MREclat: an Algorithm for Parallel Mining Frequent Itemsets

Zhigang Zhang, Genlin Ji*, Mengmeng Tang
School of computer Science and Technology
Nanjing Normal University
Nanjing, China
zzg22936@sina.com, glji@njnu.edu.cn, dreamtang1016@gmail.com

***Abstract*—Algorithm Eclat is a classical algorithm for mining frequent itemsets, which is based on vertical layout databases. It is greatly different from those algorithms based on horizontal layout databases, such as algorithm Apriori and FP-Growth. In order to improve the efficiency of mining frequent itemsets from massive datasets, parallel algorithm MREclat based on Map/Reduce framework is presented. The algorithm also overcomes the problem of memory and computational capability insufficient when mining frequent itemsets from massive datasets. In this paper, the idea of MREclat is introduced and the performance of the algorithm is studied. The experimental results show that algorithm MREclat has high scalability and good speedup.**

Keywords-Frequent Itemset Mining; Parallel Mining Algorithm; Map/Reduce; Eclat

I. INTRODUCTION

The mining of frequent itemsets is a fundamental and essential problem in many data mining applications [1]. Algorithms for mining frequent itemsets can be basically classified into two types: one is algorithms based on horizontal layout dataset such as algorithm Apriori [2] and FP-Growth [3]; the other is algorithms based on vertical layout database such as Eclat [4]. Eclat takes advantage over algorithms based on horizontal layout database. It saves much time as it doesn't need to over scanning the whole database [4-6].

Facing with the "Big Data" problem [7], using parallel or distributed algorithms to mine frequent itemsets from massive data is feasible and practical. Currently proposed parallel algorithms are CD (Count Distribution), CaD (Candidate Distribution) and DD (Data Distribution) [8]. These algorithms are based on Apriori and suffer major weaknesses at some respects of communication and synchronization [9]. However, these algorithms are instructive to the parallel frequent itemsets mining. Zaki [10] has proposed four parallel algorithms based on DEC Memory Channel technology, but, they are not suit for mining massive data sets because we can not put so much data into the shared memory. Except Zaki's work, as far as we know, there is little other work about how to parallelize Eclat algorithm. In this paper, we describe a new parallel algorithm based on Map/Reduce framework [11]. Experimental results show that our algorithm has high scalability and good speedup.

The rest of the paper is organized as follows: Section 2 introduces background knowledge. Section 3 discusses our design idea about MREclat algorithm. We present our experimental results in section 4. Section 5 concludes this paper.

This work is supported by Key Programs of Natural Science Foundation of Jiangsu Province of China (No.BK2011005).

Correspondence Author: Genlin Ji ; email: glji@njnu.edu.cn

II. BACKGROUND KNOWLEDGE

A. Frequent Itmeset

The concept of frequent itemsets mining was introduced by Agrawal in 1993. It can be formally stated as: Let $I = \{i_1, i_2,\ldots,i_n\}$ be a set of n distinct items. Let $D = \{t_1, t_2,\ldots, t_m\}$ be a set of m transactions, each transaction consists of a unique transaction identifier and a set of items, called itemset, which is a subset of set I.

Definition 1: Let I_1 I, the support of I_1 is the ratio of transactions which contain I_1. So, support(I_1) = $\|\{t \in D|\ I_1\ t\}\| / \|D\|$.

Definition 2: If the support of I_1 is larger than a minimum support threshold—*min_sup* which is set by user, then I_1 is frequent itemset.

The task of frequent Itemset mining is to find all the frequent itemsets from database. And it is the first and key step of association rule mining. With the help of frequent itemsets, we can generate rules to find useful information between items.

B. Database Layout

There are two layout formats of the target dataset for association mining: the horizontal and the vertical layout. The horizontal layout database is also called the transactional database. It consists of a list of transactions, each transaction has an identifier (TID) followed by a list of items in that transaction. This format imposes some computation overhead during the support counting step [4]. In the vertical (or inverted) layout, database consists of a list of items, each item followed by the list of tids (also called *tid-list*). So, in the vertical layout database, it is quite easy to find frequent itemsets, because we just need to count the number of tids of each item.

C. Algorithms Eclat

Algorithm Eclat has two search approaches [4]: *Bottom-up* and *Hybrid-Traversal*. As *Hybrid-Traversal* is restrict to only identifying the maximal frequent subsets, so in this paper, we only introduce the *Bottom-up* approach which can be used to find all frequent itemsets.

Input: $F_k = \{I_1, I_2, \ldots, I_n\}$ // cluster of frequent k-itemsets.
Output: Frequent l-itemsets, $l > k$.
Bottom-Up (F_k) {
1. for all $I_i \in F_k$
2. $F_{k+1} = \varnothing$;
3. for all $I_j \in F_k$, $i < j$
4. $N = I_i \cap I_j$;
5. if $N.sup \geq min\_sup$ then

978-1-4799-3261-0/14 $31.00 © 2014 IEEE
DOI 10.1109/CBD.2013.22

6. $F_{k+1} = F_{k+1} \cup N$;
7. end
8. end
9. end
10. if $F_{k+1} \neq \varnothing$ then
11. Bottom-Up(F_{k+1});
12. end
13.}

This process looks extremely like algorithm Apriori, but it differs from Apriori in step 5. In Apriori, we need to scan the whole database to compute the support of itemset *N*. However, Eclat just counts the number from *tid-list* of *N*.

D. Map/Reduce Framework

Map/Reduce [11] is proposed to support distributed computing. It's a typical implement of share-nothing architecture that can efficiently process massive dataset. Map/Reduce framework firstly partitions user's data into equal sized blocks with some replicas and distributes these blocks evenly to the distributed file system automatically. Then, Map/Reduce runs a same Map task on each of these blocks and automatically collects the outputs. Then, it uses a partition task to send these outputs to different computing nodes and runs a same Reduce task on these nodes. Finally, it collects the output of all Reduce tasks, the whole output is the result we need. Map/Reduce framework frees users from data distributing, task scheduling, fault tolerating and task communicating. Users only need to implement the map and reduce functions which are the key points of Map and Reduce task.

III. IMPLEMENT OF MRECLAT

Algorithm MREclat consists of three steps: The first is the initial step, we get all frequent 2-itemsets and their *tid-lists* from transaction database in this part; the second is the balanced group step, we partition frequent 1-itemsets into groups; the third is the parallel mining step, the data got in the first step redistributed to different computing nodes according to the group their prefix belong to. Each node run a improved Eclat to mine frequent itemsets. Finally, MREclat collects all the output from each computing node and formats the final result.

A. The Initial Step

The vertical layout, however, has a drawback. Examination of small itemsets tends to be costlier than when the horizontal layout is employed [10]. For example, a database with 1,000,000(1M) transactions, 1,000 items and a average of 10 items per transaction has *tid-list* of average size 10,000. To find frequent 2-itmsets, we have to intersect each pair of items, which requires $C_{1000}^2 \times (2 \times 10,000) \approx 10^9$ operations. On the other hand, in the horizontal format we simply need to form all pairs of the items appearing in a transaction and increment their count, require only $C_{10}^2 \times 1,000,000 = 4.5 \times 10^7$ operations. So, in this step, we take a Map/Reduce job to gather the occurrence count of all 2-itemsets directly from original transactional database. Its pseudo-code is described as follows:

// Input: *key* is the identifier of the transaction and *value* is the items of the transaction.

map (*key*, *value*) {

1. Get all items from value and sort them by lexicographic order, the sorted items are put into an array called *items* and the length of items is *n*.
2. for (*i* = 0; *i* < *n*-1; *i*++)
3. for (*j* = *i*+1; *j* < *n*; *j*++)
4. output(*items*[*i*] ∩ *items*[*j*], *key*); // form a new 2-itemset and out it with the tid
5. end
6. end
7.}

In case that this map function outputs too much intermediate data, we may a use a additional Map/Reduce job to static the frequent 1-itemsets, and in the line 1, we only keep those items who are frequent.

The input of reduce function is a 2-itemset and the list of transaction ids which contain this 2-itemset. After the reduce task, we get all frequent 2-itemsets and their *tid-lists*. The results are stored in the inverted layout database.

reduce (*key*, *value*) {
1. *tid_list* = ∅;
2. *sum* = 0;
3. for each *tid* in *value*
4. *sum*++;
5. *tid_list* = *tid_list* ∪ *tid*;
6. end
7. if *sum* / *transnum* ⩾ *min_sup* then
8. output (*key*, *tid_list*);
// only output frequent 2-itemsets
9. end
}

B. Balanced Group

In order to achieve a good load balance, we partition the items into balanced groups. Considering that the only computation in Eclat algorithm is intersection, and the number of intersect operations between frequent itemset *a* and *b* equals to the length of *tid_list* of *a* plus the length of *tid_list* of *b*. we use w_i (*w* means weight) to denote the operation complexity of getting all frequent itemsets prefixed (prefix means the first item in a itemset) by the item *A* (*i* is the index of *A* in frequent 1-itemset), which is defined as:

$$w_i = \log(n-1) + \log(\sum_{j=0}^{j<n} len_j) \quad (1)$$

Where n is the number of frequent 2-items prefixed by *A*. we sort those frequent 2-itemsets prefixed by *A* in the lexicographic order, and use len_j to denote the length of list of the *j*-th frequent 2-itemset.

After we compute all the *w* for each frequent 1-itemset, we sort the tuple consisted by the frequent 1-itemset and its weight by the value of weight in a descending order. Then MREclat uses a greedy strategy to divide the tuples into

groups, each group associated with an id. The group strategy is shown as follows:

// Input: *a* is the array of tuple which consisted with <item, weight>; M is the number of topples; N is the number of groups.

// Output: each item of *groups* is a group of frequent 1-itemset

```
divide (a[], groups[], M, N)
1. if (M<=N) then
2.    for each tuple  t in a
3.       add t.item to groups[i];
4.    end
5. else
6.    Initial array sum[] of length N with 0
7.    for (i=0; i<M; i++)
8.       add the item of a[i] to groups[i];
9.       sum[i]+= a[i].weight;
10.   end
11.  for (i=M; i<N;i++)
12.      find the index from sum with the minimum value
         and assign the index value to k;
13.      add the item of a[i] to groups[k];
14.      sum[k]+= a[k].weight;
15.   end
16. end
```

C. *The Redistribution and Parallel Ecalt Step*

In this step, firstly, we want to redistribute 2-itemsets with the same prefix into a same computing node. For example, a 2-itemset set "ab, ac, ad, bc, bd, cd…" and their *tid-lists*, "ab", "ac" , "ad" have the same prefix "a". "bc", "bd" have the same prefix "b". Then, we will send "ab", "ac" , "ad" accompany with their *tid-lists* to one computing node A, "bc", "bd" accompany with their *tid-lists* to the computing node B. Secondly, we compute frequent itemsets on different nodes separately.

This step takes a Map/Reduce job. The redistribution work is done in the map stage and its pseudo-code is described as follows:

// Input: *key* is a 2-itemset; *value* is the *tid-list* of the 2-itemset.

```
map (key, value) {
1. Get the prefix p from key;
2.  output (a, <key, value>);
3.}
```

In addition, between the Map and Reduce stage, we take the Shuffle procedure to send the output of Map stage to assigned nods with the help of group strategy. The partition function in the procedure is designed as follows:

// The input <*key*, *value* >is the output of map function; *key* is the prefix.

```
getPartition(key, value){
 1. for ( i=0; i< N; i++)
 2.    if  key is in groups[i] then
 3.       return i;  // we use the index as the id of the group
                        the key belong to here.
 4.    end
 5. end
}
```

In the *getPartition* function, we associate the prefix with the group it belongs to. After the Shuffle procedure, all the 2-itemsets with their prefix will certainly go to the computing node.

In the reduce function, we get all frequent items with the prefix *a*. Its pseudo-code described as follows:

//Input: *key* is the prefix; *value* is the 2-itemsets with its prefix.

```
reduce (key, value) {
1. a =key;
2. Get all C_a2 from value; //C_a2are 2-itemsets prefixed by a
3. Bottom-Up (C_a2)
4. Output all the frequent itemsets with the prefix a;
5.}
```

We collect all the output of reduce function, and get all the frequent itemsets.

IV. PERFORMANCE EVALUATION

Extensive experiments were conducted to assess the performance of the proposed algorithms. All the experiments were performed on a Hadoop cluster of 0.20.2 cluster of 10 nodes, where each node contains a Intel(R) Xeon(R) E5620 2.40GHz CPU, 32GB RAM, and a 500GB hard disk running CentOS 6.4. The algorithm is implemented in Java and the JDK version is 1.6.0_23. Both synthetic and real datasets were used in the experiments. The synthetic datasets were generated by the IBM Dataset Generator. The number of distinct items is 1,000 and the average length of transactions is 10. Real datasets WebDocs from FIMI were used [13]. It is a collection of web html documents which contains exactly 1,692,082 transactions with 5,267,656 distinct items. The maximal length of a transaction is 71,472.

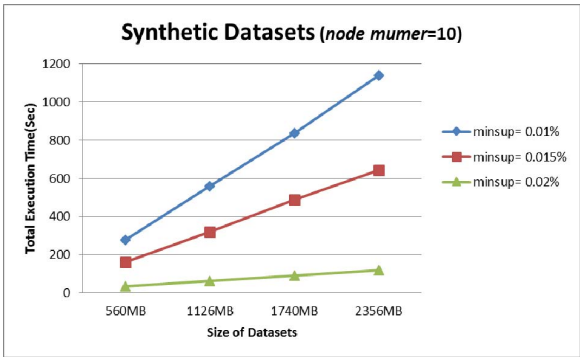

Figure 1. Scalability in different size of datasets

The result of varying minimum supports on mining synthetic datasets is shown in Figure 1. We use four datasets with size of 560MB, 1126MB, 1740MB and 2356MB. To test the performance of MREclat, we assign 3 relatively small values to *min_sup* which are 0.01%, 0.015% and 0.02%. The experiences were run on 10 computing nodes. Figure 1 shows that with the increase of size on each node, the execution time increase in a linear way; and with the decrease of minimum support threshold, the running time also increase accordingly.

We may conclude that MREclat has a performance on the synthetic dataset.

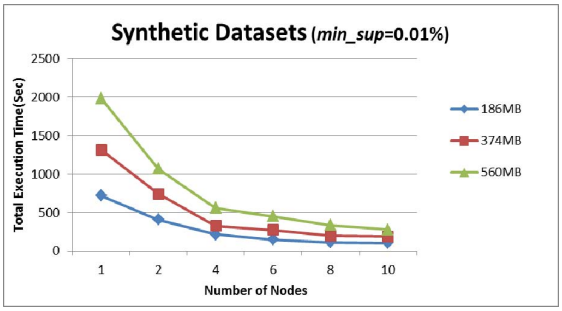

Figure 2. Execution time on different number of nodes

Figure 2 shows the execution time of another group of datasets with size of 186MB, 374MB, 560MB, these datasets separately have the amount of transactions of 1M, 2M and 3M. the experiments choose a minimum support value of 0.01%. From Figure 2, we can see that with the increase of numbers of computing node, execution time on three datasets decreases accordingly. The run time on dataset with 1M transactions can not decrease any more when the computing node number reaches 8, because the communication time will increase with the increase of computing nodes.

Figure 3 shows that MREclat achieves near linear speedup ratio on real dataset.

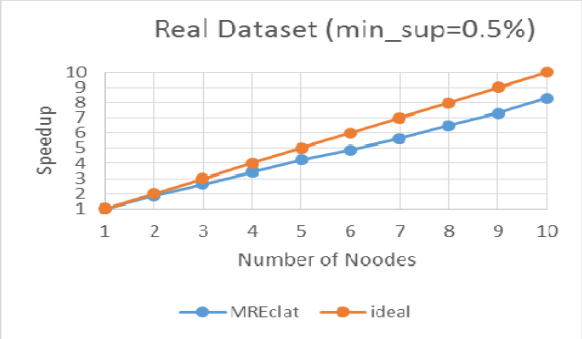

Figure 3. Speedup on WebDocs dataset

V. CONCLUSION

We proposed a parallel algorithm MREclat which based on Map/Reduce framework in this paper. MREclat converts the transactional dataset firstly, and divide the frequent 1-itemsets into balanced groups, then redistribute records of converted dataset according to their prefixes. Records with prefixes in the same group will be distributed to the same computing node. MREclat uses the improved Eclat to process data with the same prefix. As shown in the experimental results, MREclat has high scalability and good speedup ratio.

ACKNOWLEDGMENT

This work is supported by Key Programs of Natural Science Foundation of Jiangsu Province of China (No.BK2011005).

REFERENCES

[1] R. Agrawal, T. Imieliński,and A. Swami, "Mining association rules between sets of items in large databases," Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM, Vol. 22. No. 2, 1993, pp. 207-216.

[2] R. Agrawal, R. Srikant, "Fast algorithms for mining association rules," Proceedings of the 20th International Conference on Very Large Data Bases, VLDB. Vol. 1215, 1994, pp. 487-499.

[3] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," ACM SIGMOD Record. ACM, Vol. 29, No. 2, 2000, pp. 1-12.

[4] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New algorithms for fast discovery of association rules," Proceedings of the 3th International Conference Knowledge Discovery and Data Mining, vol.97, 1997, pp. 283-286.

[5] Z. F. Li, X. F. Liu, andX. Cao. "A study on improved Eclat data mining algorithm," Advanced Materials Research, vol. 328, 2011. pp. 1896-1899.

[6] B. Kotiyal, A. Kumar, B. Pant, R. H. Goudar, S. Chauhan, and S. Junee, "User behavior analysis in web log through comparative study of Eclat and Apriori," Proceedings of 7th International Conference on Intelligent Systems and Control (ISCO). IEEE, pp. 421-426, 2013.

[7] D. Howe, M. Costanzo, P. Fey, T. Gojobori, L. Hannick, and W. Hide, et al., "Big data: The future of biocuration", Nature, Vol. 455, No. 7209, 2008, pp. 47-50.

[8] M. Ashrafi, T. Zaman, S. David, and S. Kate, "ODAM: an optimized distributed association rule mining algorithm," Distributed Systems Online 1541-4922, IEEE, Vol. 5, No. 3, 2004, pp. 1-18.

[9] X. Y. Yang, Z. Liu, and Y. Fu, "MapReduce as a programming model for association rules algorithm on Hadoop," Proceedings of 3rd International Conference on Information Sciences and Interaction Sciences (ICIS). IEEE, pp. 99-102, 2010.

[10] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "Parallel algorithms for discovery of association rules," Data Mining and Knowledge Discovery, Vol. 1, No. 4, 1997, pp. 343-373.

[11] J. Dean, S. Ghemawat,"MapReduce: simplified data processing on large clusters," Communication of ACM, Vol. 51, No. 1, 2008, pp. 107-113.

[12] M. J. Zaki, K. Gouda, "Fast vertical mining using diffsets," Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2003, PP. 326-335.

[13] L. Claudio, O. Salvatore, P. Raffaele, et al. WebDocs: a real-life huge transactional dataset. http://fimi.ua.ac.be/data/webdocs.pdf.