReduce job 3, MapReduce job 4 read the result of MapReduce Job3 from disk, so MapReduce job 4 has to be executed after the finish of MapReduce job 3.

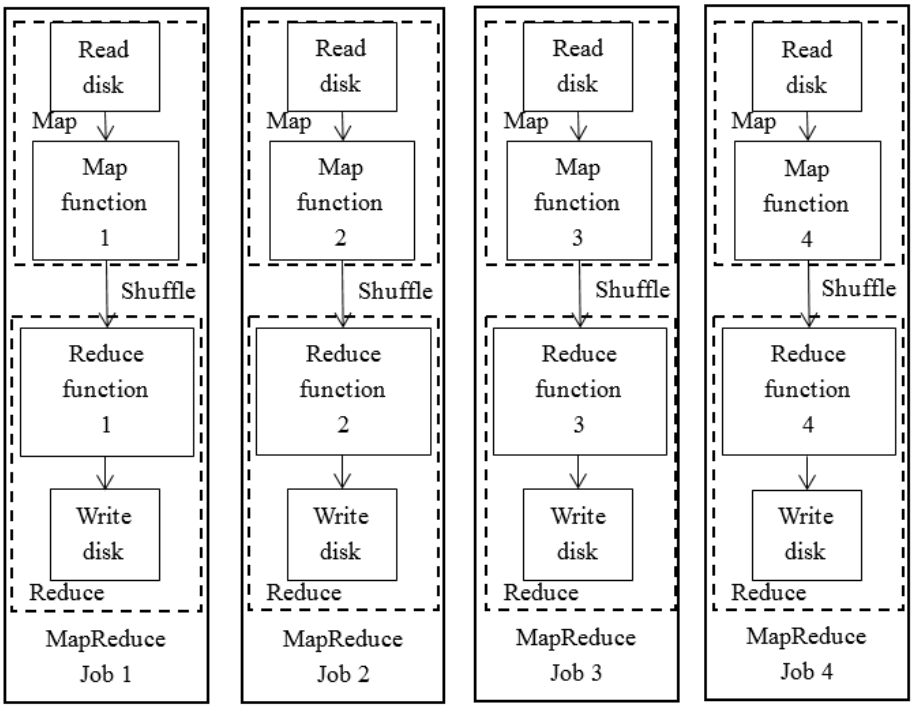

Fig. 1 Q17

In detail, Hive automatically generates four map functions: map function 1, map function 2, map function 3, map function 4, and four reduce functions: reduce function 1, reduce function 2, reduce function 3, and reduce function 4. After the execution of each map function Hive will call the MapReduce interface to send the result data of map function to the shuffle stage, then shuffle stage will send the data to the reduce function, the reduce function will call the corresponding MapReduce interface to obtain the data to be processed by the reduce function.

The optimization process described in this paper will neither rewrite the data processing part of the map and reduce functions, nor modify the MapReduce framework. We only modify the code used for transfer in the map function and reduce function, that is, we only modify the part of calling interface to read or send data, for example, the results of the map function is originally passed to shuffle by calling the MapReduce interface, now we may delete the code of calling interface to send the output data of the map function to shuffle, and add the code of sending the output data of the map function directly to the reduce function of the MapReduce job, or we may add a tag field to the key of the output key-value pair of the map function, the value of the output key-value pair of the map function is not changed, this new output key-value pair will still be sent to shuffle.

The shuffle key of the MapReduce job 1, MapReduce job 2, and MapReduce job 3 in Figure 1 of Q17 is the same, which satisfies the rule b, so we can merge the three jobs according to rule b, as shown in Figure 2, we do not modify the internal data processing part of the original four map functions and reduce function; when reading the disk, we get the data that either map function 1 or

map function 2 need, Hive will automatically determine whether to pass the data to map function 1 or map function 2 according to the file path, Assume that the output data sent by the map function 1 is (k21, v21), the output data sent by the map function 2 is (k22, v22). Since the shuffle keys of task 1 and task 2 are the same, k21 and k22 are actually one field (which can be collectively referred to as k2), we can send k2 (k21 and k22) to shuffle together in one MapReduce job, it is worth noting that slightly different from the original map function 1 and map function 2 we don't change the output data of the two map functions (We do not change the executing process of original function), but we get the output of the two original map functions, then the MapReduce rewrite module will modify the contents of k2, combining k2 and a tag into a field denoted by k2 ', we add the tag to differentiate the data from map function 1 to the data from map function 2, and finally (k2 ', v21 or v22) is passed to shuffle. When the tag is equal to 0, the content is (k2 ', v21), when the tag is 1, the content is (k2', v22). The result of shuffle is (k2 ', v21, or v22), we get the result of shuffle, determine the data source according to the tag field, and then extracts the k21 and k22 from k2' before the origin reduce function 1 and reduce function 2 execute, after which we send the data respectively to reduce function 1 and reduce function 2, the content got by reduce function 1 and reduce function 2 is the same, the data is still (k21, v21) and (k22, v22), so the data processing part of reduce function 1 and reduce function 2 is not needed to change. Assume that the data obtained by the map3 function is (k13, v13), in the case of running without optimization the map function 3 reads the result from disk stored by reduce function 1 and reduce function 2, here we can directly send the output data to map function 3, that is, the reduce function 1 and reduce function 2 directly send the output data to map function 3 instead of saving the data to disk. As before, map function 3 will automatically recognize if the input is from reduce function 1 or reduce function 2 based on the path, since the data passed to map function 3 in Figure 2 is ago sorted by shuffle and the sort key is not changed, the order does not need to be rearranged (A new shuffle stage gets the same result, and will not change the order), so the map function 3 can directly pass the output result of map function 3 to the reduce function 3, at last the output result of the reduce function 3 is save in disk and is ready to be passed to MapReduce job 2 in Figure 2.

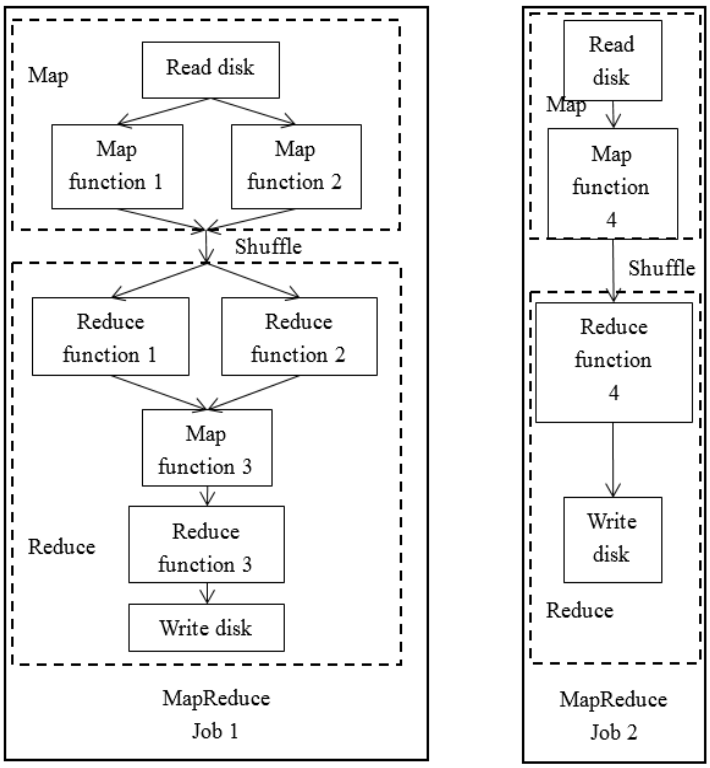

Fig. 2 Optimized Q17

In order to illustrate the rules c, we give an example, as shown in Figure 3, there are three MapReduce jobs, MapReduce job 1 and MapReduce job 2 have no child jobs, they can be immediately executed. MapReduce job 3 is the parent task of MapReduce job 1 and MapReduce job 2,

the shuffle key of MapReduce job 3 is the same as the shuffle key of MapReduce job 1, MapReduce job 1 and MapReduce job 3 can be merged according to rule c， so we rewrite the MapReduce physical plan.

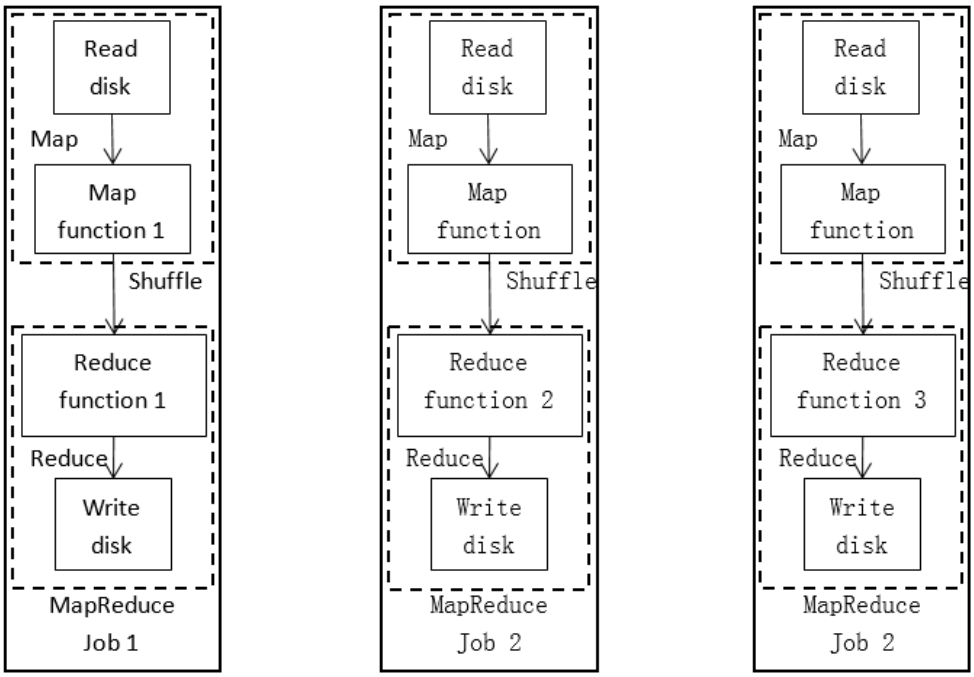

Fig. 3 An example of rule c

As shown in Figure 4, we get the table data from disk, send it to the map function 1 of MapReduce job 1, and send the output result of map function 1 to the shuffle process, after shuffle the result data will be sent to the reduce function 1 according to the tag field, then the output of the reduce function 1 of MapReduce job 1 is sent directly to the map function 3 of MapReduce job 3, and the result of map function 3 of MapReduce job 3 is sent directly to the reduce function 3 of MapReduce job 3. Meanwhile MapReduce job 2 in Figure 4 reads data from disk to get the result of MapReduce job 1 in Figure 4, sends it to the map function 3 of MapReduce job 3, gets the output result of map function 3 of MapReduce job 3, and puts the output result into shuffle process, then sends the data to reduce function 3 of MapReduce job 3.

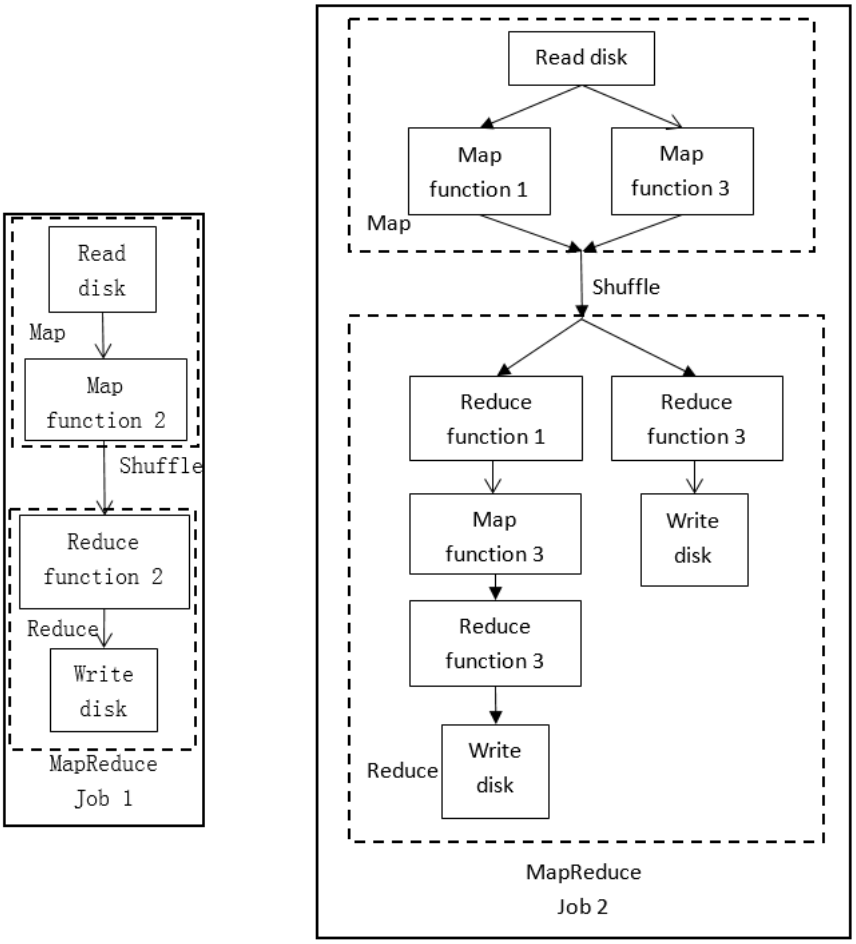

Fig. 4 An optimized example of rule c

3.5 Performance Testing

We use three virtual machines deployed in a Huawei server, the server is equipped with two 2-way 6-core 12-thread X5650 CPUs, three virtual machines are equipped with double cores, 2G memory, and 40G hard drive, The IP for master is 192.168.94.129, the IPs for slaves are 192.168.94.129, 192.168.94.130 and 192.168.94.131, the virtual machine whose IP is 192.168.94.129 is used both as master and slave, the version of Hadoop is 2.5.2, the version of Hive is 0.14.

We use TPCH Q17 as the test benchmark, both 100M data set and 1G data set were tested 6 times. The results are as follows. We can see that our system outperforms Hive in all cases. There are 2 MapReduce jobs instead of 4 MapReduce jobs after optimization, by considering the relevance of the jobs, we greatly improving the performance of the system.

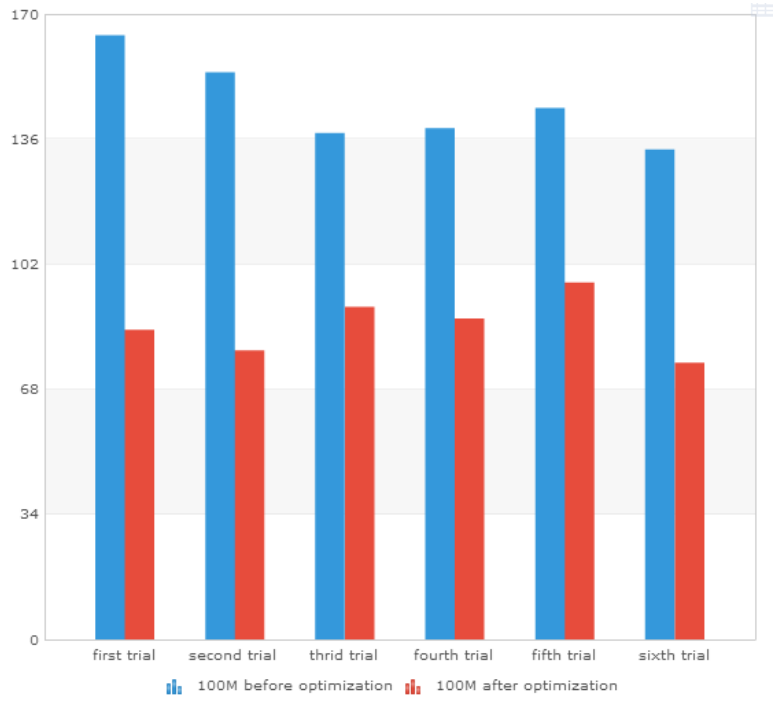

Fig. 5 Test of 100M data set

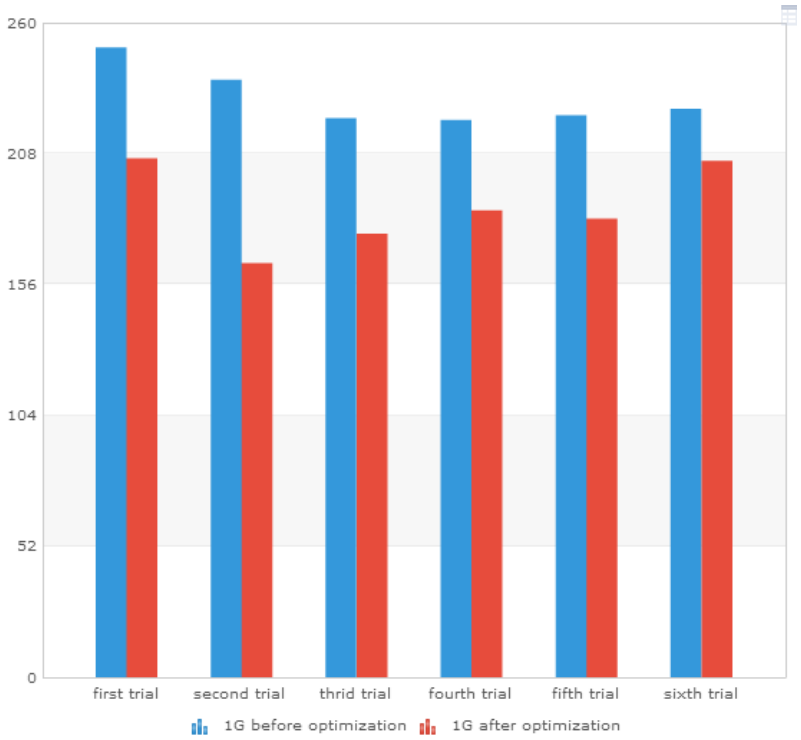

Fig. 6 Test of 1G data set

4. Summary

This paper first introduces the research background, puts forward the application requirements, the second section presents the cross-database query model, describes how the system is implemented on the basis of Hive in detail, the third section presents a high-performance computing platform, As we can see through the test, when the optimization condition meets, after optimization, the performance can be greatly improved.

In general, this paper implements a cross-database analysis model and a high-performance computing platform, the system has achieved the intended target.

Acknowledgments

The work was supported by State Grid Company Research Project under grant SGRIJSKJ[2015]1029.

References

[1]. Carr DF. How google works. Baseline Magazine 2006, p. 6.

[2]. Ghemawat S, Gobioff H, Leung S-T. The Google file system. ACM SIGOPS Operating Systems Review 2003, p. 29-43.

[3]. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM 2008. Vol. 51, p. 107-113.

[4]. Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS) 2008. Vol. 26, p. 4.

[5]. Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system. Mass Storage Systems and Technologies (MSST). 2010 IEEE 26th Symposium on 2010, p. 1-10.

[6]. Thusoo A, Sarma JS, Jain N, et al. Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment 2009. Vol. 2, p. 1626-1629.

[7]. ANTLR: http://www.antlr.org/

[8]. Sqoop User Guide: http://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html

[9]. Lee R, Luo T, Huai Y, et al. Ysmart: Yet another sql-to-mapreduce translator. Distributed Computing Systems (ICDCS). 2011 31st International Conference on 2011, p. 25-36.

[10]. Chaudhuri S. An overview of query optimization in relational systems. Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (1998), p. 34-43.

[11]. White T, Zeng Dapin, Zhou Aoying. Hadoop authoritative guide. Tsinghua University Press, 2010.