The Parallel Improved Apriori Algorithm Research Based on Spark

Shaosong Yang
College of Computer and Information
Hohai University
Nanjing, China
489271346@qq.com

Guoyan Xu
College of Computer and Information
Hohai University
Nanjing, China
gy_xu@126.com

Zhijian Wang
College of Computer and Information
Hohai University
Nanjing, China
zhjwang@hhu.edu.cn

Fachao Zhou
College of Computer and Information
Hohai University
Nanjing, China
790428547@qq.com

***Abstract*—Apriori algorithm is one of the classical algorithm in the association rule mining field, this paper analyzes the shortcomings of classical Apriori algorithm, then improves it by constructing a new data structure and optimizing the pre-pruning step. Based on the improved Apriori algorithm and combined with the Spark support for fine-grained data processing, we elaborate the idea of the improved Apriori algorithm's parallel processing, and propose the SIAP algorithms. We experimented by comparing with the Apriori algorithms based on Hadoop and the Apriori algorithms based on Spark, and the results show that the SIAP algorithm has a higher efficiency.**

Keywords—association rule; Apriori; Spark; parallel; efficiency

I. INTRODUCTION

Association reflects the dependencies or the associations between things, and association rule mining is one of the important research directions in the field of data mining. It tries to find the implicit or interest relationship in the data set, and the result is often represent in the form of frequent itemsets or association rules. Apriori algorithm [1] is one of basic algorithms in association rule mining, and its main idea is use the known low-dimensional frequent itemsets to deduce high-dimensional frequent itemsets.

With the development of the Internet, the amount of data is increasing rapidly, the traditional Apriori algorithm is unable to meet the demand. In this case, distributed processing model shows its great advantages. In the distributed computing frameworks, MapReduce model is the leading solution for distributed processing, Hadoop is an efficient open source framework, which contains the MapReduce model. The algorithm based on Hadoop solves the problem to a certain extent, but Hadoop need to save intermediate results to HDFS, thus its execution processing speed is limited; at the same time, Hadoop is also lack of support for iterative operations. UC Berkeley AMP lab proposed a memory-based parallel computing framework Spark for large data in 2009. Since the calculation is based on memory, so it can improve the real-time data processing greatly at the same time, it can also ensure the cluster's high fault tolerance and scalability in the environment of big data, and applies to the occasions that need iteration computing. This paper analyzes the shortcomings of Apriori algorithm, then improves the algorithm by establishing the mapping model and improving the efficiency of pruning, at last, we propose the SIAP algorithm based on improved Apriori algorithm and combine with the Spark's features.

This research is partially supported by the Project of Water Science and Technology of Jiangsu Province under Grant No. 2013025; the China Huaneng Technology Project under Grant No. HNKJ13-H17-04; the Natural Science Foundation of Jiangsu Province under Grant No. BK20130852;

II. RELATED WORK

Apriori algorithm is a classical algorithm in the association rule mining field, the main idea is to use a low-dimensional frequent itemsets to obtain high-dimensional frequent itemsets through iterative step by step. However, the algorithm needs to scan the database frequently, which results in the I/O overburdened; the process that producing k-frequent itemsets by the (k-1)-frequent itemsets is too large, which leads to low efficiency; the algorithm does not remove the transaction when the database contains transactions that needn't scan. Many scholars have been improved and optimized the Apriori algorithm from different angles by now. The DHP algorithm proposed by Park et al. [2] adopted the idea of dynamic hash hashing algorithms and pruning algorithm, and exclude transactions that can't generate frequent itemsets when traversing the database, then improved the efficiency of mining frequent itemsets. Brin proposed a dynamic itemsets counting algorithm DIC [3], which can effectively reduce the frequency of scanning the database. This algorithm divides the transaction database into the same size, and accesses the data blocks in turn to generate the 1-frequent itemsets then generate the candidate 2-frequent itemsets by self-join, at last it merges the 1-frequent itemsets and candidate 2-frequent itemsets that each block generated, then repeat it until there is no new itemsets or reach the limit.

The algorithms above can work well when the amount of data is small and the dimension is not high, but in the environment of big data or the dimension is high, these algorithms can't work well. Google came up with the MapReduce [4] framework in 2004, then Hadoop based on MapReduce becomes popular, and cluster-based parallel data mining gains more attention, research and application. A lot of classical data mining algorithms are parallelized on Hadoop. Based on the Apriori algorithm, Lin et al proposed several parallel algorithms SPC, FPC and DPC [5] based on MapReduce, the SPC algorithm assign the data set to all Map nodes, and execute mining operation in parallel, then in the reduce phase execute combining operation, the algorithm starts the Map and Reduce tasks only once, however the FPC and DPC algorithms require start MapReduce tasks repeatedly, determined by the dimension of frequent itemsets mining;

Li et al proposed a parallel frequent itemsets mining algorithm PApriori [6], in the Map stage scan the transaction database to count candidate frequent itemsets, and perform statistical operations to get frequent itemsets in the reduce phase, but it also need to repeatedly start MapReduce tasks. S. Hammoud proposed a parallel method [7], in each round of the iterative process, assigning slice datasets to each Map node and statistical candidate frequent itemsets, then merging to get frequent itemsets in the Reduce phase.

Because of the MapReduce framework's high latency and lack of iteration, Apriori algorithm doesn't fit the MapReduce framework well. Spark is a memory-based parallel computing framework, and it can greatly improve the real-time data processing and ensure the cluster's high fault tolerance and high scalability [8] in big data environments. Qiu H et al introduces a parallel Apriori algorithm based on Spark YAFIM [9], and the paper realized the classical Apriori algorithm in Spark environment, and the results showed that the algorithm based on Spark has greatly improved in performance than the algorithm based on Hadoop.

III. Design of SIAP Algorithm

A. Apriori Algorithm and Improvement

Apriori algorithm uses a simple data structure, and the execution process is clear, but when the dimension of affairs is large and the minimum support is small, it will reduced the execution efficiency greatly. Specifically, the shortcomings of Apriori algorithm mainly manifests in the following several aspects:

1) Each time to generate the higher dimensional itemsets there need to scan the transaction database global, when the transaction database is huge, it will lead to the I/O burden too heavy, huge amount of calculation cycle and the low efficiency of the algorithm.

2) In the process of generate candidate itemsets there need to go through the self-connection and pre-pruning steps, but both of the two operation's time complexity are high, making the low efficiency of the algorithm.

3) With the execution of the algorithm, there are a lot of transactions that don't need to be scanned in the transaction database, but Apriori algorithm doesn't remove them, which directly lead to the calculation contains a lot of redundancy, makes the algorithm less efficient.

For the problem (1), our improvement strategy is to reduce the number of scan the database to once, by the method of map the database transactions to memory through a special data structure, according to the data structures [10], this paper proposes the following storage structure, shown in Figure 1

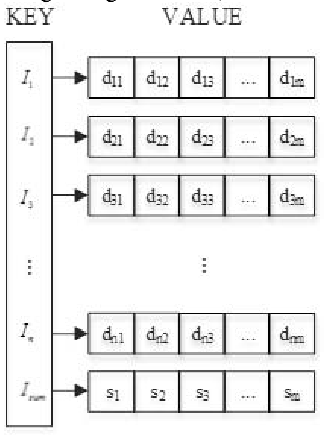

Fig. 1. Data structure to store the transaction database

$$d_{ij} = \begin{cases} 0 & I_i \in T_j \\ 1 & I_i \notin T_j \end{cases} \quad s_j = \sum_{i=1}^{|n|} d_{ij} \text{or} \, |T_j| \quad i \le |I|, j \le m, m = |D|$$

Where the key I_i represents the items that appear at least once in the transaction database besides I_{sum}, each value of the key is a one-dimensional array, d_{ij} represents whether T_j contains I_i and s_j represent the number that T_j contains in the transaction database.

By using the data structure, we can avoid repeatedly scan the database, and reduce the time complexity greatly in the pruning stage. When statistic the frequency of a candidate sets C_{temp}, we only need to take out all items of C_{temp} and treat them as the key, then fetch out the corresponding array, and do "and" operation for the same subscript elements, finally statistics the number of non-zero which is the frequency of C_{temp}.

For the problem (2), the literature [11] proposed that pruning before the item self-joins, this paper expands it and proposes the following pre-pruning idea:

Property 1 Let $L_{k-1}[i-j]$ be a (j-i)-dimension set which is comprised of $I_i, I_{i+1}, ..., I_j$, if C_k which is generated by the $L_{k\text{-}1}$ contains a k-dimension itemset $X = \{I_1, I_2, ..., I_{k-1}, I_k\}$ and exist $i \in \{i \mid 1 \le i \le k\}$ makes $|L_{k-1}[1-i]| < k-i$ to be true, then X is not a k-frequent itemset. Where $|L_{k-1}[1-i]|$ represents the number of $L_{k-1}[1-i]$.

Proof: Suppose X is a k-frequent itemset, then X can be divided into k (k-1)-frequent itemsets. First we take i items $I_1, I_2, ..., I_i$ from X, then take $k-1-i$ items from the remain. There are $C_{k-i}^{k-1-i} = C_{k-i}^{1} = k-i$ choices to compose the (k-1)-frequent itemsets. After we fix i items, the number of (k-1)-frequent itemsets constructed by X is $k-i$,that is $|L_{k-1}[1-i]| = k-i$ which contradicts the assumption, so X is not a k-frequent itemset.

B. Design of IPA Parallel Algorithm

The traditional algorithm runs on single node, the data and calculations are locally, but in the distributed environment, the data and computing are distributed, so produce the communication overhead between nodes, the common practice is to reduce the transmission of data between nodes, by distributing the computing to each node. This paper also uses the idea, parallel the proposed improved algorithm Apriori (IAP) using "transport operator" rather than "moving data".

In the Spark environment, in addition to coarse-grained global scan data, the fine-grained local data scanning operation can be well supported. To improve the performance of the proposed algorithm IAP, this paper combine the Spark's fine-grained computing model's characteristic, strike the iterative operation of frequent itemsets into the following two phases:

1) Generate local frequent itemsets. Cut the transaction database into n blocks horizontally, and distribute to m nodes, run the IAP algorithms n times on m nodes, find all the local frequent itemsets. Running times of each node is determined by the number of nodes on the data block.

2) Generate global frequent itemsets. Merge the itemsets that m nodes generated, and as a global candidate set, then through a global data scan obtained support for each candidate set, by comparison with the minimum support obtain the final global frequent itemsets.

Operations that generate global frequent itemsets are shown in Figure 2.

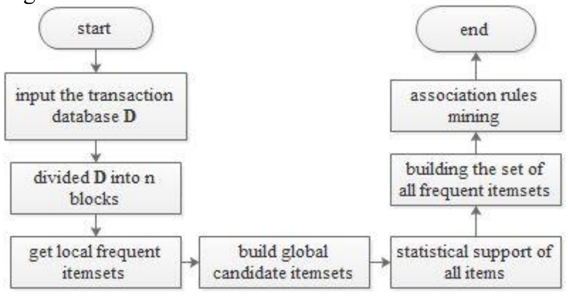
start
input the transaction database D
divided D into n blocks
get local frequent itemsets
build global candidate itemsets
statistical support of all items
building the set of all frequent itemsets
association rules mining
end

Fig. 2. Process of generating global frequent itemsets in Spark environment

In order to ensure the union set of local frequent itemsets is a superset of global frequent itemsets and the correctness of the algorithm results, we propose the following definition and nature:

Define 1: local minimum frequency: Assume that the transaction database D_i is divided into m blocks, the i-th contains $|D_i|$ information, and the global minimum support is min_sup, the i-th data blocks local minimum frequency definition as follows

$$min\_freq_i = min\_sup \times |D_i| \tag{1}$$

By formula 1, local minimum support remains min_sup, that is, in the local data block those local frequent itemsets which support are not less than min_sup are deemed candidate set of global frequent itemsets, and participating in the second stage global pruning.

According to the definition of the local minimum frequency, we propose the following property:

Property 2 Assuming the transaction database D contains n information, and divided into m pieces of data, the piece of i contains $|D_i|$ information, L is the global frequent itemsets set on the transaction database D, then for any itemsets It in L, at least one block of data make

$$\text{Support}(It)_i \ge min\_sup_i \tag{2}$$

Set up.

Proof: Suppose there is a frequent itemset $It \in L$, when $i \in \{x \mid 1 \le x \le m\}$, $\text{Support}(It)_i < min\_sup_i$ set up. That is on all of the data block, $\text{Support}(It)_i < min\_sup_i$

$$\text{Support}(It)_i < min\_sup$$

$$\Rightarrow \frac{count(It / D_i)}{|D_i|} < min\_sup$$

$$\Rightarrow \frac{count(It / D_i)}{|D_i|} < \frac{min\_freq_i}{|D_i|}$$

$$\Rightarrow count(It / D_i) < min\_freq_i$$

$$\Rightarrow \sum_{i=1}^{m} count(It / D_i) < \sum_{i=1}^{m} min\_freq_i$$

$$\Rightarrow count(It / D) < \sum_{i=1}^{m} min\_freq_i = \sum_{i=1}^{m} min\_sup \times |D_i|$$

$$\Rightarrow count(It / D) < count(It / D) \tag{3}$$

The (3) is self-contradiction, so the assumption is false, the property 2 is proved.

According to property 2 we can have the following inference:

Corollary 1 Global frequent itemsets L is a subset of the set of all local frequent item collection generated by m block of data.

Therefore, first produce the fine-grained local frequent itemsets on Spark node, then merge the local frequent itemsets, and then generate global frequent itemsets through the global scan the transaction database is feasible.

C. Implementation of SIAP

Algorithm implementation process is divided into two stages: generating local frequent itemsets; local frequent itemsets global pruning. Combine the characteristics of Spark, use the IAP algorithm to generate local frequent itemsets, in the global pruning stage by scan the transaction database to obtain statistical support information and complete pruning to get the final global frequent itemsets. Specific implementation process is shown in Figure 3:

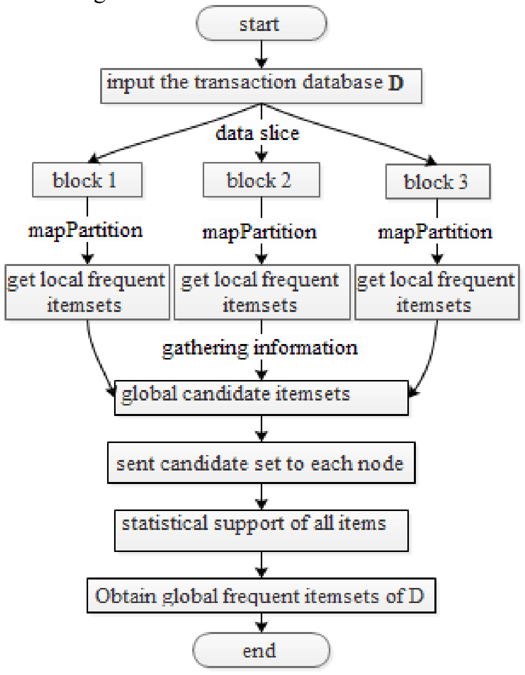
start
input the transaction database D
data slice
block 1
block 2
block 3
mapPartition
mapPartition
mapPartition
get local frequent itemsets
get local frequent itemsets
get local frequent itemsets
gathering information
global candidate itemsets
sent candidate set to each node
statistical support of all items
Obtain global frequent itemsets of D
end

Fig. 3. The implementation of IAP algorithm in Spark environment

For more detailed description of the algorithm, particularly SIAP algorithm is as follows:

TABLE I. THE IMPLEMENTATION PROCESS OF SIAP ALGORITHM

```
input: filePath, min_sup; //HDFS storage path of the transaction database, minimum support
output: global frequent itemsets L
Steps:
1. $val\ sc = \text{Initialize}\ SparkContext$ ;
2. $val\ hadoopRDD = sc.\text{textFile}(filePath)$ { //According to the filePath to access the HDFS content, and converted to hadoopRDD
3. $val\ FIRDD = hadoopRDD.\text{mapPartition}($
4. $p =>$ { //the contents of a data block
5. $I\_Apriori(p)$ // using IAP algorithm to process the data block p to obtain local frequent itemsets
6. }) ;
7. $val\ bro\_FIRDD = sc.\text{broadcast}(FIRDD\ to\ Local)$ ;}
   //Make the FIRDD localization, eliminate duplicates, and broadcast to each node
8. $val\ support\_item = hadoopRDD.\text{map}($ // Global scan the transaction database
9.     $it =>$ //Each row of data as input data
10.    $\text{getSupport}(it, bro\_FIRDD))$ //Judge whether every item appeared in it
11. $val\ L = \text{getFilter}(support\_item, min\_sup)$ //Merge the data and filter out non-compliant collections to get the final data
12. return $L$ ;
```

IV. EXPERIMENTS

A. Experimental Environment

The Spark clusters are consisted of five computers (Intel Core i5, CPU 2.4GHz, RAM 2GB) in this paper. The environmental parameters of Spark clusters are as follows:

(1) OS: Ubuntu14.04

(2) Hadoop version: 2.4.1

(3) Spark version: 1.2.0

B. Experimental Data

The experimental data sets used in this article are derived from Frequent ItemSet Mining DataSet Repository[12], and the two data sets are T10I4D100K[13] and Accidents.

C. Experimental Analysis

In order to eliminate the effect of accidental error in a single experiment, we use the approach of taking average of several experiments in this article. To evaluate the performance of the proposed SIAP algorithm, we experiment and analyze from the following three aspects:

(1) Data Scalability: Determining the support degree of data sets, the node number of cluster, then finding the relation between the copy number of data set and time;

(2) Cluster Scalability: Determining the support degree and amount of data sets, then finding the relation between the node number of cluster and time;

(3) Speedup of Cluster.

To verify the effectiveness and efficiency of the improved Apriori algorithm based on Spark (SIAP), the experimental comparison has been conducted between the algorithm based on Hadoop (MRApriori) and the algorithm based on Spark (SAP).

In order to test the performance of the algorithm to data scalability, different support degree are set depending on different data sets, and the cluster's node number is set as 5, one is the Master node, and the remaining four are the Salve nodes, the copy number of data sets is increased from 0 to 5. The comparison results are shown in Figure 4 and Figure 5.

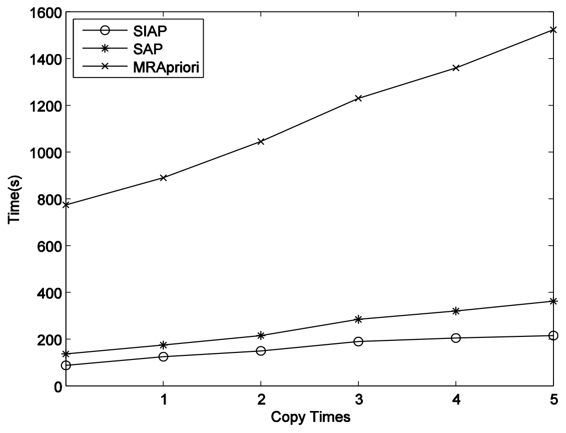

Fig. 4. T10I4D100K:min_sup=0.3%

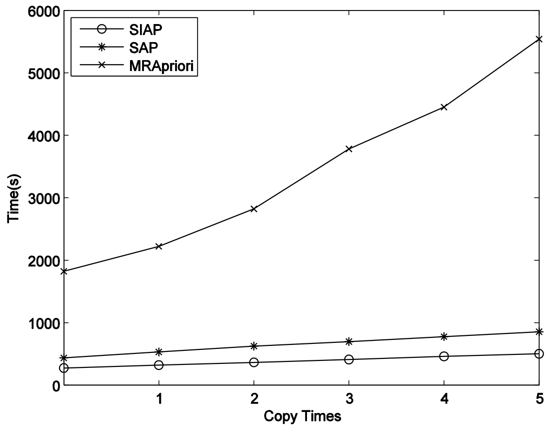

Fig. 5. Accidents: min_sup=20%

From the Figure 4 and 5, it can be seen that with the increase of copy number of the data set, the execution time of three algorithms shows an upward trend, but the rise is different: the rise degree of MRApriori algorithm is the most significant, with the increase of the copy number, and it almost rises in a linear fashion, at the same time, the rise degree will become more obviously when the amount of data is large; the degree of SAP algorithm is smaller, and the SIAP algorithm is the smallest, which almost growth in a horizontal manner. Also, it can be seen in one of the most intuitive: benefiting from the advantages and characteristics based on the memory computing and fine-grained local computing under Spark environment, the efficiency of SIAP algorithm is as much as 20 to 30 times compared to MRApriori algorithm, at the same time the efficiency of SIAP algorithm improved obviously than the SAP algorithm based on the traditional Apriori algorithm.

In order to test the performance of the algorithm to cluster scalability, different support degrees are set depending on different data sets, and the node number of cluster is increased from 1 to 5, the results of SIAP algorithm are shown in Figure 6 and Figure 7:

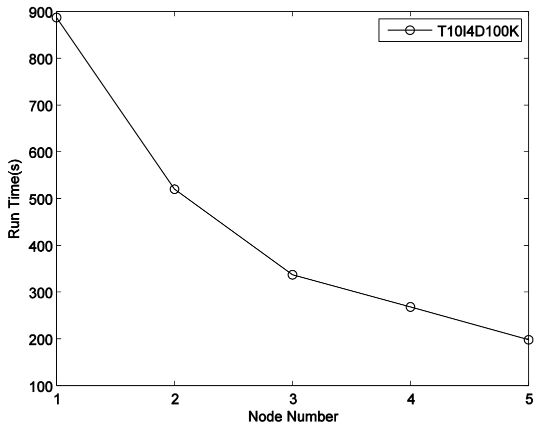

Fig. 6. T10I4D100K:min_sup=0.3%

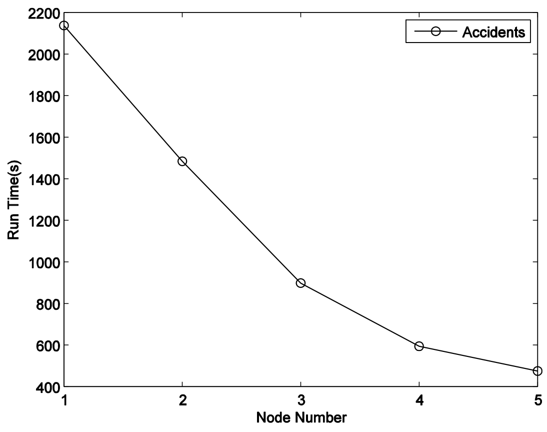

Fig. 7. Accidents: min_sup=20%

The horizontal axis is the number of nodes, and the vertical axis is the time. From Figure 6 to 7, we can see that with the increase of the node number, the running time shows a downtrend, while it can be seen the running time decreases almost linearly when the node number is large. However, it can also be seen that different data sets have different running time, and the decrease degrees are different. There are many reasons resulting in different decrease degrees: in the same distributed environment, the longitudinal and transverse dimension of data sets, the minimum support degree of the data sets, the size of the local frequent itemsets and so on. In the case of other parameters are the same, changing the minimum support degree of data sets, which are assumed as a and b, and $a < b$, therefore, the local frequent itemsets data generated on each node comply with $K_a \le K_b$. Since the local frequent itemsets need to merge by the reduce phase, in the same network conditions, the transmission time of K_b is longer than K_a, which will cause that the decline degree in condition a is likely to be greater than in condition b .

To further test the performance of SIAP algorithm, speedup is introduced to the experiment. The speedup means while maintaining the same size of the data sets, the performance which is demonstrated by algorithm parallelization via increasing cluster computer node, expressed as:

$$speedup(i) = t_1 / t_i \tag{4}$$

t_1 represent the running time on one node, and t_i represent the total running time on i nodes. In this experiment, in order to test the speedup of SIAP algorithm, keeping the experimental environment in the same, the compute node increments from 1 to 5, the results are shown in Figure 8 and Figure 9:

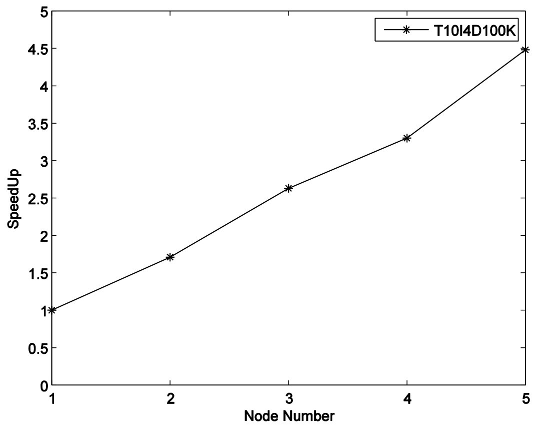

Fig. 8. T10I4D100K: min_sup=0.3%

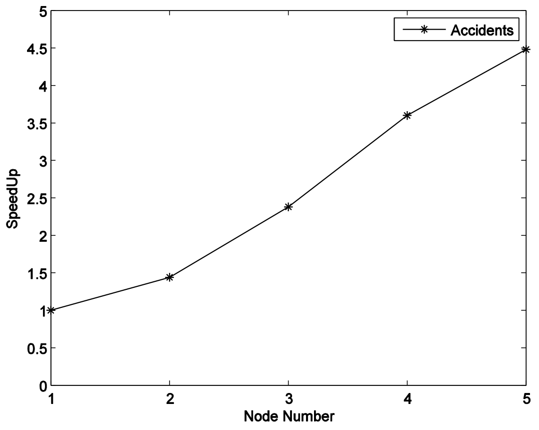

Fig. 9. Accidents: min_sup=20%

Idealized speedup should be linear, but from the above figures, the actual speedup is lower than ideal speedup, which is due to the communication between cluster computers, the time overhead brought by tasks start and schedule during iterative process makes the actual speedup lower than ideal speedup.

Through the above experiment, the SIAP algorithm has one to two orders of magnitude performance advantage compared to MRApriori algorithm, and has a better performance than SAP algorithm. Meanwhile, SIAP algorithm can adapt the scalability of the cluster size, and has a good speedup performance.

V. Conclusion and Future Works

In this paper, the concept of association rules is introduced. On the basis of analyze the Apriori algorithm, we discovered the problems that overburdened in I/O operations, high computational complexity in pruning step, haven't eliminated unnecessary scanning of affairs. To solve these problems, we proposed the IAP algorithm. Then combined with the characteristics of Spark, elaborated the idea of IAP algorithm parallelization, and based on the idea proposed SIAP algorithms and verified by comparative experiments.

The study of improved Apriori algorithm in paper, saves dataset in memory mapped using data structures. Although this data structure can compress the original data set, whereas when the data set has a lot of transaction attributes and there are large differences in latitude among the affairs, it will result in a waste of memory resources, in the further, we will try to optimize the data structure by using sparse matrix methods.

References

[1] Agrawal R , Srikant R.Fast algorithm for mining association rules[J] .Proceedings of 20th Int.Conf.Very Large Data Bases (VLDB), Morgan Kaufman Press，1994，487-499

[2] J. S. Park, M. S. Chen, P. S. Yu. Efficient parallel data mining of association rules. 4th International Conference on Information and Knowledge Management, 1995, 11: 233-235P

[3] S．Brin，et a1．Dynamic itemset counting and implication rules for market basket data．Proceedings of the ACM SIGMOD International Conference on Management of Data．1997．123-140

[4] Jeffrey Dean,Sanjay Ghemawat.Map/Reduce:Simplified Data Processing on Large Clusters[R]. OSDI'04: Sixth Symposium on Operating System Design and Implementation 2004.

[5] Lin M., Lee P. & Hsueh S. (2012). Apriori-based Frequent Itemset Mining Algorithms on MapReduce. Proc. of the 16th International Conference on Ubiquitous Information Management and Communication (ICUIMC '12). New York, NY, USA, ACM: Article No.76.

[6] Li N., Zeng L., He Q. & Shi Z. (2012). Parallel Implementation of Apriori Algorithm Based on MapReduce. In:Proceedings of the 13th ACM International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD '12). Kyoto, IEEE: 236–241.

[7] S. Hammoud. MapReduce Network Enabled Algorithms for Classification Based on Association Rules. Thesis, 2011.

[8] Yanjie Gao.Data Processing with Spark Technology, Application and Performance Optimization[M]. China machine Press. 2014-11,1-2.

[9] Qiu H, Gu R, Yuan C, et al. YAFIM: A Parallel Frequent Itemset Mining Algorithm with Spark[C]//Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International. IEEE, 2014: 1664-1671.

[10] Lin Jiaxiong, Huang Zhan. An improved Apriori Algorithm based on Array Vectors. Computer Applications and Software, 2005, 5(28):268-271.

[11] Qian Guangchao, etc al. One Optimized Method of Apriori Algorithm. Computer Engineering, 2008, 34(29) : 196-198.

[12] Frequent Itemset Mining Dataset Repository[EB/OL].[2012-10-21]. http://fimi.ua.ac.be/data/, 2012.

[13] Synthetic Data Generation Code for Associations and Sequential Patterns. Intelligent Information Systems, IBM Almaden Research Center. http://www.almaden.ibm.com/software/quest/Resources/index.shtml.