International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org, editorijettcs@gmail.com
Volume 4, Issue 3, May-June 2015 ISSN 2278-6856

Closed Frequent Itemset Mining Using Directed Acyclic Graph Based on MapReduce

Archita Bonde[1], Deipali Gore[2]

[1]M.E. Scholar, Dept. of Computer Engineering,
PES's Modern College of Engineering, Pune, Maharashtra, India

[2]Associate Professor, Dept. of Computer Engineering,
PES's Modern College of Engineering, Pune, Maharashtra, India

Abstract

In various industries, the set of frequently occurring items is required for taking important decisions. There are few algorithms which are used to extract frequently occurring item sets from the given database. The basic problem with these algorithms is the generation of candidate item sets before producing frequent item set. This causes waste of time and space. FP Growth is the most efficient and scalable approach among the existing techniques. But it still generates a massive number of conditional FP trees. So we propose an improvement for FP tree based technique which does not use conditional FP trees. It generates FP trees using Directed Acyclic Graph (DAG) structure. For this we propose an algorithm that scans the database and generates FP trees as DAG so that we can generate frequent patterns directly using DAG without generating conditional FP trees. Also the paper discusses the system with respect to parallel mining using MapReduce concept. This proves to be better in terms of time and space compared to single machine environment.

Keywords: Zero-suppressed BDD, MapReduce, Parallel Mining, Frequent Itemset Mining.

1. INTRODUCTION

Mining association rules is an important research field on Knowledge Discovery in Database (KDD) presented firstly by R. Agrawal and R. Shrikant [1]. These rules represent the interesting associations or relations between item sets in databases. Generally apriori algorithm and its improved versions were used to extract frequent item sets. These algorithms make use of character "all subsets of frequent patterns are frequent". But they generate plenty of candidate item sets in mining process, and need to scan the original database many times, thus the mining efficiency is cut down. To overcome these disadvantages FP-growth [2] algorithm is used. But this algorithm generates large number of conditional FP trees. So the efficiency of the FP-growth algorithm is not reasonable. Thus we propose a system with alternate approach called Directed Acyclic Graph [3] using MapReduce which increases efficiency of mining frequent patterns from given database.

2. RELATED WORK

2.1 Apriori algorithm

Apriori algorithm is traditionally used algorithm to mine frequent patterns from the given datasets. It works efficiently but still has disadvantages.

- Apriori algorithm generates number of candidate item sets. This requires more memory.
- Also it needs to scan dataset over and over again. This requires a lot time.

2.2 Fp-growth algorithm

FP-growth algorithm overcomes all the disadvantages of apriori. It scans the dataset only twice and also do not generate number of candidate item sets. The algorithm uses prefix tree concept. The FP tree is generated to store the item and their occurrences. The path can be detected by traversing the tree bottom-up manner. Still this approach has few disadvantages.

- The algorithm is dependent upon number of items. As the number of items increases in the dataset, it requires more memory to store the increased tree structure.
- The algorithm generates large number of conditional trees. This also requires time and memory.

3. BASIC CONCEPTS

3.1 Directed Acyclic Graph (DAG)

Binary Decision Diagram (BDD) is a compact canonical graph representation of Boolean formulae. It is a Directed Acyclic Graph data representation, similar to binary decision tree, except identical sub-trees are merged. There exist efficient BDD library routines which promote their canonicity and allow intermediate computation results to be reused. Zero-suppressed Binary Decision Diagrams (ZBDDs) [4] are a special type of BDDs which are used for efficient manipulation. The secondary data structure to store BDD is bitmap array. BDDs have two important properties:

- Equivalent sub trees are shared (canonical) ;
- Computation results are stored for future reuse (referred as BDD's caching principle).

A ZBDD is a special kind of BDD which was introduced for efficient analysis [5], [6]. More specifically, a ZBDD is a BDD with two reduction rules:

- Merging Rule: Merge identical sub trees. Figure 1 shows the merging rule.

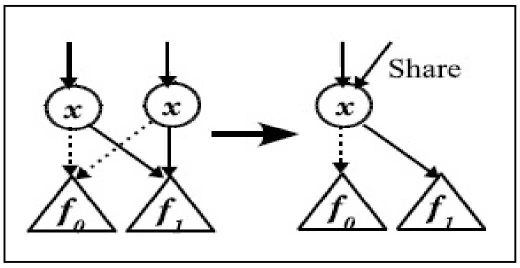

Figure 1 Merging Rule

- **Zero-suppression Rule**: Delete nodes whose 1-child node is the sink-0, and replace them with their 0-child. Figure 2 shows Zero-suppression rule.

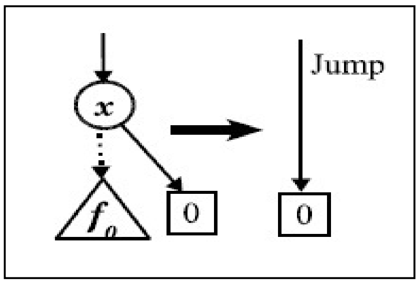

Figure 2 Zero-suppression Rule

3.2Parallel Computing Using MapReduce

MapReduce [7]-[10] is a software framework with favorable user operability and this framework can be realized by just implementing the corresponding interfaces, classes, and other operations. Applications implemented with this framework can run on large-scale cluster. The composition of the cluster is transparent for applications, and as a good fault tolerance of framework, applications can reliably handle large-scale data sets in parallel. Figure 3 shows the MapReduce framework. We use this framework upon Hadoop Distributed File System (HDFS) for efficient use.

MapReduce framework usually partitions the input data sets to several independent data blocks, and each data block will be processed by Map task in parallel. The output of Map tasks will be shuffled by the framework, and then the intermediate results will be input to the Reduce tasks. Usually the input and output of tasks will be stored in a distributed file system (Hadoop Distributed File System HDFS) and the task scheduling and monitoring of the framework are implemented by Master node (jobtracker). MapReduce framework and HDFS are running on the same set of nodes. That is, computation nodes and storage nodes are usually in together. This configuration allows the framework to schedule tasks on data nodes (tasktracker) efficiently, and improves network bandwidth utilization of the whole cluster.

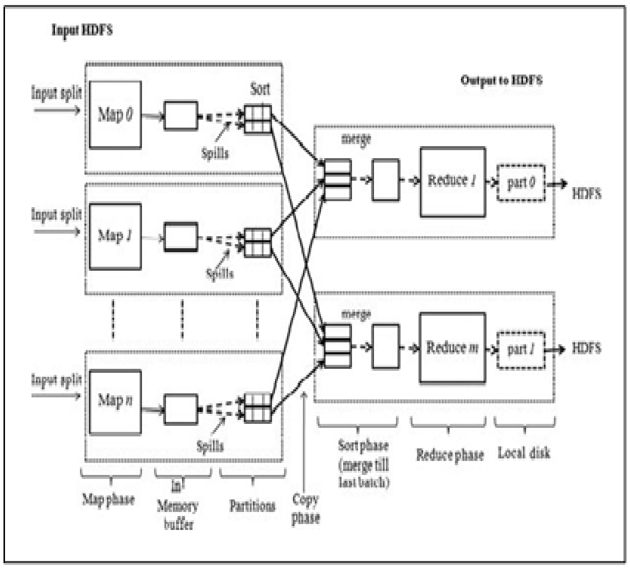

Figure 3 MapReduce Framework

4. SYSTEM DESIGN

4.1System Architectural Model

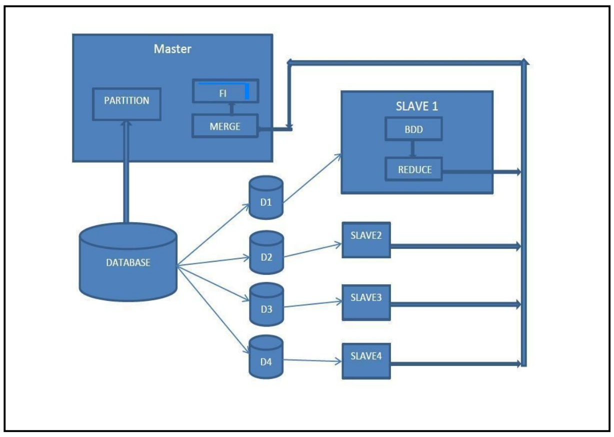

Figure 4 System Architectural Model

As we have already seen that disadvantages of FP-Growth algorithm are overcome by changing its tree data structure into Binary Decision Diagrams. Also it is seen that the diagram can further be reduced using reduction rule and forming a Zero-Suppressed Binary Decision Diagram. The representation of it is stored in the form of a bitmap array.

The above mentioned system is to be run on single machine first. Then we apply concept of MapReduce to ZBDD. In which, the datasets are divided into equal sub datasets. We use three MapReduce functions as we need to read few intermediate output files and operate on them. The output of each slave is collected to the master machine and depends on the support frequency master shows output of frequent item sets.

Figure 4 shows the system which consists of two main components which are Master and slaves. The user will first enter the dataset file. Master will partition the dataset into fixed size blocks. Here we take 4 slaves for example. Then master will transfer each partitioned dataset to the respective slave. Each slave will receive this partitioned dataset. Now the first MapReduce will take place. In this job, slave maps each item and set it's support frequency. The output of this job goes to master and master will filter these items according to minimum support frequency required by user. Now this filtered data is transferred to next MapReduce job, where each transaction of original data is altered according to this filtered data. Master will built a ZBDD based on this altered transactions and mine this Graph. The mined closed frequent itemsets are transferred as input to next MapReduce job. Here, final output is generated according to minimum support frequency.

4.2Advantages of the System

Our approach is slight modification of already existing frequent itemset mining from ZBDD.

- Parallel mining works efficiently in terms of reducing load on single machine as well as reducing total time required.
- The Zero-suppressed Binary Decision Diagram generates faster structure than prefix trees.
- The data structure is smaller as identical nodes are merged.
- Our approach only mines closed frequent itemsets. This reduces time required to traverse tree as well as memory required to store mined frequent itemsets.
- Closed frequent itemsets are more efficient to use than that of all frequent itemsets as the number of frequent itemsets can be numerous in size and sometimes user only needs itemsets containing items with higher support count.

5. Algorithmic Approach

5.1Computational Algorithm

Main() function runs on Master node and all the mapreduce jobs runs on slaves. Figure 5 shows the dataflow of algorithm.

<u>**Master Node**</u>:
Input: Dataset and minimum support frequency.

- **Step 1:** Split the dataset according to it's block size among number of slaves.
  Slave Node: Job 1 (TokenizerMapper and Reducer)
- **Step 2:** Count support frequency of each item and remove those items whose support frequency is less than that of minimum support frequency specified by the user.

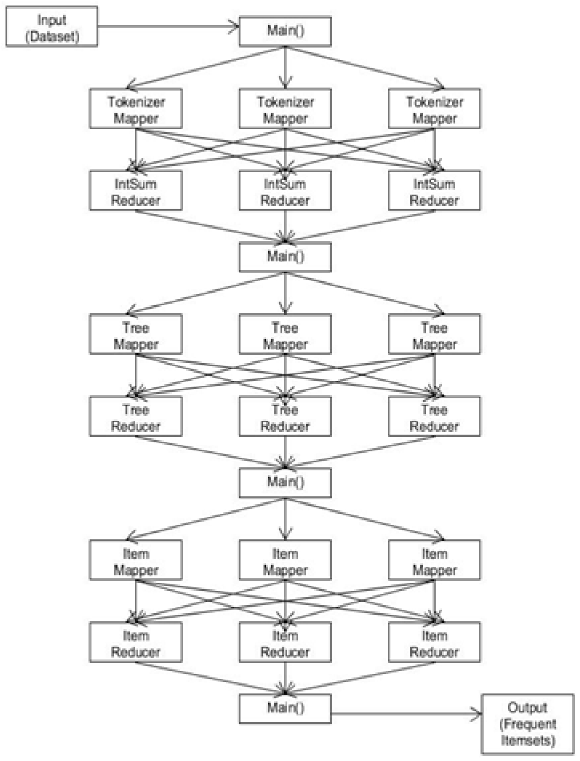

Figure 5 Dataflow of Algorithmic Approach

<u>**Master Node:**</u>

- **Step 3:** Sort all the items according to their support frequency (ItemOrder).

<u>**Slave Node:**</u> Job 2 (TreeMapper and Reducer)

- **Step 4:** Filter and organize each transaction from the original dataset according to ItemOrder.
- **Step 5:** Remove redundancy by adding support count of duplicate ordered transactions.

<u>**Master Node:**</u>

- **Step 6:** Construct a ZBDD using these ordered transaction one by one.
- **Step 7:** Mine all closed frequent itemsets from ZBDD by traversing each node's high path and low path (freq_itemsets).

<u>**Slave Node:**</u>

- **Step 8:** Calculate support frequency for each itemset from freq_itemsets by checking it's presence in orederd transactions.
- **Step 9:** Remove itemsets whose support frequency is less than minimum support frequency required by user.

<u>**Master Node:**</u>

- **Step 10:** Merge frequent itemsets from all the slave nodes and display the output.

5.2Pseudo Code for Algorithm

```
Main()
{ Input: dataset D,  min_support;
//Job 1: find support for each item
TokenizerMapper(){
// Map each item
context.write(item, 1);}
TokenizerReducer(){
//Remove redundancy
context.write(item, sum);}// end of Job 1
 get_freq_items(String path){
Hashtable<String, Integer> itemset;// stores each item
and it's support frequency
String itemOrder; // stores items according to decreasing
support frequency }
// Job 2: find ordered transactions from original dataset
TreeMapper(){
for each transaction in D{
for each item in itemOrder{
            if (present)
               write to OrderedItems[];
               next item;
            else
               next item;}}
        String str ← (String) OrderedItems[];
context.write(str, 1);}
TreeReducer(){
//remove redunadancy
    context.write(str, sum);}//end of Job 2
Hashset<String> get_results(String path, string
itemOrder[], int min_support){
Hashtable<String, Integer> item_transac;
// contains ordered transaction and it's support frequency
     ZddTree = zdd.union(ZddTree, str);
        //construct ZBDD for each ordered transaction
     Hashset<String> freq_itemsets;
        //contains frequent itemsets mined from ZBDD
freq_itemsets = mine_deep(ZddTree, null,
freq_itemsets,itemOrder)
     return freq_itemsets;}
Hashset<String> mine_deep(int zd, String prefix, Hashset
items, itemOrder){
high = high node of (zd);
low = low node of (zd);
var = prefix + itemOrder[variable of (zd)];
push var to items;
if(var.length() > 1){
      for(j = 1 ; j< var.length(); j++){
         add(var.subString(j, var.length()));}}
if(last low node){
      mine_deep(low, prefix, items, itemOrder);}
if(last high node){
      mine_deep(high, var, items, itemOrder);}
return items;}
//Job 3: find frequent itemsets with support greater than
or equal to min_support
ItemMapper{
    For each itmset{
      For each ordered transaction{
         if(present)
            context.write(itemset, 1);
      else
            next itemset;}}}
ItemReducer{
//remove redundancy and remove items whose support
less than min_support
context.write(itemset, sum);}//end of Job 3
Output: Frequent itemsets with support greater than
min_support
}
```

6. RESULTS

6.1Working of Algorithm using Example

Table 1 shows the dataset we are using as input to our system. Minimum support frequency is considered as 3.

Table 1: General Dataset

TID	Transactions
1	f, a, c, d, g, i, m, p
2	a, b, c, f, l, m, o
3	b, f, h, j, o
4	b, c, k, s, p
5	a, f, c, e, l, p, m, n

After job 1 is finished, the output can be seen as table 2. It contains the items with support frequency greater than or equal to minimum support frequency.

Table 2: Frequent Items

Item	Support Frequency
a	3
b	3
c	4
f	4
m	3
p	3

After job 2 is finished, the output can be seen as table 3. It contains the ordered transactions containing only items present in table 2.

Table 3: Ordered Transactions

Ordered Transactions	Support Frequency
c, b, p	1
f, b	1
f, c, m, a, p	2
f, c, m, b, a	1

The Zero-suppressed binary decision diagram (ZBDD) created using ordered transactions from table 3 is shown in figure 6.

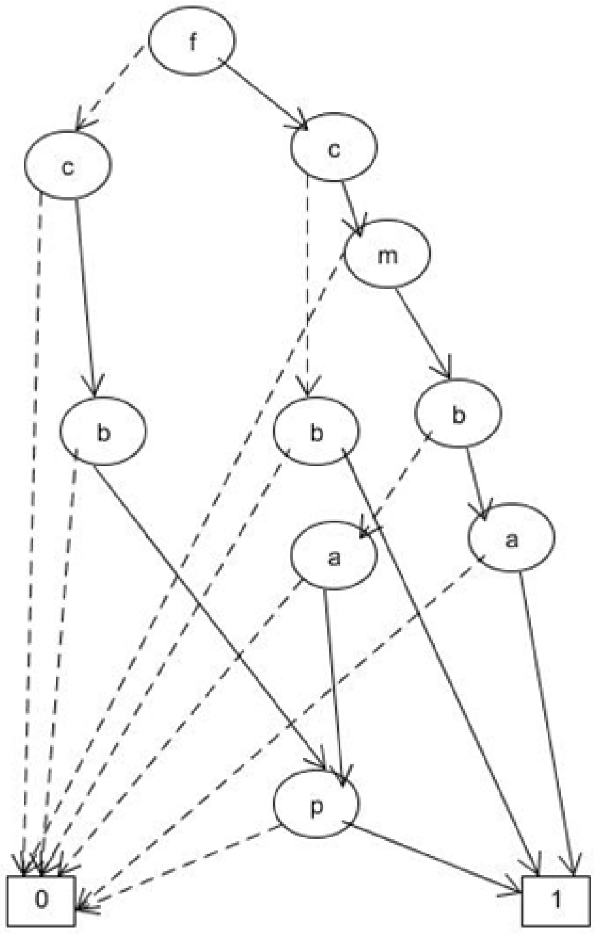

Figure 6 Zero-suppressed Binary Decision Diagram

All the closed frequent itemsets mined from ZBDD are as,
Freq_itemsets={ cb, cmba, fcmb, fcma, bp, ma, mb, map, fcmba, f, fcmap, ba, b, c, cbp, a, fcm, mba, m, fb, cmb, cm, ap, p, cmap, cma}
The output of job 3 i.e. final output of system is showm in table 4. It contains the closed frequent itemsets whose support frequency is greater than or equal to minimum support frequency.

Table 4: Closed Frequent Itemsets

Closed Frequent Itemsets	Support Frequency
c, m	3
f, c	3
f, c, m	3

6.2Performance Analysis

We tested our system performance by using input datasets of different sizes. We used 1.6 Mb, 3.1 Mb, 6.3 Mb and 12.5 Mb datasets for this. The time required for all these datasets are shown in table 5 according to single machine environment and two slave environment. The minimum supports given to all these datasets are also shown.

Table 5: Performance Analysis

Dataset Size (MB)	Minimum Support Frequency	Time for Single Machine (sec)	Time for Two Slave Machines (sec)
1.6	25000	69	76
3.1	50000	72	111
6.3	100000	92	78
12.5	200000	115	89

Based on the total time required for system to get the desired output on single as well as distributed environment, the graph shown in figure 7 and 8 is plot.

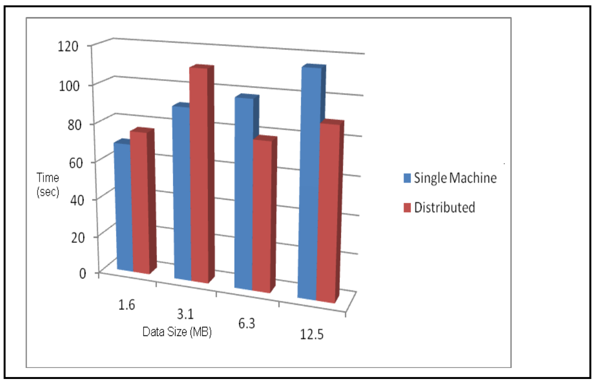

Figure 7 Bar Chart

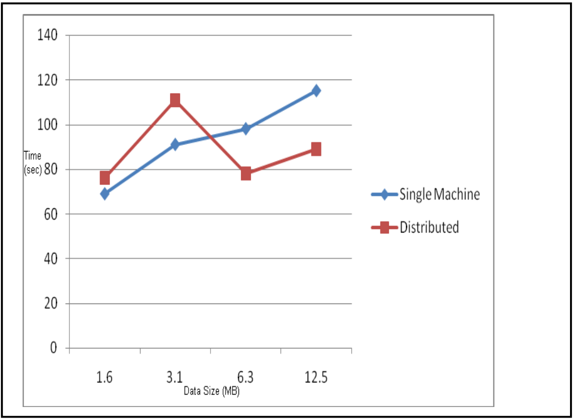

Figure 8 Performance Graph

It is observed that for small size of dataset, the time required to carry out whole approach on single machine environment is less than that on distributed environment using two slaves. Here, we can say that, single machine environment is efficient for smaller datasets. But as the dataset sizes goes on increasing, the time required by system on distributed environment is less than that of single machine environment. So the system proposed by us is very efficient for mining closed frequent itemsets from big data.

7. Conclusion

Our system is efficient in data structure as well as performance. The improvement in itemsets mining from ZBDD instead of prefix trees and mining only closed frequent itemsets instead of whole lot gives remarkable advantages. Also we used distributed parallel mining approach which reduces load as well as time required to perform operations on big data.

References

[1] R.Agrawal, and R.Srikant, Fast Algorithms for Mining Association Rules, Proc.of the 20th Intl Conference on Very Large Databases, Santiago, Chile, 1994.
[2] Zhang Wei, Liao Hongzhi, Zhao Na, Research on the FP growth algorithm about association rule mining, ISBIM 2008, December 19,2008, pp. 315-318.
[3] Bryant, R. E. (1986), Graph-based algorithms for boolean function manipulation, IEEE Transactions on Computers 35(8), 677691.
[4] Loekito E. , Bailey J. , Fast mining of high dimensional expressive contrast patterns using ZBDDs, in Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD06), pp. 307316.
[5] Minato S. , Zero-suppressed BDDs for set manipulation in combinatorial problems, in Proceedings of the 30th International Conference on Design Automation, pp. 272277, 1993.
[6] Minato, S. Arimura, H. , Frequent pattern mining and knowledge indexing based on Zero-suppressed BDDs, in The 5th International Workshop on Knowledge Discovery in Inductive Databases (KDID06), pp. 8394.
[7] Premchaiswadi W. Romsaiyud W., Optimizing and Tuning MapReduceJobs to Improve the Large-Scale Data Analysis Process.
[8] Xin Yue Yang, Zhen Liu, Yan Fu, MapReduce as a Programming Model for Association Rules Algorithm on Hadoop.
[9] J. Dean, S. Ghemawat, MapReduce:A Flexible Data Processing Tool.
[10] Hui Chen, Tsau Young Lin, Zhibing Zhang, Jie Zhong, Parallel Mining Frequent Patterns over Big Transactional Data in Extended MapReduce, IEEE International Conference on Granular Computing (GrC), 2013.

AUTHOR

Archita Bonde received the B.E. degree in Computer Engineering from Mumbai University, Maharashtra, India in 2011 and currently pursuing M.E. from Pune University, Maharashtra, India.