# PARALLELIZING APRIORI ALGORITHM USING OPENMP AND ITS APPLICATION IN FOREST FIRE PREDICTION

Rahul Pabba
*Department of Computer Science*
*California State University Long Beach*
Long Beach, CA, U.S.A.
Rahul.pabba01@student.csulb.edu

Prudhvi Raju Madam
*Department of Computer Science*
*California State University Long Beach*
Long Beach, CA, U.S.A.
PrudhviRaju.Madam01@student.csulb.edu

Charitha Reddy Mandli
*Department of Computer Science*
*California State University Long Beach*
Long Beach, CA, U.S.A.
Charithareddy.Mandli01@student.csulb.edu

Geethika Penukula
*Department of Computer Science*
*California State University Long Beach*
Long Beach, CA, U.S.A.
Geethika.Penukula01@student.csulb.edu

## ABSTRACT

Forests are places where lands are dominated by trees and plants. It is home to many spies of plants and animals. Destroying forests has a direct impact on global warming. So, it is our duty to preserve it. Cataclysmic calamities such as forest fire can burn large acres of land and endanger animals stuck in it. If there are any human communities near to forests, then there is a loss of infrastructure as well. The meteorological infrastructure for detecting fire is accessible in a timely manner in farfetched locations. There are no long-established methods to detect fire for isolated places and we cannot get time-to-time updates of the place. To stop this, we must find the factors which cause fires to the forest. So that we can reduce the ratio of catching fire. The best mining algorithm, which is called the Apriori algorithm is an algorithm which is useful in finding itemsets in the database. User is given constraints to find itemsets in terms of support and confidence. The algorithm's repetitive nature leads to decreased efficiency as dataset size increases. If a dataset has 'n' items, in the first iteration we have to make C2 combinations and reduces the intermediates which opposes the support count, which is also time taking. By using openMP threads, the efficiency of our project will be increased. To split the transaction database into many parts, we use data decomposition, and everyone is selected by the thread to find the support count and by these, we can find the transaction allocated to a particular thread. For instance, the primary goal of this project is to predicate the fire in the forest and transactional databases has many phenomena's and in that few transactions have forest fire phenomena and these transactions are carried out with respect to another item-sets of the transactions. Therefore, if we happen to receive an additional transaction from the user, we can determine the confidence of the forest fire using that transaction.

## 1. INTRODUCTION

In modern world, everything is interlinked to data. A huge quantity of data are generated daily. Data mining is a method of evaluating and extracting relevant information from acquired data.

Huge databases/datasets are scanned and the raw data present in them is run through various algorithms to produce something with a context, which is useful for industrial as well as research purposes. Understanding the relationships between database objects or tuples is a popular way to deduce from data. This data mining technique is called ARM (Association Rule Mining).

It involves identifying different sets of items within the dataset through the utilization of diverse algorithms in which associations can be formed. The ARM algorithm, which is a kind of the Apriori algorithm, aims to identify numerous frequent itemsets within a database. Users define constraints for discovering itemsets, which include support (the percentage of transactions with an item-set) and confidence.

However, the algorithm's efficiency diminishes significantly with larger datasets due to its highly iterative nature. This project seeks to enhance efficiency by leveraging OpenMP threads. Data decomposition separates the transaction database into segments, each managed by a unique thread. The candidate item-set is computed by support count inside the allocated transactions.

Suppose we start with a small data containing 'n' transactions then during the 1st iteration we generate nC2 combinations and chuck out the intermediate transactions which are less than the minimal support. That means for n value of 200 we have 19,900 first generated item sets which have to be scanned individually which consumes a lot of time. Giving a few transactions to each thread and running them parallel decreases the execution time by a fraction of the number of threads taken into the code. The scope of our project is not restricted to forest fires alone. Any application which requires the analysis of large amount of data where an event

is dependent of a set of known factors contributing to it in varied amounts in varied times, our project can be used to tailor the requirement of the particular application. The most famous being the Market Basket Analysis.

## 1.1 Association Rules

Association rules are used to identify correlations and cooccurrences between datasets. Used to describe trends in data from independent sources, such as relational and transactional databases. Data mining functionality includes association rule mining (ARM). Association rule mining evaluates big databases to find common relationships between items. [2]

One typical method for extracting insights from data is to investigate the links between distinct instances or tuples in the dataset. This approach, known as Association Rules Mining (ARM), comprises using multiple algorithms to discover links between different item sets in the database. The Apriori method, a well-known ARM technique, is critical in detecting many frequently occurring itemsets in the database.

The challenge with the Apriori algorithm lies in its significant reliance on iteration, leading to a sharp decline in efficiency as the dataset's size or dimension grows. Our project seeks to enhance the algorithm's efficiency by leveraging parallel computing concepts and implementing a few changes in the algorithm.

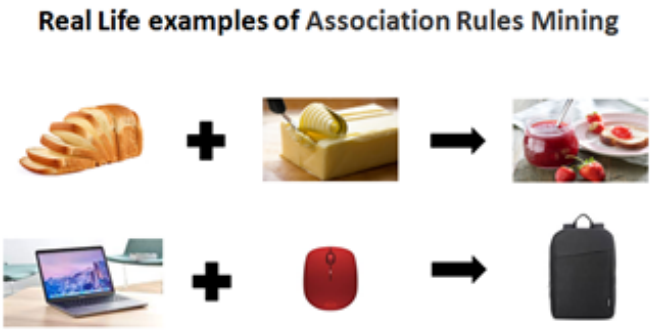



**Figure 1: Example Image**

## 1.2 Apriori Algorithm

The challenge with the Apriori algorithm lies in its significant reliance on iteration, leading to a sharp decline in efficiency as the dataset's size or dimension grows. Our project seeks to enhance the algorithm's efficiency by leveraging parallel computing concepts and implementing a few changes in the algorithm.

The Apriori Algorithm has applications to recognize common item sets for Boolean association rules. It expands association rules based on past knowledge of frequently used item characteristics and sets. The primary objective of developing the Apriori algorithm is to operate with transactional databases. This procedure includes going through through the database using "JOIN" and "PRUNE" methods for item sets. It then determines the item sets with the lowest support value.

If an item set's count is less than the minimum support number, we delete it from the database table and link the L(k-1) and L(k-1) item sets. The algorithm must collaborate on k-2 elements and decide if all subsets of an item set are frequent. If item sets are rarely used, remove them. To find Boolean association rules in frequent itemsets, the Apriori Algorithm uses two factors: support and confidence.

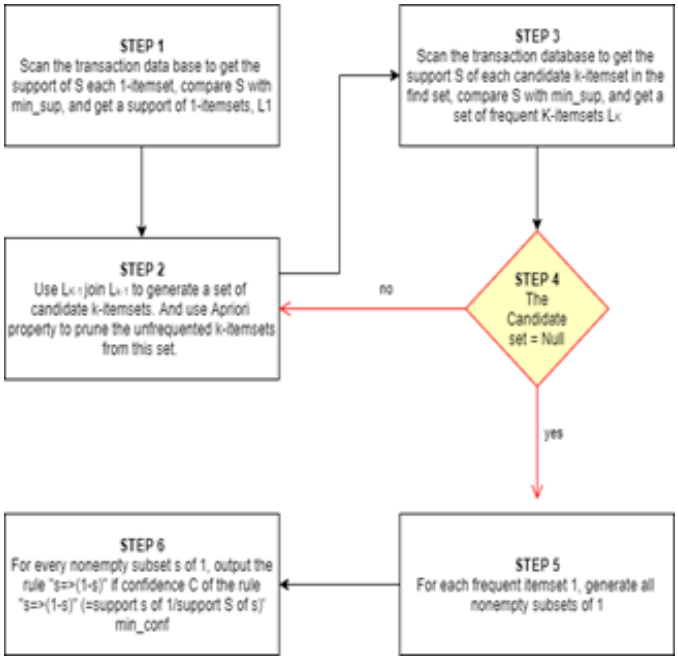
### *1.2.1* Key Concepts:

- Frequent Itemsets: Group of items which has minimum support (denoted by Li for ith-Itemset).
- Apriori Property: select a subset from the set of frequent item-sets.
- Join Operation: to discover Lk, a collection of candidate k-itemsets is formed by combining Lk-1 with itself.

### *1.2.2* Pseudo Code:

**Algorithm 1** Apriori Algorithm

$C_k$: size of the candidate itemset here is $k$
$L_k$: size of the frequent itemset here is $k$
$L_1 = \{\text{frequent items}\}$
**for** $k = 1; L_k \neq \emptyset; k++$ **do**
    $C_{k+1}$ = generated candidates from $L_k$
    **for** each transaction $i$ in db **do**
        Increase the count of all candidates found within itemsets in $C_{k+1}$ that are included in $i$
    **end for**
    $L_{k+1}$ = Candidates in $C_{k+1}$ with min_support
**end for**




### *1.2.3* Example of Apriori Algorithm:

Let's analyze the provided dataset to identify frequent item sets and derive association rules from it.

Minimum support value count is 2.

Minimum confidence is 50%.

| TID | Items |
|---|---|
| T1 | I1,I2,I4 |
| T2 | I2,I4 |
| T3 | I2,I3 |
| T4 | I1,I2 |
| T5 | I2,I3 |
| T6 | I1,I2,I3,I5 |
| T7 | I1,I2,I3 |
| T8 | I3,I4 |

**1. when** $K = 1$

(i) Let's craft a table showcasing the support count for each item found in dataset C1, our candidate set.

| Itemset | Sup_count |
|---|---|
| L1 | 4 |
| L2 | 7 |
| L3 | 5 |
| L4 | 3 |
| L5 | 1 |

(ii) Then, we'll assess the support count of each item inthe candidate set against a minimum support count, set at 2. If an item's support count falls below this threshold, it will be eliminated. This process yields our item set L1.

| Itemset | Sup_count |
|---|---|
| L1 | 4 |
| L2 | 7 |
| L3 | 5 |
| L4 | 3 |

**2. when** $K = 2$

(i) We'll proceed by creating another table to form the candidate set C3 by combining the item sets in L2 through a joining process. The criterion for joining Lk-1 and Lk-1 is that they must share (K-2) elements in common. Hence, for L2, the first element must align to initiate the joining process.

| Itemset | Sup_count |
|---|---|
| L1,L2 | 4 |
| L1,L3 | 2 |
| L1,L4 | 1 |
| L2,L3 | 4 |
| L2,L4 | 2 |
| L3,L4 | 1 |

(ii) Next, evaluate the support count of each item in thecandidate set against a minimum support threshold. For instance, if the minimum support is set at 2, any candidate set items with a support count below this threshold are eliminated. This step results in the formation of item set L2.

| Itemset | Sup_count |
|---|---|
| L1,L2 | 4 |
| L1,L3 | 2 |
| L2,L3 | 4 |
| L2,L4 | 2 |

**3. when** $K = 3$

(i) Let's generate another table to create candidate set C3 using L2 (join step). The condition for joining Lk-1 and Lk-1 is that they should have (K-2) elements in common. So, in this case, for L2, the first element should match.

| Itemset | Sup_count |
|---|---|
| L1,L2,L3 | 2 |
| L1,L2,L4 | 1 |
| L2,L3,L4 | 0 |

(ii) Now, assess the support count of each item in the candidate set against the specified minimum support threshold. For

example, if the minimum support is set at 2, any candidate set items with a support count lower than this threshold are discarded. This process generates item set L3.

| Itemset | Sup_count |
|---|---|
| L1,L2,L3 | 2 |

**4. when** $K = 4$

Now create table that generate candidate set C4 using L3. Condition to join Lk-1 and Lk-1 is that it should have (K-2) elements in common. In order to form L4 we have to add item sets of L3 that first two elements should match. But we should stop here because as we don't have further items in candidate set C3.

Now we have all frequent item sets and have to generate strong Association rule. For which we have to calculate confidence of each rule

## Confidence

A confidence level of 50% indicates that half of the customers who purchased bread and butter also bought jam.

Confidence(A→B) = $\frac{\text{Support\_count}(A \cup B)}{\text{Support\_count}(A)}$

So, the item set generated is from L3 is L1, L2, L3, so the rules derived from this are

[I1^I2]=>[I3] //confidence = sup(I1^I2^I3)/sup(I1^I2) = 2/4*100=50%
[I1^I3]=>[I2] //confidence = sup(I1^I2^I3)/sup(I1^I3) = 2/2*100=100%
[I2^I3]=>[I1] //confidence = sup(I1^I2^I3)/sup(I2^I3) = 2/4*100=50%
[I1]=>[I2^I3] //confidence = sup(I1^I2^I3)/sup(I1) = 2/4*100=50%
[I2]=>[I1^I3] //confidence = sup(I1^I2^I3)/sup(I2) = 2/7*100=28%
[I3]=>[I1^I2] //confidence = sup(I1^I2^I3)/sup(I3) = 2/5*100=40%

Therefore, if the minimum confidence is set at 50%, then the initial 4 rules can be deemed as robust association rules.

### 1.3 Open MP

OpenMP is an enactment of multithreading. Where, the threads are sub-divided, and tasks are assigned to every individual thread. Then they run simultaneously with runtime environment and each thread is assigned to an individual processor among themselves. In many programming languages parallel programming is done with the help of OpenMP. OpenMP helps in supporting shared memory multiprocessing. OpenMP has the ability to create many threads as per the processor need. It is the developer choice to select the number of threads needed. It supports several operating systems like Windows, Linux, MAC OS etc. OpenMP has mater thread and salve thread. Master thread uses one thread which runs from the beginning to the end. Whereas slave thread uses parallel sections of the program which is the source for additional threads to the fork. It is the more efficient way for parallel programming. OpenMP let the developer to control the scheduling of the threads and they are classified as static, dynamic and guided. OpenMP has the ability to execute the iteration of the loop in an ordered way.

Used functions in OpenMP:

1. Pragma: It is used to divide additional threads to achieve the work encompass in parallel. Pragmas are developed like, they are works perfectly, even if the compiler does not support them.

2. Share: If a variable is shared among all the threads, the only property of the particular variable is shared which is needed.

3. Omp_set_dynamic: It has the ability of enabling and disabling the adjustments of threads for the execution of parallel executions. It does not affect the implementations, if it does not have dynamic adjustments of number of threads.

4. Num_threads: It is one of the ways to determine the number of threads while creating parallel regions. It is specified while the execution of the program.

5. Collapse: It has the ability to the collapse the specified loops and divided them accordingly by the schedule clause. It is useful in the conversion of nested loop into a normal loop.

## 2. PROJECT DESIGN

### 2.1 The transaction generator:

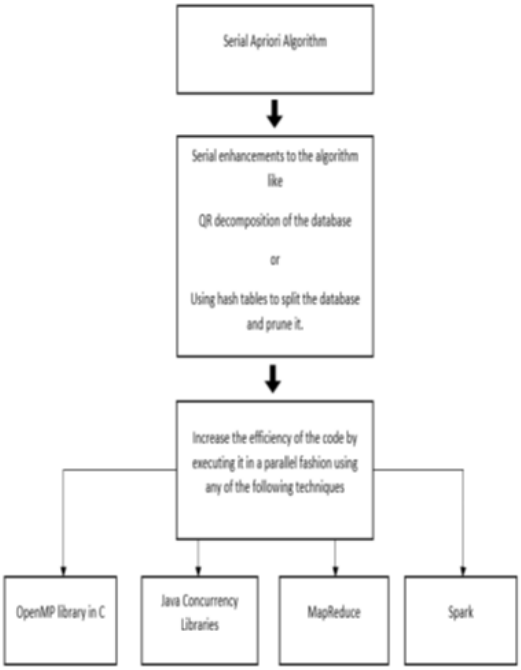
For the sake of experimentation with numbers and providing context, we generated a .txt file. In this file, we created random entries of transactions using the srand() and time() functions available in the C++ libraries.

### 2.2 Creation of Threads:

After checking the working of the serial code, we used open MP to create threads and then assign each thread a specific number of transactions to form an association rule which would be later checked for support count. Since all the transactions are independent of each other there is no ambiguity of assigning it to a thread.

### 2.3 Calculating Support and Confidence:

AAfter the rules are association rules are generated, the support is calculated and checked with the required number by the following formula:




## 2.4 Organizational Rules:

The set of items is defined as I = I1, I2, I3..., Im, and the number of items is defined as m. DB = T1, T2, T3..., Tn, and TiI denotes the entire transaction database. If XTi, we call the itemset X found in Ti. The association rule X=>Y implies that XI, YI, and XY= are all valid simultaneously. Support and confidence are the most often used measurements of association rule strength.

Quantitative data is converted into qualitative data, for the association algorithm to process the later one. The transform procedure is as follow, first, the desired data got from database with SQL, then the data will be divided into two kinds, one is that the fire number is zero, and the other is not, at last, the dataset, whose value is zero, will be divided further, from which the maximum and the minimum values, and the difference value is divided by 3, so all the results are divided into 3 groups according to the quotient, and each group will be set a value as high, medium or low. Full records in the subset have been divided into four kinds according to its fire number, that are frequent and rare. These complete data are processed with the Apriori Algorithm, and some rules between the weather factors and the wildfire.

Apriori comes under the association rule algorithm which can be used to mine the Boolean dataset [12]. The principle is to state that the prior knowledge of the dataset to search the dataset iterates to explorer the k+1 items with k items.

## 2.5 Forest Fire Application

An application of the Apriori algorithm allows for the detection of whether a forest fire will occur following a specific sequence of events.

The database comprises numerical values representing various incidents within the forest. In our case, value of 12 signifies the prediction of the forest fire.

As a result, the confidence score is calculated as the chance of a forest fire developing after a series of attacks. If we denote the new transaction as T, then,

Probability (Forest Fire) = $\frac{\sup(T,12)}{\sup(T)}$

The code uses the function 'float predict(string input)'. The Apriori algorithm initiates all layers, which accumulate using the structure F. This function parses the incoming transaction as a parameter and turns it into a template vector, changing all characters to integers. The function from 'F' then computes the transaction count, resulting in the database's support/frequency/count value.

With two support values now available, we can determine the confidence (or probability) of the occurrence of 12 following the given transaction.

### *2.5.1 Serial*

We implemented the Serial Apriori method in C++ and estimated the execution time using the time() function. This algorithm initiates by scanning the text file to generate the first candidate list and computes the support value for all unique items in the database. Afterward, the candidate list is checked against the minimum support value, and the first round of the algorithm is processed. Subsequently, items in this round are combined to create a candidate list for the subsequent levels, with 'n' items in each level, where 'n' ranges from 1 to the maximum level. The Apriori algorithm then eliminates all itemsets from this candidate set whose subsets are not frequent itemsets. This iterative process is repeated for each level that can be generated from the collection of items in the transaction.

### *2.5.2 Parallel*

To address the impracticality of the Apriori algorithm's need to execute many database rounds to construct the list, we built a parallel approach utilizing the OpenMP package in C++. The method repeatedly reads the database to determine support for entries in the candidate list. As a result, this part of code has been optimized and parallelized.

Data decomposition requires dividing the database in accordance with the number of working threads. Each thread starts at a certain location in the file and scans until it reaches another thread's starting point. These threads calculate the support count for the itemsets in the candidate list that used this function. Finally, counts are aggregated using OpenMP procedures. This method drastically shortened the duration of the Apriori algorithm.

# 3. IMPLEMENTATION

We have included a code snippet below which describes the pruning method, generate_itemset and the support count method. These methods are integral components of our implementation for efficiently discovering the frequent itemsets.

### 3.1 Prune Method:

This function, prune, filters candidate itemsets by checking if their subsets meet the minimum support requirement. It iterates through each candidate itemset, checks each of its subsets, and eliminates those that do not satisfy the support criterion.

```
    void prune (vvi nset, int sz) {
for (auto i : nset) {
bool ok = true;
for (int j = 0; j < sz; j++) {
vi dk;
for (int k = 0; k < sz; k++) {
if (k == j) continue;
dk.emplace_back (i[k]);
}
if (itemsets.count(dk) == 0) {
ok = false;
break;
}
}
if (ok) check_sup_count (i);
}
}
```

### 3.2 Generate Itemset Method:

This method, generate_itemset, constructs itemsets of a specified size by combining smaller itemsets. It iterates over the previous itemsets (pset), generates new itemsets (nset) by merging pairs of itemsets, and then filters out those that don't meet the minimum support criteria before repeating the process to create larger sets.

```
    void generate_itemset (int items) {
vvi pset, nset;
for (auto i : itemsets)
 if (i.F.size() == items - 1)
  pset.emplace_back (i.F);
if (pset.empty()) {
max_items = items - 2;
return;
}
int i;
#pragma omp share (i) for collapse(2)
for (i = 0; i < pset.size (); i++) {
for (int j = i + 1; j < pset.size(); j++) {
set <int> ps;
#pragma omp share (ps) parallel
{
#pragma omp for nowait
for (int k = 0; k < (int) pset[i].size(); k++)
  ps.insert (pset[i][k]);
#pragma omp for nowait
for (int k = 0; k < (int) pset[j].size(); k++)
  ps.insert (pset[j][k]);
  }
if (ps.size() == items) {
vi tk (ps.begin(), ps.end());
nset.emplace_back (tk);
}
}
}
prune (nset, items);
generate_itemset (items + 1);
}
```

### 3.3 Support Count Method:

This function, check_sup_count, calculates the support count for a given itemset by counting its occurrences in the transactions. If the support count meets the minimum threshold, it updates the itemsets map with the count value.

```
    void check_sup_count (vi arr) {
int cnt = 0;
#pragma omp share (cnt) for
for (int i = 0; i < (int) transactions.size(); i++) {
if (present (transactions[i], arr)) cnt++;
}
if (cnt >= min_sup) itemsets[arr] = cnt;
}
```

## 4. RESULT

```
[04/20/24]seed@VM:~$ g++ -std=c++11 fire.cpp -fopenmp
[04/20/24]seed@VM:~$ ./a.out 5 65 fire.in
Association Rules :
Low Temperature ^ Average Wind Speed ==> Low Rainfall ^ Weak Fire
Low Temperature ^ Low Rainfall ==> Average Wind Speed ^ Weak Fire
Low Temperature ^ Average Wind Speed ^ Low Rainfall ==> Weak Fire
Normal Temperature ^ Average Wind Speed ==> Low Rainfall ^ Normal Fire
Normal Temperature ^ Average Wind Speed ^ Low Rainfall ==> Normal Fire
High Temperature ^ Average Wind Speed ==> Low Rainfall ^ Fast Spreading Fire
High Temperature ^ Average Wind Speed ^ Low Rainfall ==> Fast Spreading Fire
Time Taken : 0.01543532
[04/20/24]seed@VM:~$
```

**Figure 2: Output Image**

Upon execution with 90 transactions, 15 states with a minimum support of 5, and a minimum confidence of 65%, the time taken to execute was 0.015435532 seconds.

## 5. CONCLUSION

The Apriori algorithm's parallel implementation speeds up database scanning, making it more efficient and suitable for real-time applications. However, this parallel implementation is merely the first step in optimizing efficiency.

Moving future, new measures can be implemented to increase efficiency. Implementing a data structure that fetches the entire dataset and keeps it in memory can cut processing time by eliminating frequent database reading. Using improved pruning strategies to remove unnecessary item sets from the candidate list before producing the Apriori level improves efficiency.

While our discussion thus far has primarily focused on the basic Apriori algorithm, it's worth noting that various algorithmic variations have been developed, including alternative frequent data mining algorithms. Some of these variations outperform the original Apriori algorithm in certain scenarios.

## REFERENCES


[1] Hu, L., Zhuo, G., & Qiu, Y. (2009, August). Application of Apriori algorithm to the data mining of the wildfire. In Fuzzy Systems and Knowledge Discovery, 2009. FSKD'09. Sixth International Conference on (Vol. 2, pp. 426-429). IEEE.

[2] Korde, N. S., & Shende, S. W. (2014). Parallel Implementation of Apriori Algorithm.

[3] IOSR Chai, S., Yang, J., & Cheng, Y. (2007, June). The research of improved apriori algorithm for mining association rules. In Service Systems and Service Management, 2007 International Conference on (pp. 1-4). IEEE.

[4] Changsheng, Z., Zhongyue, L., & Dongsong, Z. (2009, March). An improved algorithm for apriori. In Education Technology and Computer Science, 2009. ETCS'09. First International Workshop on (Vol. 1, pp. 995-998). IEEE.

[5] F. Lastname1 Spandana, K., Sirisha, D., & Shahida, S. (2016). Parallelizing Apriori Algorithm on GPU. International Journal of Computer Applications, 155(10).

[6] Rathee, S., Kaul, M., & Kashyap, A. (2015, October). R-Apriori: an efficient apriori based algorithm on spark.

[7] Parsania, V., Kamani, G., & Ghodasara, Y. R. (2014). Mining Frequent Itemset Using Parallel Computing Apriori Algorithm.

[8] Li, N., Zeng, L., He, Q., & Shi, Z. (2012, August). Parallel implementation of apriori algorithm based on mapreduce. In Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on (pp. 236-241). IEEE.

[9] Shah, A. (2016, July). Association rule mining with modified apriori algorithm using top-down approach. In Applied and Theoretical Computing and Communication Technology.

[10] (iCATccT), 2016 2nd International Conference on (pp. 747-752). IEEE.