Research on Data Query Optimization Based on SparkSQL and MongoDB

Yujun Chen
College of Computer and Information
Hohai University
Nanjing, China
EthanChen@hhu.edu.cn

Yuansheng Lou, Feng Ye
College of Computer and Information
Hohai University
Nanjing, China
EthanChen911@yeah.net

***Abstract*—With the arrival of the era of big data, the analysis and processing of massive data has become a very critical computing problem. This paper proposes a query optimization method based on SparkSQL and MongoDB. It analyzes the principle and compares it with other literature in order to draw the conclusion. The conclusion shows that when dealing with problems such as interactive SQL queries, the Apache Spark engine can reasonably decompose the tasks based on the dependencies between the massive data, thereby reducing the data query processing time and improving the operating efficiency. Also it is very suitable for storing some simple data with large amount due to flexible query and index of MongoDB. Obviously, the combination of the two can significantly improve the query speed of massive data.**

Keywords-component; massive data analysis; apache spark; mongodb; query optimization

I. INTRODUCTION

With the widespread use of computers and the rapid development of the Internet, data has been growing explosively. How to deal with large-scale data has become a research hotspot [1]. In this context, the traditional data warehouse application has been difficult to meet the OLAP (Online Analytical Processing). Therefore, it is imperative to seek a new type of highly scalable data warehouse and efficiently analyze and process the data. Due to the emergence of Apache Spark which is new big data framework based on in-memory computing and can be deployed on the Hadoop platform, many companies gradually start using the Apache Spark platform. The most important function of the data warehouse is to perform OLAP and provide users with decisions, so the performance of SQL queries is very important. Then big data needs to be stored. MongoDB is a product between a relational database and a non-relational database. Its supported data structure is very loose, so it can store more complex data types. The biggest feature of MongoDB is that the powerful language query and high performance [2].

At present, there are many researches on the query method that combines the big data engine and the database. The literature [3] proposes a Hadoop-based distributed double-level index structure [3] to establish different indexes for different data types. Experiments show that The level of data retrieval is still slow; Literature [4] proposed Hadoop-based interactive big data analysis and query method, mainly to connect the HDFS, Hive and Hbase query test, the limitation is that only Hadoop and database Join queries, no optimization [4]; Literature [5] used Impala's big data query analysis [5], and literature [6] used Hbase for query method [6]. The above methods all have limitations not make deep research. Based on the above work, this paper proposes an optimization method to improve the query speed of massive data, and enhances the availability of the system.

II. RELATED TECHNOLOGY

A. Apache Spark

Apache Spark, developed by the AMP Lab at the University of California, Berkeley in 2009, is a large data parallel computing framework based on in-memory computing. The basic flow of Spark operation is as follows: First, SparkContext is created by the Driver to perform resource application, task allocation and monitoring. The SparkContext constructs a DAG (Directed Acyclic Graph) based on the RDD (Shared Memory Abstraction) dependency. The DAG is submitted to the DAGScheduler for parsing into the Stage, and then each TaskSet is submitted to the TaskScheduler for processing; Executor apply for a task to SparkContext. Task Scheduler Issues Task to Executor and provides application code. The Task runs on the Executor, feeds the results back to the TaskScheduler, and then feeds back to the DAGScheduler [7][8].

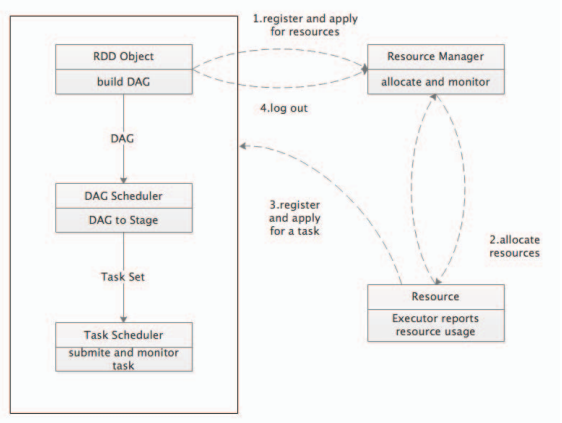

Figure 1. Apache Spark operation basic flow chart

B. MongoDB

MongoDB is written in C++ and is an open source database system based on distributed file storage. It is a

978-1-5386-7445-1/18/$31.00 ©2018 IEEE
DOI 10.1109/DCABES.2018.00046

type of NoSQL database that runs on Unix, Windows, and OSX platforms, supports 32-bit and 64-bit applications [9]. MongoDB is a scalable, high-performance, next-generation database that features high performance, ease of deployment, ease of use, and convenient storage of data.

MongoDB is a product between a balance relational database and a non-relational database. The traditional relational database generally consists of three levels of concepts: database, table, and record. MongoDB consists of three levels: database, collection, and document object. The data format it stores is a collection of key-value pairs. The key is a string, and the value can be any type in the collection of data types, including arrays and document objects. This data format, called BSON, is a binary serialization document similar to JSON [10].

III. OPTIMIZATION

The optimization method of this article includes two parts: paging method of MongoDB and SparkSQL.

A. Where-limit Optimization Paging Method

The query optimization method provided in the MongoDB database is relatively simple. When the system processes a query request, the system selects the fastest query plan to execute, and the skip and limit functions are not considered when executing the query plan. Now database systems all store huge amounts of data. When paging queries are made, if the data that needs to be queried is behind a sorted collection, skipping a large amount of data is required to affect the data query efficiency, so this paper proposes a where-limit optimization paging method in large-scale data sets.

This article divides the where-limit method into four steps:

Step1: According to the query conditions to obtain the value of the keyword field, and store it in the keyword array;

Step2: Use the count function to determine the total number of records in the database, and then calculate the total number of pages to display based on the number of records to be displayed per page;

Step3: Get the keyword array according to the query conditions;

Step4: Paging query by keyword in keyword array.

The where-limit method is better than the skip-limit method because it avoids using the skip function and the system does not need to spend a lot of time to skip large amounts of data [11]. When using the skip-limit method for paging queries, the data offset of the paging (skip function parameter value) is prioritized, but with different offsets, the query time per page will be different: the larger the offset, the longer the query time will be. Where-limit uses a method to find an array of keywords to solve this problem. In the query process, the user only needs to find out the keyword array according to the query conditions, and then decide to skip number of records according to the index of the keywords in the array. The algorithm flow chart is shown in Figure 2.

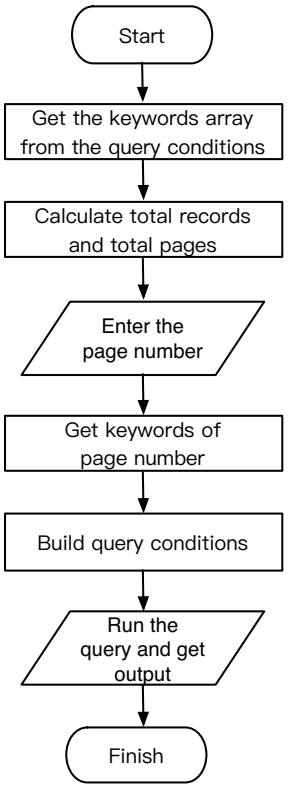

Figure 2. Where-limit optimization method flow chart

B. SparkSQL Optimization

This problem often occurs in SparkSQL: After performing an aggregate operation such as join operation on an RDD, most tasks execute quickly, but individual task execute extremely slowly. For example, there are a total of 100 tasks. 97 tasks are executed within one minute, but the remaining tasks take ten minutes. This situation is called data skew [12] and is very common in SparkStreaming and SparkSQL modules.

For example, the ‘hello’ key, corresponding to a total of seven data on three nodes (as shown in Figure 3). These data will be pulled to handle the same task, and the ‘world’ and ‘yes’ keys correspond to one data. That is, the other two tasks only need to deal with a few of data. The first task at this time may be seven times as long as the other two tasks. The speed of the entire stage is determined by the slowest task.

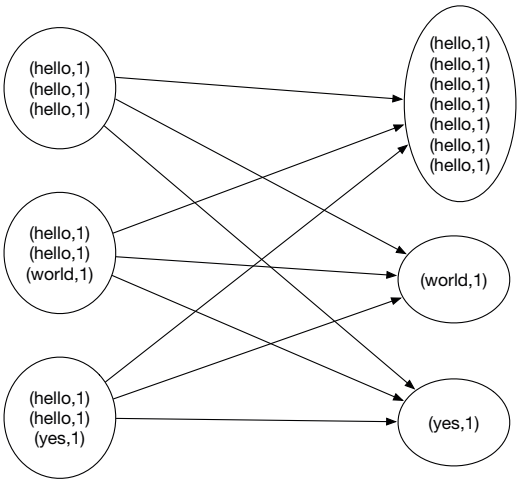

Figure 3. SparkSQL task allocation chart

This situation will only occur in the shuffle process. Generally used operators that trigger the shuffle operation are: distinct, groupByKey, reduceByKey, aggregateByKey, join, cogroup, repartition, and so on. In general, the most straightforward and effective method for data skew is to reduce the number of such operators that trigger the shuffle operation. However, some data operations will still use these operators more or less, so this paper proposes a two-stage aggregation for optimization.

The core idea of this method is to perform two-stage aggregation (as shown in Figure 4). The first step is local aggregation, firstly add a random number to each key, such as a random number within ten, then the original same key becomes different. For example, (hello, 1) (hello, 1) (hello, 1) (hello, 1) (hello, 1) (hello, 1) becomes (1_hello, 1) (1_hello, 1) (2_hello, 1) (2_hello, 1) (3_hello, 1) (3_hello, 1). Then, after adding the random prefix, perform the aggregation operation. The result will become (1_hello, 2) (2_hello, 2) (3_hello, 2). Then remove the prefix of each key and will become (hello, 2) (hello, 2) (hello, 2). Make the aggregation operation again will get the final result (hello, 6).

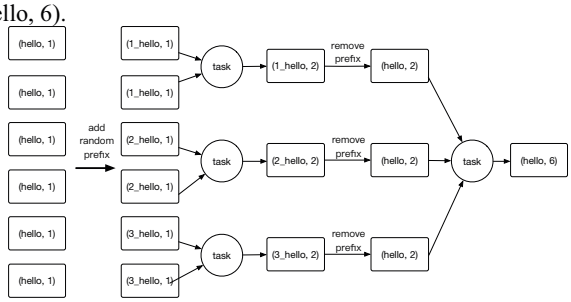

Figure 4. SparkSQL aggregate instance diagram

The method is to change the original key to add a random prefix, so that the data originally processed by one task can be distributed to multiple tasks to perform local aggregation. It can solve the problem of processing massive data of a single task. Then remove the random prefix and perform global aggregation again to get the final result. The advantage of this method is that the data skew caused by the shuffle operation of the aggregation operation is better, and improve the performance of the Apache Spark significantly.

IV. EXPERIMENT

A. Experimental Environment and Datasets

The experimental data set is the real-time water level data set of the Chu River from January 1, 2015 to July 24, 2017, with a total of 18,910,865 records. The experimental environment is a small cluster of three computers. The processor is an AMD Ryzen 7 1700X, the memory is 32G Corsair DDR4 3000Mhz, the hard disk is the Samsung sm961 128g SSD, and the graphics card is ASUS. GeForce GTX 1060.

The database selects MySQL and MongoDB. The Apache Spark version is 2.2.0, MongoDB version is 3.6, MySQL version is 5.7, and the integrated development tool is IntelliJ IDEA.

B. Experimental Analysis

The experiment firstly was conducted by MySQL and MongoDB to import the dataset of Chu river. Then make some read and write operation. Finally, use the IDEA to connect to MongoDB through Spark, and invoke the SparkSQL and where-limit optimization methods to do research.

(1) The time consumed by the two types of database to import the Chu river data set

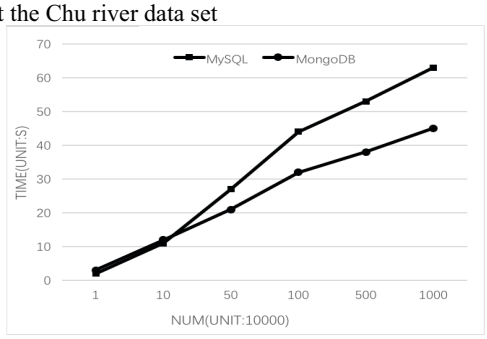

Figure 5. MySQL and MongoDB import data time consuming

When the amount of data is small, the efficiency of the two kinds of databases is almost the same. After the data volume level increases to tem million levels, the advantages of MongoDB become more and more obvious. Due to the multithreading operation recommended by MongoDB official (numInsertionWorkers), in essence, it can split the insert task into multiple threads. As you can see, MongoDB imports data faster than MySQL.

(2) Query the lowest water level detected by each monitoring station

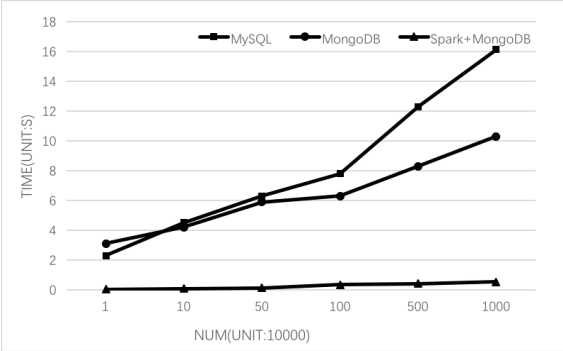

Figure 6. Check the lowest water level detected by each monitoring station time consuming

There are 70 monitoring stations in the data set of this paper. On average, there are 250,000 data in one monitoring station. If the query on stand-alone database with a data size of tens of millions, it will take a long time. However, under the Apache Spark platform and the optimized method, apparently the operating efficiency is very high. Moreover, this data size has not yet reached the Apache Spark performance bottleneck,

This paper also compares the query speeds of other papers. In literature [4], Hbase and SparkSQL were used to query the dataset on the Hadoop cluster platform. The query execution time using the original method in literature [4] and the optimization method proposed in this paper is shown in Figure 7.

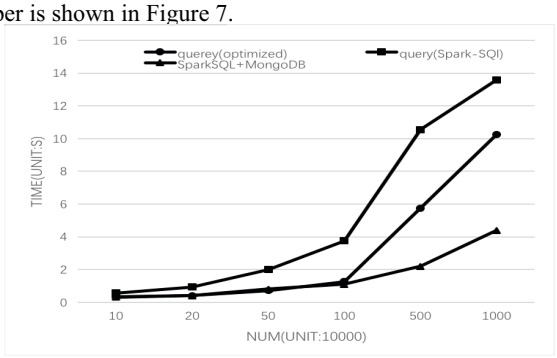

Figure 7. Comparison experiment diagram

It can be seen that the method optimized in this paper has better query speed in this experimental environment and can satisfy the problem of slow query operation under massive data. When dealing with lightweight data, the Apache Spark platform can achieve very satisfactory results based on the advantages of its memory operations. Of course, due to the stability of the system and memory consumption issues, it has limitations when it comes to extreme high-level data.

So the SparkSQL+MongoDB query optimization method proposed in this paper, this method is really helpful for improving the query speed, and can effectively improve the database query efficiency.

V. CONCLUSIONS

This article first introduced the knowledge of Apache Spark and MongoDB, and use optimization methods proposed in this paper to conclude that it can effectively improve the speed of data query. Because data set and hardware configuration conditions are limited, it can not fully utilize the capabilities of Apache Spark. This paper aim to share experimental results and related knowledge for related users who are not yet understood and learning. Obviously, compared to simply using MySQL or MongoDB, the optimization method proposed in this article is a big lead in the query speed. Of course, this method still has limitations, hope to work hard to make improvements in the future.

ACKNOWLEDGMENT

This work was partly financially supported through grants from the National Science and Technology Support Project of China (2013BAB05B00), 2013 Jiangsu Province hydrological science and technology project (2013025), “Six Talents Peak” project in 2017 Jiangsu Province (xydx-078), and the 2017 Jiangsu Province postdoctoral research funding scheme (1701020C).

REFERENCES

[1] S. J. Walker, Big Data: A Revolution That Will Transform How We Live, Work, and Think [C]. Mathematic & Computer Education, 2013, 47(17): 181-183.

[2] J. Dean, and S. Ghemawat. Simplified Data Processing on Large Clusters [M]. ACM, 2008, 51(1):107-113.

[3] J. Feng, W. G. Xu, D. Q. Feng. Hadoop-based Interactive Big Data Analysis Query Processing Method [J]. Computer and Modernization, 2017, (10):29-35.

[4] C. Y. Li, R. G. Wang, X. H. Liang. Hadoop-based Interactive Big Data Analysis Query Processing Method [J]. Computer Technology and Development, 2016, 26(8):134-137.

[5] C. Guo, B. Liu, W. W. Lin. Big Data Query Analysis and Computation Performance of Impala [J]. Application Research of Computers, 2015, (5): 1330-1334.

[6] X. D. Du. Hbase-based Distributed Query Optimization in Big Data Environment [J]. Computer Disk Software and Application, 2014, (8): 22-24.

[7] M. Zaharia, R. S. Xin, P. Wendell, et al. Apache Spark: A Unified Engine For Big Data Processing [J] Communication of the Acm, 2016, 59(11):56-65.

[8] X. R. Meng. Spark SQL: Relational Data Processing in Spark [C]. ACM SIGMOD International Conference on Management of Data ACM, 2015:1383-1394.

[9] Z. Parker, S. Poe, S. V. Vrbsky. Comparing NoSQL MongoDB to an SQL DB [C]. ACM Southeast Conference, ACM, 2013:1-6.

[10] J. S. Zhu, J. X. Wang, J. F. Zhang. Research and Application of Big Data Query Technology in NoSQL Database [J]. China Railway Science, 2014, 35(1):135-141.

[11] H. Li, R. B. Wang. Design Method for Multi-conditional Paging Query Optimization [J]. Computer Engineering, 2010, 36(2):51-52.

[12] Z. Wang, Q. Chen. MapReduce Data Equalization Method Based on Incremental Partition Strategy [J]. Chinese Journal of Computers,2016(1):19-35.