International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 10 (2015) pp. 26615-26628

http://www.ripublication.com

A Novel Parallel Frequent Pattern Mining Algorithm

[1]Yamuna Devi. N, [2]Devi Shree J

[1]Assistant Professor (SG), Department of MCA, Coimbatore Institute of Technology, Coimbatore, India;
[2]Assistant Professor (SG), Department of EEE, Coimbatore Institute of Technology, Coimbatore, India.
E-mail : [1]yamunaacit@gmail.com, [2]devishreecit@gmail.com

Abstract

Mining frequent patterns is a process of extracting frequently occurring patterns from very large data storages. Sequential and parallel versions of frequent pattern mining algorithms have been explored in vast. Almost all of the sequential algorithms work on repeated passes over the input database to filter frequent patterns which incurs high I/O cost. In parallel algorithms, reduction phase at the end of each pass incurs high synchronization cost. In view of compensating both of these bottlenecks, this paper proposes a parallelized frequent pattern mining algorithm using hadoop map-reduce with single reduction phase. As a part of map function at each node, it is executed an algorithm namely, Direct-Vertical, to predict some of frequent patterns in advance. This parallel algorithm requires declined number of mappers through the passes because it uses equivalence class partition of input records. It performs only one reduction phase separating two map functions. This algorithm produces the frequent pattern set in fast manner with less synchronization, memory and I/O overhead.

Key words: frequent pattern mining, parallel algorithm, map-reduce algorithm, direct-vertical algorithm

Introduction

Due to auto generation and collection of huge volume of data in the domains such as cloud computing, mobile applications and social networks, digging out interesting patterns from these huge data becomes important process. This helps to identify customer behavior pattern, predicting forthcoming events and trends and many more. Extracting interesting patterns from high volume of data is an eternal process in data mining which is known as knowledge discovery. Among many techniques of data mining, Association rule mining is well known and heart task to generate frequent

patterns and association rules. A lot number of single-node algorithms have been developed to find frequent patterns which based on Apriori algorithm, vertical data format etc[2][4][13][16] . Following that parallel versions of frequent pattern mining algorithms that are classified as GPU based, Hadoop based and others are implemented in distributed environment.

MapReduce is a parallel programming model introduced by Google and inspired by Hadoop in which the computation is specified by two functions, Map and Reduce. The underlying MapReduce library automatically parallelizes the computation, and handles complicated issues like data distribution, load balancing and fault tolerance. This leads to reduced execution time and minimum requirement of memory to hold the input data.

In this paper, we proposed a parallelized map-reduce base algorithm to mine frequent patterns. The proposed algorithm uses two map functions and one reduce function. This algorithm uses a new ECLAT based single-node algorithm namely, Direct-vertical [21], which is to be executed in first map function as a part. This further reduces execution time than the algorithms that use simple map-reduce technique. The proposed parallel algorithm uses equivalence class concept to divide the input records to individual mapper. It scans the input database only once. The reducer executes one time after which it requires declined number of mappers as it grows in phase since equivalence class concept is used. The new algorithm is compared with parallelized ECLAT algorithm, single-node Direct-Vertical algorithm and proved for better performance in memory utilization and execution time.

Theoretical Analysis

Frequent Patterns

Association Rules are generated in two steps. As first step, frequent patterns are generated. The frequent patterns are those itemsets whose occurrences exceed a predefined threshold support value in the database. The second step is to generate association rules from those large frequent itemsets with the constraints of minimal confidence. The first step can be done in turn two sub-steps. They are, candidate itemsets generation and frequent itemsets generation by pruning process. The new algorithm proposed in this paper focuses on showing improvement in generation of candidate large itemsets and pruning process.

Formally, as defined by Jiawei Han et al [7], Let $I = \{I_1, I_2, \ldots I_m\}$ is the text attributes, known as the item. Given a transaction database D, let $D=\{t_1, t_2, \ldots t_n\}$. Let $t_i \subseteq I$, where i= 1,2,…n. Every transaction has an identifier, known as TID. An association rule is an implication of the form $\alpha \Rightarrow \beta$, where $\alpha \subseteq I$, $\beta \subseteq I$ and $\alpha \cap \beta = \phi$. In support-confidence framework, each association rule has support and confidence to confirm the validity of the rule. The support (s%) denotes the occurrence rate of an itemset in D, and the confidence (c%) denotes proportion of data items containing β in all items containing α in D.

Set minimum support T and minimum confidence Z, if s% ≥T and c%≥Z , then $\alpha \Rightarrow \beta$ is a strong association rule. Defined in terms of equations,

Sup(I) = Count(I) / Count(D)

$Sup(\alpha \Rightarrow \beta) = Sup(\alpha \cup \beta) = s\%$

$Conf(\alpha \Rightarrow \beta) = Sup(\alpha \cup \beta) / Sup(\alpha) = c\%.$

Map Reduce Model

MapReduce is a parallel programming model and an associated implementation introduced by Google and its open-source counterpart is Hadoop. MapReduce is an attractive model with high scalability for parallel data processing by partitioning a job into numerous small tasks running on multiple machines in high performance cluster computing environments. Google had created over ten thousand distinct MapReduce programs performing a variety of functions, including large-scale graph processing, text processing etc. in 2010[10].

In this programming model, a user specifies the computation by two functions, Map and Reduce. The map function takes a key, value pair and outputs a list of intermediate values with a new key. The map function is written in such a way that multiple map functions can be executed at once. The reduce function then takes the output of the map functions, and does some process on them, usually combining values, to generate the desired result in an output file.

In Hadoop, the job is sent to a master node, the job tracker, which divides the map and reduce tasks amongst multiple slave nodes, or task trackers that they all work in parallel and report to it and ask for new work whenever they are idle. The job tracker keeps track of task trackers, so that the failed tasks can be redistributed to other task trackers, which incurs only a slight increase in execution time. It manages not to distribute a small piece of a larger input to every process node, so that the bandwidth of a large number of hard disks is able to be utilized in parallel. In this way, the performance of Hadoop may be able to be improved by having the I/O of nodes work more concurrently, providing more throughput. It automatically handles fault tolerance and gathering of results across the multiple nodes, at last returns a single result or set. The structure is shown in figure 1.

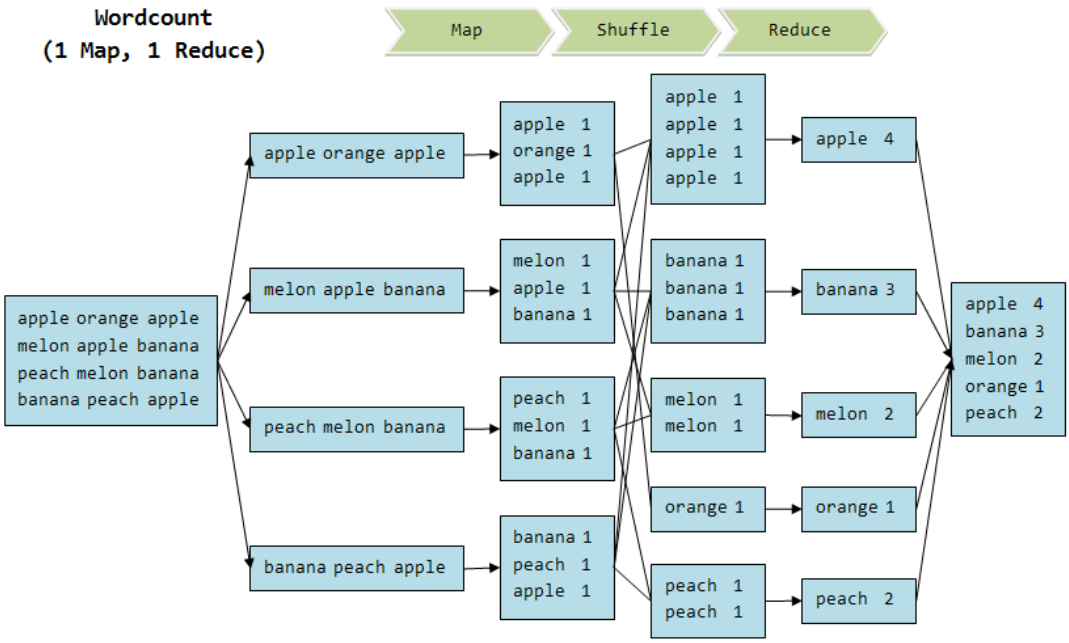

Figure 1: Map-Reduce Model

Why Map/Reduce

The Map/Reduce programming model is suitable for algorithms or applications which are large-scale data intensive, requires the data in the format of (key, value) pair. Though these features are considered as issues in most of the applications, however, it justifies the proposed parallel algorithm with Map/Reduce approach as the input database is highly compatible with the required format.

Equivalence Clause

Though proposed algorithm has join step as in other mining algorithms, it does not require prune step. The join step of the algorithm is optimized using the concept of equivalence class. For any sorted list, the values can be partitioned into equivalence classes based on the common prefix length of the values. Let *L*2 = {AC, AD, AE, BC, BD, BE, CD, CE, DE}. When it is considered as normal set, the customary way of joining consists of $\binom{9}{2}$ = 36 combinations. Considering that L_k is lexicographically sorted and the itemsets in L_k are partitioned into equivalence classes based on their common L_{k-1} prefixes. Formally, the equivalence class of $x \in L_{k-1}$, is specified as

$$[x] = \{y \in L_k \mid x[1 : k-1] = y[1 : k-1]\}$$

The L_2 set can be partitioned into equivalence classes such as E1 = [A] = {AC, AD, AE}, E2 = [B] = {BC, BD, BE}, E3 = [C] = {CD, CE}, E4 = [D] = {DE}. From this, by removing the common prefix, the above classes can be written as □([A]) = {C, D, E}, □([B]) = {C, D, E}, □([C]) = {D, E}, □([D]) = {E}. The candidate C_3 itemsets can be formed by taking all $\binom{|l|}{2}$ pairs of the items only from the same class and prefixing the class identifier. As an example, consider the items in □([B]). By

forming the pairs, the set is {CD, CE, DE}, and after prefixing, the candidate sets are {BCD, BCE, BDE}. The identifier □([D]) is eliminated because the set consists of only one element using which no pairs can be formed. Thus only $\binom{3}{2} + \binom{3}{2} + \binom{2}{2} = 7$ combinations alone are considered. The candidate itemsets produced by [A] are independent of those produced by the class **[B].** Formally, $\sum_{i=0}^{n} \binom{|\square i|}{2}$ number of combinations is considered instead of $\binom{|Lk-1|}{2}$ number of combinations.

The concept of equivalence partitioning was used by Zaki, et al [12] to parallelize the candidate generation step in CCPD. It was also used in *Candidate Distribution by* Agrawal and Shafer, [1] to partition the candidates into disjoint sets.

Benefits of Equivalence Class In Proposed Model

At the end of first map function, the results are consolidated to find the 2-item frequent sets (L_2) using a reducer function. The usage of equivalence class is used in this proposed model after finding 2-item frequent sets. The equivalence classes are found from 2-item frequent sets and further in all the following levels. In each level, the k-itemsets are partitioned into equivalence class records and are fed as input to each mapper for further processing. The results of these mappers are independent and there is no common itemsets as a consequence of equivalence class concept. This helps to have one time execution of the reduce function.

Existence Survey

Parallel ECLAT algorithm

The ECLAT algorithm was suggested with only one database scan compared with Apriori algorithm with multiple scans, and transforms the input database into vertical, {Itemset, TID}, form, where item is the name/id of the item and {TID} is the set of transaction identifiers containing the item. It does intersection with the {Itemset, Tid} sets to make support count of each candidate itemset instead of database scan [3] [19].

The parallel version of Eclat was implemented by many researchers. Manalisha Hazarika and Mirzanur Rahman developed parallel version of Eclat using equivalence class concept in synchronous and asynchronous phases[11]. Sandy proposed Dist-Eclat, a distributed Eclat algorithm which partitions the input database among nodes and discussed with communication costs [15]. Another parallel method of Eclat algorithm is discussed by Jianwei Li et al using equivalence classes [6]. Apart from these, many parallel versions of Apriori and FP-Tree growth algorithms were developed and implemented. All these algorithms were compared with corresponding sequential algorithms for their performance.

Direct-Vertical algorithm

The direct-vertical algorithm follows the vertical data format and intersection process as in ECLAT. But, it is totally different in reading the input transaction database and generation of ordered combinations instead of candidate sets. This algorithm reads one transaction at a time and does finding frequent itemset on the fly in that transaction. An itemset combination is verified for support count using intersection

method at first occurrence. The second occurrence is considered as 'exist' combination and intersection is avoided and TID is appended to that existing combination. This leads to reduced number of intersections in direct vertical algorithm compared to ECLAT algorithm. This algorithm uses combinatorial method to mine frequent patterns which avoids generation of candidate sets and pruning process [21]. It was already proved that direct vertical algorithm outperforms ECLAT algorithm.

The algorithm works in stages as, it reads the current transaction and generates all possible ordered combinations of items in that transaction. These combinations are verified for minimum support using intersection method. All combinations that satisfy support count are considered as frequent itemsets and are stored in *frequent itemset table*. This process is repeated for each transaction. Finally, the algorithm constructs a *1-itemset table* for 1-itemset frequent sets and *frequent itemset table* for k-itemsets where k>=2 in vertical form as {itemset, Tid}.

This algorithm requires only one scan of the transaction database to generate the set of all frequent itemsets without generating any candidate sets and subsets. All infrequent itemsets will be filtered on the fly. This qualifies the efficiency of the algorithm. The algorithm is given in Algorithm 1.

Algorithm 1. *Direct_Vertical (D, min_Support)*

```
Initialize i=1, support=0;
while (i<= n) do//for each transaction Ti in D & n //is the total number of
transactions in D
{
 read transaction Ti;
 for each item Ij ∈ Ti,
 //1<=j<=m, m is the number of different items
 append Ti in 1-itemset table against Ij and increment support of Ij;
 count all Ij in Ti that have support >= minSupport and move all Ij to frequent list
S;
 if count<2 then {i=i+1; Break;}            //eligibility to make combinations
 else { Ck = Produce_Combinations (S)
//Ck is the set of all possible ordered combinations //of frequent 1-itemsets from S,
where //2<=k<=count. Refer Section 4.1
 for each C ∈Ck {
 if C already exist in frequent itemset table, then append Ti against C;
 else { S1 = result of intersection of
 corresponding TID-set of individual item in C;
 if count(S1) >= minSupport then
 include C in frequent itemset table
 and append Ti against C;
 }
 }
 } i=i+1;
}
```

Other parallel algorithms

Xianfeng Yang and Liming Lian proposed a new Fastman parallel algorithm based on map reduce structure to mine frequent patterns [20]. In this paper, they compared the performance with clara algorithms. M.J.Zaki et al also proposed a parallel data mining algorithm for association rule generation[22]. Madhavi Vaidya discussed clustering concept in mining using map reduce model [10]. Jongwook Woo used map reduce structure for solving market basket problem [8]. Mohammed J Zaki et al elaborated a parallel algorithm for association rule mining. They used itemset clustering method and equivalence class concept in their proposal [12]. Ning Li et al implemented a parallel apriori algorithm based on map reduce [14]. Syed Khairuzzaman Tanbeer et al developed parallel frequent pattern tree mining algorithm which was implemented in distributed environment [17]. Map reduce was used for text processing by Lin and Dyer [9].

Proposed Method

In the proposed algorithm, there are two map functions and one reducer function. The first map function results <key1, value1> pairs to the reducer. The reducer executes to produce <key2, value2> and store them in a global output file. The second map function reads the input from the global output file and it updates it by executing the second map function.

Map function I: The original input dataset is split into parts and are globally broadcasted to all mappers where each record is read as a string. Consequently, as a first step, this map function executes direct-vertical algorithm which generates a set of k-itemset frequent sets and infrequent sets. All the frequent k-itemsets, where k>1, are stored in a global output file FREQ_TAB. As a second step, it generates 2-itemset candidate sets and filters frequent 2-itemsets to store them in FREQ_TAB.

The map function results with all 1-itemsets and infrequent k-itemsets, where k>1. The algorithm is given in Algorithm 2.

Algorithm 2: Map I Function

Input Phase:

1. Generate split files.

Map Phase:

1. Execute Direct-vertical algorithm and generate the data set (<V1>,<V2>,….,<Vn>) where each Vi : <key_itemset$_i$, value_(list of transaction, count)$_i$>.
2. Loop For each <key_itemset> (except 1-itemsets)
   2.1 If count>=min_support, move <Vi> to FREQ_TAB.
   2.2 Else Write Vi into output file.
3. End Loop For
4. Generate 2-itemsets and do intersection to generate the data set (<U1>,<U2>,….,<Un>) where each Ui : <key_2-itemset$_i$, value_(list of transaction, count)$_i$>.

5. Loop For each <key_2-itemset>
   5.1 If count>=min_support, move <Ui> to FREQ_TAB /update <Ui> in FREQ_TAB.
   5.2 Else Write Ui into output file.
6. End Loop For

Reduce function: The input of the reduce function is the data obtained from the map function of each host. In reduce step, all 1-itemsets and all infrequent itemsets are reduced by summing up the count and the reduce function filters all frequent 1-itemsets and 2-itemsets alone and stores them in FREQ_TAB. In both map and reduce functions, the frequent itemsets alone are moved to the global output file FREQ_TAB, for further process. The algorithm for reduce function is shown in Algorithm 3.

Algorithm 3. Reduce function

1. Read the output file of map function I and reduce to generate the result set (<W1>,<W2>,….,<Wn>) where each Wi : <key_k-itemset$_i$, value_(combined list of transaction, added count)$_i$>.
2. Loop For each <key_k-itemset>
   2.1 Check the existence of <key_k-itemset> in FREQ_TAB.
   2.2 Case 'True' : Append list of transactions and add count value.
   2.3 Case 'False': When count>=min_support, move <Wi> to FREQ_TAB.
3. End Loop For

Map function II : The second map function is responsible for generating all frequent k-itemsets, where k>2. The input phase in second map function reads the dataset from FREQ_TAB and divides the k-itemsets into equivalence classes, k=2, 3,..n. The map function starts execute with reading 2-itemset equivalence classes and generates 3-itemset candidates and filters frequent 3-itemsets to store them into FREQ_TAB. It goes for iteration with k+1-itemsets until there are no items in FREQ_TAB. Each mapper executes individually without overlapping, as the input records are equivalence class records which are exclusive. The algorithm is given in Algorithm 4.

Algorithm 4. Map II function

Input Phase:

1. k=2;
2. Loop while FREQ_TAB has k-itemsets
   2.1 Read all k-itemsets from FREQ_TAB and divide them into equivalence classes as split files.

Map Phase:

3. Read input as equivalence class.
4. Generate k+1-itemsets and do intersection to generate the data set (<X1,X2,…,<Xn>) where each Xi : <key_k+1-itemset$_i$, value_(list of transaction, count)$_i$>.
5. Loop For each <key_k+1-itemset>
   5.1 when count>=min_support, move <Xi> to FREQ_TAB /update <Xi> in FREQ_TAB.

5.2 End Loop For
6. k=k+1
7. End While loop

Benefits of Using Direct-Vertical Algorithm In Proposed Model

- One time scan of the input database
- It is an algorithm based on combinatorial logic which cause no generation of candidate sets.
- As the pruning process is avoided, there is no subsets generation
- Frequent itemsets to the point of transactions are found on the fly while the transaction is read from the input database.
- Addition of new transaction can be done at any time, which will not affect execution process.
- Part of the k-item frequent sets be owned by the partition of transactions is found outside of the map function.
- Map and reduce functions are written only to find the frequent itemsets from consolidated transaction set.

Experimental Results

The Datasets and Evaluation Measures

The proposed algorithm executes in parallel on n-nodes which includes execution of direct-vertical algorithm in each node. In this section, we evaluate the performance of our proposed algorithm in terms of speed by comparing it with parallel Eclat and direct-vertical in single node system. The two dataset namely T5I4D100K and T10I6D400K are considered for performance evaluation with various support values. The datasets are picked from UCI machine learning repository [18] and is listed in Table 1.

Performance experiments were run on two computers, with 2.8GHz cores and 4GB memory. Hadoop version 0.20.2 and Java 1.5.0_14 are used as the MapReduce system for all the experiments. Experiments were carried on many times to obtain stable values for each data point.

Table 1: Database Properties

Data set	Database	T	I	D	Size
Data set1	T5I4D100K	5	4	50,000	3.2MB
Data set2	T10I6D400K	10	6	1,50,000	7.4MB

Performance and Analysis

The speedup measure and corresponding performance of the proposed algorithm is evaluated for throughput in terms of execution speed by comparing it with parallel ECLAT, Direct-vertical algorithm. Table 2 and Table 3 show the comparison of speed

between the above said algorithms for various support values on dataset1 and dataset2 respectively.

Table 2: Execution time on dataset1

Support(%)/ Algorithm	Parallel ECLAT	Direct-Vertical	Proposed Method
20	2.21	3.57	1.43
30	2.13	3.5	1.3
40	1.89	3.41	0.97
50	1.34	3.36	0.81
60	1.17	3.29	0.67

Table 3: Execution time on dataset2

Support(%)/ Algorithm	Parallel ECLAT	Direct-Vertical	Proposed Method
20	2.48	4.9	1.94
30	2.35	4.51	1.7
40	2.01	4.17	1.49
50	1.65	3.93	1.02
60	1.32	3.5	0.89

Figure 5 and Figure 6 show the performance of the algorithms on dataset1 and dataset2 respectively. Clearly, the proposed algorithm speeds well as it executes parallel on data nodes. Parallel ECLAT algorithm generates candidate itemsets and uses equivalence class concept in synchronization and asynchronization phases[11]. The proposed algorithm executes direct-vertical algorithm to generate some of the frequent k-itemsets in advance and only the infrequent itemsets are used to make candidate itemsets. Direct-vertical algorithm consumes negligible time on small partition of database. Due to this reason, the proposed algorithm outperforms the parallel ECLAT algorithm.

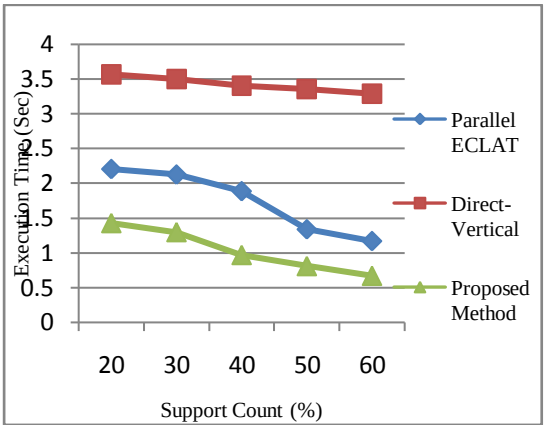

Figure 5: Execution time on dataset1

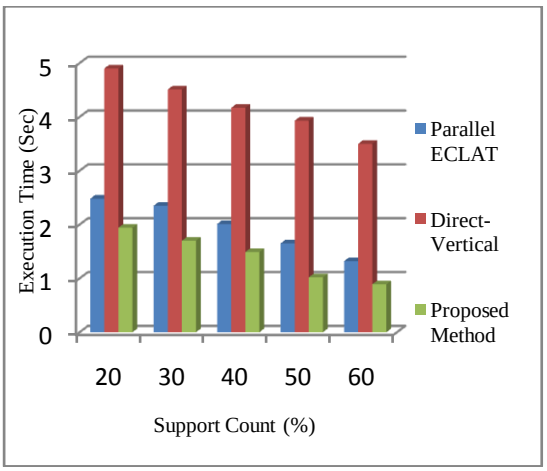

Figure 6: Execution time on dataset2

The Scaleup measure is the ability of the algorithm against increase in both the system and the dataset size. Scaleup is defined as the ability of the algorithm to execute in n-nodes with larger datasets in the same run-time as the original system. Scaleup measure is evaluated on direct-vertical algorithm to convince its efficiency in proposed algorithm. The execution time of direct-vertical algorithm in different number of nodes is compared in Table 4 to evaluate scaleup measure. It is proved that the scaleup efficiency of using direct-vertical algorithm in the proposed algorithm helps to reduce execution time. If it is not executed in each node, the time consumption of the proposed algorithm increases considerably. There is insignificant variation which attests the good scaleup measure and efficiency of its usage in the proposed algorithm.

Table 4: Execution time of direct-vertical algorithm

Support(%)/ Algorithm	Direct-Vertical (2-Nodes) on dataset1	Direct-Vertical (4-Nodes) on dataset2
20	0.57	0.59
30	0.52	0.55
40	0.40	0.41
50	0.31	0.33
60	0.22	0.22

The sizeup measure is evaluated by keeping the number of nodes constants and increase the size of dataset. The performance is evaluated for our proposed algorithm and is shown in Table 2 and 3. It shows linearity and an insignificant difference in time taken to generate frequent itemsets between the two datasets. It proves that our algorithm works for big datasets without much increase in execution time. If numbers of node are increased, our algorithm delivers the result in significant reduced time.

Conclusions

Exploring frequent patterns in transactional databases is judged one of the most important data mining problems. This task requires a lot of computation power and memory. In this paper, it is proposed a new parallel algorithm which works on map reduce structure of Hadoop for mining frequent patterns. The algorithm uses direct-vertical algorithm and equivalence class concept for fast generation of frequent itemsets. The datasets are selected with the notion to measure speed, scaleup and sizeup performance. The algorithm is evaluated for the performance by executing it on a 2-node and 4-node clusters. The algorithm minimizes I/O overheads by scanning the local database portion only one time as direct-vertical algorithm is used. Further, the algorithm uses only simple intersection operations to compute frequent itemsets and doesn't have to maintain or search complex structures such as hashing. The experimental results show that our algorithm outperforms than any other sequential algorithm and similar types of parallel algorithms. This assures that the algorithm can be implemented directly on general purpose habitual database systems as discussed by Holsheimer, et al. [5]. In future, scalability can be further improved by increasing number of nodes and the size of datasets. The Direct-vertical algorithm itself can also be parallelized.

References

[1] Agrawal, R.; Shafer, J.: Parallel mining of association rules, *IEEE Trans. on Knowledge and Data Engineering.*, 8(6), 962–969 (1996).

[2] Agrawal Rakesh; Srikant Ramakrishnan: Fast Algorithms for Mining Association Rules, Proceedings of twentieth International Conference on Very Large Data Bases, 487-499 (1994).

[3] Christian Borglet : Efficient implementation of Apriori and Eclat, FIMI (2004).
[4] Goswami, D.N.; Chaturvedi Anshu; Raghuvanshi, C.S.: An Algorithm for Frequent Pattern Mining Based on Apriori, International Journal on Computer Science and Engineering, 2, 942-947 (2010).
[5] Holsheimer, M.; Kersten, M.; Mannila, H.; Toivonen, H.:.A perspective on databases and data mining, 1st Intl. Conf. Knowledge Discovery and Data Mining, (1995).
[6] Jianwei Li; Ying Liu; Wei-keng Liao; Alok Choudhary:.Parallel data mining algorithms for association rules and clustering, CRC Press, (2006).
[7] Jiawei Han.; Micheline Kamber; Jian Pei.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers. USA, (2012).
[8] Jongwook Woo.: Market basket analysis algorithm on Map/Reduce in AWSEC2, International Journal of Advanced Science and Technology, 46, (2012).
[9] Lin, J.; Dyer, C.: Data-Intensive Text Processing with MapReduce, Tutorial at the 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics, (2010).
[10] Madhavi Vaidya.: Parallel processing of cluster by Map Reduce, International Journal of Distributed and Parallel Systems, 3(1), (2012).
[11] Manalisha Hazarika; Mirzanur Rahman.: Mapreduce based Eclat algorithm for Association rule mining in Data mining: MR_ECLAT, International journal of computer science and engineering, 3(1), 19-28 (2014).
[12] Mohammed J Zaki; Srinivasan Parthasarathy; Mitsunori Ogihara: Parallel Algorithms for discovery of Association rules, Data Mining and Knowledge Discovery, 1, 343–373 (1997).
[13] Nagesh, H.R.; Bharath Kumar, M.; Ravinarayana, B.: Improved implementation and performance analysis of Association rule mining in large databases, Springer book on Advances in Computing, Communication and Control, 94-105 (2013).
[14] Ning Li; Li Zeng; Qing He; Zhongzhi Shi: Parallel Implementation of Apriori Algorithm Based on MapReduce, International Journal of Networked and Distributed Computing, 1(2), 89-96 (2013).
[15] Sandy Moens; Emin Aksehirli; Bart Goethals: Frequent Itemset Mining for Big Data, IEEE International Conference on Big Data- Workshop on Scalable Machine Learning: Theory and Applications, (2013).
[16] Sheila A. Abaya: Association rule mining based on Apriori algorithm in minimizing candidate generation, International Journal of Scientific and Engineering Research, 3, 1-4 (2012).
[17] Syed Khairuzzaman; Tanbeer Chowdhury; Farhan Ahmed; Byeong-Soo Jeong: Parallel and Distributed algorithms for frequent pattern mining in large databases, IETE Technical Review, 26(1), 55-65 (2009).
[18] UCI Repository of Machine Learning Databases. Department of Information and Computer Science, University of California, Irvine, CA, USA, (1998).

[19] Vijayarani, S.; Sathya, P.: Mining Frequent item sets over Data Streams using Eclat Algorithm, International Conference on Research Trends in Computer Technology, 27-31 (2013).

[20] Xianfeng Yang; Liming Lian: A New Data Mining Algorithm based on MapReduce and Hadoop, International Journal of Signal Processing, Image Processing and Pattern Recognition, 7(2), 131-142 (2014).

[21] Yamuna devi, N.; Devi shree, J.: A Combinatorial tree based frequent pattern mining, Journal of Theoretical and Applied Information Technology, 63(3), 781-789 (2014).

[22] Zaki, M.J.; Ogihara, M.; Parthasarathy, S.; Li, W.: Parallel data mining for association rules on shared-memorymulti-processors, Technical report 618, University of Rochester, (1996).