An Efficient Algorithm Based on MapReduce For Computing Frequent Item sets

JIN Da-wei, SONG Jie, LIU Bin, ZHAO Cheng
PLA University of Science & Technology
Nanjing, China
dave_nj@163.com, adamsongs@163.com, l.bin.2008@gmail.com, nanjzc@189.cn

Abstract--**As traditional method mining for association rules between items in large and grand data sets is inefficient. In this paper we present an efficient method called BPMRA which is based on mapreduce and partition. We have compared BPMRA algorithm based multi-node and partition based single node method and performed some experiments. It turns out that BPMRA possesses high parallelism good stability and scalability, especially suitable for mining for association rules in large and grand data sets.**

Keywords--association rules; frequent item sets; mapreduce; partition algorithm

I. INTRODUCTION

The mining for association is one of basic questions in data mining, and the core issues of the mining for association are that how to find the frequent item sets. The Apriori algorithm [1] is a classical algorithm of the mining for association, but there are also some inadequacies in it, for example, it requires multiple scans to the transaction data sets, and generates a larger set of candidates. Especially in the case of large data sets, it's very inefficient. So in this paper we presents a efficient algorithm for computing frequent item sets based on Partition[3] and MapReduce[2]–BPMRA(Based Partition and MapReduce Algorithm).

II. BASIC CONCEPT

A. Partition algorithm

For the Apriori algorithm's lack of data mining in the large-scale data sets, A.Savasere presents the partition algorithm; the core idea of the algorithm is to divide the original big data sets into series small parts that contains only small amount of data sets logically. According to the given minimum support, it can compute the frequent item sets of each part, merge the frequent item sets of all the parts, then get the global candidate frequent item sets, by scanning the original data sets, compute the support count of each item set. At last, the complete set of frequent item sets will be generated. The following two properties ensure the correctness of the partition algorithm.

Property Ⅰ If an item set is frequent in the whole situation, then it must be frequent in one part at least.

Property Ⅱ If an item set is not frequent in all parts, and then it must not be frequent in the whole situation.

The proof:

Set D: data sets; Divided D into n parts; D_i: ($1 \le i \le n$): part i; $|D|$: the total number of transactions in D; *min_sup*: minimum support; I^D: the item set I in D.

Property Ⅰ: Using reduction ad absurdum, set item set I is frequent in the whole situation, assume item set I is not frequent in all parts. It means $I^{Di}.count<|D_i|*min\_sup$ ($1 \le i \le n$), sum to n in both sides respectively, can get easily:

$$\sum_{i=1}^{n} I^{D_i}.count < min\_sup * \sum_{i=1}^{n} |D_i|$$

It can be got from the question set:

$$\sum_{i=1}^{n} I^{D_i}.count = I^{D}.count$$

And

$$min\_sup * \sum_{i=1}^{n} |D_i| = |D| * min\_sup$$

It can release $I^D.count<|D|*min\_sup$. It is in contradiction with the question set which is $I^D.count \geqslant |D|*min\_sup$. This is the end of proof.

Property Ⅱ: The item set I is not frequent in all parts, it means $I^{Di}.count<|D_i|*min\_sup$ ($1 \le i \le n$), sum to n in both sides respectively, we can get easily:

$$\sum_{i=1}^{n} I^{D_i}.count < min\_sup * \sum_{i=1}^{n} |D_i|$$

It can be got from the question set:

$$\sum_{i=1}^{n} I^{D_i}.count = I^{D}.count$$

And

$$min\_sup * \sum_{i=1}^{n} |D_i| = |D| * min\_sup$$

It can release $I^D.count<|D|*min\_sup$. This is the end of proof.

B. MapReduce programming model

MapReduce is a distributed parallel programming model which deals with large-scale data. It abstracted the progressing into one operating platform and two user-defined functions: Map and Reduce. The Map function is responsible for processing sub-data set and generate intermediate results; the Reduce function is responsible for reduction of the intermediate results and generates the final results. The operating platform is responsible for the scheduling, fault-tolerant, data-managing of Map and Reduce mission.

MapReduce largely reduce the difficulty of a distributed program written to deal with large-scale data. The data processing task is completed so that users need not care about the underlying details of the case. The application of MapReduce is very wide, for example, sequence, word count, Web connection diagram reversal, log analysis, inverted sort index build, document clustering, machine learning and the machine translation based on statistical.

Currently, in addition to the MapReduce framework of Google, Hadoop [4] of Apache also achieve the analogous MapReduce framework.

III. BPMRA ALGORITHM

A. Main idea

According to the partition algorithm, BPMRA algorithm is also divided into two stages:

Stage Ⅰ: According to the data sets D and appointed parts n it can be generated n Map mission, at the same time, start n' （n' is related to the computing node numbers j, commonly $n'=j*2$，$n'\le n$） Map mission parallel compute frequent item set, until the n missions are all over. Then a Reduce mission is started to merge the results of all Map missions, in order to generate the global candidate frequent item sets;

Stage Ⅱ: According to the data sets D, appointed parts m(m can be the same with n in stage Ⅰ)and the complete set of the global candidate frequent item sets C^G generated in stage Ⅰ, it can be generated m Map missions, at the same time, start m' Map missions parallel compute C^G the support count of each item set, until m missions are all over. Then start a Reduce mission to merge the support count of the same frequent item sets, the sum of the support count in C^G can be got. At last, according to given minimum support and the sum of the total number of transactions, it can be computed to get the complete set of frequent item sets L^G.

B. Algorithm description

In order to describe the algorithm expediently, define the sign below:

$p_i \subseteq D, p_i \cap p_j (i \neq j)$: p_i is the part of D;

C_k^G: The k item in the global candidate frequent item sets;

C^G: The complete sets of the global candidate frequent item sets;

$C_{p_i}^G$: is same with C^G, but each frequent item set all have its support count in the part p_i;

$L_k^{p_i}$: The k item frequent item set in part p_i;

L^{Pi}: The complete sets of the frequent item sets in part p_i;

L^G: The complete sets of the global frequent item sets;

min_sup: minimum support;

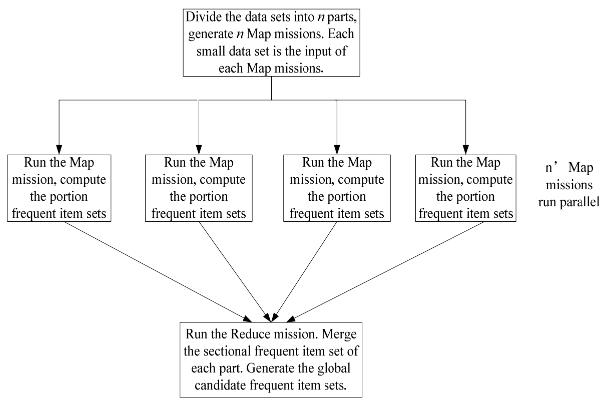
$|p_i|$: The total number of transactions in part p_i;

$|D|$: The total number of transactions in the data sets.

a. *Stage Ⅰ*

Input: data sets D, the minimum support *min_sup* and the part n;

Output: The complete sets of the global candidate frequent item sets C^G.


Divide the data sets into n parts, generate n Map missions. Each small data set is the input of each Map missions.
Run the Map mission, compute the portion frequent item sets
Run the Map mission, compute the portion frequent item sets
Run the Map mission, compute the portion frequent item sets
Run the Map mission, compute the portion frequent item sets
n' Map missions run parallel
Run the Reduce mission. Merge the sectional frequent item set of each part. Generate the global candidate frequent item sets.

Stage Ⅰ executing schematic diagram

In the Map mission of computing frequent item set, adopt vertical data format ({item: tidlist}), item is the name of the item, and the tidlist is the set of identifier affairs which concludes item. The advantage of this method is that the tid set of each k item set have the complete information to compute the support. So it needs not to scan the data sets when it computes the k+1($k\ge 1$) item set.

The description of Map mission：

```
read partition $p_i$ ($1\le i\le n$)
generate frequent 1 itemsets $L_1^{p_i}$ and every itemset with tidlist
for ($k$=2; $L_{k-1}^{p_i} \neq \varnothing$; $k$++) do
    for (int $i$=1; $i$<size of $L_{k-1}^{p_i}$; $i$++)do
        $l_1$=item $i$ of $L_{k-1}^{p_i}$
        for (int $j$=$i$+1; $j\le$ size of $L_{k-1}^{p_i}$; $j$++)do
            $l_2$=item $j$ of $L_{k-1}^{p_i}$
            if(($l_1[1]=l_2[1]$)∧($l_1[2]=l_2[2]$)∧…∧($l_1[k-2]=l_2[k-2]$)∧($l_1[k-1]<l_2[k-1]$)) then
                $c=l_1[1]\cdot l_1[2]\cdot l_1[k-2]\cdot l_2[k-1]$
                $c$.tidlist=$l_1$.tidlist∩$l_2$.tidlist
                if(|c.tidlist|≥$min\_sup$*| $p_i$ |) then
                    $L_k^{p_i} = L_k^{p_i} \cup \{c\}$
            endfor
        endfor
    endfor
get result $L^{p_i} = L_l^{p_i} \cup \bigcup_k L_k^{p_i}$ and transfer it to reduce task.
```

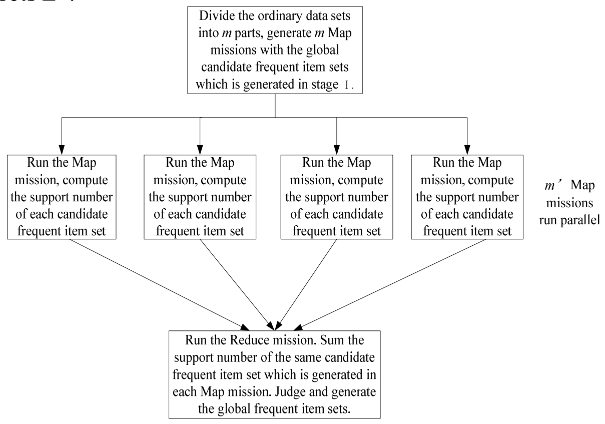
The description of Reduce mission：

```
for($k$=1; $L_k^{p_i} \neq \varnothing$; $k$++)do
    $C_k^G = \bigcup_{i-1}^{n} L_k^{p_i}$
endfor
get $C^G = \bigcup_k C_k^G$ and transfer it to second phrase's map task.
```

b. *Stage Ⅱ*

Input: data sets D, The complete sets of the global candidate frequent item sets C^G and parts m;

Output: The complete sets of the global frequent item sets L^G.


Divide the ordinary data sets into m parts, generate m Map missions with the global candidate frequent item sets which is generated in stage 1.
Run the Map mission, compute the support number of each candidate frequent item set
Run the Map mission, compute the support number of each candidate frequent item set
Run the Map mission, compute the support number of each candidate frequent item set
Run the Map mission, compute the support number of each candidate frequent item set
m' Map missions run parallel
Run the Reduce mission. Sum the support number of the same candidate frequent item set which is generated in each Map mission. Judge and generate the global frequent item sets.

Stage Ⅱ executing schematic diagram

The description of Map mission：

```
read partition $p_i$ ($1 \le i \le m$) and candidate frequent item sets $C^G$
for every transaction record $t \in p_i$ do
  for every candidates $c \in C^G$ do
  if t contains $c$ then
    c.count++
  endfor
endfor
get $C^G_{p_i}$ ( $c \in C^G$ with its support count ) and transfer it to reduce task
```

The description of Reduce mission：

```
for all candidates $c \in C^G$ do
  $c.count = \sum_{i=1}^{m} c.count(c \in C^G_{p_i})$
  if($c.count \ge min\_sup * |D|$) then
      $L^G = L^G \cup \{c\}$
endfor
```

C. Algorithm summary

There are several advantages below in the BPMRA algorithm:

1)Compare with the Apriori algorithm, the BPMRA algorithm uses vertical data format when it computes the frequent item set, avoid multiple scanning to the data sets, the BPMRA algorithm only need to scan twice, raises the efficiency greatly.

2) The BPMRA algorithm uses the computing method of “Big into small, small parallel”; the stability is well when the amount of data increases and the computational complexity rise.

3) The BPMRA algorithm possesses high scalability and parallelism, by means of constantly adding compute nodes to extend, and through extending to raise the compute parallelism.

4)The BPMRA algorithm requires low with machines, only need several ordinary PC machine, and will be able to have a strong computing power。

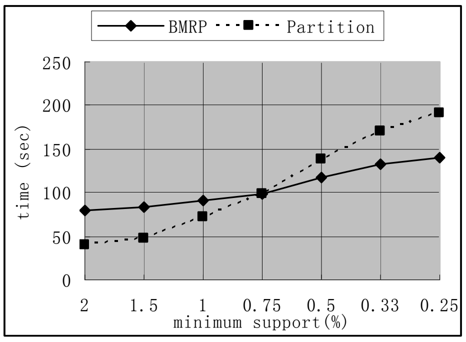
Certainly, there are also several disadvantages in the BPMRA algorithm:

1)Because of using vertical data format when it computes the frequent item sets, it involves two big collection seeking intersection operator when it seeks the support of the item set, so it may need to spend slot of time, currently the time complexity is $O(n+m)$ (n、m are two collection for intersection).

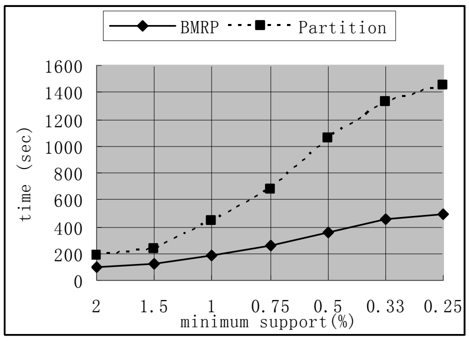
2) Because the master node needs to coordinate individual compute nodes, it will produce certain network time consuming, especially it will consume more when the quality of network is poor. But generally this situation can be avoided in the internal LAN.

IV. EXPERIMENT AND ANALYSIS

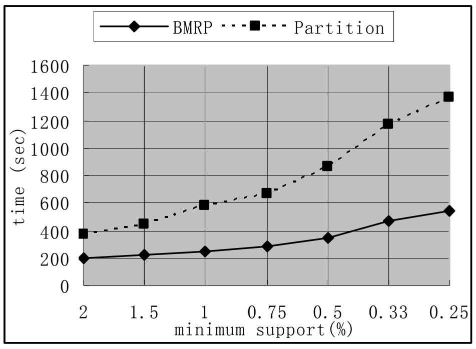
In order to further verify the efficiency of the BPMRA algorithm, we have done some experiments in this paper. The experiment environment is: There are four DELL PC machine. The CPU frequency is 2.4GHz, the memory is 1GB, and the system is Fedora9 Linux; one machine is master node, responsible for the coordination of compute nodes; the other three machines are slave nodes, responsible for computing. The algorithm is written based on Hadoop version 0.20.0 and Java. The experiment data is generated by IBM Data Generator; there are three data sets altogether: T5.I2.D1000K, T5.I2.D5000K, and T20.I4.D1000K. We will compute respectively by the BPMRA algorithm and the Partition algorithm based on one node, the divided parts are the same. The following graphs are the comparison of the result:



Graph 1. The result of T5.I2.D1000K data set



Graph 2. The result of T5.I2.D5000K data set



Graph 3. The result of T20.I4.D1000K data set

We can conclude by the experiment that the BPMRA algorithm is not well when the amount of data is not big and the computational complexity is not difficulty, for example, when the data set is T5.I2.D1000K and the support is greater than 0.75%(Graph 3)，the BPMRA algorithm is not better than the Partition algorithm which is based on one node. This is mainly caused by the network time-consuming between each computing nodes and master node, and less time consumed by the calculation itself. With the increasing of the amount of data and the rising of the computational complexity, the time consumed by the calculation will more than the time consumed by the network(Graph 4 and Graph 5). With the lowering of the minimum support, the increase of the BPMRA algorithm is more stable, but the increase of the Partition algorithm based on one node is very obvious. In contrast, the BPMRA algorithm shows a clear advantage.

V. CONCLUTION

In this paper, as the traditional computing algorithm for frequent item set is inefficient when the amount of data is big, we present the BPMRA algorithm, this algorithm runs in several computing nodes and achieve to compute highly parallel. We have done several experiments; the result shows that the BPMRA algorithm is viable and efficient when it is mining frequent item sets whether the huge amounts of data or high computing complexity data.

REFERENCES

[1] AGRAWAL R, IMIELINSKI T, SWAMI A. Mining Association Rules Between Sets of Items in Large Databases[C]//In Proceedings of 1993 ACM SIGMOD International Conference on Management of Data. Washington, DC: ACM Press, 1993:207-216

[2] JEFFREY DEAN, SANJAY GHEMAWAT. MapReduce: Simplified Data Processing on Large Clusters [C]//In Proceedings of the 6th Symposium on Operating System Design and Implementation. San Francisco: Google Inc., 2004:10-23.

[3] SARVASERE A, OMIECINSKY E, NAVATHE S. An Efficient Algorithm for Mining Association Rules in Large Databases[C]//In Proceedings of the 21st International Conference on Very Large Databases. San Francisco: [s.n.], 1995:432-444.

[4] Hadoop project [EB/OL]. (2009). http://hadoop.apache.org.

[5] TOM WHITE. Hadoop: The Definitive Guide [M]. California：O'Reilly Media, Inc.，2009.

[6] JIAWEI HAN, MICHELINE KAMBER. Data Mining Concepts and techniques, Second Edition [M], 2008.