A MapReduce-Based Parallel Mining for Frequent Itemset

Nian Liu¹ and Jinxu Guo²

¹ Department of Information and Communication Engineering, Wuhan University of Technology, Faculty of Information Engineering, 122 LuoShi Road, Wuhan, China. Email: 893780601@qq.com
² Department of Information and Communication Engineering, Wuhan University of Technology, Faculty of Information Engineering, 122 LuoShi Road, Wuhan, China. Email: 1003580870@qq.com

Abstract. Frequent itemset mining is the most important step of association rule mining. Due to the large size of datasets, many parallel mining methods have been introduced to divide datasets or to distribute mining processes, which improve the efficiency of mining. In this paper, we propose a parallel algorithm of PrePost⁺ based on MapReduce (we call our parallel algorithm MR-PrePost⁺) to mine frequent itemsets. In our parallelization approach, the Reduce task performs a set of independent mining tasks, which eliminates data dependencies and computational dependencies between machines. The experimental results show that our MapReduce-based parallel algorithm MR-PrePost⁺ performs better than PFP in running time, which is one of the best frequent itemset mining parallel algorithms.

1. Introduction

Frequent itemset mining, first proposed by Agrawal, Imielinski, and Swami (1993), has become one of the most important technologies for data mining. A great deal of research work has been done and a lot of algorithms have been proposed. However, designing efficient algorithms is still one of several key issues yet to be solved in frequent itemset research, especially for big data environment.

PrePost⁺ is a high-performance algorithm for mining frequent itemsets proposed in recent years. It employs N-list to represent itemsets and uses a set-enumeration search tree to find frequent itemsets. Especially, it employs an efficient pruning strategy, named Children-Parent Equivalent, which greatly reduces the search space. PrePost⁺ has good performance when dealing with small datasets. But, the performance of PrePost⁺ starts to decrease as soon as the dataset size increases.

Working with big data usually requires parallel and distributed mining methods to improve the efficiency of mining. Hadoop is a well-known framework for processing big data in parallel, it provides the environment of distributed storage and computation across clusters of computers. Therefore, this paper is going to propose an efficient parallel algorithm based on MapReduce to mine frequent itemsets for big data.

In summary, the contributions of this paper are as follows:
(1) We design and implement a Hadoop-based parallel algorithm of PrePost⁺, called MR-PrePost⁺, which uses the MapReduce to mine frequent itemsets in parallel.
(2) We compare MR-PrePost⁺ with a state-of-art parallel algorithm (PFP) in terms of running time. The experimental results show that the MR-PrePost⁺ algorithm has better performance.

The rest of this paper is organized as follows. In Section 2, we study the background and related work for frequent itemset mining. Basic principles of PrePost⁺ are analyzed in Section 3. We describe MR-PrePost⁺ algorithm in Section 4. Experiment results are shown in Section 5 and conclusion is given in Section 6.
2. Related Work

Frequent itemset mining was initially proposed for market basket analysis in order to solve the problem of mining association rules. A great deal of research work has been done and a lot of algorithms have been proposed.

Most of the previously proposed algorithms for mining frequent itemsets can be divided into two groups: the Apriori-like method and the FP-Growth method. The Apriori-like method uses an iterative approach to discover frequent itemsets. It generates candidate itemsets of length \( k+1 \) by using frequent itemsets of length \( k \), and then scans the dataset to count the supports of these candidate itemsets. One typical algorithm is the Apriori algorithm proposed by Agrawal, Imielinski, and Swami (1993). Previous studies indicate that the Apriori-like method has two drawbacks. The one is that it repeatedly scans the dataset. The another is that it generates a large set of candidates.

In order to avoid scanning the dataset repeatedly, Zaki and Gouda (2003) proposed the Eclat algorithm which is a vertical algorithm. The Eclat converts the dataset into a vertical data format, and computes the supports of itemsets by using the intersection of transactions, which need not to scan the entire dataset. The Eclat has proven to be a very efficient algorithm. But when the dataset is large, converting the dataset to a vertical data format may not be possible.

In order to avoid generating candidates, Han, Pei and Yin (2000) proposed the FP-Growth algorithm. The FP-Growth compresses the dataset into a compact data structure, called FP-tree, and employs the divide-and-conquer strategy to mine frequent itemsets. The FP-Growth does not generate candidates and only needs to scan the dataset twice, its performance is significantly better than the performance of the Apriori-like method. However, the FP-Growth needs to recursively build FP-tree to mine frequent itemsets, it will create large amounts of FP-trees when the dataset is large, the computing memory may not be enough in that case.

In recent years, an efficient algorithm for mining frequent itemsets, called PrePost, has been proposed by Deng, Wang and Jiang (2012). The PrePost algorithm employs a data structure named N-list to represent itemset, and discovers frequent itemsets through the intersection of N-lists. The PrePost can directly find frequent itemsets without generating candidates in some cases by using the single path property of N-list. However, the PrePost belongs to the Apriori-like approach, it still has too many candidates.

In order to avoid the problem of PrePost, Deng and Lv (2015) proposed PrePost+, an improved algorithm for the PrePost. The PrePost+ algorithm uses the set-enumeration search tree to find frequent itemsets and proposes an efficient pruning strategy (Children-Parent Equivalent) which greatly prunes the search space and reduces the candidates. MapReduce is a parallel computing model for big data, and many studies have been done to parallelize traditional well-known frequent itemset mining algorithms based on MapReduce. The first MapReduce-based parallel frequent itemset mining algorithm is the PFP (Li, Wang, Zhang et al., 2008), which is still one of the best parallel algorithms. The MRApriori is another MapReduce-based parallel algorithm that parallelizes the Apriori algorithm (Yahya, Hegazy and Ezat, 2012). The Dist-Eclat is a parallel algorithm for the Eclat (Moens, Aksehirli and Goethals, 2013).

3. Basic Principles

PrePost+ employs PPC-tree to generate the N-list. PPC-tree is a tree structure which looks like an FP-tree, each node in PPC-tree consists of item name, count, children-list, pre-order, and post-order. The pre-order and the post-order respectively represent pre-order and post-order rank of node. Figure 1 shows the PPC-tree generated from the dataset in Table 1 (the minsup is 2). PrePost+ uses PP-code to represent node information, which is \( ((N, \text{pre-order}, N, \text{post-order}) : \text{count}) \). PrePost+ employs N-list to represent itemset information, N-list is a sequence of PP-codes. Figure 2 shows the N-list of frequent items.

PrePost+ counts itemsets’ supports by the intersection of N-lists. For PP-codes in two N-lists of two itemsets, if they satisfy the ancestor-descendant relationship, a new PP-code can be created for the N-list of the new itemset which is generated by these two itemsets. The ancestor-descendant relationship of two PP-codes is that \( N_1 \) is an ancestor of \( N_2 \) if and only if \( N_1, \text{pre-order} < N_2, \text{pre-order} \) and
PrePost+ calculates the support for all PP-codes in the N-list of itemset to get the itemset frequency. Figure 3 shows the intersection process of generating the N-list of \( bk \) from the N-list of \( b \) and \( k \).

### Table 1. A transaction dataset.

| ID | Transactions | Ordered frequent items |
|----|--------------|------------------------|
| 1  | \( a, b, c, e, f, g, p \) | \( b, c, a, e, f, g \) |
| 2  | \( a, b, c, e, k, q \) | \( b, c, a, e, k \) |
| 3  | \( a, b, c, g, k, l \) | \( b, c, a, k, g \) |
| 4  | \( b, c, e, k, o \) | \( b, c, e, k \) |
| 5  | \( b, c, d \) | \( b, c \) |
| 6  | \( b, c, m \) | \( b, c \) |
| 7  | \( a, e, f, g, h \) | \( a, e, f, g \) |
| 8  | \( a, f, k, n \) | \( a, k, f \) |

Figure 1. The PPC-tree generated from the dataset in Table 1.  
Figure 2. The N-list of frequent items in Table 1.

PrePost+ directly discovers frequent itemsets using set-enumeration search tree, and reduces the search space with an efficient Children-Parent Equivalence pruning strategy. Figure 4 shows the details, due to the support of \( \{afg\} \) and \( \{efg\} \) is equal to the support of \( \{fg\} \), we need not to create child nodes \( a \) and \( e \) for \( \{fg\} \), the nodes connected by solid lines represent the created child nodes, the nodes connected by dotted lines do not need to be created, and the support of itemsets they represent can be obtained by the support of node \( b \) and \( c \). Therefore, in some cases, PrePost+ can directly discover frequent itemsets without generating candidates. PrePost+ continues to create child nodes for nodes \( b \), \( c \) and \( k \) until there are no child nodes to be create. In this process, PrePost+ discovers all frequent itemsets suffixed with \( fg \). Similarly, PrePost+ does the same for the other 2-itemsets, and discovers all frequent itemsets.

### 4. Mr-Prepost+: The Proposed Method

The MR-PrePost+ proposed in this paper uses two MapReduce tasks to complete frequent itemset mining. Figure 5 depicts the three steps of MR-PrePost+. The first MapReduce task counts the supports of all items and stores frequent items in \( F_i \). The second MapReduce task generates group-dependent transactions and mines frequent itemsets for each group-dependent shard. There is a crucial step before the second MapReduce task, it is dividing all items in \( F_i \) into groups, this step can complete on a single computer because \( F_i \) is small.
Figure 3. The N-list of $bk$ in Table 1.

Figure 4. PrePost+ discovers frequent itemsets.

Figure 5. MR-PrePost+ framework.

Figure 6. Grouping items for $F_1$ in Table 1.

4.1. Parallel Counting

Counting is a classical feature for MapReduce. For each item, say $a_i$ in a transaction $T$, the Mapper outputs a key-value pair $\langle key = a_i, value = 1 \rangle$. The MapReduce infrastructure collects the set of corresponding values of $a_i$, and feeds the Reducer with these key-value pairs. The Reducer counts the supports of $a_i$ and checks for frequency. Algorithm 1 presents the pseudo code of the first MapReduce task.
4.2. Grouping Items
This paper considers the load of Hadoop cluster and proposes a load balancing grouping strategy. The load balancing grouping strategy first calculates the load value of each item according to the equation (1), in which \(\text{index}\) represents the negative index of item in \(F\), \(\text{support}\) represents the support of item and \(x\) is the base of the logarithm, and then successively adds the item to the group with the least load value. Figure 6 shows the grouping items process of \(F\) in Table 1.

\[
\text{item\_load} = \log_x (\text{index} + \text{support})
\]

4.3. Parallel PrePost+
This step is the key in our MR-PrePost+ algorithm. We convert transactions into several group-dependent transactions, therefore, local PPC-tree built from different group-dependent transactions are independent during generating N-lists of itemsets. For each transaction \(T\), the Mapper outputs a series of key-value pairs, like \((\text{key}, \text{value}) = \{a_i[0], a_i[1], ..., a_i[L]\}\), based on the grouping items, and ensures the independence of \(\text{gid}\) data. MapReduce infrastructure collects the set of corresponding values of \(\text{gid}\) and feeds the Reducer with these key-value pairs, in which the \(\text{value}\) is a group-dependent shard. For each group-dependent shard, the Reducer constructs the local set-enumeration search tree, during this process, it outputs found frequent itemsets, which are suffixed with the group items. Algorithm 2 presents the pseudo code of this MapReduce task. Figure 7 shows the process of using our MR-PrePost+ algorithm on the dataset in Table 1.

```
algorithm 1 Parallel counting
1: procedure Mapper (key, value = \(T_i\))
2:    for each item \(a_i\) in \(T_i\) do
3:        Call Output((\(a_i["1"]\));
4:    end for
5: end procedure

6: procedure Reducer(key = \(a_i\), value = \(S(a_i)\))
7:    \(C \leftarrow 0;\)
8:    for each \(a_i\) in \(S(a_i)\) do
9:        \(C \leftarrow C + 1;\)
10:   end for
11:   if \(C \geq \text{minsup}\) then
12:       Call Output((null, \(a_i[0] + C\))
13:   end if
14: end procedure

algorithm 2 Parallel PrePost+
1: procedure Mapper (key, value = \(T_i\))
2:    Load \(G\) - list; \(\backslash\text{created in step 2}\)
3:    Create HashTable \(H - map\), map for \(\text{gid}\) with items
4:    hasGroup \(\leftarrow \phi\)
5:    \(a[\backslash 1] \leftarrow \text{Split}(T_i);\)
6:    for \(j = |T_i| - 1\) to 0 do
7:        \(\text{gid\_now} \leftarrow \text{getHashNum}(H - \text{map}, a[j]);\)
8:        if \(\text{gid\_now}\) not in hasGroup then
9:            Call Output((\(\text{gid\_now}, a[0] + a[1] + ... + a[j]\));
10:        hasGroup \(\leftarrow \text{hasGroup} \cup \{\text{gid\_now}\};\)
11:    end if
12: end for
13: end procedure

14: procedure Reducer(key = \(\text{gid}, value = DB_{\text{gid}}\)
15:    Load \(G\) - list;
16:    groupItems \(\leftarrow G\) - list\_{\text{gid}};
17: \(\backslash\text{change PrePost+ algorithm, let it mine frequent}\)
18: \(\backslash\text{items suffixed with items in the groupItems}\)
19: \(F \leftarrow \text{Call PrePost + (DB_{\text{gid}}, groupItems)};\)
20: for each frequent itemset \(v\) in \(F\) do
21:    Call Output((null, \(v + \text{supp}(v))\);
22: end for
23: end procedure
```
5. Experimental Results

In this section, we report our experimental results on the performance of our MR-PrePost+ algorithm compared with PFP algorithm in terms of running time. In the experiments, we performed it on a Hadoop (version 2.7.3) cluster of 4 computers with 2.8 GHz Core quadcore processor, 8 GB RAM, we evaluate the performance using two datasets, kosarak and webdocs (from http://fimi.cs.helsinki.fi/data/), with sizes of 32.1M and 1515.5M, respectively. Figure 8 and Figure 9 show the experimental results of running MR-PrePost+ and PFP on mentioned datasets.

As can be seen in Figure 8 and Figure 9, the runtime of both algorithms increases with the decrease of the minimum support. In two datasets, MR-PrePost+ performs better for all minimum support, and PFP consumes longer runtime.

6. Conclusions

In this paper, we propose MR-PrePost+, a parallel algorithm for mining frequent itemsets for big data environment, based on MapReduce framework. In addition, considering the load of Hadoop cluster, we propose a load balancing grouping strategy to group items, which promotes the efficiency of the entire mining task. The experimental results show that our MR-PrePost+ performs better than PFP, which is one of the best frequent itemset mining parallel algorithms.
7. References

[1] Agrawal R, Imielinski T and Swami A. 1993. Mining association rules between sets of items in large databases. In: The 1993 ACM SIGMOD International Conference on Management of data (SIGMOD’93), Washington. 207-216

[2] Han J W, Pei J and Yin Y W. 2000. Mining frequent itemsets without candidate generation. In: The 2000 ACM SIGMOD International Conference on Management of data (SIGMOD’00), New York. 1-12

[3] Zaki M J and Gouda K. 2003. Fast vertical mining using diffsets. In: The 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD’03), Washington. 326-335

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

[5] Deng Z H and Lv S L. 2015. PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via Children–Parent Equivalence pruning. *Expert Systems with Applications*, **42**(13): 5424-5432

[6] Li H, Wang Y, Zhang D, Zhang M and Chang E. 2008. PFP: Parallel FP-Growth for query recommendation. In: The 2008 ACM Conference on Recommender Systems. 107-114

[7] Yahya O, Hegazy O and Ezat E. 2012. An efficient implementation of Apriori algorithm based on Hadoop-MapReduce model. *International Journal of Reviews in Computing*, **12**: 59-67

[8] Moens S, Aksehirli E and Goethals B. 2013. Frequent itemset mining for big data. In: The 2013 IEEE International Conference on Big Data, Silicon Valley, CA. 111-118