Received: April 24, 2018

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Performance Improvement of PrePost Algorithm Based on Hadoop for Big Data

Yassir Rochd [1]* Imad Hafidi [1]

[1] *Laboratory of Process Engineering and Optimization of Industrial Systems, National School of Applied Science, Hassan 1st University, Khouribga, Morocco*
* Corresponding author's Email: y.rochd@gmail.com

Abstract: With the blasting growth in data, uptake data mining techniques to mine association rules, and then find useful information hidden in large data has become ever more important. Several existing data mining techniques often through mining frequent itemsets draw association rules and get to relevant knowledge, but with the rapid arrival of the era of big data, traditional data mining algorithms have been impossible to meet large data's analysis needs. Lately, the PrePost algorithm has been suggested, a new algorithm for mining frequent itemsets based on the idea of N-lists. PrePost in most cases outperforms other present state-of-the-art algorithms. In mind of this, we present the HPrePostPlus algorithm. A better version of PrePost based on Hadoop, that utilization a HashMap to traverse effectively the PPC tree, and improve the process of creating the N-lists related with 1-itemsets. We combine also the characteristic of Hadoop with a view to process large data. Experience has demonstrated that HPrePostPlus algorithm is greater than the state-of-the-art methods in terms of performance and scalability.

Keywords: Frequent itemset mining, PrePost, Hadoop, Big data.

1. Introduction

The past ten years has seen the outstanding growth of Internet contact technology particularly mobile Internet and detector system to perceive and obtain details. Organizations from industry, administration, and academia possess and store large volumes of data with enormous importance. The ability value of big data [1] cannot be uncovered by simple gathering or statistical analysis, currently referring to big data. Advanced big data analytics and applications require special technologies to successfully cope with massive amounts of data. Data mining techniques [2] are now outline care from the practitioners of all data related industries for this purpose. The aim of data mining is to look into data by searching and interpreting unforeseen trends or patterns and then verify the results with the detected patterns applied to new subsets. Since data collected from various data sources is often a series of solitary data, correlation analysis has hence become a major basis for data mining and big data science [3]. Association rules mining [4] was suggested to find out certain interesting correlation relationships among the data itemsets. Thus, frequent itemset mining [5] is an essential stage in the process of association rule mining.

Most of the suggested algorithms for frequent itemsets can be grouped into Apriori method [6] and FP-growth method [7]. The Apriori mode scans the database to find frequent itemsets by generating a large set of a candidate. Whereas, FP-growth mode does the scan twice to mine frequent itemsets without generating a candidate. The FP-growth uses FP-tree data structure to store database and utilize a divide-and-conquer strategy to find frequent itemsets, which is much more efficient than Apriori mode.

Lately, the algorithm for mining frequent itemset, PrePost [8, 9], has been suggested. It's based on the notion of PPC tree (Pre-order Post-order Code tree), which is an FP-tree type structure. PrePost operates as follows. A tree building algorithm is accustomed to construct a PPC-tree. Then, the N-lists are generated. Each component of this N-lists is associated with a 1-itemset in the tree. An N-list of

k-itemset is a compact form of transaction ID list (TID list). A divide and conquer strategy is then used for mining frequent itemsets.

These algorithms working on single computer have shown good achievement in handling with small amount of data. Nevertheless, traditional procedures has faced considerable defiance when computing power and remembrance space are restricted to big data era. Some applications and attempts have been made to mine frequent itemset from massive data by using parallel computing technologies.

Parallel programming frameworks are divided into two categories: memory sharing and distributed architectures (share nothing). Although it's easier to make algorithms implemented, the scalability of parallelism on memory sharing framework is not satisfactory enough [10]. Message passing interface (MPI) [11], a common framework for scientific distributed computing, takes the advantage of memory locality. Thanks to certain MPI advantages in iterative computation, some researchers apply it to mine frequent itemset [12]. And yet, its drawbacks are its high communication load due to data exchanges between different computer nodes and the lacking of fault tolerance.

MapReduce [13], a framework embedded in Apache Hadoop to process large amounts of distributed data in parallel, was designed to support distributed computing in a cloud computing paradigm, turning out to be an efficient platform for parallel data mining of large scale datasets.

A number of distributed frequent itemset mining methods [14, 15-16-17-18-19] which are usually simple extensions of a sequential method using distributed data processing frameworks.has been proposed,

Although the existing distributed methods can partially solve the limit on scalability, they still face some problems. First, they do not have good scalability due to workload skewness. The existing distributed methods divide the search space of patterns (i.e., enumeration tree) to be explored into multiple pieces (a subtree) and assign each piece to each machine. Each subtree of the enumeration tree tends to have different size, i.e., different amount of workload. In particular, the distributed methods based on Eclat and FP-Growth have this problem noticeably. As a result, the existing methods tend not to improve performance proportionally to the number of machines used. Second, they do not have good scalability due to high network communication overhead. The existing methods usually perform frequent itemset mining by redistributing intermediate data via network. This approach could largely degrade the performance and scalability as the amount of data transferred among machines increases.

In this paper, we propose an improved version of PrePost, based on Hadoop itemset mining method for big data called HPrePostPlus.
HPrePostPlus solves the above problems, and so, can find frequent patterns on much larger datasets compared with the existing distributed methods.

Unlike FP-tree-based approaches, HPrePostPlus algorithm does not build additional trees on each iteration; it mines frequent itemset directly using the N-list concept. The efficiency of HPrePostPlus is achieved because: (i) N-lists are much more compact than previously proposed vertical structures, (ii) the support of a candidate frequent itemset can be determined through N-list intersection. This process is more efficient than finding the intersection of TID lists because it avoids unnecessary comparisons.

For solving the problem of network communication overhead, HPrePostPlus broadcasts only frequent itemset F_k via network, which size is much smaller than that of intermediate data. As a result, HPrePostPlus shows much higher performance than the state-of-the-art MapReduce based methods.

The main contributions of this paper are the following:
(i) We propose HPrePostPlus, a scalable Hadoop based method for frequent itemset mining that has no intermediate data, and small network communication.
(ii) We use HashMap to traverse efficiently the PPC tree and to speed up the process of creating the N-lists associated with frequent 1-itemsets.

Experiments show that HPrePostPlus outperforms the state-of-the-art MapReduce-based methods in terms of speed and scalability.

The rest of the paper is organized as follows. Section 2 presents the basic concepts. Section 3 outlines survey of related works. Section 4 gives proposed approach. Then, section 5 gives results and discussion and talk at last in section 6 shows the conclusion.

2. Preliminaries

2.1 Frequent itemset mining

Suppose that $I = \{I_1, I_2, \ldots, I_m\}$ is an itemset composed of m items. A database D consists of a series of transactions. Each transaction is a subset of I and has a unique label denoted by TID. A set of items is referred to as an itemset. An itemset that

contains k items is a k-itemset. The occurrence frequency of an itemset is the number of transactions that contain the itemset. Given an itemset X, the support number of X is the number of transactions in D that contain X. If the support number of X is greater than or equal to the specified minimum support threshold, then the itemset X is labelled as a frequent itemset. The purpose of frequent itemset mining is to find all frequent itemset in a given database.

2.2 Hadoop and mapreduce

Encouraged by benefits of parallel execution in the distributed environment, the Apache Foundation came up with open source platform, Hadoop, for faster and easier analysis and storage of different varieties of data [21]. HDFS and MapReduce programming model are two integral parts of it. Google File System gave birth to HDFS (Hadoop Distributed File system), which mainly deal with storage issues. Contrary to the RDBMs, it follows WORM (write-once read-many) model in order to split large chunk of data to smaller data blocks then join them to the free node available [22]. Stored Input data blocks are kept in more than one node in order to achieve high performance and fault tolerance.

MapReduce which is inspired by Google's MapReduce [13] is known to be a linearly adaptable programming model. It contains two main functions a map ( ) function and a reduce ( ) one, both of which work in a synchronous manner in order to operate on one set of key value pairs, and that, to produce the other set of key value pairs. These functions are equally valid for any size of data irrespective of the degree of the cluster. MapReduce uses the feature known as data locality to collocate the data with the compute node, so that data access is fast. It follows shared nothing architecture which eliminates the burden from the programmer of thinking about failure. The architecture itself detects failed map or reduce task and assigns it to a healthy node.

2.3 PrePost algorithm

PrePost algorithm [8,9] 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 frequent 1-itemsets (FIM1). In the mining process, the database does not require rescanning, 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 shows 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 pre-order
5. Post-order: represent order of node when post-order.

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 .PPC-Tree's main purpose is to construct N-list liking shown by Fig. 2, then find all the frequent itemsets 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 FIM 1, 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 items in 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 pre-order traversal and post-order traversal, set pre-order and post-order of each node and establishes N-list of I-frequent itemsets.

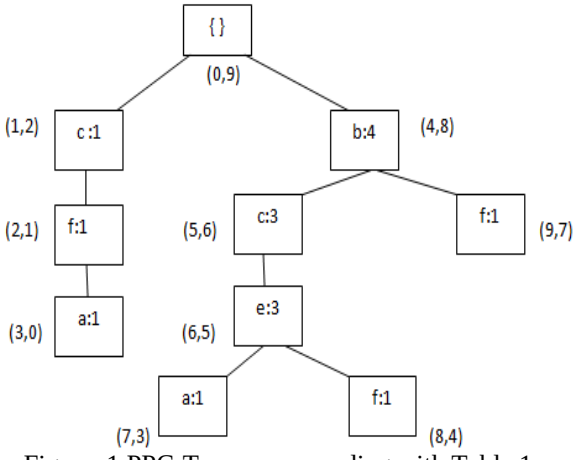

Figure. 1 PPC-Tree corresponding with Table 1

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

b → <(4,8) : 4>

c → <(1,2) : 1>-----<(5,6) : 3>

e → <(6,5) : 3>

f → <(2,1) : 1>-----<(8,4) : 1>-----<(9,7) : 1>

a → <(3,0) : 1>-----<(7,3) : 1>

Figure. 2 N-list of frequent 1-itemsets

4. Mining frequent itemsets based on N-list using the method liking Apriori Algorithm.
5. Table 1 shows a transaction database, corresponding to Fig. 1 for PPC-Tree, assuming the minimum support is 3.

3. Related work

The precedent proposed algorithms for mining frequent itemsets classed into three groups, generate candidate, frequent pattern growth and Hybrid approach.

Recently, three types of structure have been suggested for representing itemsets: Node-list [23], N-list [8], and Node set [24, 25], to facilitate the mining of frequent itemsets. They are founded on a prefix coding tree, which save the sufficient information about frequent itemsets. Node-list and N-list is founded on a PPC-tree, which is a prefix tree with every node encoded by its pre-order number and post-order number. The N-list (or Node-list) of an itemsets is a set of nodes in the PPC-tree. The solely difference between N-list and Node-list lies in that the Node-list of an itemset consists of descendant nodes while its N-list consists of ancestor nodes.

N-lists (or Node-lists) have two important specifications: First, the support of an itemset is the sum of counts registering in the nodes of its N-list (or Node-list). Second, the N-list (or Node-list) of a (k + 1)-itemset can be formed by joining the N-lists (or Node-lists) of its subset with length of k with linear computation complexity. Compared to the vertical structures for representing itemsets, such as diffset, the size of N-list or Node-list is much smaller.

Compared with FP-tree [7], they are more simple and flexible. Therefore, the algorithms based on N-list or Node-list demonstrate high efficient and outperform the existing classic algorithms, such as Eclat and FP-growth. Compared with Node-lists, N-lists have two advantages. The first one is that the length of the N-list of an item-set is much smaller than the length of its Node-list. The other one is that N-lists have property called single path property, which can be utilized to directly mining frequent itemsets without generating candidate itemsets in few cases. These make that PrePost [8], the mining algorithm based on N-lists, is high effective than PPV [19], the mining algorithm based on Node-lists. Recently, PrePost has been improved by utilizing various very effective pruning techniques [9]. Although N-list and Node-list are efficient structures for mining frequent itemsets, they need to include pre-order and post-order number, which is memory-consuming.

More of the variants of PrePost algorithm were developed to employ for small size of data in a single machine system. With the apparition of big data for last some years, single-machine system shows to be incapable to treat big data. A great number of researches has been realising for frequent pattern mining in multi-machine environment, i.e., distributed computing environment [10]. Hadoop is one of the important distributed computing frameworks, which is adopted by many researchers for frequent pattern mining in big data.

There have been proposed a lot of MapReduce-based methods for finding frequent itemsets on large-scale data [14, 15, 16, 17, 18, 19, 20]. Table 2 summarizes the characteristics of the major existing MapReduce-based methods [26], SPC, BigFim and PFP.

Table 2. Resume of the characteristics of the main methods based on MapReduce

Methods	Intermediate data size	Speed of support conting	Scalability
SPC	Small	Slow	Good
BigFIM	Large	Fast	Bad
PFP	Large	fast	Bad

There have been suggested a lot of MapReduce based methods for finding frequent itemsets on large-scale data [14, 15, 16, 17, 18, 19, 20]. Table 2 resumes the properties of the major existing MapReduce-based methods [26], SPC, BigFim and PFP.

There are various Apriori-based methods on the MapReduce framework. Lin et al. [16] proposed three distributed Apriori methods on MapReduce: SPC, FPC, and DPC. SPC iteratively performs the candidate generation and testing steps as a MapReduce round. At the k-th iteration, every mapper reads a partitioned database, generates candidate itemsets, and calculates support counts of them for the partitioned database. Then, the reduce step aggregates the support counts of the same candidate itemset and tests them against minsup. The result of the reduce step is broadcasted for being utilized in the next iteration. FPC reduces the number of MapReduce rounds by utilizing the map function that processes the candidate k-itemsets, (k+1)-itemsets, (k+2)-itemsets together in a single MapReduce round. DPC dynamically gathers candidate itemsets of consecutive multiple lengths to be processed by the mappers in a single MapReduce round according to the number of candidate itemsets. By comparing these Apriori-based methods, HPrePostPlus performs support counting much faster from the intersection of N-lists, avoiding needless comparisons.

Moens et al. [17] proposed BigFIM, which is a hybrid approach between Apriori and Eclat. It first finds frequent itemsets of short lengths using the distributed algorithm of the Apriori approach and generates conditional databases, i.e., equivalence classes whose prefixes are the itemsets previously found. After that, it performs the sequential Eclat algorithm on each conditional database independently in each machine. Compared with SPC, its support counting is fast by using an efficient sequential algorithm, Eclat. However, since the sizes of conditional databases are quite different with each other, i.e., there is workload skewness, mining task tends to fail due to lack of memory in a certain machine, or takes too long time due to the machine having the largest workload. In addition, it generates a large amount of intermediate data and incurs large network communication overhead during generating conditional databases. Therefore, BigFIM tends to show bad scalability as the number of machines increases.

PFP [20] and its variations [27] are the distributed methods based on the FP-Growth approach. They first project an input database and build independent FP-Trees, which are kind of conditional databases, using the projected databases. Then, they perform frequent itemset mining on each FP-Tree independently in each machine. Like BigFIM, PFP and its variations can find frequent itemsets from FP-Trees by using an efficient sequential algorithm, FP-Growth.

However, similarly with BigFIM, PFP and its variations have several drawbacks such as workload skewness, large intermediate data size, and large network communication overhead. Therefore, they tend to fail due to lack of memory, and show bad scalability. Comparing to BigFIM and PFP, HPrePostPlus shows much better scalability as the number of machines increases, since it does not intermediate data, and small network overhead.

Liao et al. [18] presented a MRPrePost algorithm a parallel algorithm adapted for mining big data based on Hadoop platform under Mapreduce, the algorithm uses N-list data structure, which enhances PrePost by way of adding a prefix pattern. An enhanced PrePost algorithm with hadoop platform suggested by Thakare et al.[26] based on N-list data structure and improved by implementing compact PPC tree.

Comparing to the precedent versions of PrePost based on hadoop [18, 19], general tree method is utilized to traverse the tree PPC tree. The general tree method utilized linked list which is an implementation of the List interface. It provides sequential access and effective for inserting and deleting items in the list. But, it became less efficient while accessing items in the list. In HPrePostPlus algorithm, general tree method is implemented with HashMap which is an implementation of the Map interface. It provides an efficient and fast for locating value based on the key. It does not save the item in the order and it provides an easy way to access and delete items on the basis of key value pairs. The HPrePostPlus algorithm uses also a HashMap to improve the process of creating the N-lists associated with 1-itemsets and combines the features of Hadoop in order to process large data.

4. HPrePostPlus algorithm

4.1 HPrePostPlus design

The HPrePostPlus algorithm is a data mining algorithm for frequent itemsets which uses N-list data structure to represent the itemsets. All the required information of the itemsets is to be saved by N-list. Efficacy of the HPrePostPlus algorithm is achieved by using the method of generating frequent itemsets without generation of candidate itemsets. The HPrePostPlus algorithm uses also a HashMap to

improve the process of creating the N-lists associated with 1-itemsets from the PPC tree and combines the features of Hadoop in order to process large data.

The HPrePostPlus algorithm is implemented with Hadoop to enhance its performance. We store the big transactional data in Hadoop distributed file system (HDFS) of Hadoop framework, and multiple partitions of data are distributed across cluster nodes. The complete algorithm is divided into three phases, which are described as follows:

Phase 1: The data file is given as input to the Hadoop. It divides whole input file into fixed size blocks called shard, and map it to the different DataNode in Hadoop cluster. DataNode counts the number of items in each block. Then, apply support count and arrange all items in the descending order. Then, reducer combines data from all DataNode and generate list called F1 list. The F1 list is mapped to different DataNode with the distributed cache. The main input file is rearranged according to F1 list. Here, uses the concept of distributed cache to compare two files with Map. Then, generate the list of frequent 1-itemset by descending called FL1 list. The Pseudo-code for the complete process of phase 1 is presented in Fig. 3.

Algorithm of parallel statistical 1-frequent itemsets and sort them

Input: D = Transactional Dataset, minsup= Minimum Support Threshold, I = item
Ouput**:** FL_1= the set of frequent l -itemsets by descending order
1. Procedure Mapper(key,value=T)
2. For each item I in T do
3. Output (key=I, value=1)
4. End
5. End Procedure
6. Procedure Reduce (key=I,value=S(I))
7. Sum=0
8. For each 1 in S(I) do
9. Sum=Sum + 1
10. End
11. If (sum>=minsup) Output(key=I,value=Sum)
12. Then Call function Sort(Fim1)
13. End if
14. Output (FL_1)
15. End Procedure

Figure. 3 Pseudo code of parallel statistical 1-frequent itemsets and sort them

Phase2: All the non-frequent items are removed from the original input data, which reduces the data size. Only the FL1 list is passed as input to reduce network communication overhead, and generate a compressed tree called PPC tree similar like FP tree. Post-order traversal effectively the tree to determine post-order and preorder the tree to determine preorder, and then use the HashMap created to speed up the process of creating the N-lists associated with 1-frequent items. The Pseudo-code is shown in Figs. 4 and 5.

Algorithm of constructing PPC-Tree and corresponding HashMap

lnput: shard and FL_1
1. Output: PPC –Tree, H_1 the HashMap of FL_1
2. Create H_1
3. Procedure Mapper(key, value=T)
4. for each Transaction T in D do
5. select the frequent item in T and sort out them according to the order of FL_1
Let the sorted frequent-item list in T be a path [p|P] as the value to output <key, [p|P]>
where p is the first element and P is the remaining list.
6. End for
7. Procedure Reduce (key, [p|P])
8. Create root of a PPC –tree, and label it as “null”
9. For each [p|P]
10. Call insert_tree([p|P],T).
11. End for
12. Scan PPC-tree to generate the post-order of each node
13. Return H_1
14. Function insert_tree([p|P],T)
15. if T has a child N such that N.item-name = p.item-name
16. Then increase N’s count by 1;
17. Else create a new node N, with its count initialized to 1, and add it to T’s children-list;
18. If P is nonempty then call insert tree(P,N) recursively.
19. End if
20. End if

Figure. 4 Algorithm of constructing PPC Tree

Algorithm of generating N-List of 1-frequent itemsets from the HashMap

Input: PPC-tree and FL_1 the set of frequent 1-itemsets, H_1 the HashMap of FL_1
Output: NL_1, the set of the N-lists of frequent 1-itemsets.
1. Procedure N-lists construction (R, H_1)
2. Let C=(R.pre-order,R.post-order,R.count)
3. Add C to H_1 [R,name] count by C.count
4. For each child in R.children do
5. N-lists construction(child)
6. End for

Figure. 5 Pseudo code of generating N-List of 1-frequent itemsets

Phase3: The N-lists of 1-frequents itemsets NL_1 are distributed over cluster nodes as a group of lists for loading balance on the cluster. For example from PPC-tree of Fig. 2:

NL_1G1 = {b → {< (4,8): 4 >}, f → {< (2,1): 1 >, < (8,4): 1 >, < (9,7): 1 >}},

NL_1G2 = {c → {< (1,2): 1 >, < (5,6): 3 >}, a → {< (3,0): 1 >, < (7,3): 1 >}},

NL_1G3 = {e → {< (6,5): 3 >}.

We store thus the N-list of 1-frequent itemsets in a distributed cache, which is shared among all the nodes. Each node independently depth-first traversals every frequent item in the group assigned, until all frequent item sets with the current prefixes sub-tree are located far. For example, for b in group 1, 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 prefix subtree merge process, normally when b and c are combined, the original algorithm generates PPCode <(b.preorder, b.postorder): c.count> when the condition is b.preorder <c. preorder && b.postorder> c.postorder. But, this paper will generate PPCode as <c.preorder, c.postorder): c.count> in the same condition. As a result of the depth-first and prefix subtree policy, we must promise the new added element and the current prefix subtree on the same path, necessary and sufficient condition is the new element added and the last element of the current prefix subtree are on the same path.This's the reason why we generate PPCode as <(b.preorder, b.postorder): c.count>.Finally, reduce combines output. The Pseudo-code of phase 3 is presented in Fig. 6.

Algorithm of mining frequent itemsets

Input: NL_1G [i]=group i of NL_1 and shared the NL_1 to be saved in distributed cache
Output: FL_k =frequent k-itemsets F
1. For each mapper do
2. For each NL_l of NL_1G [i] do
3. Call mining_fim_k(NL_l, FL_k,NL_1, minsup)
4. end for
5. end for
6. Function mining_fim_k(NL_k, NL_1,minsup)
7. For i = 0 to NL_1 do
8. If (NL_k.count >= |DBI|* minsup)
9. F=F U L_k
10. If (NL_kcount >= NL_1[i].count)
11. Assume L_k= $x_1x_2….x_k$, L[i].item = x_{k+1} , supp(x_k) > supp(x_{k+1})
12. FL_{k+1} = FL_k+FL_1[i] // FL_{k+1} = $x_1x_2….x_kx_{k+1}$
13. FL_k= FL_{k+1}
14. Compare N-list ofNL_k with N-list of NL_1[i]
15. If (NL_k.preorder < NL_1 [i].postorder && NL_k . postorder > NL_1[i].preorder|)
16. NL_k+l.N-list.add (NL_1[i].prepost, NL_1[i].postorder.count):NL_1[i].count)
17. End if
18. End If.
19. End if
20. End for

Figure. 6 Pseudo code of mining frequent itemsets

Table.3 The properties of datasets used in experiment

Dataset	Size	Transactions	Items	Average length
T10I4D100K	3.8 MB	100.000	870	10
T40l10D100K	14 MB	100.000	1000	40

5. Experiments

In this section, the algorithm HPrePostPlus was compared with its original version PrePost [8], three state-of-the art algorithms negFin [25], MRPrePost [18] and the well-known PFP [20]. We evaluated the speed performance by analyzing the running time and scalability.

The experiments were conducted on a Hadoop cluster of 3 nodes where each node contains Intel® Core ™ i5- 3230M CPU@2.60GHz processing units and 12.00GB RAM . HDFS was used for storage of input dataset and output frequent itemsets. The datasets T10I4D100K and T40l10D100K are used for experiments. These two real datasets were presented at the first IEEE ICDM workshop on Frequent Itemset Mining (FIMI' 03) [28]. Table 3 shows the detail of the two datasets.

The running time with different support degree for dataset T10l4D100K and T40l10D100K is shown in Figs. 7 and 8 separately. The x-axis denotes the support degree and y-axis represents the running time. The support degree grows from 0.1% to 0.5%.

The experimental results with respect to the runtime experiments are presented in Figs. 7 and 8. Figure 7 reflects that the performance of the parallel algorithms HPrePostPlus, MRPrePost and PFP, is not as good as negFIN and 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, negFIN and PrePost has an advantage of data localization. But when the dataset is large, the sequential methods negFIN and PrePost at a lower support threshold cannot be performed

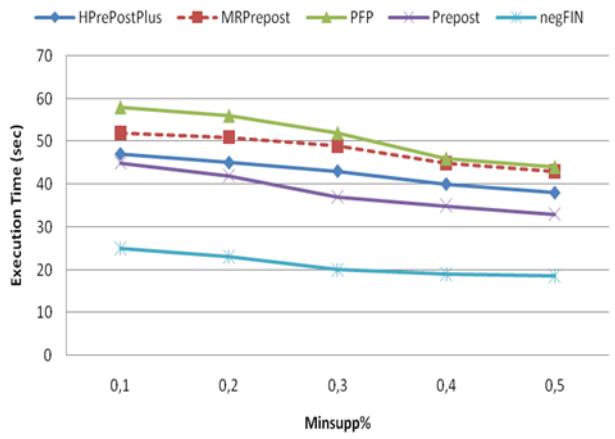

Figure. 7 The running time of T10I4D100K

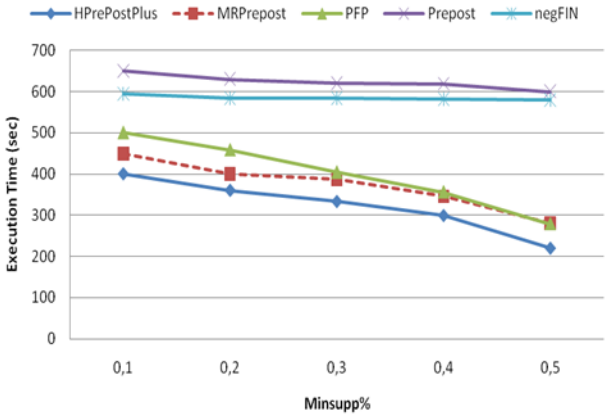

Figure. 8 The running time of T40l10D100K

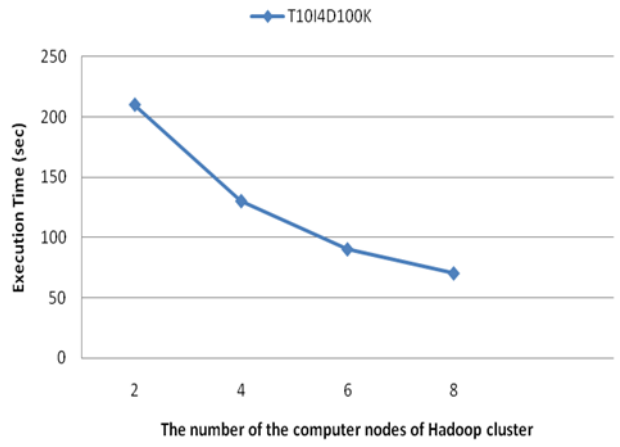

Figure.9 The running time with different computer nodes

due to memory overflow, On the other hand, the distributed algorithm, can still frequent itemsets mining, which is the purpose of PrePost algorithm parallelization, the main purpose of parallelization is to handle large dataset, which cannot be processed on standalone.

The results in Fig. 8 reflect this view. In addition, we can also know from Fig. 8, HPrePostPlus, MRPrePost and PFP are significantly superior to negFIN and PrePost, and achieved 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 HPrePostPlus is shorter than MRPrePost and PFP, because of sharing cache on Hadoop when HPrePostPlus conducts a depth-first strategy, which reduces the communication. However, using a HasMap to speed up the process of creating the N-lists associated with frequent items from PPC tree is very effective.

In Fig. 9, x-axis represents the number of computer nodes of Hadoop cluster and y-axis represents the running time of HPrePostPlus algorithm. Fig. 10 illustrates the running time with different numbers of computer nodes. With more computer nodes, HPrePostPlus needs less execution time, and the curve of HPrePostPlus has a nearly linear decline. HPrePostPlus shows a characteristic of near-linear scalability.

6. Conclusion

This paper has suggested the HPrePostPlus algorithm as an effective algorithm for mining frequent itemsets using the N-list. First, we proposed several ameliorations on the previously published PrePost algorithm: (i) use of a HasMap to improve the process of creating the N-lists associated with the frequent 1-itemsets from PPC tree and (ii) implementate a scalable Hadoop-based method for frequent itemset mining that has no intermediate data, and small network communication. HPrePostPlus does not improve over the negFIN and PrePost with respect to small datasets but the time gap is not significant. With respect to large datasets, HPrePostPlus is faster. Besides, the runtime of HPrePostPlus is always faster than MRPrePost and PFP. Also, the experimental results indicated that the proposed algorithm shows better efficiency and scalability.

For future work we will focus on applying our approach for mining frequent closed itemsets and maximal itemsets.

References

[1] R. Sandhu and S.K. Sood, “Scheduling of big data applications on distributed cloud based on QoS parameters”, *Cluster Computing*, Vol. 18, No. 2, pp. 817–828, 2015 .

[2] L. Han and H.Y. Ong, "Parallel data intensive applications using MapReduce: a data mining case study in biomedical sciences", *Cluster Computing*, Vol. 18, No. 1, pp. 403–418, 2015.
[3] Y. Chen, F. Li, and J. Fan, "Mining association rules in big data with NGEP", *Cluster Computing*, Vol. 18, No. 2, pp. 577–585, 2015.
[4] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using fp-trees", *IEEE Transactions knowledge and Data Engineering*, Vol. 17, No. 10, pp. 1347–1362, 2005.
[5] R. Agrawal and J.C. Shafer, "Parallel mining of association rules", *IEEE Transactions Knowledge and Data Engineering*, Vol. 8, No. 6, pp. 962–969, 1996.
[6] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules ", In: *Proc. of the 20th international Conference on Very Large Data Bases*, Vol. 1215, pp. 487–499 , 1994.
[7] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation", In: *Proc. of the International Conference on Management of Data*, Vol. 29, No. 2, pp. 1-12, 2000.
[8] Z.H. Deng, Z.H. Wang, and J.I. Jiang, "A new algorithm for fast mining frequent itemsets using N-lists", *Science China Information Sciences*, Vol. 55, No. 9, pp. 2008-2030, 2012.
[9] Z.H. Deng and S.L. Lv, "PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via Children–Parent Equivalence pruning", *Expert Systems with Applications*, Vol. 42, No. 13, pp. 5424-5432, 2015.
[10] D. Apilatti, E. Baralis, T. Cerquitelli, P. Garza, F. Pulverenti, and L. Venturini, "Frequent itemset mining for big data: A Comparative analysis", *Big Data research*, Vol.9, pp.67-83, 2017.
[11] S. Li, T. Hoefler, C. Hu, and M. Snir, "Improved MPI collectives for MPI processes in shared address spaces", *Cluster Computing*, Vol. 17 , No. 4, pp. 1139–1155, 2014.
[12] M.G. Kaosar, Z. Xu and X. Yi, " Distributed Association rule mining with minimum communication overhead", In: *Proc. of the Eighth Australasian Data Mining Conference*, Vol. 101, pp. 17–23, 2009.
[13] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", *Communicaton of the ACM*, Vol.51, No. 1, pp. 107–113, 2008.
[14] C. Bhat and C.K. Bhendadia, " Mining Big Data Using Modified Induction Tree Approach", *International Journal of Intelligent Engieneering and Systems*, Vol.9, No.2, pp.14-20, 2016.
[15] K. Chon and M. Kin, "BIGMiner: a fast and scalable distributed frequent pattern miner for big data", *Cluster Computing*, 2018.
[16] M.Y. Lin, P.Y. Lee, and S.C. Hsueh, "Apriori-based frequent itemset mining algorithms on MapReduce", In: *Proc. of the 6th International Conference on Ubiquitous Information Management and Communication*, article 76, 2012.
[17] S. Moens, E. Aksehirli. and B. Goethals, " Frequent Itemset Mining for Big Data", In: *IEEE International Conference on Big Data*, pp. 111-118, 2013.
[18] J. Liao, Y. Zhao, and S. Long, "MRPrePost-A parallel algorithm adapted for mining big data", In: *Proc. of IEEE Workshop Electronics, Computer and Applications*, pp. 564-568, 2014.
[19] S. Thakare, S. Rathi S and RR. Sedamkar, "An Improved PrePost Algorithm for Frequent Pattern Mining with Hadoop on Cloud", In: *Proc. of the 7th International Conference on Communication, Computing and Virtualization*, 2016.
[20] H. Li H, Y. Wang Y, D. Zhang, M. Zhang, and E.Y. chang, "Pfp: parallel fp-growth for query Recommendation", In: *Proc. of the ACM Conterence on Recommender Systems*, pp.107-114, 2008.
[21] Apache Hadoop. http://hadoop.apache.org.
[22] Yahoo developer network. Hadoop tutorial. https://developer.yahoo.com/hadoop/tutorial/.
[23] Z.H. Deng and Z.H. Wang, "A new fast vertical method for mining frequent itemsets", *International Journal of Computational Intelligence Systems*, Vol. 3, No. 6, pp. 733–744, 2010.
[24] Z.H. Deng, " DiffNodesets: An efficient stucure for fast mining frequent itemsets", *Applied Soft Computing*, Vol. 41, pp. 214–223, 2016.
[25] N. Arybarzan, B. Bidgoli, and M. Reshnehlab, "negFIN: An efficient algorithm for fast mining frequent itemsets", *Expert Systems with Applications*, Vol.105, pp.129-143, 2018.
[26] Y. Rochd, I. Hafidi, and B. Ouartassi, "A Review of Scalable Algorithms for Frequent Itemset Mining for Big Data Using Hadoop and Spark", In : *Real-Time Intelligent Systems, Advances in Intelligent Systems and Computing*, Vol .756, pp.90-99, Mai 2018.
[27] Y. Xun, J. Zhang, X. Qin, and X.Zhao, "Fidoop-dp: data partitioning in frequent itemsetmining on hadoop clusters", *IEEE*

Transactions on Parallel and Distributed Systems Journal, Vol. 28, No.1, pp.101–114, 2017.

[28] B. Goethals and M.J. Zaki, " FIMI'03: Workshop on frequent itemset mining implementations ", In: *Proc. of the Third IEEE International Conference on Data Mining Workshop on Frequent Itemset Mining Implementations*, pp.1–13, 2003.