2014 IEEE Workshop on Electronics, Computer and Applications

MRPrePost—A parallel algorithm adapted for mining big data

Jinggui Liao
Computer Science and Engineering, South China University of Technology
Guangzhou, China
e-mail: zhaolab@126.com

Yuelong Zhao*, Saiqin Long
Computer Science and Engineering, South China University of Technology
Guangzhou, China
e-mail: zhaolab@126.com

***Abstract*—With the explosive growth in data, using data mining techniques to mine association rules, and then to find valuable information hidden in big data has become increasingly important. Various existing data mining techniques often through mining frequent itemsets to derive association rules and access to relevant knowledge, but with the rapid arrival of the era of big data, Traditional data mining algorithms have been unable to meet large data's analysis needs. In view of this, this paper proposes an adaptation to the big data mining parallel algorithms—MRPrePost. MRPrePost is a parallel algorithm based on Hadoop platform, which improves PrePost by way of adding a prefix pattern, and on this basis into the parallel design ideas, making MRPrePost algorithm can adapt to mining large data's association rules. Experiments show that MRPrePost algorithm is more superior than PrePost and PFP in terms of performance, and the stability and scalability of algorithms are better.**

Keywords- big data; data mining; PrePost algorithm; parallel iz-ation; MRPrePost algorithm

I. INTRODUCTION

Big Data is the most popular word in the IT industry, which refers to the amount of data involved is huge to not pass the current mainstream software tools, to capture, manage, process, and organize datasets into useful information within a reasonable time. Big Data has 4V characteristics-Volume, Variety, Velocity, and Value. Big data mining techniques to existing data poses a severe technical challenges, effective data analysis for commercial and academic research is increasingly important.

Agrawal[1] in 1993 first proposed mining customer transaction database itemsets problem, now FIM (frequent itemsets mining) has become an essential part of data mining. Most of the current algorithms can be grouped into two categories[2]: Apriori-like algorithm and FP-growth algorithm. Apriori[1,3,4,5] by repeatedly scanning the database to prune candidate sets. the main advantage of FP-Growth[6] algorithm is FP-Tree. When faced with large data, these two algorithms are not well adapted to. For the above algorithm, a solution is to consider only the large threshold value, the number of candidates can be reduced, but this will lead mining association rules out inaccurate due to low utilization data[7].

Based on above, this paper propose a data mining algorithms based on Hadoop framework parallel called MRPrePost,which is a hybrid of big data mining method. It combines the Dis-Eclat[8] basis PrePost[2] algorithm. By uniformly partitioning the search space way instead of the original database, which is a good solution to the problem of mining large data response time bring. In order to verify the efficiency of the algorithm, it's run on two datasets with PrePost and PFP-Growth, conducted experiments comparing, Experimental results show that, MR-PrePost algorithm than several other mining algorithm is faster.

The next form of paper is organized as follows: The second part is related to work, and the third section details MR-PrePost algorithm, the fourth part is the comparison of experimental results and performance analysis, the fifth part of the paper summarizes

II. RELATION WORK

A. Problem Description

Let $I = \{i_1, i_2, \cdots, i_n\}$ be a set items, a transaction is defined as $T = (tid, X)$ where tid is a transaction identifier and X is a set of items over I. A transaction database D is a set of transactions. Its vertical database D' is a set of pairs that the are composed of an item and the set of transactions C_j that contain the item:

$$D' = \left\{\left(i_j, C_{i_j} = \{tid | i_j \in X, (tid, X) \in D\}\right)\right\}. \quad (1)$$

C_{i_j} is also called the cover or tid-list of i_j. The support pf an itemset Y is the number of transactions that contain the itemset. Faormally,$\text{support}(Y) = |\{\text{tid}|Y \subseteq X, (\text{tid}, X) \in D\}|$, or in vertical database format $\text{support}(Y) = |\bigcap_{i_j \in Y} C_{i_j}|$. An itemset is said to be frequent if its support is greater than a given threshold δ, which is called minimum support. Frequency is a monotonic property w.r.t. set inclusion, meaning that if an itemset is not frequent, none of its supersets are frequent. Similarly, if an itemset is frequent, all of its subsets are frequent.

B. Correlation algorithm analysis

The current main association rule mining algorithms are Apriori, FP-growth, Eclat, MRApriori, PFP-growth, Dis-Eclat, etc.

Apriori[1] is the most classical algorithm, the main idea is to generate k+1-frequent itemsets based on k-candidate itemsets. By traversing the database to statistics candidate collection, then according to support threshold to prune candidate itemsets. The pruning strategy is that if an itemset is not frequent, its superset so is. We can be seen from the algorithm ideas, from k-itemset to k+1-itemset and uses a

breadth-first strategy prefix mode. The algorithm is simple, but many times to traverse the database and generate a large number of candidate sets, time and memory overhead will become a bottleneck[9,10]. Traversing the operation of the database is distributed to each node, while reducing the candidate set of statistical time, but increases network traffic overhead.

Comparing with Apriori, FP-growth[6] is an improved algorithm. Its main advantage is that only needs to scan the database twice, and construct a compressed data structure - FP-Tree, which reduces the search space, while no candidate set, improved memory utilization. From the algorithm thought,we can see it adopts to depth-first mode policy. However, it constructs a large number of conditions pattern tree when recursive, When faced with large amounts of data, the memory is difficult to accommodate all of the pattern tree[11], and the tree traversal algorithm whose time complexity is higher. PFP[11] is based on the Hadoop[12] parallel algorithms, PFP groups the itemsets, as a condition database divided to each node, each node independently generates the FP-Tree and mines frequent itemsets. PFP reduce the traffic between nodes,increases the degree of polymerization of node. However, algorithm is not efficient if the database is discrete[2]. In addition, grouping strategy relates to the efficiency of the algorithm, a distributed programming difficulty lies in load balancing, distributed algorithm execution time is determined by the maximum execution time of a node, therefore, how to make data uniform distributed related to the algorithm efficiency. Reference literature[8] mentioned a Round-Robin strategy is basically to achieve load balancing effect, this paper also uses this strategy.

Eclat[13] by depth-first traversal and prefix pattern tree generates frequent item sets, which is also based on if an itemset is not frequent, none of its supersets are frequent. the time complexity is O(M*N) when Eclat each time adds a prefix, M and N are TD-list length of a prefix sub-tree with the newly added item. The algorithm needs to save D' in memory. Dis-Eclat unlike the previously mentioned algorithm dividing database, but the search space will be allocated to each node, which eliminates the communication between nodes. But as the previous analysis, the time complexity is O(M*N), and each prefix subset must store all records containing the subset.

From the above analysis, the current method has advantages and disadvantages, The advantage of Eclat algorithm is to use a vertical database can quickly calculate support of frequent itemsets. The advantage of FP-growth algorithm is to generate a compressed data structure FP-Tree, and does not produce candidate set in the mining process. MRPrePost proposed algorithm combines the advantages of both, MRPrePost also established similar to FP-Tree data structure called PPC-Tree, then by PPC-Tree to structure similar to the structure of the vertical TD-list database N-list. The specific implementation details will be described detail in part III.

III. MRPrePost Algorithm

A. PrePost Algorithm

PrePost algorithm presents a data structure named N-list, which is a modification of the vertical database, storing the association rule mining all the information needed. PrePost also need to scan the database twice to construct a PPC-Tree, and make use of PPC-Tree to generate the N-list of FIM1. In the mining process, the database does not require re-scanning, only need to intersect the merger N-list, and the complexity of the algorithm is O(m+n), m and n are the length of two N-list, Each element of N-list composed by PrePost-Code, which is called after the sequence encoding the preamble, the composition in the form of <(pre-order, post-order: count)>, PrePost-Code is based on the PPC-Tree respectively from the previous order traversal and post order traversal. Fig. 1 show the PPC-Tree, which is similar to FP-Tree, and the construction process is the same with the FP-Tree. but not the same as the composition of the node, PPC-Tree node consists of five components.

1. item-name: represent node name
2. count: represent node count
3. children-list: represent a children collection of the node
4. pre-order: represent order of node when preorder
5. post-order: represent order of node when postorder

Table 1 shows a transaction database, corresponding to Fig.1 for PPC-Tree, assuming the minimum support δ is 3.

Table 1. Transaction database

ID	Items	Ordered frequent items
1	a, c, g, f	c, f, a
2	e, a, c, b	b, c, e, a
3	e, c, b, i	b, c, e
4	b, f, h	b, f
5	b, f, e, c, d	b, c, e, f

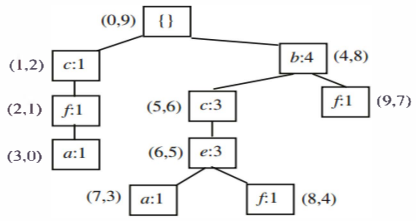

Figure 1. PPC-Tree corresponding with Table 1

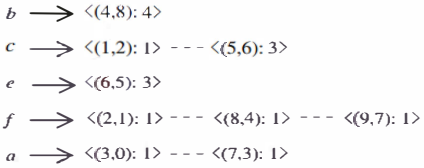

Figure 2. N-list of FIM1

Each k-frequent itemsets F_k corresponds to a N-list, which in ascending order according to the pre-order, at the same time must also be ascending according to post-

order(concrete proof is in [2]). PPC-Tree's main purpose is to construct N-list liking shown by Fig. 2, then find all the frequent item sets based on N-list. We can then delete the PPC-Tree to reduce memory overhead. The main steps of the PrePost algorithm：

1. Scan transaction database named D, output the FIM1, And in descending order according to the number of its support to generate F_1.
2. Scan D again, Select the frequent items in each record and arrange them in the order of F_1, assuming list of itemsin each record is [p|P], p is the first item in the list, P is the rest of the items. Call the function insert_tree ([p|P], T_i).
3. Tree formed on the second step, respectively preorder traversal and postorder traversal, set pre-order and post-order of each node and establish N-list of 1-frequent itemsets.
4. Mining frequent itemsets based on N-list using the method liking Apriori Algorithm.

For a detailed description of Step 4, we give an example. For frequent itemsets bf, by the b, f merger. N-list of b is <(4,8):4>, and N-list of f is <(2,1):1>-<(8,4):1>-<(9,7):1>, because 4>2, 8>1, this case, f and b are not on the same path, then traverse the next PP-Code of f, we find 4<8 and 8>4, this case b and f are on the same path, so the PP-Code:<(4,8):1> added to the N-list of bf. Similarly, since 4 <9 and 8> 7, <(4,8): 1> added to the N-list of bf, in which case the item sets bf N-list: <(4,8): 1>-<(4,8): 1>, for the same pre-order, post-order to get the merger of PP-Code of bf's N-list structure <(4,8): 2>. Implementation and certification process are in [2].

B. *Outline of MRPrePost*

MRPrePost proposed by this article mainly uses three MapReduce[14,15] to parallelize PrePost, Fig. 3 describes the specific process.

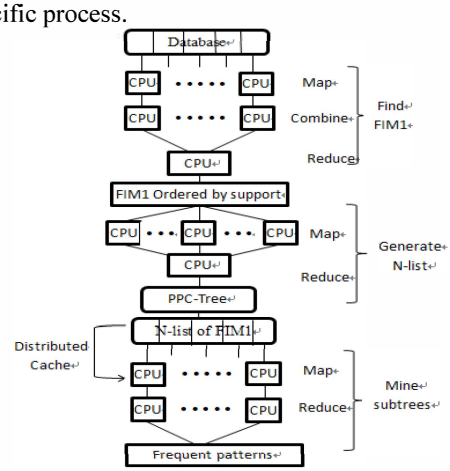

Figure 3. Flowchart of MRPrePost

Data mining is divided into three stages by MRPrePost algorithm:1. Statistic 1-frequent itemsets, similar to the process of word frequency statistics, raw data sets scattered on each worker nodes, each node independently perform map function, reduce function combined statistical results, and according frequent threshold cropped infrequent items; 2. Using a method similar to building FP-Tree to build PPC-Tree, traverse and generate N-List frequent one set; 3. The search space division, the N-list is distributed to each worker nodes, in order to ensure the cluster load balance, we make use of Round-Robin[8] to act as patition function, each node using the prefix pattern generates independently frequent itemsets.

C. *MRPrePost algorithm pseudo-code and analysis*

Step 1: Block the database level, this process can use the default file block policy of Hadoop, then the data block is called shard, which is allocated on each worker node. Count the number of items in each shard set in map stage,reduce merges output of map stage, and generates FIM1 according frequent support threshold δ, then descends FIM1 based on support of FIM1 to generate F-list. The algorithm pseudo code shows in Fig. 4.

```
Input: A transaction database DB and a minimum support δ
Out ut: F-list (the set of frequent 1-itemsets order by desc)
Procedure: Mapper (key,value = Ti)
for each item ai in Ti do
    Output (<key=ai, value=1>)
end
Procedure: Reducer (key=ai, value=S(ai))
sum = 0;
for each 1 in S(ai do
    sum += 1;
end
if(sum>=δ) Output (<key=ai, value=sum>); //output the FIM1
then call function Sort(FIM1); output the F-list
```

Figure 4. Pseudo code of parallel statistical 1-frequent itemsets and sort them

Step 2: Each map filters shard based on F-list, for each transaction of shard, sort frequent item based on sequence of F-list and output the result as value. Then, reduce constructs the compressed tree similarly constructing FP-Tree. Postorder traversal the tree to determine postorder and preorder the tree to determine preorder, then generate N-list of 1-frequent itemsets. Pseudo code is shown in Fig. 5 and Fig. 6.

```
Input: shard and F-list
Output: PPC-Tree
Procedure: Mapper (key, Ti)
for each Ti
    selec the frequent items in Ti sort out the according to the order of F-list
them
    produce a path [p|P] as the value to output <key, [p|P]>
end for
[PPC-Tree construction]
Procedure: Reducer (key, [p|P])
create the root of a PPC-Tree, and label it as "NULL"
for each [p|P]
    call insert_tree([p|P], Tr)
end for
[Fun tion insert_tree([p|P],Tr)]
if Tr has a childNsuch thatN.item-name=p.item-namethen
    increaseN's countby 1;
else
    create a new node N, with its count initialized to 1, and add it toT schildren-list;
    if P is nonempty then
        call insert_tree(P,N) recursively.
    end if
end if
```

Figure 5. Pseudo code of constructing PPC-Tree

```
Input: PPC-tree and L1, the set of frequent 1-itemsets.
Output:NL1, the set of the N-lists of frequent 1-itemsets.
Procedure N-lists_construction(PPC-tree)
first scan PPC-Tree to generate the post-order of each node
then create NL1, let NL1[k] be the N-list of L1[k].
for each node N of PPC-tree accessed by pre-order traversal do
    generate the pre-order of each node
    if (N.item-name=L1[k].item-name)then
        insert< (N.pre-order,N.post-order) :N.count> into NL1[k]
    end if
end for
```

Figure 6. Pseudo code of generating N-list of 1-frequent itemsets

Step 3: Save the N-list of 1-frequent itemsets in a distributed cache, to be shared by all map.While group N-list using Round-Robin[12], which is a modular arithmetic remainder strategy. Assuming the number of groups is m,the set of N-list of 1-frequent itemsets is $P = \{p_1, p_2, p_3, \cdots, p_k\}$, p_i represents a N-list of the i-th frequent itemsets, $\text{group}(p_i) = \text{i mod m}$. This can try to ensure that the cluster loads balance. According to this strategy, assuming m is 3, then:

g1 = {b → {< (4,8): 4 >},f → {< (2,1): 1 >,< (8,4): 1 >,< (9,7): 1 >}}
g2 = {c → {< (1,2): 1 >,< (5,6): 3 >},a → {< (3,0): 1 >,< (7,3): 1 >}}
g3 = {e → {< (6,5): 3 >}. Fig. 7 shows the pseudo code.

```
Function group(NL1,group_size)
for i=0 to NL1.size()
    groups[ i % group_size].add(NL1[i])
end for
```

Figure 7. Pseudo code of grouping N-list

```
Input: group_i and shared the NL1 to be saved in distributed cache
Output: frequent k-items F
for each mapper do
    for each NL_1 of groups[i] do
        call mining_fim_k(NL_1,Lk,NL1, δ)
    end for
end for
Function mining_fim_k(NL_k,NL1,δ)
for i = 0 to NL1 do
  if(NL_k.count ≥ |DB| * δ)
    F = F ∪ Lk
    if(NL_k.count ≥ NL1[i].count)
      Assume Lk = x1x2x3....xk, L[i].item = xk+1, supp(xk)>supp(xk+1)
      Lk+1 = Lk+L1[i] // Lk+1 = x1x2.....xk+1
      Lk = Lk+1
      compare N-list of NL_k with N-list of NL1[i],
      if NL_k.preorder<NL1[i].prepost&&NL_k.postorder>NL1[i].postorder
      then
      NL_k+1.N-list .add(<(NL1[i].prepost , NL1[i].postorder,count ):NL1[i].coun >)
      end if
    end if
  end if
end for
```

Figure 8. Pseudo code of mining frequent itemsets

Step 4: This step is a key in the whole algorithm. The pseudo-code shown in Fig. 8. Each map independently depth-first traversals every frequent item in the group assigned, until all frequent itemsets with the current prefixes sub-tree are located far. For b in group1, the current prefix is b, when c and e are added to the prefix sub-tree to generate 2-frequent itemsets {bc,be}(bf and ba are not frequent itemsets). To bc, be prefixed to continue the operation, eventually get all the frequent item sets on b. {b,bc,be,bce}. In the process of merging the prefix sub-tree, the paper made a modification to the original algorithm, for example, when b and c are combined, original algorithm generate PPCode < (b.preorder, b.postorder): c.count > when the condition is b.preorder < c.preorder && b.postorder > c.postorder. But, this article will generate PPCode as <(c.preorder, c.postorder): c.count> at the same condition. As a result of the depth-first and prefix sub-tree policy, we must promise the new item added and the current prefix sub-tree on the same path, necessary and sufficient condition is the new added item and the last item of the current prefix sub-tree are on the same path. This's the reason why we generate PPCode as <(c.preorder, c.postorder): c.count>. Finally, reduce combines output.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

Three sets of experimental environment for desktop computers equip ubuntu10.0, one as the master node, the other two as slaver node. The three computers have same configuration, CPU is AMD Athlon dual-core processor, clocked at 2.11GHz, memory size 2G. T10I4D100K and Pumsb act as experimental data. They can be downloaded at http://fimi.ua.ac.be/data/, the first is small and the second is large. We compare the runtime of three algorithms PrePost, PFP and MRPrePost when they are performed on the two datasets. Source code of PFP is provided by machout[16].

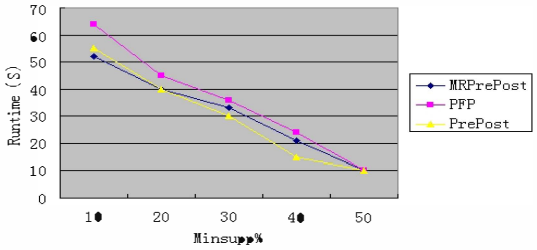

Figure 9. Comparison of the three algorithms on T10I4D100K

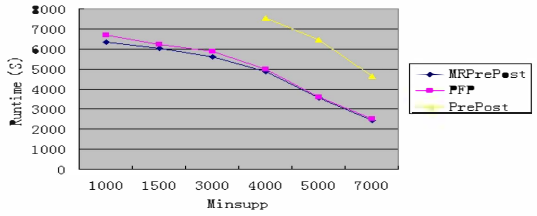

Figure 10. Comparison of the three algorithms on Pumsb

From experimental results shown by Fig. 9 and Fig. 10, we know the runtime will become shorter when support increases. It's evident. Fig. 9 also reflects performance of the parallel algorithm is not as good as PrePost on small dataset. The reason is each node needs to send message to others in clusters, but delay of network bandwidth is unpredictable, so I/O operation occupies main runtime, thus affecting the performance of the algorithm. Contrarily, PrePost has an advantage of data localization. But when the dataset is large, PrePost at a lower support threshold can not be performed due to memory overflow. The MRPrePost and PFP can still frequent itemsets mining, which is the purpose of PrePost algorithm parallelization, the main purpose of parallelization is to handle large dataset, which can not be

processed on standalone. The results in Fig. 10 reflects this view.

In addition, we can also know from Fig. 10, MRPrePost and PFP are significantly superior to PrePost, PrePost can not be calculated at a low support, but MRPrePost and PFP still have a good performance. At this time, communication time between the nodes in a distributed cluster is not a major factor, but data processing time. Parallelization is to use multiple processors independently to process small-scale data, so the algorithm superior performance compared to a stand-alone environment. The results also reflect, whether on a large or small datasets, runtime of MRPrePost is shorter than PFP's. Mainly because of sharing cache when MRPrePost conducts a depth-first strategy, which reduces the communication. Another reason is MRPrePost sub-tree to add a prefix entry time is linear, these measures save a lot of running time.

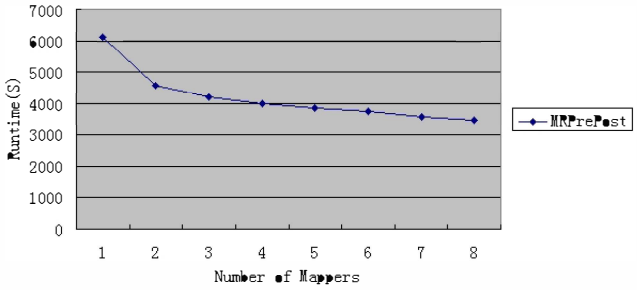

Figure 11. The number of different map affects the running time

In order to validatethe scalability of the algorithm, this paper dynamically adjusts the number of map on each node, the experimental result shown in Fig. 11. From the experimental results, With the increasing number of map, less and less time to run the algorithm, but with the increase in the map, the running time is reduced more slowly, tends almost unchanged due to the limited capacity of a single node, the calculation efficiency can not unlimited upgrade, number of map can not upgrade performance when increase to a certain number. Then the number of node should be increased instead of increasing the number of map in a single node. Through the above analysis can be drawn, MRPrePost proposed algorithm has good scalability.

V. CONCLUSION

Big Data mining is now a hot research question, MapReduce brings a simple distributed programming. The paper proposes a parallel algorithm based on MapReduce called MRPrePost on the basis of PrePost. and describes in detail the implementation of the algorithm.

In the face of mining large data sets, the parallelization is a good solution, experimental results prove this point. Through experiments, we compare the performance of PrePost and MRPrePost, and also with the PFP algorithm horizontal contrast, MRPrePost algorithm is the fastest of the three algorithms. The results meet our expectations, and analyzes the causes of the experimental results in the paper. Since MRPrePost algorithm is based on MapReduce, so we can simply add nodes to extend the algorithm without the need for any modifications to the algorithm. It can be concluded, MRPrePost proposed algorithm suitable for large-scale data sets for mining association rules.

ACKNOWLEDGMENT

This research is financially supported by the National Natural Science Foundation of China (60573145), Doctoral Fund of Ministry of Education funded project (200805610019), Guangzhou Municipal Science and Technology Project (2010Y1-C681) and the Fundamental Research Funds for the Central Universities.

*E-mail the corresponding author: zhaolab@126.com

REFERENCES

[1] Agrawal R, Srikant R. Fast algorithms for mining association rules[C]/Proc. 20th int. conf. very large data bases, VLDB. 1994, 1215: 487-499.

[2] Deng Z H, Wang Z H, Jiang J J. A new algorithm for fast mining frequent itemsets using N-lists[J]. Science China Information Sciences, 2012, 55(9): 2008-2030.

[3] Savasere A, Omiecinski E, Navathe S. An efficient algorithm for mining association rules in large databases. In: The 21th International Conference on Very Large Data Bases (VLDB'95), Zurich, 1995. 432–443.

[4] H.Mannila, H.Toivonen, and A.Verkamo. Efficient algorithm for discovering association rules. AAAI Workshop on Knowledge Discovery in Databases, pp.181-192, Jul.1994.

[5] Shi Yue-mei. Hu Guo-hua. A Sampling Algorithm for Mining Association Rules in Distributed Database[C]. In:2009 First International Workshop on Database Technology and Applications, 2009, 431-434.

[6] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[C]/ACM SIGMOD Record. ACM, 2000, 29(2): 1-12.

[7] Mobasher B, Dai H, Luo T, et al. Effective personalization based on association rule discovery from web usage data[C]/Proceedings of the 3rd international workshop on Web information and data management. ACM, 2001: 9-15.

[8] Moens S, Aksehirli E, Goethals B. Frequent Itemset Mining for Big Data[C]/2013 IEEE International Conference on Big Data,. IEEE, 2013: 111-118.

[9] Li N, ZengL, HeQ, etal. Parallel implementation of apriori algorithm based on MapReduce[C] ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel& Distributed Computing (SNPD), 2012 13th. IEEE, 2012: 236-241.

[10] Hammoud S. MapReduce network enabled algorithms for classification based on association rules[J]. 2011.

[11] Li H, Wang Y, Zhang D, et al. Pfp: parallel fp-growth for query recommendation[C]/Proceedings of the 2008 ACM conference on Recommender systems. ACM, 2008: 107-114.

[12] Apache hadoop. http://hadoop.apache.org/, 2013.

[13] Zaki M J, Parthasarathy S, Ogihara M, et al. Parallel algorithms for discovery of association rules[J]. Data Mining and Knowledge Discovery, 1997, 1(4): 343-373.

[14] Assunçao J, Fernandes P, Lopes L, et al. Distributed Stochastic Aware Random Forests--Efficient Data Mining for Big Data[C] /2013 IEEE International Congress on Big Data (Big Data Congress). IEEE, 2013: 425-426.

[15] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J].Communications of the ACM, 2008, 51(1):107-113.

[16] Apache mahout. http://mahout.apache.org/, 2013.