MapReduce as a Programming Model for Association Rules Algorithm on Hadoop

Xin Yue Yang
Department of Computer Science and Engineering
University of Electronic Science and Technology of China
Chengdu, China
e-mail: yyyxinyue@163.com

Zhen Liu
Department of Computer Science and Engineering
University of Electronic Science and Technology of China
Chengdu, China
e-mail: quake@uestc.edu.cn

Yan Fu
Department of Computer Science and Engineering
University of Electronic Science and Technology of China
Chengdu, China
e-mail: fuyan@uestc.edu.cn

***Abstract*— As association rules widely used, it needs to study many problems, one of which is the generally larger and multi-dimensional datasets, and the rapid growth of the mount of data. Single-processor's memory and CPU resources are very limited, which makes the algorithm performance inefficient. Recently the development of network and distributed technology makes cloud computing a reality in the implementation of association rules algorithm. In this paper we describe the improved Apriori algorithm based on MapReduce mode, which can handle massive datasets with a large number of nodes on Hadoop platform.**

Keywords-Association rules; Apriori; Hadoop; MapReduce;KDD

I. INTRODUCTION

Known as knowledge discovery in databases (KDD), data mining can automatically extract useful hidden information from the datasets which are massive, noise, and vague. Data mining, as an efficient way to extract the knowledge, has draw attention from both the industrial and research communities. Association rule is one of the important themes and essential of data mining which can find out the relationship between item sets in the database. We can use the interesting association relationships which are extracted among huge amounts of data. But, the discovery of association rule is a direct mass-oriented database system that often has hundreds of properties and millions of records, contains a complex relationship between data tables, and remains a time-consuming process. This will inevitably lead to a great surge in search of dimension and space.

Traditional association rule mining algorithms encounter many difficulties when deal with massive data. For example the traditional Apriori algorithm [13] faces the following difficulties: It spends so much time dealing with particularly large number of candidate sets since candidate (k + 1) itemsets are constructed through the self-join of frequent k-itemsets. Its growth is exponential. To get all frequent sets, it needs to repeatedly scan the database. As Apriori employs an iterative approach, which incurs high I/O overhead for scanning, there are certain limitations regarding in handling of transactions in large database.

Considering above problems, parallel association rule algorithms with high performance are best choice. Currently proposed parallel algorithms are CD (Count Distribution) CaD (Candidate Distribution) and DD (Data Distribution) [3]. These algorithms have major weaknesses at some respects of Communication and synchronization. Now MapReduce model can be used in Machine Learning on Multicore[5] where has many advantages to solve these problems. A key benefit of MapReduce is that it automatically handles failures, hiding the complexity of fault-tolerance from the programmer [11]. In this paper we use the MapReduce model which provides a parallel design pattern for simplifying application developments in distributed environments to improve the Apriori algorithm as to obtain a better parallel performance. This model can split a large problem space into small pieces and automatically parallelize the execution of small tasks on the smaller space [6]. We use it to reduce the communication overhead without having to take into account the synchronization operations between nodes. The contribution of this paper includes:

- An improved parallel Apriori was proposed. We calculate the count of the candidate items based on Mapreduce framework to achieve parallelism which significantly improved the efficiency. After the redesign of map and reduce function, through the iterations of invoking map and reduce function, then gets frequent itemsets.
- Using Apache's MapReduce implementation –Hadoop, we have compared the performance of the improved parallel Apriori algorithm on the Hadoop platform using different nodes.

II. BACKGROUND: ASSOCIATION RULES, MAPREDUCE AND HADOOP

A. Association Rules

Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of items. Let $DB = \{t_1, t_2, \ldots, t_n\}$ be a database of transactions, where each transaction t_i is a set of items such that $t_i \subseteq I$. Given an item set $X \subseteq I$, a transaction T

contains X if and only if $X \subseteq T$. An association rule is an implication of the form X=>Y, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \varnothing$ .The Support and Confidence are common indicators to measure strength of association rules. To select interesting rules from the sets of all possible rules, constraints on various measures of significance and interest can be used. The best-known constraints are minimum thresholds on support and confidence. The support supp(X) of an itemset X is defined as the proportion of transactions in the dataset which contain the itemset. The confidence of a rule is defined as $conf(X \subseteq Y) = \sup p(X \cup Y) / \sup p(X)$ [4].

A user-specified minimum support and a user-specified minimum confidence are needed in the Association rules at the same time. It consists of two steps. First, minimum support is used in order to find all frequent itemsets in a database. In a second step, applying the minimum confidence and these frequent itemsets, we can form rules. The first step should be played more attention to improving as the second step is straight forward. Apriori is applied to count the support of itemsets with a breadth-first search strategy and generate candidate itemsets to accomplish the first step.

B. MapReduce

As a data parallel model, MapReduce is a patented software framework introduced by Google to support distributed computing on large datasets on clusters of computers [1]. Known as a simple parallel processing mode, Map-reduce has many advantages: such as, it is easy to do parallel computation, to distribute data to the processors and to load balance between them, and provides an interface that is independent of the back-end technology [10]. MapReduce is designed to describe the process of parallel as Map and Reduce. The user of the MapReduce library expresses the computation as two functions: Map and Reduce [2].

Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function.

The Reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user's reduce function via iterator. This allows us to handle lists of values that are too large to fit in memory.

The Map and Reduce functions are both defined with respect to data structured in (key, value) pairs. MapReduce provides an abstraction that involves the programmer defining a “mapper” and a “reducer”, with the following signatures [9]:

Map::(key1) ⇒ list(key2,value2)

Reduce::(key2,list(value2) ⇒ list(value2)

C. Hadoop

Hadoop is a popular open source implementation of MapReduce, a powerful tool designed for deep analysis and transformation of very large data sets which is inspired by Google’s MapReduce and Google File System [2]. It enables applications to work with thousands of nodes and petabytes of data. Hadoop uses a distributed file system called Hadoop Distributed File System (HDFS) which creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable reliable, and has extremely rapid computations to store data as well as the intermediate results [6]. Hadoop schedules map and reduce tasks to distributed resources, which handles many tough problems, including parallelization, concurrency control, network communication and fault tolerance [8]. Furthermore, it performs several optimizations to decrease overhead involved in the scheduling, network communication and intermediate grouping of results.

III. MAPREDUCE FOR APRIORI

A. Description of the traditional Apriori

Apriori is a traditional algorithm for association rule and designed to operate on database containing transactions. The main idea of Apriori is producing candidate items, and then scanning the database so as to decide whether they meet the count. It use a "bottom up" approach, frequent subsets are extended one item at a time (a step known as candidate generation), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found. It can be shown as follows:

Input: D, (Database of transactions), min_sup (minimum support threshold),

Output: L, (frequent itemsets)

Method:

(1) L_1 =find_frequt_1-itemsets(D);

(2)for(k=2; L_{k-1} != $\varnothing$;k++){

(3) C_k = Apriori_gen (L_{k-1} ,min_sup);

(4) for each transaction t$\in$ D{//scan D for counts

(5) C_t =subset(C_k ,t); //get the subsets of t

(6) for each candidate c$\in$ C_t

(7) c.count++;

(8) }

(9) L_k ={c$\in$ C_k |c.count>min_sup}

(10) }

(11)return L=L$\cup_k$ L_k ;

The function of Apriori_gen(L_{k-1} ,min_sup) is presented in the paper [7].

B. MapReduce for Apriori

The Apriori consists of two steps: the first is generating all frequent itemsets, the second is generating confident association rule from the frequent itemsets. We deploy MapReduce to parallel the first step and have the data stored in the file in order to deal with in the Hadoop Distributed File System (HDFS).

We use Hadoop components to perform job execution, file staging and workflow information storage and use the files replace the database to store datasets. In the files each line can be seen as a transaction, each item is separated with white space (a space or Tab). The datasets in files are split into smaller segments automatically after stored in HDFS and the map function is executed on each of these data segments. At first, the candidate sub item are put out with the counts number after the execution of map function, then the frequent itemsets are found after the execution of map function.

Input/output	Map function	Reduce function
Input: ***key/value pairs***	***Key:Line No.; Value：one row of data***	***Key: candidate item sets;*** ***Value:1***
Output : ***key/value pairs***	***Key: candidate itemsets;*** ***Value:1***	***Key:*** ***frequent subitems;*** ***Value:*** ***support***

Fig.1. key/value pairs for map function and reduce function

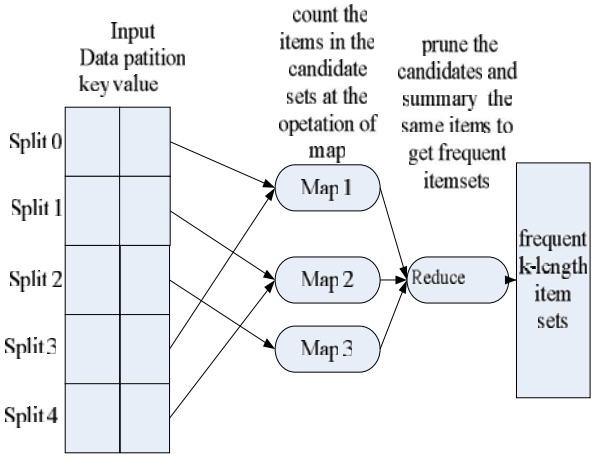

Fig.2. Dataflow of parallel Apriori

The definition of key/value is the first step to implement the function of MapReduce. The pairs of key/value are shown in Fig.1, using the key/value pairs, we can deign the simple map function and reduce function [12]. The map function mainly collects the count of every item in candidate itemsets and the reduce function prunes the candidate itemsets which have an infrequent sub pattern. Each frequent item is generated through one execution of map and reduce function. After stored in HDFS, the datasets are split into smaller segments and then transformed to datanodes. Map function is executed on these data segments and produces <key, value> pairs for every records it encounters. The framework groups all the pairs, which have the same item and invokes the reduce function passing the list of values for candidate items. The reduce function adds up all the values and produce a count for the candidate item as a one-time synchronization. This algorithm's advantage is that it doesn't exchange data between data nodes, it only exchanges the counts. In every scan, each map function generates its local candidate items, then the reduce function gets global counts by adding local counts. Fig.2 shows the data flow and the phases of the MapReduce framework for the improved Apriori algorithm. Multiple iterations of MapReduce computations are necessary for the overall computation. An iteration of the MapReduce produces a frequent itemset. The iteration continues until there aren't frequent itemsets further. Parallel Apriori algorithm is shown as follows:

Input: D(data in HDFS), min_sup(minimum support threshold),

Output: L, frequent itemsets

Method:

(1) L_1 =find_frequt_1-itemsets(D);

(2)for(k=2; L_{k-1} != $\varnothing$;k++){

(3) C_k =candidate_gen(L_{k-1} ,min_sup);

(4) for each row of data t$\in$ D{//scan D for counts

(5) C_t = map();//get the subset of the candidate itemsets

(6) }

(7) L_k =Reduce();// get the subset of the frequent itemsets

(8) }

(9)return L=L$\cup_k L_k$;

IV. EXPERIMENTS

In our experiments, we used Hadoop version 0.20.0, running on a cluster with 9 machines (1 master, 8 slaves). Each machine has two single-core processors (running at 2.60GH) and 4GB memory. The experiments datasets generated by IBM Data Generator are changed in the form of transactions as in the database.

We use speedup as criterion to measure our algorithm's superiority. When more than one node is used, computation and transfer is performed in parallel, and the speedup is over one. In the case of more than one data node, speedup increases with the increase of number of data in certain circumstances, which can prove the high efficiency of the parallel Apriori based on MapReduce. Dataset also mainly determines the performance of the parallel algorithm. If we experiment with the small datasets, its performance turned out to be lower for the reasons; extra communication time occupying a large proportion compared to the total execution time. This is easily predicted from experiments where we noticed that the more data a node processes, the less significant proportion becomes

the communication time. The opposite effect is simply seen here, larger datasets would have shown even better speed up characteristics. Clearly the parallel algorithm scales very well, being able to keep speedup almost larger as the datasets and data nodes sizes increase as show in Fig1. Like the CD algorithm, the algorithm's communication overhead is O (|C| ·n) at each phase, where |C| and n are the size of candidate itemsets and number of data sets, respectively [3].

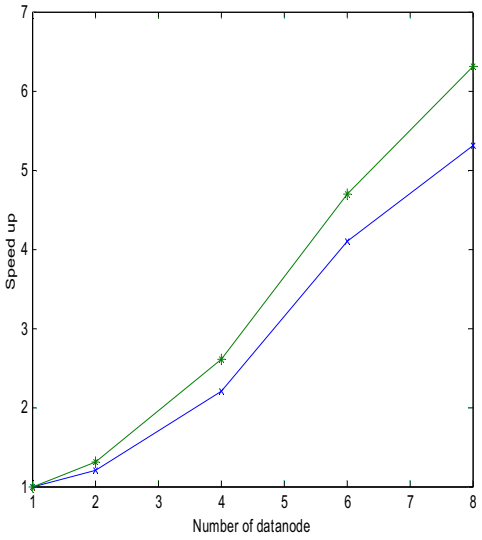

Fig. 3. speedup of parallel Apriori (Blue:100,000 transactions, Green:1000,000 transactions,) support = 1%).

V. CONCLUSION

In this paper, we mainly address the challenge of using the MapReduce model to parallelize Apriori. Hadoop, the underlying cloud computing framework, can handle many tough problems, including parallelization, concurrency control, network communication and fault tolerance. The novelty of the algorithm lies in the ability of parallel Apriori to take full advantage of what Hadoop can provide. It can be easily applied to many commodity machines to deal with mass data without consider the synchronization problem.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of the National Natural Science Foundation of China (60903073).

REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org/

[2] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters, ACM Commun, vol. 51, Jan. 2008, pp. 107-113.

[3] Ashrafi, M.Zaman, T.David, S.Kate. ODAM. An optimized distributed association rule mining algorithm. IEEE Distributed Systems Online.

[4] Association rule learning. http://en.wikipedia.org/wiki/Association_rule_learning

[5] J.Ekanayake, S.Pallickara, G.Fox. Map-Reduce for Machine Learning on Multicore. IEEE International Conference on In eScience, 2008.

[6] J.Ekanayake, S.Pallickara, G.Fox. MapReduce for data intensive scientific analyses. Proceedings - 4th IEEE International Conference on eScience.

[7] Zhang C.S, Li Z.Y, Zheng D.S, An Improved Algorithm for Apriori. Fourth. Proceedings of the 1st International Workshop on Education Technology and Computer Science, ETCS 2009, v 1, p 995-998, 2009.

[8] C.Jin, C.Vecchiola, R.Buyya. MRPGA: An Extension of MapReduce for Parallelizing Genetic Algorithms.Fourth IEEE International Conference on eScience 2008.

[9] T.Elsayed, J.Lin, and Douglas W. Oard. Pairwise Document Similarity in Large Collections with MapReduce. Proceedings - 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009.

[10] J.H.C. Yeung, C.C. Tsang, K.H. Tsoi, B.Kwan, C. Cheung, A.P.C. Chan, P.H.W.Leong. Map-reduce as a Programming Model for Custom Computing Machines. Proceedings of the 16th IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM'08.

[11] M.Zaharia, A.Konwinski, A. D. Joseph, R. Katz, I. Stoica. Improving MapReduce Performance in Heterogeneous Environments. EECS Department University of California, Berkeley Technical Report No. UCB/EECS-2008-99 August 19, 2008.

[12] B. Panda, J.S. Herbach, S.Basu, R.J. Bayardo. Massively Parallel Learning of Tree Ensembles with MapReduce http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.149.4871.

[13] R.Agrawal and R.Srikant. Fast Algorithms for Mining Association Rules. In Proc.of the 20th Intl. Conf. on Very Large Data Bases(VLD'94).