Journal of Information Hiding and Multimedia Signal Processing ©2019 ISSN 2073-4212
Ubiquitous International Volume **10**, Number **1**, January **2019**

An Improved FP-Growth Algorithm Based on Projection Database Mining in Big Data

Le Zhang[1], Su Xu[1,*], Xiangjun Li[1,2*] and Xiaoliang Wu[1]

[1]School of Information Engineering, [2]School of Software
Nanchang University
No. 999 Xuefu Road, Honggutan District, Jiangxi Nanchang, China
zhangle@ncu.edu.cn
*Corresponding author: xusu@ncu.edu.cn, lxjun_alex@163.com

Pei-Chann Chang

Department of Information Management
Yuan Ze University
135 Yuan Tung Road, Chungli 32003, Taiwan
iepchang@saturn.yzu.edu.tw

Received April 2018; revised August 2018

Abstract. *The traditional FP-Growth frequent itemset mining algorithm and some improved algorithms are faced with a problem of being unable to store the huge FP-tree in standalone memory. Therefore, a parallel mining algorithm is raised in this paper. Transaction databases are extracted according to each frequent 1-item, and a corresponding projection database is generated for each frequent 1-item; then, projection databases are respectively distributed to a node machine, and the improved algorithm is used in each node machine to generate partial frequent itemsets by parallel mining; finally, all frequent itemsets are obtained by summarization. This algorithm has no need to generate a FP-tree for transaction databases, and solves the storage problem of standalone memory. Besides, the use of the MapReduce programming model and parallel processing technology increases the efficiency of frequent itemset mining.*
Keywords: Frequent Itemset, FP-Growth, Big Data, MapReduce Programming Model

1. **Introduction.** The Frequent Pattern means a pattern that occurs frequently in a dataset. A set of data items that occurs frequently in a dataset is known as a frequent itemset. Frequent itemset mining on datasets can further derive the association rules among data items so as to help us find out the association between transactions [1].

R.Agrawal put forward the classic Apriori algorithm to dig the frequent pattern in datasets [2]. Han et al. put forward a FP-Growth algorithm based on the frequent pattern tree (FP-tree) for frequent pattern mining [3]. By comparison, the Apriori algorithm has two defects. First, it needs to scan the transaction database repeatedly, resulting in huge network load or I/O load; second, it may produce a huge candidate set, and take up a lot of memory space. However, the FP-Growth algorithm only needs to scan the transaction database twice to generate a FP-tree, then adopt the recursive algorithm to dig the FP-tree and generate all frequent itemsets, without any candidate set. Thus, the algorithm is more efficient[4].

However, faced with massive data in the era of big data, the storage and computing ability in the standalone environment will become the bottleneck of data mining[5].

Therefore, improving the traditional algorithm and utilizing big data, parallel computing technologies and so on for frequent itemset mining becomes a research focus. For example, parallel FP-Growth algorithm based on multithreading[6]. Though the algorithm effectively shows the parallelism of the FP-Growth algorithm and makes full use of processor resources, it still shares a single memory space. As a result, it fails to solve the defect of insufficient standalone memory. Another example is the parallel FP-Growth mining algorithm on general PC cluster [7]. By comparison with the algorithm in Literature [6], this algorithm has a large improvement, truly realizes distributed parallel computation, and makes full use of the computing and storage resources of multiple nodes. But it fails to consider the communication load among cluster machines, resulting in low performance on the whole. Parallel computing based on MapReduce programming model is used in Literature [8], [9] ,[10],[11],[12] and [13].

It effectively solves a computing bottleneck, but still needs to construct a FP-tree and recursively construct a conditional pattern tree, failing to truly solve the problem of the traditional FP-Growth algorithm in storing a huge FP-tree in the context of big data. Motivated by this, in this paper, we propose an improved parallel partitioning method based on FP-Growth, and realized by means of the MapReduce programming model and parallel processing technology.

The rest of this paper is organized as follows. A brief description and limitation of traditional FP-Growth algorithm is introduced in Section 2. The improved partitioning method based on FP-Growth is proposed in Section 3. The improved parallel frequent itemset mining with two stages is provided in Section 4. Section 5 describes the experimental results and their analysis of the improved FP-Growth algorithm performance, and finally, the conclusion of the research is shown in Section 6.

2. **Traditional FP-Growth Algorithm.**

The FP-Growth algorithm uses a data structure known as the frequent pattern tree (FP-tree), which is a special prefix tree composed of a frequent item header table and an item prefix tree. It compresses the entire transaction database into a frequent pattern tree. In the process of FP-tree construction and frequent itemset mining, it only needs to scan the transaction database twice, thereby, greatly reducing I/O load and improving mining efficiency.

The transaction database is shown in Table 1, with the minimum support count minsupport=2.

Table 1. Transaction Database D

TID	Transaction ID List
T100	I_1, I_2, I_5
T200	I_2, I_4
T300	I_2, I_3
T400	I_1, I_2, I_4
T500	I_1, I_3, I_6
T600	I_2, I_3
T700	I_1, I_3
T800	I_1, I_2, I_3, I_5
T900	I_1, I_2, I_3, I_7

The main steps of the algorithm are described as follows:

(1) The transaction database D is scanned for the first time to obtain all frequent 1-itemsets, and the collection of frequent 1-items is sorted in a descending sequence of

support count. In this way, the frequent 1-item header table L (as shown in Table 2) is obtained. The support count of I_6 and I_7 is 1, below the min-support. They belong to non-frequent items, so they will not occur in the frequent 1-itemset.

Table 2. Frequent 1-Itemset L

Item	Support Count
I_2	7
I_1	6
I_3	6
I_4	2
I_5	2

(2) FP-tree construction: First, the root node of the tree is created, tagged with null. Then, the transaction database D is scanned for the second time. A branch is created for each transaction in the FP-tree, and each node in a branch corresponds to each item in the transaction (deleting non-frequent items) and is linked according to the sequence of Table L. Besides, the count of each item in the branch increases by 1. At last, the association between the frequent 1-itemset header table and the FP-tree is established, and the FP-tree as shown in Figure1 is obtained.

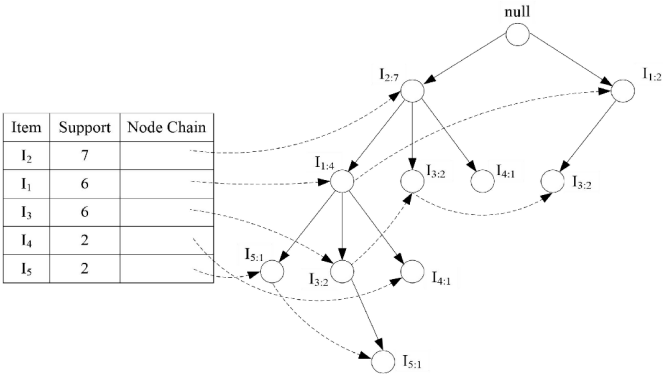

Figure 1. FP-Tree for transaction database D

(3) A conditional pattern base and a conditional pattern tree are generated for each frequent 1-item. For each item in the header table, the FP-tree is traversed from bottom to top to obtain the conditional pattern base corresponding to each item. The conditional pattern bases of all items are shown in Table 3.

Table 3. Conditional Pattern Base

Item	Conditional Pattern Base
I_5	$\{<I_2, I_{1:1}>, <I_2, I_1, I_{3:1}>\}$
I_4	$\{<I_2, I_{1:1}>, <I_{2:1}>\}$
I_3	$\{<I_{2:2}>, <I_{1:2}>, <I_2, I_{1:2}>\}$
I_1	$\{<I_{2:4}>\}$

(4) The conditional FP-tree of each item is constructed according to the conditional pattern base thereof obtained in (3). At last, frequent itemset mining is conducted by the conditional FP-tree.

For example, the conditional pattern base of I_5 comprises two prefix paths, $<I_2, I_{1:1}>$ and $<I_2, I_1, I_{3:1}>$; the conditional FP-free of I_5 obtained therefrom only includes a single path $<I_{2:2}, I_{1:2}>$, and does not contain I_3, because its support count is 1, below the minimum support count. The single path generates the following frequent itemsets: { I_2, $I_{5:2}$}, { I_1, $I_{5:2}$}and { I_1, $I_{5:2}$}.

In another example, the conditional pattern base of I_3 only contains three prefix paths $< I_{2:2}>$, $< I_{1:2}>$ and $< I_2, I_{1:2}>$. Its conditional FP-tree has two branches, $< I_{2:4}, I_{1:2}>$ and $< I_{1:2}>$. The conditional FP-tree generates the following frequent itemsets: { I_2, $I_{3:2}$}, { I_2, $I_{3:2}$} and { I_2, I_1, $I_{3:2}$}.

3. **Improved FP-Growth Algorithm.** The FP-Growth algorithm builds a FP-tree by scanning a transaction database. It comprises the information of all frequent itemsets. Then, based on the FP-tree, it builds the corresponding conditional FP-tree for each frequent 1-item, and digs the frequent itemset in each conditional FP-tree. At last, the set of frequent items dug from all conditional FP-trees is the set of all frequent items corresponding to the entire transaction database. This approach is effective in the event that the transaction database is not large in size and the FP-tree can be stored in standalone memory. However, in the context of big data, faced with massive databases, the FP-tree constructed cannot be stored in standalone memory. In this case, this approach loses effectiveness.

Therefore, we improved the traditional FP-Growth algorithm. We still take the aforesaid transaction database D as an example to expound the specific improvement method.

(1) The transaction database D is scanned for the first time to obtain all frequent 1-itemsets, and the collection of frequent 1-items is sorted in a descending sequence of support count. In this way, the frequent 1-item item header table L (as shown in Table 2) is obtained. This step is the same as that of the traditional FP-Growth algorithm.

(2) First, data cleaning is conducted for the transaction database D, and all non-frequent 1-items in D are deleted. Then, D is extracted according to each frequent 1-item (excluding the first item I_2 in Table L); a projection database for each frequent 1-item is created, in which, all transactions contain such item; each projection database is distributed to a node machine. The corresponding projection databases of I_1, I_3, I_4 and I_5 are respectively shown in Tables 4-7.

Table 4. Corresponding Projection Database of I_1

TID	Transaction ID List
T100	I_1, I_2, I_5
T400	I_1,I_2, I_4
T500	I_1, I_3
T700	I_1, I_3
T800	I_1, I_2, I_3, I_5
T900	I_1, I_2, I_3

Table 5. Corresponding Projection Database of I_3

TID	Transaction ID List
T300	I_2, I_3
T500	I_1, I_3
T600	I_2, I_3
T700	I_1, I_3
T800	I_1, I_2, I_3, I_5
T900	I_1, I_2, I_3

(3) Each node machine first scans the distributed projection database to construct the FP-tree of the corresponding item. We need to improve the traditional FP-Growth algorithm for FP-tree construction. We assume that the k_{th} node machine processes the frequent 1-item I_k, and the corresponding projection database is D_k. During transaction processing of D_k, the items in transactions are sorted according to the sequence of Table L; I_k and all subsequent items are deleted; branches are generated for the remaining items in the FP-tree to be constructed. The specific algorithm is described as follows:

Table 6. Corresponding Projection Database of I_4

TID	Transaction ID List
T200	I_2, I_4
T400	I_1, I_2, I_4

Table 7. Corresponding Projection Database of I_5

TID	Transaction ID List
T100	I_1, I_2, I_5
T800	I_1, I_2, I_3, I_5

① Create the root node of the FP-tree, tagged with null.

② Scan the database D_k, and execute the following steps for each transaction therein:

a. The items in the transaction are sorted according to the sequence of L, I_k and all subsequent items are deleted; the formed transaction item list is denoted as [p | P], wherein, p is the first item element, and P is the list of remaining item elements.

b. Call insert tree ([p | P], T). The executing process of insert tree ([p | P], T) is as follows: if T has a child N, with N.item-name=p. item-name, then, the count of N increases by 1, and the support count of the corresponding item in the header table increases by 1; or create a new node N, set its count as 1, and link it to its parent node T. In addition, add an item to the header table, and set the support count as 1. If P is non-null, recursively call insert_tree (P, T).

The FP-tree generated by the improved algorithm will be much smaller than that generated by the traditional algorithm, which is called projection FP-tree. The projection FP-trees constructed for I_1, I_3, I_4 and I_5 are respectively displayed in Figures 2-5.

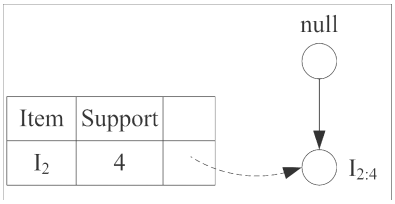

Figure 2. Projection FP-Tree of I_1

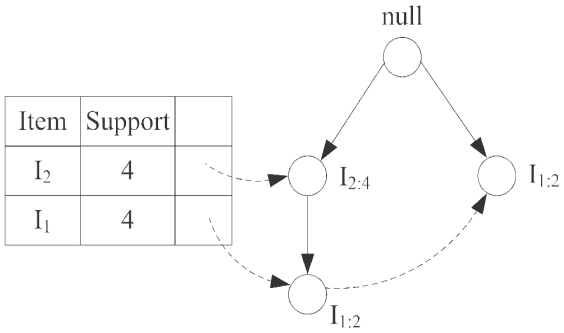

Figure 3. Projection FP-Tree of I_3

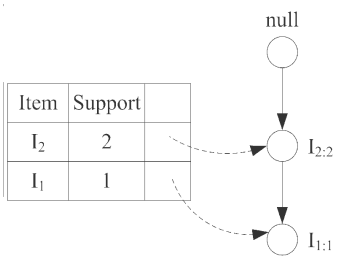

Figure 4. Projection FP-Tree of I_4

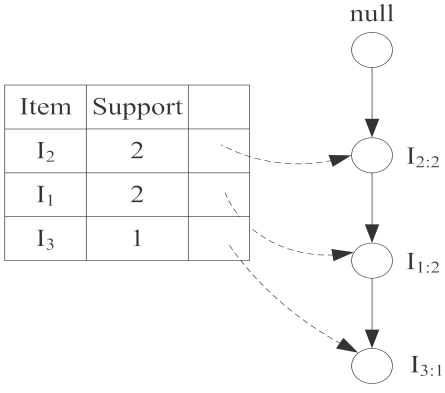

Figure 5. Projection FP-Tree of I_5

(4) Pruning. Each node machine prunes the projection FP-tree constructed, deletes the items with support count below the minimum support count (min-support=2) in the

header table, and removes the nodes thereof in the FP-tree. In the above four projection FP-trees, only the FP-trees of I_4 and I_5 need to be pruned. The pruned projection FP-trees are shown in Figure 6 and Figure 7.

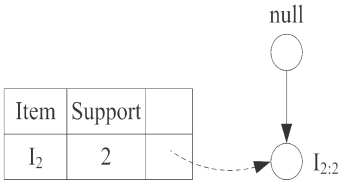

FIGURE 6. Pruned Projection FP-Tree of I_4

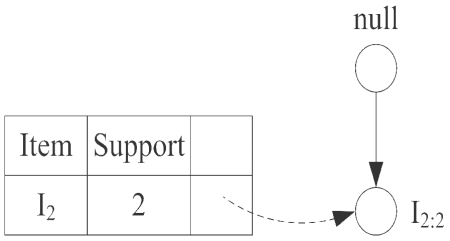

FIGURE 7. Pruned Projection FP-Tree of I_5

The FP-tree generated upon pruning is actually the corresponding conditional FP-tree of each frequent 1-item.

(5) Each node machine digs the conditional FP-tree constructed to generate the corresponding frequent itemset. We assume that the processing item of a node machine is I_k, and its corresponding conditional FP-tree is T_k, then, the algorithm of frequent itemset mining by the conditional FP-tree is described as follows:

FOR Each path starting from null root node of T_k is denoted as R;
  FOR Each node in R is denoted as p;
    Frequent pattern = p$\cup I_k$
  FOR Each node combination in R is denoted as P;
    Frequent pattern = P
    Frequent pattern = P$\cup I_k$

(6) At last, combining the frequent items generated by each node machine, we can obtain all frequent itemsets.

4. **Parallel Frequent Itemset Mining.** The improved frequent itemset mining includes two stages. In Stage 1, it generates all frequent 1-itemsets, and creates the frequent 1-item header table L. We can use the MapReduce programming model for parallel realization [14,15]. In Stage 2, pluralities of node machines execute frequent itemset mining in a parallel way. Finally, all frequent itemsets are summarized.

(1) Stage 1: First, the transactions in the transaction database are divided into n identical data blocks. Then, n data blocks are sent to n Map nodes with parallel processing; the partial 1-itemset and the support count thereof are calculated in each Map node. At the end of Map node processing, the identical items are combined by the function Combiner. The results are sent to a Reduce node to calculate the global frequent 1-itemset and the support count thereof, and the frequent 1-itemset is sorted in a descending sequence of support count. Finally, the sorted result set L is obtained (as shown in Table 2). The algorithm procedures in the Map and Reduce process in Stage 1 are shown in Figure 8.

(2) Stage 2: To improve the traditional FP-Growth algorithm, we do not construct the entire FP-tree, but extract the transaction database according to each frequent 1-item to generate the corresponding projection database for each frequent 1-item. Then, we distribute the projection database to each node machine, and enable them to respectively generate a partial frequent itemset. Finally, the entire frequent itemset is obtained by summarization. The mining process in Stage 2 is shown in Figure 9.

In practical application, the number of items in a transaction database is often much more than that of actually distributable node machines. In this case, to balance the

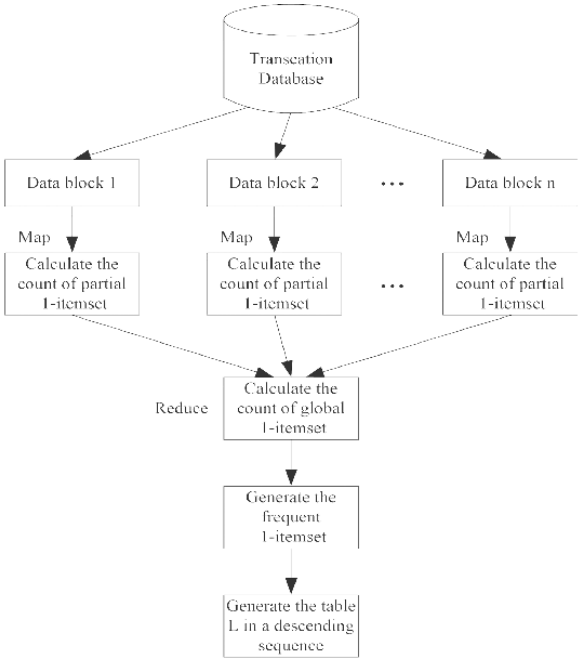

FIGURE 8. Frequent 1-Item Mining Based on MapReduce

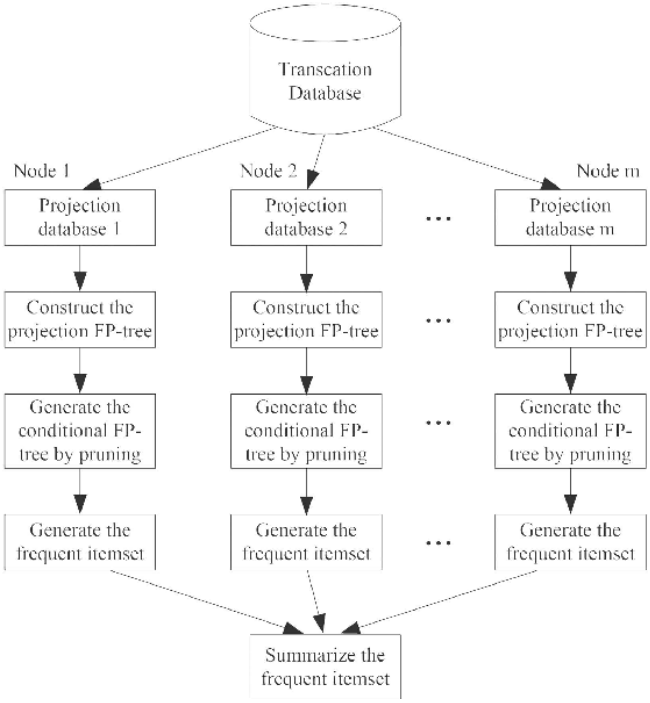

FIGURE 9. Mining Process in Stage 2

processing workload of node machines, when distributing processing items to each node machine, we first distribute node machines according to the ascending sequence of Table L. After completing a round of distribution, we will distribute node machines according to the descending sequence of Table L.

For some commercial applications, records in transaction databases will continue to increase dynamically, the transaction database will also continue to expand. Although this algorithm can solve the problem of expanding database mining by adding node machine,

but the resources that they use will also continue to expand. Therefore, the actual practice is to set a certain time interval to mine.Of course, the longer the time interval, the higher the accuracy for data mining.

5. **Experimental Analysis and Summary of the Algorithm.** To verify availability and efficiency of the parallel frequent itemset mining algorithm based on MapReduce proposed in this paper, we adopted the java language to realize the traditional FP-Growth algorithm and the proposed improved algorithm, and carried out experiments and analyses. In the experiments, 6 nodes are configured in the Hadoop cluster environment, and the client-server architecture is applied, including 1 control node and 5 data nodes. Each node is configured with Intel core i5-2450M CPU 2.50 GHz, 4GB memory.

First of all, we need to verify a problem of the traditional FP-Growth algorithm, i.e., being unable to store a huge FP-tree in standalone memory. We first selected the data set T1014D100K.dat in Frequent Itemset Mining Data Repository [16]. The data set is 3.84MB, including 100,000 records. We set the minimum support as 20%, 10% and 5% respectively, and used the traditional FP-Growth algorithm for mining. The algorithm routine could be executed smoothly. Then, we selected a 10-million-level database, 345MB in size. The algorithm routine still could be executed smoothly under the support of 20%, 10% and 5%. When we selected a 100-million-level database, the algorithm routine could be executed smoothly under the support of 20% and 10%; but insufficient memory occurred under the support of 5%, resulting in a failure to complete normal data mining. Thus, we used the proposed algorithm, and performed it in 5 node environments. In this way, the algorithm routine could smoothly complete data mining.We find that the larger the database and the smaller the support, the more the frequent 1-items it contains, so the bigger the FP tree is.

We further verified the efficiency of the improved algorithm proposed in this study, selected a 100 thousand records transaction database, with the support of 5%, and performed the FP-Growth algorithm in a standalone machine, and the improved algorithm on 2, 3, 4 and 5 node machines respectively. Their running time is respectively shown in Fig.10. Then, we selected a 10 million records transaction database with the support of 5%, and performed the FP-Growth algorithm in a standalone machine and the improved algorithm on 2, 3, 4 and 5 node machines respectively. Their running time is respectively shown in Fig.11.

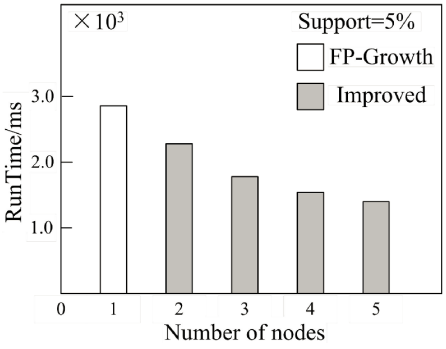

FIGURE 10. RunTime for 100,000 records

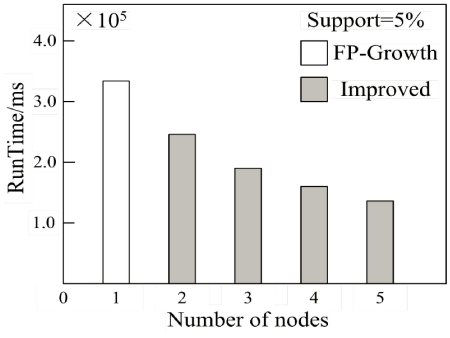

FIGURE 11. RunTime for 10,000,000 records

As shown in Figure 10, the algorithm adopts the MapReduce parallel computing mode and the divide-and-conquer strategy, and uses a plurality of node machines to conduct improved frequent itemset mining for the projection database in a parallel way. Thereby, the efficiency of mining is improved, but the time is not proportionally reduced along

with the number of nodes increases, the reason for this is that, when there is not enough data in the transaction database, the time required for the improved algorithm used to generate a projection database for each frequent item and to distribute to each node is relatively large, it affects the time that is reduced.

When the records of the transaction database reached 10 million level, the FP tree generated by the traditional FP-Growth algorithm is very large, and so recursive mining of frequent items by FP trees will also be more time-consuming. Thereby, the time required for the improved algorithm used to generate a projection database for each frequent item and to distribute to each node is relatively reduced, so the time that is reduced is becoming more obviously. It can be seen that the improved algorithm is more efficient for the frequent item mining of the massive transaction database in the big data environment.

6. **Conclusion.** To solve the problem that the traditional FP-Growth frequent itemset mining algorithms are faced with a problem of being unable to store the huge FP-tree in standalone memory, in this paper, we propose an improved parallel mining method based on FP-growth, and realized by means of the MapReduce programming model and parallel processing technology. According to the research focus of the proposed algorithm, it fundamentally solves the problem of the traditional FP-Growth algorithm and some improved FP-Growth parallel algorithms, i.e., being unable to store the FP-tree in standalone memory and unable to realize frequent itemset mining for a massive transaction database. However, as the proposed algorithm requires extracting the transaction database based on each frequent 1-item, generating a corresponding projection database for each frequent 1-item, and distributing the projection database to each node machine, it increases I/O load to some extent.

Acknowledgement. The authors would like to thank all the referees for their constructive and insightful comments on this paper. The contributions of this research are the following: the National Natural Science Foundation of China (No. 61762062, 61601215, 61862042); Science and Technology Innovation Platform Project of Jiangxi Province (No. 20181BCD40005); Major Discipline Academic and Technical Leader Training Plan Project of Jiangxi Province (No. 20172BCB22030); Primary Research & Development Plan of Jiangxi Province (No. 20181ACE50033, 20171BBE50064, 2013ZBB E50018, 20111BB E50008); Jiangxi Province Natural Science Foundation of China (No.20142BAB207011); Science & technology research project of Education Department of Jiangxi Province (No. GJJ150104, GJJ161387 and YC2015-S035).

REFERENCES

[1] P. N. Tan, M. Steinbach, V. P. Kumar, Introduction to Data Mining, *Posts & Telecom Press*, 2011.

[2] R. Agrawal, T. Imielinski, A. Swami, Mining Association Rules between Sets of Items in Large Data Bases, *Proceedings of the 1993 ACM-SIGMOD International Conference on Management of Data, Washington DC:ACM Press,* 1993.

[3] H. Jiawei, P. Jian, Y. Yiwen, Mining Frequent Patterns Without Candidate Generation, *Proceedings of the ACM-SIGMOD International Conference on Management of Data. Dallas, Texas, USA: ACM Press,* 2000.

[4] R. C. Agarwal, C. C. Aggarwal , V. V. V. Prasad , A Tree Projection Algorithm for Generation of Frequent Item Sets, *Journal of Parallel & Distributed Computing,* vol.61,no.3,pp.350-371,2001.

[5] L. J. Zhou, X. Wang, Research of the FP-Growth Algorithm Based on Cloud Environments, *Journal of Software,* pp.676-683,2014

[6] L. Li, E. Li , Y. M. Zhang, Optimization of Frequent Itemset Mining on Multiple-core Processor, *Proceedings of the 33rd International Conference on Very Large Data Bases. Vienna, Austria: VLDB Endowment,*pp.1275-1285,2007.

[7] M. Chen, H. F. Li, FP-Growth Parallel Algorithm in Cluster System, *Computer Engineering,*no.10,pp.71-75,2009.
[8] A. Bechini, F. Marcelloni, Armando Segatori, A MapReduce solution for associative classification of big data, *Information Sciences,*pp.33-55,2016.
[9] Li Haoyuan, Wang Yi, Zhang Dong, PFP: Parallel FP-Growth for Query Recommendation, *Proceedings of the 2008 ACM Conference on Recommender SystemsLousanne, Switzerland:,*pp.125-137,2008.
[10] A. Segatori, A. Bechini, P. Ducange , et al, A Distributed Fuzzy Associative Classifier for Big Data, *IEEE Transactions on Cybernetics,* , no. 99, pp.1-14, 2017.
[11] M. Chen, X. D. Gao , H. F. Li, An efficient parallel FP-Growth algorithm, International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery. *IEEE,*pp.283-286,2009.
[12] X. S. Chen, Z. Shuai,T. Hao , et al, FP-Growth Algorithm Based on Boolean Matrix and MapReduce, *Journal of South China University of Technology,*vol.42,no.1,pp.135-141,2014.
[13] S. G. Totad, R. B. Geeta , Reddy P V G D P, Batch incremental processing for FP-tree construction using FP-Growth algorithm, *Knowledge & Information Systems,*vol.33,no.2,pp.475-490,2012.
[14] M. Qiang, Yang Jinmin. A Parallel Frequent Itemsets Mining Algorithm Based on MapReduce, *Computer Applications and Software,* 2015.
[15] M. Yuekun, L. Pengfei, et al, Improved FP-Growth Algorithm and Its Distributed Parallel Implementation, *Journal of Harbin University of Science and Technology,*2016.
[16] F. Itemset Mining Dataset Repository[EB/OL][2012-10-21], http://fimi.ua.ac.be/data, 2012.