International Journal of Database and Theory and Application
Vol.9, No.2 (2016), pp.153-160
http://dx.doi.org/10.14257/ijdta.2016.9.2.17

Query and Analysis of Data on Electric Consumption Based on Hadoop

Jianjun[1] Zhou and Yi Wu[2]

[1]*Information Science and Technology in Heilongjiang University*
[2]*School of Information Science and Technology, Heilongjiang University Harbin, Heilongjiang,150080, China,*
Zhou_1969@tom.com, wy51cn@aliyun.com

Abstract

Traditional data management is usually based on relational databases, which are capable of managing small amounts of data. But relational databases have some difficulty in inquiry, management, and analysis of large amounts of data and magnanimity data. The method of effective management of magnanimity data is a problem deserving of study. In this paper, traditional relational databases are moved to Hadoop, in order to implement query and analysis on Hadoop. This paper changes the amount of data record and number of nodes in clusters, and records the query time in different conditions. Advantages and disadvantages of query on Hadoop can be analyzed by comparing the statistics with the query time on relational database Oracle. The factors affecting the time of query on Hadoop can be found by analysis. Furthermore, the result is also a reference material of future research and data managements on cloud platforms.

Keywords*: Hadoop cluster, parallel computing, distributed file system, MapReduce Module, data warehouse*

1.Introduction

In modern society, with the development of information technology, people produce data every day, everywhere. The value of data will get greater as technology develops. But this value cannot be found unless it is handled properly. Every single data has immeasurable value, but data needs storage and management. It's a important question to manage, inquire and update the data according to the demand. Nowadays, data management is mostly based on relational databases. But the query and processing of magnanimity data is beyond the supporting range, or the high efficiency range of relational database. So the improvement of available data management system is urgently needed to satisfy the demand.

After investigation, this paper finds that, distributed file system and cloud computing can deal with the amounts and the efficiency problem. Among numerous cloud platforms, Hadoop, an open source cloud platform designed by Apache has the most influence. Hadoop is made up of distributed file system HDFS and MapReduce module. But distributed file system normally supports unstructured data. People's conception toward data management is always related with relational databases because they have been dominated for a long time. It will cost some time for people to change to a new conception. And most data is structural in databases. Most firm or corporation wouldn't like to change the structural data into unstructured, because this process may lose a lot of valuable information. In view of the reasons above, though Hadoop has a database Hbase, which fits the storage and operation of unstructured data based on key and

ISSN: 1975-0080 IJDTA

value (like Google's Bigtable), it cannot be used in the migration and query of data in relational databases. Thus, it is really worth researching that how to query relational databases on cloud platforms.

This paper applies the basic principle of cloud computing and cloud storage, and studies Hadoop—a cloud platform which is designed by Apache based on Google's ideas of cloud computing—and its subproject, Hive. Finally, it achieves query and analysis of data on Hadoop.

Hadoop cloud platform is suitable for handling big amounts of data. When dealing with magnanimity data, advantages of Hadoop would appear. But Hadoop, as well as its subproject Hive do not perform very well when handling normal data. Thus, this paper's aim is to analyze the possible factors that may affect query of relational databases on cloud platforms, by comparing the statistics obtained in the experiment. It will also give a reference to later research and improvement[1][2].

In the experiment of this paper, the number of nodes in Hadoop clusters is changed (from 1 to 5, limited by experimental conditions), and the size of query records are changed (the numbers of query records are 1million, 3million, 8million, 12million, 30million). Different query time is recorded under these changes. Bottlenecks of small-amount query are found through analysis of the statistics. Comparing the results with the same query on Oracle, this paper shows the advantages of cloud platform when dealing with magnanimity data.

2. Interrelated Research

2.1 Hadoop Platform

Hadoop, which is developed by Apache foundation, is a infrastructures of distributed system. Users can develop distributed programs and fully exploit the high-speed computing and storage of the clusters even if they don't know the distributed bottom layer. Hadoop is a software platform that is easy to use and process magnanimity data. Hadoop realizes a distributed file system HDFS (Hadoop Distributed File System). HDFS has a good fault tolerance, and is designed to be disposed in cheap hardware. It provides high transaction rates to access the data of application programs. Hadoop is very suitable to programs which have huge amounts of data[3][4][5].

2.2 HDFS

HDFS (Hadoop Distributed File System), the basic layer of Hadoop system, is responsible for the storage, management, and error handling of data. The idea of HDFS is consulted in Google's file system GFS. From being the infrastructure of Apache Nutch search engine, to being a subproject of Hadoop, HDFS has absorbed the strongpoint of many distributed systems. HDFS is a distributed file system with a high fault tolerance, and it is suitable for cheap hardware. It not only can provide high-throughput data access, but also has easiness in expansibility. So it can be deployed on massive clusters, and be used to deal with magnanimity data.

HDFS is the cornerstone of distributed computing. HDFS stores all of the data files on Hadoop. It is produced with Master/Slave structure, which is made of a NameNode, a backup of NameNode (usually is secondary NameNode) and more than one DataNodes. NameNode, which is the manager in the file system, is mainly responsible for the management of the namespaces of the file system, and the coordination of users' access to the data. It will store the descriptive metadata of the file system as lists in the memory, so that users can have immediate access to them. DataNode, the actual storage of data, partitions the local disc into many blocks to store data, and saves the metadata of them in the memory. DataNode reports to NameNode periodically. Meanwhile, NameNode

separates the data into many parts and stores them in these blocks. HDFS file system achieves high reliability by using copy storage strategy.

2.3 MapReduce

MapReduce is a programming model, as well as a high-efficient scheduling model. MapReduce comes from the concept of functional programming Map and Reduce. Map means mapping, Reduce is to control[2]. With MapReduce, programmers can run their programs on distributed systems without knowing distributed programming. In function Map, programmers can assign the procedure for the processing of data in each block, and in function Reduce, they can control the intermediate result from the processing of data in each block. Users only need to assign function Map and Reduce to write distributed programs. They don't need to care about how to partition, allot and schedule the data inputted[6][7][8].

2.4 HBase

HBase, a distributed columnar open source database, is a distributed storage system which has high reliability, capability and flexibility. With HBase, large scale structural storage clusters can be set up on very cheap PC servers.

HBase has Hadoop HDFS as its file storage system, uses Hadoop MapReduce to handle magnanimity data stored in HBase, and has Zookeeper as correspondence.

However, HBase does not fit the problem in the background of this paper, for the reason below. HBase is between NoSQL and RDBMS. It retrieves data only by row key and the range of row key, and does not support coarse index. But the relational databases usually inquire data by their properties. HBase only supports transactions with a command line, but there is a lot of linking in relational databases. Considering the next step is to analyze massive data, Hive, which will be mentioned below, is more suitable.

2.5 Hive

Hive, a data warehouse tool based on Hadoop, can map structural data files to a database table, and provides entire SQL inquiry function. Hive can also change SQL statements to MapReduce tasks and run them.

Hive does not have a specific storage format or an index for the data. Users can organize the data table very freely. The only thing Hive need to analyze data is row delimiters and column delimiters when making the table. All the data in Hive is stored in HDFS. The data models in Hive are Table, External Table, Partition, and Bucket.

The Table in Hive is similar to normal Table in relational databases in concept, which means that every table in Hive has a relevant catalog to store data. For example, a Table named rdb_tb, whose path in HDSF is: /xxx/rdb_tb. And xxx is the catalog of data warehouse indicated by ${hive.metastore.warehouse.dir} in hive-site.xml, all the data in this Table (excluding External Table) are saved under this catalog.

Partition corresponds to Dense Indexes of columns of Partition in relational databases. But the structure of Partition in Hive is quite different. In Hive, a Partition in Table corresponds to a catalog in Table, and each of the Partition is stored in the catalog which it corresponds to.

2.6 Sqoop

Sqoop (SQL-to-Hadoop) is used to move data between Hadoop and relational databases. Sqoop can import the data from relational databases to HDFS on Hadoop, and import the data from HDFS to relational databases.

3. Processing and Analysis on Hadoop

3.1 Preparation of the Data

The data used in this experiment is electric consumption of an area during a period of time. They were stored in relational databases. If processing these data on a cloud platform, we must import them into Hive. Sqoop is used to finish this task.

```
14/05/09 12:03:20 INFO mapred.JobClient:   File Input Format Counters
14/05/09 12:03:20 INFO mapred.JobClient:     Bytes Read=0
14/05/09 12:03:20 INFO mapred.JobClient:   Map-Reduce Framework
14/05/09 12:03:20 INFO mapred.JobClient:     Map input records=2514859
14/05/09 12:03:20 INFO mapred.JobClient:     Physical memory (bytes) snapshot=80822272
14/05/09 12:03:20 INFO mapred.JobClient:     Spilled Records=0
14/05/09 12:03:20 INFO mapred.JobClient:     CPU time spent (ms)=37400
14/05/09 12:03:20 INFO mapred.JobClient:     Total committed heap usage (bytes)=558366
14/05/09 12:03:20 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=395927552
14/05/09 12:03:20 INFO mapred.JobClient:     Map output records=2514859
14/05/09 12:03:20 INFO mapred.JobClient:     SPLIT_RAW_BYTES=87
14/05/09 12:03:20 INFO mapreduce.ImportJobBase: Transferred 267.0477 MB in 187.8825 se
14/05/09 12:03:20 INFO mapreduce.ImportJobBase: Retrieved 2514859 records.
14/05/09 12:03:20 INFO manager.OracleManager: Time zone has been set to GMT
14/05/09 12:03:20 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM RD
14/05/09 12:03:20 WARN hive.TableDefWriter: Column SHIBIAO had to be cast to a less pr
14/05/09 12:03:20 WARN hive.TableDefWriter: Column CBSJ had to be cast to a less preci
14/05/09 12:03:20 WARN hive.TableDefWriter: Column ZXZYGDN had to be cast to a less pr
14/05/09 12:03:20 WARN hive.TableDefWriter: Column SJZT had to be cast to a less preci
14/05/09 12:03:20 WARN hive.TableDefWriter: Column FL1DN had to be cast to a less prec
14/05/09 12:03:20 WARN hive.TableDefWriter: Column FL2DN had to be cast to a less prec
14/05/09 12:03:20 WARN hive.TableDefWriter: Column FL3DN had to be cast to a less prec
14/05/09 12:03:20 WARN hive.TableDefWriter: Column FL4DN had to be cast to a less prec
14/05/09 12:03:20 INFO hive.HiveImport: Removing temporary files from import process:
14/05/09 12:03:20 INFO hive.HiveImport: Loading uploaded data into Hive
14/05/09 12:03:21 INFO hive.HiveImport: WARNING: org.apache.hadoop.metrics.jvm.EventCo
14/05/09 12:03:22 INFO hive.HiveImport: Logging initialized using configuration in jar
14/05/09 12:03:22 INFO hive.HiveImport: Hive history file=/tmp/namenode/hive_job_log_n
14/05/09 12:03:25 INFO hive.HiveImport: OK
14/05/09 12:03:25 INFO hive.HiveImport: Time taken: 2.984 seconds
14/05/09 12:03:25 INFO hive.HiveImport: Loading data to table default.rdj_tb
14/05/09 12:03:26 INFO hive.HiveImport: OK
14/05/09 12:03:26 INFO hive.HiveImport: Time taken: 0.569 seconds
14/05/09 12:03:26 INFO hive.HiveImport: Hive import complete.
14/05/09 12:03:26 INFO hive.HiveImport: Export directory is empty, removing it.
namenode@hadoopnamenode:~$
```

Figure 1.Data Importation

We can see from Fig. 1 that, MapReduce tasks is generated when Sqoop imports data from relational databases into Hive, and Hadoop executes these tasks.

After the importation, we execute a query:

select * from rdj_tb where yhbh = 0701042082;

We can see the result in the following figure.


```
2013-07-08 00:00:00.0   0701042082      0701481068      2340713 2013-07-08 00:11:55.0
2013-07-07 00:00:00.0   0701042082      0701481068      2340713 2013-07-07 00:11:37.0
2013-07-03 00:00:00.0   0701042082      0701481068      2340713 2013-07-03 00:11:32.0
2013-07-09 00:00:00.0   0701042082      0701481068      2340713 2013-07-09 00:11:39.0
2013-07-04 00:00:00.0   0701042082      0701481068      2340713 2013-07-04 00:11:54.0
2013-07-11 00:00:00.0   0701042082      0701481068      2340713 2013-07-11 00:11:30.0
2013-07-10 00:00:00.0   0701042082      0701481068      2340713 2013-07-10 00:11:43.0
2013-07-12 00:00:00.0   0701042082      0701481068      2340713 2013-07-12 00:11:40.0
2013-07-14 00:00:00.0   0701042082      0701481068      2340713 2013-07-14 00:11:32.0
2013-07-13 00:00:00.0   0701042082      0701481068      2340713 2013-07-13 00:11:51.0
2013-07-15 00:00:00.0   0701042082      0701481068      2340713 2013-07-15 00:11:47.0
2013-07-16 00:00:00.0   0701042082      0701481068      2340713 2013-07-16 00:11:40.0
2013-07-17 00:00:00.0   0701042082      0701481068      2340713 2013-07-17 00:11:38.0
2013-07-18 00:00:00.0   0701042082      0701481068      2340713 2013-07-18 00:11:41.0
2013-07-20 00:00:00.0   0701042082      0701481068      2340713 2013-07-20 00:11:35.0
2013-07-19 00:00:00.0   0701042082      0701481068      2340713 2013-07-19 00:11:44.0
2013-07-21 00:00:00.0   0701042082      0701481068      2340713 2013-07-21 00:11:38.0
2013-07-23 00:00:00.0   0701042082      0701481068      2340713 2013-07-23 00:11:56.0
2013-07-22 00:00:00.0   0701042082      0701481068      2340713 2013-07-22 00:11:36.0
Time taken: 12.605 seconds
hive>
```

Figure 2. Result of the Query

From the result we can tell that the data have been successfully imported

3.2 Comparison and Analysis

To find the influence of the amount of data and the number of the nodes in Hadoop cluster on the efficiency of the query, this paper did some experiments. These experiments are about ‘count’, ‘group by’, and ‘order by’ (three statements in SQL).

The numbers of data in the experiments are 1 million, 3 million, 8 million, 12 million, and 30 million. The number of nodes in Hadoop cluster varies from 1 to 5.

3.2.1 Experiment of ‘count’

The statement of this query is:

select count(*) from rdj_tb;

Time efficiency of the query is showed in the figure below.

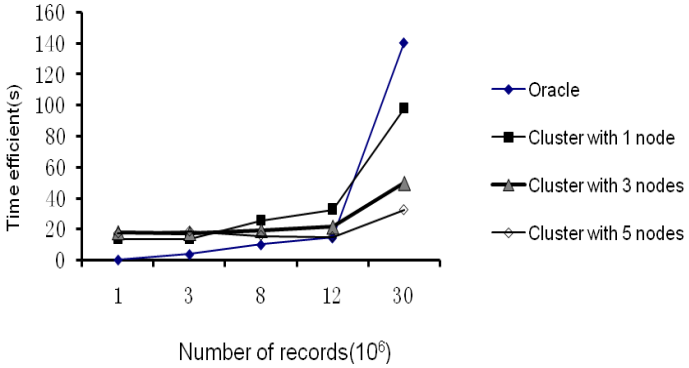

Figure 3. Line Chart of Time in Experiment 1

3.2.2 Experiment of 'group by'

The statement of this query is:

select gdjbh,sum(fl4dn) from rdj_tb group by gdjbh;

Time efficiency of the query is showed in the figure below

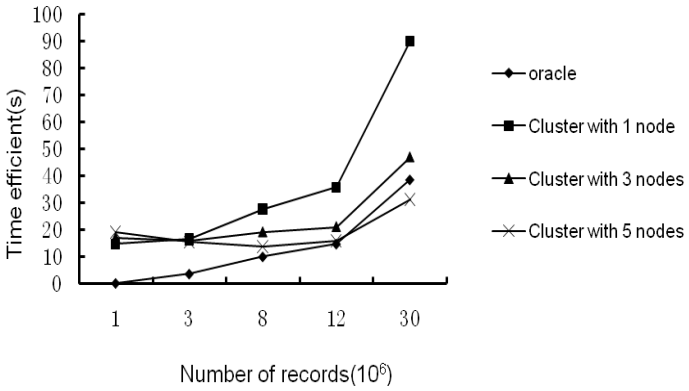

Figure 4. Line Chart of time in Experiment 2

3.2.3 Experiment of 'order by'

The statement of this query is:

select * from rdj_tb order by SHIBIAO;

But when the number of data records is 30 million, Oracle causes a system crash, so that we name its cost of time is infinite. And the rest of the results are showed in the figure below.

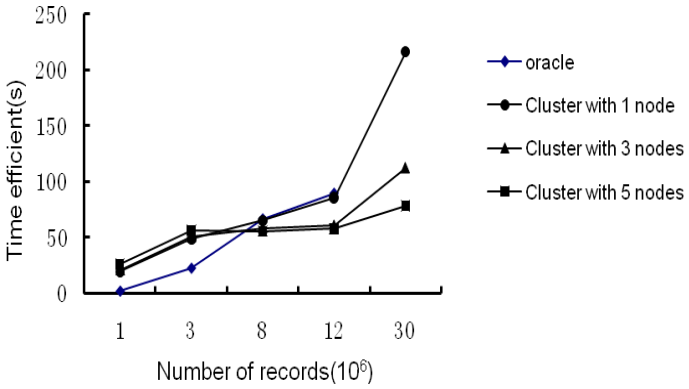

Figure 5. Line Chart of Time in Experiment 3

3.3 Result of the Experiments

From the statistics above, we can see that when the number of the data records is over 3million, the time of each query decreases sharply as the number of the nodes in the cluster. The descent velocity of time falls when the number of nodes rises.For example, when the number increases from 1 to 2, the cost of time is reduced by half. But when the number is 3, the cost of time is reduced by less than 40%. What's more, when the number of nodes is varied from 3 to 5, the cost of time changes even less.

Besides, Hadoop does not show the same advantage in processing small amounts of data as it deals with magnanimity data, because the MapReduce costs a lot of time in the

procedure, and Hive also takes time during the query, such as analyzing SQL statements, dispatching MapReduce tasks.

We come to the conclusion that, when dealing with small amounts of data, the dispatching part of Hadoop and Hive on it costs much more time comparing with relational databases. But when the amount of data is quite huge, the dispatching time is negligible. And under this circumstance, the advantage of Hadoop appears.

3.4 Comparison of Query Efficiency

When the data amount is tiny (for example, 1 million in this experiment), Oracle spends very little time to do the query. But as the number of record increases, the time cost rises sharply. We can see that when dealing with the data whose number of records is 12 million, Hadoop cluster with 5 nodes performs the same as Oracle, or even better when executing 'count' and 'group by' query. In addition, the querying speed of relational database decreases as the data amount increases, but the speed of Hadoop cluster is quite stable. Cloud platform shows a clearly advantage especially in 'order by' query, in which the querying is much faster than relational database if the number of data records is over 8 million.

It turns out that relational databases truly havea very high efficiency when dealing with small amount of data, because relational databases have a lot of optimization over the storage of data, which is better than cloud platforms. But when it comes to magnanimity data, Hadoop platform puts relational databases in the shade. The amount of data is not very large in this experiment, and the performance of Hadoop is as good as relational database Oracle, or even better. It can be clearly seen that even if relational databases support magnanimity data, their efficiency is inferior to cloud platforms.

3.5 Analysis of Results

Through analyzing and comparison above, we can come to the conclusion that the factors that may affect the efficiency of Hadoop and Hive are as follows:

1).After the user submits a MapReduce task, the task will be sent to JobTracker node, which has many TaskTrackers under each node, and waits JobTracker to assign the task.A table is maintained inside JobTracker node to record which TaskTracker nodes are active and which are idle. JobTracker selects idle nodes among them to assign tasks according to the table. During the assignment, the chosen nodes need to copy command into itself first, and then execute the task. In this stage, assigning tasks and copying commands use a plenty of time, especially the copying work.

2).Files in HDFS are stored in different nodes based on the block. So JobTracker may assign the data on one node to other nodes. In this time, the chosen TaskTracker node needs to extract data from other nodes, which also spends a lot of time.

3).MapReduce nodes on Hadoop can only deal with key-value pairs, for example, <key, value>. Before further processing of data in the experiment, texts need to be changed in to key-value pairs.

4).MapReduce model includes the Map stage and the Reduce stage. The nodes in each stage are called Map nodes and Reduce nodes. After Map nodes execute the task, result will be saved on local hard drive. But the nodes that process Reduce work may be other nodes. Thus, it also takes a lot of time for Reduce nodes to extract data over the network.

5).Hive needs to analyze SQL statements, to generate MapReduce tasks. This step also needs some time to finish.

6).When the amount of data is tiny, it will take munch time finish work above. But the processing time of data in each node is very short. So Hive based on Hadoop has a poor efficiency when dealing with small amount of data.

7).Some of the work above needs to be done only one time during the whole query job, such as the analysis of SQL statements, the generation of MapReduce tasks, and the

translation of key-value pairs. When Hadoop deals with magnanimity data, the time to do the whole job is very long. So the time to do the work above is not very much compared to the total time of the job. What's more, Hadoop has a very high parallelism. Therefore Hadoop has a higher efficiency when processing magnanimity data[9][10].

4. Conclusion

By the comparison and analysis of the experiment above, I personally think that to improve the query of structural database on cloud plat forms, following ways can be considered:

1).Choose a suitable algorithm. When JobTracker dispatches TaskTracker, it should choose the closest nodes to process the data. This will reduce the time wasted by network.

2).Consider to build indexing mechanism of data in every block. This will make each block seem like a simple database. This way, when TaskTracker needs to seek information in blocks, it can act like query relational databases. Although it may cost more space to manage information in blocks if we set blocks in HDFS larger, we can make up the waste of space by having higher time efficiency.

3).Develop specific MapReduce model aiming at inquiring relational databases. Runtime will be shortened if we have a specific MapReduce model.

Acknowledgment

This work is supported by the Nature Science Foundation of Heilongjiang Province with the grant number: F201325. The author would like to thank for their support.

References

[1] K. Ullah and M. N. A. Khan, "Security and Privacy Issues in Cloud Computing Environment: A Survey Paper", International Journal of Grid and Distributed Computing, vol. 7, no. 2, **(2014)**, pp. 89-98.
[2] K. H. K. Reddy, D. S. Roy and M. R. Patra, "A Comprehensive Performance Tuning Scheduling Framework for Computational Desktop Grid", International Journal of Grid and Distributed Computing, vol. 7, no. 1, **(2014)**, pp. 149 -168.
[3] "Apache Hadoop", http://hadoop.apache.org., **(2010)**.
[4] K. Zheng and Y. Fu, "Research on Vector Spatial Data Storage Schema Based on Hadoop Platform", International Journal of Database Theory and Application, vol. 6, no. 5, **(2013)**, pp. 85-94.
[5] G. Zhao, "A Query Processing Framework based on Hadoop 261", International Journal of Database Theory and Application, vol. 7, no. 4, **(2014)**, pp. 261-272.
[6] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", Commun ACM, vol. 51, **(2008)**, pp. 107-113.
[7] W. Lang, M. J. Patel, "Energy Management for MapReduce Clusters", PVLDB, **(2010)**, pp. 129-139.
[8] J. Lin and C. Dyer, "Data-intensive text processing with MapReduce", US: Morgan & Claypool, **(2010)**, pp. 1-177.
[9] G. Hu, H. Lin and D. C. Yang, "Marcinkiewicz integrals with non-doubling measures", Integral Equation and Operator, vol. 58, **(2007)**, pp. 205-238.
[10] F. Wen, J. Qiu and Y. Jie, "Hadoop high availability through metadata replication", New York: Suite 701 New York NY USA, **(2009)**, pp. 114-133.

Authors

Jianjun Zhou, He was born in July 12th, 1969. In 1992, graduated from the Computer Science Course of Heilongjiang University with a bachelor's degree and then got his master's degree of Software Engineering in Northeastern University in China.

He is now a member of the School of Information Science and Technology in Heilongjiang University, in Harbin, China. His research interests are database, parallel processing, big data, etc.

