Research on Improved Apriori Algorithm based on MapReduce and HBase

Dongyu Feng[1,2] ,Ligu Zhu[1,2] ,Lei Zhang[1,2]
1. College of Computer, Communication University of China
2. Beijng Key Laboratory of Big Data in Security & Protection Industry
Beijing, China
fengdy1225@163.com, liguzhu@cuc.edu.cn, zhanglei@cuc.edu.cn

***Abstract*—In order to improve the efficiency of Apriori algorithm for mining frequent item sets, MH-Apriori algorithm was designed for big data to address the poor efficiency problem. MH-Apriori takes advantages of MapReduce and HBase together to optimize Apriori algorithm. Compared with the improved Apriori algorithm simply based on MapReduce framework, timestamp of HBase is utilized in this algorithm to avoid generating a large number of key/value pairs. It saves the pattern matching time and scans the database only once. Also, to obtain transaction marks automatically, transaction mark column is added to set list for computing support numbers. MH-Apriori was executed on Hadoop platform. The experimental results show that MH-Apriori has higher efficiency and scalability.**

Keywords—Apriori; MapReduce; HBase; data mining; association rules

I. INTRODUCTION

With the rapid development of Internet, the explosive growth of data launched a challenge to IT industry. Traditional association rule mining technology has been unable to meet the needs of big data, thus the integration and improvement of the existing technology has become the attention focus of many scholars. Data mining refers to the process of extracting useful, potential and specific knowledge from a large amount of data[1], which is the most important step in KDD(knowledge discovery[2]), and Apriori is one of the most classic data mining algorithms.

Apriori algorithm was first proposed in 1994 by Agrawal. By repeatedly scanning the database, Apriori algorithm searches candidate item set layer by layer, and finally it iteratively generates frequent item sets. According to the priori property of the Apriori algorithm, all nonvoid subsets of a frequent item set must be frequent[4]. According to this property, the essence of Apriori algorithm can be summarized as follows: it generates frequent item set Lk whose length is the kth scan, and connects items within Lk to generate candidate item set C_{k+1}; during the (k+1)th scan, it counts the number of L_k generating candidate item set C_{k+1} whose length is k+1 until there is no more frequent item set. Therefore each time to find a frequent item set L_k, it needs to scan a full database; besides the amount of candidate item sets is also very large. For instance, a frequent item set with length 1000 $\{x_1,x_2,x_3,\ldots,x_{1000}\}$ will generate 21000 candidate item sets. Thus the time consumption of Apriori algorithm mainly exists in the following 3 aspects:

(1) The number of candidate item sets generated by items connect within frequent item sets is large.

(2) To calculate the support and mine frequent item sets, it is necessary to carry out a number of global scanning of massive database.

(3) The huge time consumption of scanning database for pattern matching between the candidate set and the transaction.

In order to make up the low efficiency of Apriori algorithm, researchers have selected all kinds of technical methods to improve Apriori algorithm. Teseng and Luo[5-6] implemented Apriori algorithm on Hadoop platform based on the MapReduce framework, while in terms of big data these methods increase new load in other aspects, impacting the significantly improve of Apriori algorithm. Cloud computing provides a new method to improve the efficiency of massive data mining. MapReduce, BigTable and GFS are three core technology of Google cloud computing. Some researchers have improved Apriori algorithm based on MapReduce theory[7]. With the efficient search and query function of MapReduce, Apriori algorithm can quickly generate candidate item sets, solving the bottleneck problem (1) above of the Apriori algorithm. Therefore among the improved Apriori algorithm of many researchers, this paper selected Apriori algorithm based on MapReduce for further improvement.

This paper introduced MapReduce framework and HBase database of Hadoop platform to realize the parallel processing of Apriori algorithm. It inherits the advantages of MapReduce model with the three-dimensional attributes of HBase database to improve Apriori algorithm, and solves the bottleneck problem (2) and (3). Finally, the improved algorithm is implemented on the Hadoop platform. The experimental results show that the introduction of HBase database improves the efficiency of the Apriori algorithm to a new level.

II. RELATED RESEARCH

A. Introduction of Classical Apriori Algorithm

Apriori algorithm is one of the most classical algorithms for mining frequent item sets of association rules. According to the priori of frequent item sets, it takes iterative method to search layer by layer, obtaining all frequent item sets generated by transactions via scanning the database[8]. Apriori Algorithm is described as follows.

978-1-4673-9613-4/16/$31.00 ©2016 IEEE

Input: transaction database D, the minmum of the support minsuppport

Output: frequent item sets L in D

Setps:

1.L_1=find_frequent_1-items(D);

2.For(k=2;Lk-1!=∅;k++){

3.C_k=apriori_gen(L_{k-1});//generate candidate k-item set from frequent (k-1)-item set L_{k-1}

4.For each transaction t D{//scan D to count the number

5.countC_t=subset(C_k,t)//obtain the candidate item set

6.For each candidate c∈C_t

7.c.count++;}

8.L_k={c∈C_k|c.count≥minsupport};}

9.return L=L∪L_k;

B. Apriori Algorithm based on MapReduce framework

The thought of MapReduce is "divide-conquer". MapReduce framework partitions large data sets into small data blocks. Map function parallel processing each small data set, and transfers intermediate result to Reduce function. The Map function transforms the input data into key-value pairs <key, value> and sorted by the value of key. The Reduce function merges key-value pairs of the same key, and transfers to Hadoop platform for parallel processing to improve the execution velocity of algorithm.

MapReduce is mainly used to generate candidate item sets. According to the nature of Apriori, two (k-1)-frequent item sets with the same first (k-2) items connect to generate k-candidate item sets. Thus this paper takes first (k-2) items of (k-1)-frequent items as the key, and the last item as the value, these key-value pairs are output by Map function. Combiner function merges key-value pairs with identical key to the same Reduce function, which enables connection operation finished rapidly. The efficiency of this method increases obviously when generate K_3 candidate item set, solving the problem (1) in the introduction[9]. For instance, there are 4 frequent item sets: <A,B,C>,<A,B,D>,<A,B,E>,<A,C,D>, the result from Map function is <<A,B>,C>,<<A,B>,D>,<<A,B>,E>,<<A, C,D >, and MapReduce framework sorts them into <<A,B>,[C,D,E]>、<<A,C>,[D]>. Then it takes these results as input of Reduce function and output result is <A ,B ,C ,D >、<A ,B , C ,E >、<A ,B ,D ,E >, which is the newly generated candidate item set. Because there is no item set with the same first 2 item as <<A ,C>,[D]>, hence no new candidate item set is generated.

C. improvement thought of Apriori algorithm based on HBase database

In big data environment, Apriori algorithm only based on MapReduce, will generate a large amount of <itemset,1> key-value pairs during calculate the support of candidate set, which not only cost much time, but also call for a high requirement to the hardware environment. HBase is a distributed database based on Big-table, which is used for parallel processing of large scale data and also multi-dimensional sort Map. The three dimension are row key, column key and time stamp. In the table of HBase, it can store different versions of each data which are distinguished and indexed by time stamp. The time stamp can be automatically assigned by system or by client. HBase will generate transaction items automatically when generating frequent item sets and candidate item sets. The unique time stamp can replace <itemset,1> key-value pairs of MapReduce, as consequence it needs scan the database only once. And there is no need to return the transaction database for pattern matching, greatly reducing the time consumption of Apriori algorithm.

Meanwhile, HBase table can be divided into Tablets automatically based on the row key, and laod balancing technology enables Tablets be distributed to each node for parallel processing with the obvious improvement of reading efficiency[10]. Through the Wrapper class, as the input and output of MapReduce framework, HBase has efficiently solved the problem (2) and (3) of low efficiency of the Apriori algorithm in the introduction.

III. DESIGN OF IMPROVED APRIORI ALGORITHM BASED ON MAPREDUCE AND HBASE

A. The description of improved Apriori algorithm based on MapReduce and HBase

The algorithm is illustrated with an example of the transaction database D in Table 1, and we set minimum support minsupport＝2.

TABLE I. TRANSACTION DATABASE

TID	Items
1	ACD
2	BCE
3	ABCE
4	BE

Setp 1: Generating candidate 1-item set C_1 shown in table 2. Calculating support and deleting items whose support is less than minsupport, then obtain frequent 1-item set L_1 shown in table 3. The system automatically assigns a time stamp for each record when the data is imported into HBase. In this paper, we take time stamp of each transaction as its transaction code, which is convenient for indexing data with its time stamp. For example, the time stamp of the transaction 1 is 1. When generating candidate 1-item set C_1, MH-Apriori algorithm add a column TID to represent the id of transaction where each item is stored. Therefore when generating new frequent item sets there is no need to scan transaction database D repeatedly to calculate support and pattern matching.

TABLE II. CANDIDATE 1-ITEM SET C_1

Items	Sup	TID
A	2	1,3
B	3	2,3,4
C	3	1,2,3
D	1	1
E	3	2,3,4

TABLE III. FREQUENT 1-ITEM SET L_1

Items	Sup	TID
A	2	1,3
B	3	2,3,4
C	3	1,2,3
E	3	2,3,4

Step 2: generate candidate 2-item set C_2 and frequent 2-item set L_2 based on frequent 1- item sets L_1. Items in table 3 connect each other to generate items of candidate 2-item set C_2 in table 4. Due to HBase assign time stamp to each transaction the same value as the transaction label. Thus when finding which transaction the item exist in, it just need call time stamp index to search for the transaction of the original item set that generated the item set, then the transaction with the same time stamp is what it searches for. Then calculate the number of per item in TID columns in table 4, the result is the support of this item set. For example, item AC in table 4, due to AC is generated by the connection between A and C, then return to table 3 for the TID column of A and C, in which the TID of A is 1,3 and B is 1,2,3. Then it calls time stamp index, then extract transaction with the same time stamp, thus the transaction code of AC is 1,3 and the support is 2. Delete item set less whose support less than minsupport in Table 4, then obtain frequent 2-item set L_2 in table 5.

TABLE IV. CANDIATE 2-ITEM SET C_2

Items	Sup	TID
AB	1	3
AC	2	1,3
AE	1	3
BC	2	2,3
BE	3	2,3,4
CE	2	2,3

TABLE V. FREQUENT 2-ITEM SET L_2

Items	Sup	TID
AC	2	1,3
BC	2	2,3
BE	3	2,3,4
CE	2	2,3

Setp 3: Connection and pruning to generate candidate 3-item set C_3 in table 6 and frequent 3-item set L_3 in table 7. Based on the method introduced above, connect items in table 5 to generate Items columns in table 6. In MapReduce framework, import data in table 5 to Map function, it sets the first item as key, the last item as value and output from Map function. Then MapReduce merges data with the same key to one Reduce function to connect items quickly. After items in table 6 generated, it repeats step 2 to calculate support and transaction item. After pruning, table 7 can't regenerate a new candidate item set, thus the calculation is finished.

TABLE VI. CANDIATE 3-ITEM SET C_3

Items	Sup	TID
ABC	1	3
ACE	1	3
BCE	2	2,3

TABLE VII. FREQUENT 3-ITEM SET L_3

Items	Sup	TID
BCE	2	2,3

B. pseudo code of improved algorithm

Step 2: generate candidate 2-item set C_2 and frequent 2-item set L_2 based on frequent 1- item sets L_1. Items in table 3 connect each other to generate items of candidate 2-item set C_2 in table 4. Due to HBase assign time stamp to each transaction the same value as the transaction label. Thus when finding which transaction the item exist in, it just need call time stamp index to search for the transaction of the original item set that generated the item set, then the transaction with the same time stamp is what it searches for. Then calculate the number of per item in TID columns in table 4, the result is the support of this item set. For example, item AC in table 4, due to AC is generated by the connection between A and C, then return to table 3 for the TID column of A and C, in which the TID of A is 1,3 and B is 1,2,3. Then it calls time stamp index, then extract transaction with the same time stamp, thus the transaction code of AC is 1,3 and the support is 2. Delete item set less whose support less than minsupport in Table 4, then obtain frequent 2-item set L_2 in table 5.

Input: input_data_path，minsup

Output: k_output_data_path

```
Setup()
    k=1;
outputpath=GetFirstFreq(inputdata-path,k,trans_num,minsup);
//Get a frequent item set and store result in outputpath
    candidatesetlist;// Read frequent k-item set
        Map(key1=frequent(k-1)_itemset,value1=support_count)
        P=frenquent(k-1)_itemset;
output(key2=(p[0],p[1],…,p[k-2]),value2=p[k-1]);//Set the first k-2 items as key and the last as value, and output from Map function.
Reduce(key2=(p[0],p[1],…,p[k-2]),iterable(value2)=iterable(value2))
    Item[]=iterable(value2);
    sort(item);//Sort item set by row.
for i:0 to item.length do
    for j:i+1 to item.length do
        itemset=(p[0],p[1],…,p[k-2],item[i],item[j]);
//generate new candidate (k+1)-item set.
                TIDlist=foreach(linkitem in ItemsList)
                {linkitem[i].timetamp=linkitem[j].timetamp}
            suppot_count=count(TIDlistset);//index item with identical time stamp to generate transaction to calculate support.
            k_outputpath(key3=itemset,value3=suppot_count);
    end
end
```

IV. EXPERIMENTAL AND ANALYSIS

A. Experimental environment

This paper built Hadoop cluster in Linux environment, and Hadoop version is 0.20.0. The Hadoop cluster includes 5 nodes, which is 1 master node and 4 slave nodes. The OS version of the node is CentOS 6.3 and JDK version is 1.6.45. Test data selected IBM DataGen data set, and the size of data is 1.8GB, with 2492070 transactions and 53 different items, thereinto the longest transaction includes 42 items. In this paper we implemented the improved algorithm based on MapReduce and Hbase on the data set to conduct experiment. And compare with Apriori algorithm only based on MapReduce, MH-Apriori possesses excellent advantages in algorithm performance.

B. Experimental results and analysis

Figure 1 shows the comparison of time consumption between MH-Apriori parallel algorithm based on MapReduce and Hbase database and original Apriori algorithm based on MapReduce in mining frequent K-item sets. As is shown in figure 1, with the increasing of the number of mining frequent item set, the time consumption of the original Apriori algorithm is rising rapidly far larger than MH-Apriori algorithm's, which reflects the advantage of MH-Apriori algorithm. That is because the original Apriori algorithm will generate a large number of <itemset,1> key/value pairs. As the increase of the number of frequent item sets to be mined, the number of candidate item sets even increases exponentially. Especially in processing massive data it greatly increased the load of MapReduce framework. For instance if the final number of support for a candidate item set is 1000, then the candidate item set will generate 1000 <itemset, 1> key / value pairs and send to Reduce function for merging to get support of 1000. While MH-Apriori algorithm searches data items with identical time stamp to generate transactions, it only needs to scan database once to finish the pattern matching between candidate item sets and transactions, and calculate the support of transaction automatically and significantly improve the algorithm efficiency.

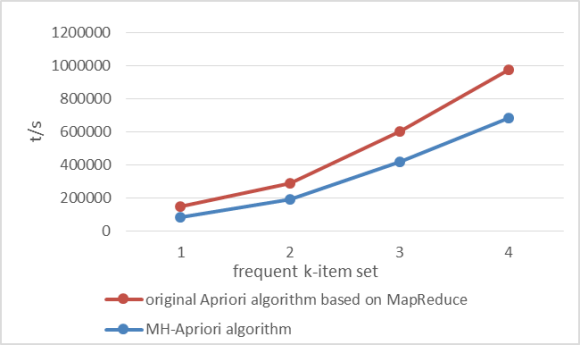

Fig. 1. Time consumption of frequent k-item sets

Figure 2 shows the comparison of time consumption between MH-Apriori algorithm and original Apriori algorithm based on MapReduce in different cluster scale. As is shown in figure 2, the larger cluster scale the less time consumption. That is because HBase will automatically divided data into Tablet according to the line key, with the load balancing function in master node to enable the Tablet can be efficiently and sequentially distributed to different slave nodes, improving the algorithm efficiency. Nevertheless with the increasing of the cluster scale over 6 nodes, here is no much improvement for reducing time consumption. In figure 2, it reflects that under the same cluster scale the time consumption of MH-Apriori algorithm is less than the original Apriori algorithm's, embodying the advantage of improved algorithm.

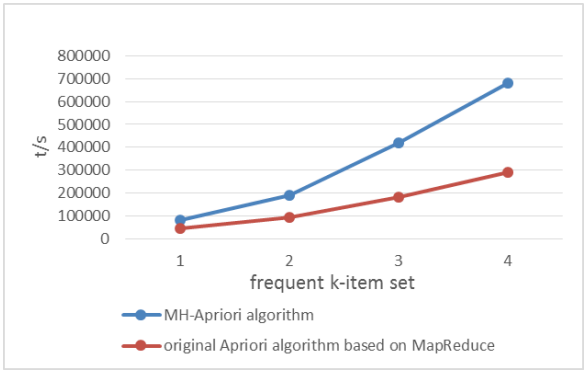

Fig. 2. Time consumption of two algorithms in different cluster scale

V. CONCLUSION

This paper researched the improved Apriori algorithm based on MapReduce framework, and introduced HBase database to design MH-Apriori algorithm based on MapReduce and HBase and implemented the MH-Apriori algorithm in Hadoop cluster. The experiment result shows that MH-Apriori algorithm scan the database only once to finish the pattern matching of candidate item sets, avoiding a great deal of <itemset,1> key/value pairs and improving the efficiency of Apriori algorithm. MH-Apriori algorithm integrates distributed system resources to improve the efficiency of data mining especially suitable for large scale data set, which shows that MH-Apriori algorithm has well research value and applied foreground in the area of big data processing.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation Project 'Key techniques research of continuous data storage system energy-saving'(6137006).

REFERENCES

[1] Han E H,Karypis G, Kumar V' Scalable parallel data mining for association rules[M]. ACM,1997.

[2] Savasere A, Omiecinski E R,Navathe S B. An efficient algorithm for mining association rules in large databases[J].1995, 1-24.

[3] Zhang S, Du Z, Wang J T L. New Techniques for Mining Frequent Patterns in Unordered Trees[J]. Cybernetics IEEE Transactions on, 2015, 45:1113-1125.

[4] Tassa T. Secure Mining of Association Rules inHorizontally Distributed Databases[J]. IEEE Transactions on Knowledge & Data Engineering, 2014, 26(4):970-983.

[5] Tseng V S, Wu C W, Fournier-Viger P, et al. Efficient Algorithms for Mining the Concise and Lossless Representation of High Utility Itemsets[J]. IEEE Transactions on Knowledge & Data Engineering, 2015, 27(3):726 - 739.

[6] Luo X W, Wang W. Improved Algorithms Research for Association Rule Based on Matrix[C]// Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on. 2010:415-419.

[7] Yang S, Xu G, Wang Z, et al. The Parallel Improved Apriori Algorithm Research Based on Spark[C]// 2015 Ninth International Conference on Frontier of Computer Science and Technology (FCST). IEEE Computer Society, 2015:354-359.

[8] Yadav C, Wang S, Kumar M. An approach to improve apriori algorithm based on association rule mining[C]// Computing, Communications and Networking Technologies (ICCCNT),2013 Fourth International Conference on. IEEE, 2013:1-9.

[9] Sun D, Lee V C, Burstein F, et al. An efficient vertical-Apriori Mapreduce algorithm for frequent item-set mining[C]// Industrial Electronics and Applications (ICIEA), 2015 IEEE 10th Conference on. IEEE, 2015.

[10] Li N, Zeng L, He Q, et al. Parallel Implementation of Apriori Algorithm Based on MapReduce[C]// Proceedings of the 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. IEEE Computer Society, 2012:236-241.