Research on Optimization of Association Rules Algorithm Based on Spark

Chengang Li[a], Yu Liu[b,*], Zeng Li[c]

College of Information Science and Engineering, Guilin University of Technology, Guilin, China

[a]Lca1107@163.com, [b]Lewis_5709@163.com, [c]Lz20508@163.com

Keywords: Association rules; Aprior; Spark; pruning

Abstract: Aiming at the bottleneck of traditional association rule algorithm (Apriori), such as processing speed and computing resources, as well as the problem of accessing disk in the MapReduce computing framework on Hadoop platform. The traditional association rules are transferred to the memory based Spark computing framework, and the optimization algorithm under the framework of Spark is given. By comparing the Apriori algorithm under MapReduce, the algorithm can greatly improve the mining efficiency of the large data association rules. At the same time, the algorithm can reduce the I/O overhead when facing a large number of data. In the cluster, both the extensibility and the acceleration ratio are better than the traditional Apriori algorithm.

1. Introduction

With the advent of the era of big data, the explosive growth of data volume is becoming more and more concerned about the application of mining technology in large data environment. Apriori algorithm is the most classical algorithm in association rules. Its status is very important, but the algorithm's initial serial computing has been unable to meet the requirements of the data era. In today's parallel computing model, many scholars choose a new model of large data mining.

In recent years, with the wide application and open source of Hadoop and Spark, the parallel computing framework of cloud computing is applied to distribute and parallelize traditional algorithms, so as to improve the efficiency of algorithm implementation. It has become the preferred research direction of many scholars. In this paper, the traditional Apriori algorithm is improved on the basis of the Spark framework, and the new improved algorithm can be applied to the spark distributed parallel computing framework. At the same time, Spark based computing kernel RDD is used to store data in a specific data structure, thereby reducing the number of scanned databases[1]. In addition, through the introduction of global pruning, a large number of redundant candidate sets are reduced and the efficiency of the algorithm is further improved.

In view of the shortcomings of the traditional Apriori algorithm, the parallel computing framework based on spark enables the algorithm to deal with the big data effectively, and improves the efficiency of the traditional algorithm and reduces the I/O cost[2]. The remaining contents of this paper are as follows: the second part gives the spark related technologies, as well as the shortcomings of the traditional Apriori algorithm and the improvement strategy. The third part gives the implementation and theoretical proof process of the algorithm. The fourth, fifth part gives the operation and result analysis of the experiment, and makes a prospect for the future work.

2. Research on Spark and Association Rules

2.1 Spark Computing Framework

Spark was born in UC Berkeley AMP lab, which is similar to the MapReduce computing framework in Hadoop, but its performance is better than MapReduce[3]. The Spark computing framework is based on memory, compared to the MapReduce results stored in the disk, and the middle result of the Spark task is stored in memory[4]. This allows the use of the results without the need to read and write HDFS repeatedly. Therefore, it can better use its performance to achieve the iterative algorithm in big data. Its characteristics are as follows[5-7]:

- efficient. The memory computing engine provides the Cache mechanism to support repeated iterations or multiple data sharing, and also reduces the IO overhead of data reading; The unique DAG (Directed Acyclic Graph) engine reduces the read and write overhead from multiple intermediate results to HDFS; multi line task pool boot, reduce unnecessary sort operation in shuffle process and reduce the disk IO operation. Therefore, the calculation speed is 10~100 times faster than that of MapReduce.
- easy to use. Spark provides a wealth of API, supports Java, Scala, Python and R four languages, in addition, the operation on the Spark, the amount of code is less than MapReduce 2~5 times.
- multipurpose. Since Spark is integrated with Hadoop, Spark can be used seamlessly with many tools on Hadoop, providing convenience for developers.

The running process of Spark is shown in Figure 1. The whole process is composed of Spark Context, Cluster Manager, Work Node and Executor. Among them, Driver users write data processing logic; Cluster Manger is responsible for central deployment and resource management; Work Node is responsible for processing Driver commands and reporting status to Driver; Task is the computing unit, and Executor is the executor. The Spark program will create a Spark Context interactive interface, through the Cluster Manager resource scheduling, so that the computing node access to resources, and ultimately perform tasks in Executor.

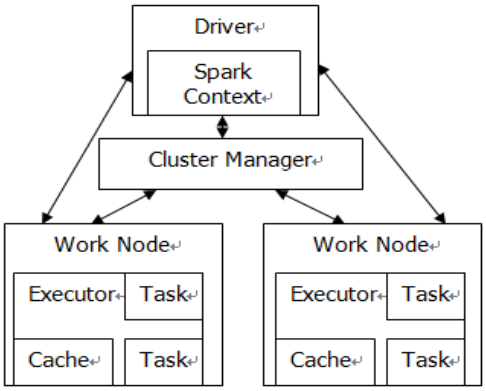

Figure.1. Spark Program Flow Chart

2.2 Improved Apriori Algorithm Based on Spark

Apriori algorithm uses a simple data structure, the execution process is clear, but when the transaction dimension is large and the minimum support is very small, the execution efficiency will decrease. Therefore, there are several problems in the traditional Apriori algorithm [8-10]:

- A large number of candidate items will be generated by the algorithm, which directly results in a large amount of redundancy, which reduces the computational efficiency of the algorithm.
- Repeatedly scanning transaction database, when generating high-dimensional itemsets, we need to scan the database repeatedly, resulting in too much I/O overhead, long computation cycle and low algorithm efficiency.
- The time complexity is high, due to the frequent itemsets one candidate process needs to link and the pruning operation, the time complexity is very high, the efficiency of the algorithm is low.

For second points, an improved strategy is to store a specific data structure to reduce the number of times the database is scanned. Therefore, according to the RDD calculated by the Spark framework, the data storage structure in the Apriori algorithm is transformed into the storage structure shown in Figure 2.

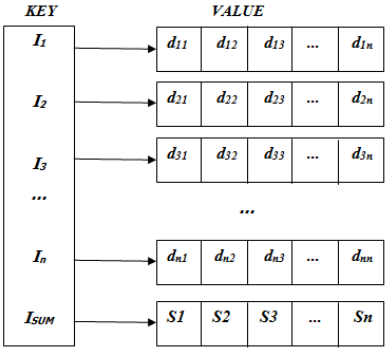

Figure 2. Data structure storage diagram

In the data structure given by Figure 2, when the support degree of statistical candidate set is only needed, support is taken as Key value, then the array of corresponding subscript elements is used to do "join" operation. Finally, the number of non-zero statistics is the support of candidate set. The formula is (1):

$$d_{ij} = \begin{cases} 0, I_i \in T_j \\ 1, I_i \notin T_j \end{cases} S_j = \sum_{i=1}^{|D|} d_{ij} or \left|T_j\right|,$$
$$i \le \left|I\right|, i \le m, m = \left|D\right| \tag{1}$$

In view of third points, the main changes are made as follows.

$$L_{k-1}[i-j] = \{I_i, I_{i+1}, \cdots, I_j\} \tag{2}$$

$$X = \{I_1, I_2, \cdots, I_{k-1}, I_k\} \in C_k \tag{3}$$

$\exists i \in \{1 \le i \le k\}$ makes $\left|L_{k-1}[1-i]\right| < k-1$, and X is not a frequent k item set. $|L_{k-1}[1-i]|$ is the total number of $L_{k-1}[1-i]$.It is proved as follows:

Assuming that X is a frequent k item set, then X can produce k frequent $k-1$ item sets. A i dimension project is taken from the X, and then the $k-1-i$ items are extracted from the remaining items, At this time, there are $C_{k-1}^{k-1-i} = C_{k-1}^{1} = k-1$ options to form a $k-1$ item set. So, when the i project is selected, the number of $(k-1)$ frequent itemsets composed of X is $k-i$, that is, $\left|L_{k-1}[1-i]\right| = k-1$.This is inconsistent with the hypothesis, so X is not a frequent k item set.

3. The Theoretical Proof And Implementation Process Of The Algorithm

3.1 Algorithm Theory Proof

On this basis, combined with the Spark parallel computing framework, the algorithm is optimized. The algorithm has two main steps:

A. produce local frequent itemsets.

B. generate global frequent itemsets.

In the improved algorithm, we first block the data. In order to ensure the correctness, we also define the local minimum support degree to the block. Suppose that the database D_i is divided into m blocks, $i-th$ is $\left|D_i\right|$ data block, and $\min\_sup$ is a global minimum frequent itemset, and then the minimum frequent itemsets for $i-th$ block are:

$$\min\_freq_i = \min\_\sup \times |D_i| \tag{4}$$

The local minimum support is still $\min\_\sup$, that is to say, in the local dataset, when the local frequent itemsets are less than $\min\_\sup$, they are considered as global frequent itemsets and participate in the next stage of pruning operation.

In addition, for database D_i, it is divided into m blocks, and L is a global frequent itemset, so It for any block of data in L needs to be satisfied:

$$\sup(It)_i \geq \min\_\sup_i \tag{5}$$

It is proved as follows:

Suppose:

$It \in L, \exists i \in \{x \mid 1 \leq x \leq m\}, \sup(It)_i < \min\_\sup_i$

Then:

$$\begin{aligned}
&\sup(It)_i < \min_\sup_i \\
&\Rightarrow \frac{count(It / D_i)}{|D_i|} < \min_\sup \\
&\Rightarrow \frac{count(It / D_i)}{|D_i|} < \frac{\min_freq_i}{|D_i|} \\
&\Rightarrow count(It / D_i) < \sum_{i=1}^{m} \min_freq_i \\
&\Rightarrow \sum_{i=1}^{m} count(It / D_i) < \sum_{i=1}^{m} \min_freq_i \\
&\Rightarrow count(It / D) < \sum_{i=1}^{m} \min_freq_i = \sum_{i=1}^{m} \min_\sup \times |D_i| \\
&\Rightarrow count(It / D) < count(It / D)
\end{aligned} \tag{6}$$

It can be seen from the proof that formula (5) is established, so we can launch a set of global frequent itemsets L, which can generate all the local frequent itemsets in m block data, so it is feasible to generate global frequent itemsets through local frequent itemsets.

3.2 The Execution Flow of The Optimization Algorithm on Spark

First, transaction database is partitioned into blocks, each block is defined with minimum support degree, and then allocated to each computing node for local computation to generate local frequent itemsets. The generated local frequent itemsets are aggregated to the global candidate item sets, and then distributed to the statistical global support of each node, thus getting the global frequent itemsets.

The running process is shown in Figure 3:

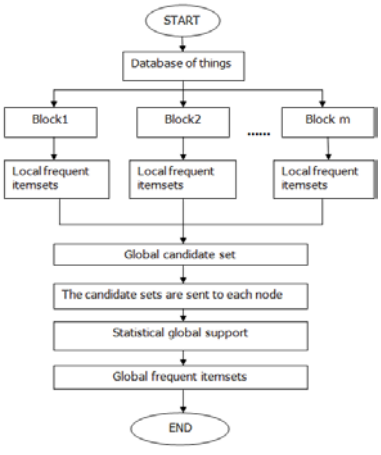

Figure 3. Flow chart of algorithm implementation

4. Experimental Results And Analysit

The hardware experiment platform selected in this paper is a number of models of the same computer. The operation system is CentOS 6.5, Spark version is 2.2.0, the processor is Intel I7 eight core processor, 8G runs memory, 1T hard disk.

The data set used in the experimental data set is the data provided by the Fourth National University of cloud computing application innovation competition. The size of the data is 1.6G, and the average transaction length is 42M, with a total of 2000 Items. They are divided into three databases, 0.3G, 0.5G, and 0.7G, and then tested.

This paper is mainly to achieve the optimization of association rules algorithm based on Spark, so in experiment, we measure the efficiency of the algorithm by comparing the efficiency of traditional MapReduce algorithm. In addition, in the same test environment, we test different data to prove the scalability and speedup of the algorithm. Finally, verify the scalability of the algorithm by testing the same data set with different nodes.

4.1 Experiment A

Under the same configuration of the experimental machine, the same data set is executed in the Spark computing framework and the MapReduce computing framework respectively. Then the time used in each of them is counted respectively. Run the three data, respectively, to get the time for the calculation of the three databases on Spark and MapReduce. The minimum support degree is min_sup=0.3.

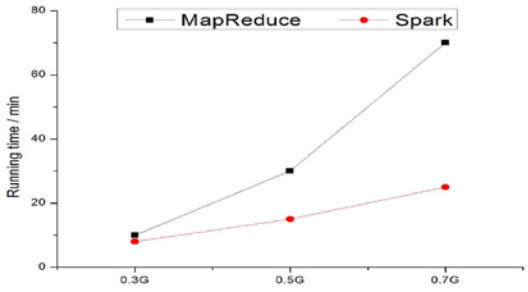

Figure 4. Comparison of running time between MapReduce and Spark

As you can see in Figure 4, the same data is distinctly different on different computing frameworks. For small data, its running time is not very different. But with the increase of data, for MapReduce, it will increase the number of disk access and increase its I/O cost, so it will waste too much time. The calculation efficiency of the Spark framework is significantly lower than that of the traditional MapReduce framework. And with the growth of the data, the calculation efficiency is not a positive proportion, so it shows that the results will be better when the data is larger.

4.2 Experiment B

The purpose of the experiment is to prove the scalability of the optimization algorithm, and to verify the execution time of each database on different nodes by calculating the increase of nodes. Then compare three different groups of databases to verify the scalability of the algorithm.

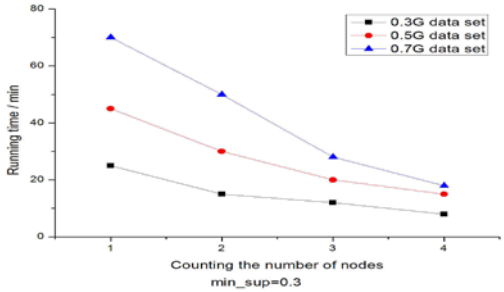

Figure 5. Comparison of operating efficiency under different number of nodes

As you can see from Figure 5, for the three sets of data sets, the algorithm runs down as the number of nodes increases. But from each set of data, the rate of decline of different datasets varies with the increase of nodes. With the increase of nodes, the efficiency of smaller data sets is not obvious. For larger data sets, the increase of their nodes will greatly reduce the running time. It shows that the optimization algorithm has better scalability and scalability on Spark. For the large data age, with the increase of data, the efficiency of the algorithm is obviously improved.

5. Conclusions And Future Works

In this paper, we study the traditional association rule algorithm, apply the traditional association rule Apriori algorithm to Spark, and optimize the data structure and parallel implementation of the algorithm. The experiment shows that the association rule algorithm based on Spark is more efficient than the traditional algorithm. In addition, the algorithm has good scalability and is suitable for a larger amount of data. But the data in the experiment did not reach a larger level of data. The purpose of this paper is to optimize the traditional association rules algorithm, which is suitable for the association rules of large data and processing data faster. In the follow-up study, a larger amount of data will be used to optimize the algorithm proposed in this paper.

Acknowledgement

This research work was supported by Guangxi key Laboratory Fund of Embedded Technology and Intelligent System and The National Natural Science Foundation of China under research project 41264005.

References

[1] Lu Ye, Research on Improved Association Rules Algorithm Based on spark,[D],Taiyuan University of Science and Technology, 2017

[2] LIN, ZENG L, HE Q, Parallel implementation of Apriori algorithm based on MapReduce[C]. 2012 13th ACM International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing, IEEE Computer Society. Kyoto, 2012:236-241

[3] MENG X F, CI X. Big data management: concepts, techniques and challenges [J]. Journal of Computer Research and Development, 2013, 50(1) : 146 - 149

[4] Schafer JB, Dan F, Herlocker J, et al. Collaborative filtering recommender systems. The Adaptive Web, Methods and Strategies of Web Personalizationk, 2010: 46–45.

[5] G. Linden, B.Smith and J.York, “Amazon.com recommendations: Item-to-item collaborative filtering” IEEE Internet Comput., vol. 7,no. 1, pp. 76–80, 2003.

[6] A.Davidson and A.Or, “Optimizing shuffle performance in Spark,” University of California , Berkeley-Department of Electrical Engineering and Computer Sciences, Tech. Rep, 2013

[7] J. Dai, “Experience and lessons learned for large-scale graph analysis using GraphX,” 2015, Spark Summit East.[Online]. Available: https://spark-summit.org/east-2015/talk/ experienceand-lessons-learned-for-large-scale-graph-analysis-using-graphx

[8] Guo Jian. Research on improved Apriori algorithm based on coding and MapReduce[C].2013 10th Web Information System and Application Conference (WISA 2013),IEEE Computer Society.2013:294-299

[9] Qiu Hongjian, Gu Rong, Yuan Chunfeng. YAFIM:A parallel frequent item set mining algorithm with Spark[C].2014 IEEE 28th International Parallel & Distributed Processing Symposium Workshops, IEEE Computer Society,2014:1664-1671.

[10] Mayer-Schonberger V,Cukier K. Big Data: A Revolution That Will Transform How We Live, Work and Think[J].John Murray Publishers,2015,3(1):19-21