A PARALLEL APRIORI-TRANSACTION REDUCTION ALGORITHM USING HADOOP-MAPREDUCE IN CLOUD

AL Sayeth Saabith[1], Elankovan Sundararajan[2], Azuraliza Abu Bakar[3]

[1,2] Centre for Software Technology and Management, [3] Center for Artificial Intelligence and Technology, Faculty of Information. Science and Technology, Universiti Kebangsaan Malaysia, UKM Bangi, 43600, Selangor-DE, Malaysia.

ABSTRACT

Apriori algorithm is a classical algorithm of association rule mining and widely used for generating frequent item sets. However, the original Apriori algorithm has some limitation such as it needs to scan the dataset many times to discover all frequent itemset and generate huge number of candidate itemset. To overcome these limitations, researchers have made a lot of improvements to the Apriori such as candidate generation, without candidate generation, transaction reduction, partitioning, and sampling. When it comes to mine massive data, these algorithms failed to prove efficiency because limitation of the processing capacity, storage capacity, and main memory constraints. Therefore, parallel and distributed algorithms are developed to perform large-scale computing in ARM on multiple processors. However, the problems with most of the parallel and distributed framework are overheads of managing distributed system, lack of high level parallel programming language, and node failures. Hadoop-MapReduce is an efficient, scalable, and simplified programming model for massive data processing and it also available on cloud environment. Cloud computing offers huge computing resources, and capacities to solve big data challenges. Recently many parallel algorithms have been proposed on Hadoop-MapReduce to enhance the performance of Apriori algorithm but there are some drawbacks: since multiple scan over the dataset is needed to generate candidate itemset, it consume more execution time. The aim of this study is to propose a parallel Transaction Reduction MapReduce Apriori algorithm (TRMR-Apriori) which is reduce unnecessary transaction values and transactions from the dataset in parallel manner to overcome above problems. The experiments show that TRMR-Apriori is able to achieve better execution time to discover frequent itemset those of previous sequential ARM algorithms such as Apriori, AprioriTid, Eclat, and FP-Growth and the previous parallel algorithms such as PApriori, MRApriori, and Modified Apriori with different condition on homogeneous computing environment using Hadoop-MapReduce platform in cloud. Overall, the TRMR-Apriori shows the strength to extract the frequent itemset from massive dataset in cloud.

Keywords: [Apriori, Hadoop-Mapreduce, ARM, Parallel, Cloud, Transaction Reduction]

1. INTRODUCTION

Data mining and knowledge discovery has emerged to extract useful, interesting, and unknown patterns and knowledge from huge amount of database. Association Rule Mining (ARM) is one of the most important and popular technique of data mining which find interesting correlation or association between set of items or attributes and also frequent patterns in large database [1,2]. The most typical application of ARM is in market basket analysis which analyses the purchasing behaviour of customers by finding the frequent items purchased together. In addition to the many business application, it is also applicable to bi-informatics, medical diagnosis and text analysis [3].

**Corresponding author: Email: XYZ@ABC.COM*

The Apriori algorithm is one of the most widely used algorithm in ARM that collects the item sets which frequently occur in order to discover association rule in massive datasets. The original Apriori algorithm is for the sequential (single node or computer) environment. This Apriori algorithm has many drawbacks to process huge datasets such as single machine memory, CPU and storage capacity are insufficient [4].

Parallel and distributed computing is a better solution to overcome the above problems. Many researches have been carried out for parallelizing the Apriori algorithm. Apriori parallel algorithms handles gigantic dataset on various platforms with different configurations. There are many major challenges such as to achieve efficacious load balancing, utilize total memory system, intended new algorithm for memory utilize, consider different data layout, produce effective parallel algorithm, reduce communication cost among processors, system failure, data recovery, simplify parallel programming issue, manage scalability, and high availability.

Cloud systems which can be effectively employed to handle parallel mining since they provide scalable storage and processing services, as well as software platforms for developing and running data analysis environments. We exploit Cloud computing platforms for running big data mining processes designed as a combination of several data analysis steps to be run in parallel on Cloud computing elements.

To overcome the above stated major challenges, the Hadoop-MapReduce platform is more suitable solution. Hadoop is an open source programming framework is capable to running applications for large scale processing and storage on large clusters of commodity hardware.

Numerous techniques have been proposed to improve the efficiency of classical Apriori algorithm, such as direct hash and pruning (DHP), transaction reduction, partitioning, sampling, dynamic itemset counting (DIC), vertical layout techniques, and FP-Growth [5,6,7,8,9,10,11,12,13]. The transaction reduction is one of the best way to extract all frequent itemset from massive dataset. Transaction reduction is more suitable for big and dense dataset in parallel and distributed environment because the transaction reduction technique perform for generating k-frequent itemsets, if the length of a transactions is less than k, the transaction is not necessary to scan and C_k is the set of entire items that contain k-frequent itemsets, while generating $k+1$ frequent itemsets, any item called A including in $(C_k - C_{k-1})$ is not essential to scan in all transactions[6]. So we can reduce unnecessary transaction value as well as transaction from the dataset in parallel manner. In this thesis we propose transaction reduction method to improve the efficiency of Apriori algorithm in parallel and distributed on cloud using Hadoop-MapReduce platform.

The association rule mining (ARM) is very important task within the area of data mining [14]. The main task of every ARM algorithm is to discover the set of items that frequently appear together, the frequent itemsets. Finding frequent itemsets in transaction databases has been demonstrated to be useful in several business field [15]. Apriori is one of the most popular data mining approaches for finding frequent itemsets from transactional datasets. The Apriori algorithm is the basis of many other well-known algorithms and implementations. However, it has some limitations such as it needs to scan the dataset many times and to generate many candidate itemsets. The main challenge faced by researchers in frequent itemset mining has been to reduce the execution time. Most of the current data mining algorithms like Apriori are good for the databases that are small in size. Unfortunately, when the data size is huge, both memory use and computational cost can still be very expensive.

In addition, single processors memory and CPU resources are very limited, which make the algorithm performance inefficient. Furthermore, exponential growth of worldwide information, organizations have to deal with ever growing amount of data. As these data grow past

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

hundreds of gigabytes towards a terabyte or more, it becomes nearly impossible to process them on a single machine. The solution for the above problems is parallel and distributed computing.

Parallel and distributed computing offer a potential solution for the above problems if the efficient parallel and distributed algorithm can be implemented. Most of the parallel ARM algorithms are based on parallelization of Apriori that iteratively generates and tests candidate itemsets from length $1\ to\ k$ until no more frequent itemsets are found. This algorithm can be categorized into three, those are Count Distribution, Data Distribution and Candidate Distribution [16].

Experiment show that the Count Distribution method exhibits better performance and scalability than the other two methods [17,18,19]. Cloud computing platforms as a possible solution for mining and analyzing massive data. With the cloud computing paradigm, the data mining and analysis can be more accessible and easy due to cost effective computational resources.

The Apache-Hadoop framework is evolving as the best new approach for analyzing big data in parallel and distributed cloud environment [20,21]. Hadoop is an open source distributed framework which is design based on the Google MapReduce programming model. Hadoop-MapReduce is a programming model for easily and efficiently writing applications which process vast amount of data in parallel on large clusters of commodity hardware in a reliable, fault tolerance manner.

A MapReduce programme partitions the input dataset into independent splits which are processed by the map tasks in a completely parallel manner. The Hadoop frameworks combines and stores the maps output as a set of intermediate key/value pairs which are then fetched as an input for the reduce tasks.

From literature study it is clear that parallel implementation of Apriori algorithm is more efficient and has good performance compare to the sequential Apriori algorithm. The use of Hadoop-MapReduce framework provides further advantages such as the Apriori algorithm requires frequent transaction set scanning; Hadoop provides HDFS storage with good parallel reading-writing characteristics, thus the time repeatedly read transaction set is greatly shortened compared with the traditional storage system, which makes it possible to mine large amount of data on Hadoop with Apriori algorithm. The calculation with Apriori algorithm can be regarded as process of counting, and this process is suitable for MapReduce model, so Apriori algorithm has the natural characteristics of MapReduce [22].

1.1 Problem Statement

Apriori algorithm effectively discover all the frequent itemsets from the data repositories however it suffers some limitation such as Apriori is time consuming to hold a vast number of candidate sets with many frequent itemsets, Apriori algorithm checks many sets from candidate itemsets and it scans database many times repeatedly for finding candidate itemsets, and Apriori will be very slow and inefficient when memory and storage capacity is limited with large number of transactions. Researchers have been proposed new algorithms based on the Apriori algorithm and try to improve the efficiency by making some modifications, such as reducing the number of passes over the database; reducing the size of the database to be scanned in every pass; pruning the candidates by different techniques such as direct hash pruning, transaction reduction, partitioning, sampling, dynamic itemset counting, vertical dataset layout, and frequent pattern tree [8,17,21,23].

Most of the data mining sequential algorithms are suffering to handle the data dimension and massive dataset. Sequential algorithms are inefficient to improve run time performance for

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

such huge databases [17,21,24,25]. Therefore, parallel and distributed algorithms are developed to perform large-scale computing in ARM on multiple processors. Researchers have been proposed numerous algorithms using partitioning, frequent pattern tree, direct hash pruning techniques on Hadoop-MapReduce platforms to enhance the performance of Apriori algorithm such well-established algorithms are PApriori, MRApiori, and Modified Apriori, but there are some drawbacks: since multiple scan over the dataset is needed to generate candidate itemset, it consume more execution time [8,19,20,21,25,26,27,28,29,].

Transaction reduction is one of the best technique to reduce the execution time for discover all frequent itemset from massive transaction dataset by reduces the transaction that does not contain any frequent k+1 itemset and such transaction may be marked and removed. This technique is more significant when implement in parallel and distributed environment [6,7,8]. This study aims to propose a transaction reduction parallel Apriori algorithm using Hadoop-MapReduce platform in cloud and investigate the proposed algorithm performance with existing various sequential ARM algorithms as well as parallel algorithms were developed using Hadoop-MapReduce platform under the homogeneous system environment.

The objective of this study are

- To investigate the performance of different sequential ARM algorithms in cloud environment and parallel Apriori algorithm using Hadoop-MapReduce in cloud.

- To propose a transaction reduction algorithm for parallel Apriori based on Hadoop-MapReduce.

The outcomes of this study are to set-up and configure the cloud environment to test big data analysis and improving parallel Apriori algorithm using transactional reduction method.
Cloud Data Mining offers tremendous potential for analyzing and extracting the useful information in various fields of human activities: finance, banking, medicine, genetics, biology, pharmacy, and marketing. Cloud provides technology that can handle huge amounts of data, which cannot be processed efficiently and at reasonable cost using standard data mining technologies and techniques. Hadoop-MapReduce is the current trend in the field of huge data processing. Based on the algorithms and technologies developed by large Internet companies, there is a quite widespread ecosystem of solutions for processing and analysis of huge amounts of data. Proposed transaction reduction algorithm may increase the performance compared to earlier well developed versions of sequential and parallel versions Apriori algorithms.

2. METHODOLOGY

This study consists 6 phases to accomplish this research: problem identification, massive dataset collection (benchmark data), set-up and configure cloud environment, implement sequential algorithms in cloud, Test the proposed transaction reduction Apriori algorithm as well as existing some parallel Apriori in cloud environment., evaluation.

Phase 1: Deep and through understanding of related problems area such as ARM problem description, sequential Apriori algorithm and their drawbacks, way of improvement, parallel Apriori algorithm using Hadoop-MapReduce by the critical and in depth evaluation of previous research.

Phase 2: Collect massive dataset from online to evaluate the proposed transaction reduction MapReduce Apriori algorithm and pre-process the dataset into suitable format that is convenience to Hadoop-MapReduce framework.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

Phase 3: Set-up and configure the cloud environment to test proposed transaction reduction MapReduce Apriori algorithm on Amazon web service (AWS) and create Hadoop elastic MapReduce instances on AWS and upload the pre-processed dataset to simple storage service (S3) to evaluate the performance of proposed algorithm.

Phase 4: Implement the different techniques versions of sequential Apriori algorithm with varying min_sup value in cloud.

Phase 5: Test the proposed transaction reduction Apriori algorithm as well as existing some parallel Apriori MapReduce based algorithm in cloud environment varying min_sup, number of transaction, and number of nodes.

Phase 6: The proposed transaction reduction Apriori algorithm is evaluated and compared with existing well known versions of sequential and parallel Apriori algorithm with varying condition min_sup value as well as existing parallel Apriori algorithm with varying condition such as min_sup, number of nodes, and number of dataset transactions in unique cloud environment.

2.1 PROPOSED TRANSACTION REDUCTION MAPREDUCE APRIORI ALGORITHM (TRMR-APRIORI)

We propose an improved Apriori algorithm based on Hadoop-MapReduce model, called TRMR-Apriori. It reduces the number of transactions values and number of transactions along with reducing the number of candidate item sets generated. We used pruned and transaction reduction theory as follows:

A. Theorem 1: In k-frequent itemset $T = \{i_1, i_2, i_3, .., i_k\}$, if there is an item i_j that does not satisfy the condition $countitem(i_j) < k-1$ then T is infrequent itemset where $j < k$ and $countitem(i_j)$ are the number of itemsets containing the item i_j in the collection of $k-1$ frequent itemset, it is typically denoted as L_{k-1}
Proof: - Assume T is a k-frequent itemset, then all $k, k-1$ subset of *T* are in L_{k-1}, and every item of T named i_j will exit in $k-1$ subsets of T. it means that the number of itemsets containing item i_j is $k-1$, so every item named i_j of T could make the value of $countitem(i_j)$ to be no less than $k-1$ because the collection named L_{k-1} includes all subsets of T, but this is contradicts with the condition, so the assumption does not hold, so T is not a frequent itemset.

B. Theorem 2: For generating k-frequent itemsets, if the length of a transactions is less than k, the transaction is not necessary to scan.

   Proof:- generating k-frequent itemsets must scan the transaction database to calculate the frequency of frequent itemsets, if the length of a transaction is less than k. it means the number of items it contains is no more than k, indicates that the transaction cannot contains the k-frequent itemsets, so there is no need to scan this transaction.

C. Theorem 3: Assume C_k is the set of entire items that contain k-frequent itemsets, while generating $k+1$ frequent itemsets, any item called A including in $(C_k - C_{k-1})$ is not essential to scan in all transactions.

   Proof: - for any item called A including in $(C_k - C_{k-1})$, A is not subsist in k-frequent itemsets, moreover, A is not subsist in $k+1$ frequent itemsets definitely, so A

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

cannot help the generation of $k+1$ frequent itemsets, therefore not necessary to scan the item A.

In general, theorem 1 is using for pruned optimization that is generating *k*-candidate itemsets, using current L_{k-1} frequent itemsets, we can count and save the all frequency items from the transaction database. If the frequency of any item in the transaction database is less than $k-1$, this item cannot be utilized to produce the $k-itemsets$, so we eliminate all the frequent itemsets enclosing the item in L_{k-1}, this can helps to efficiently trim the scale of L_{k-1} and produce C_k more effectively. The theorem 2 and 3 based on transaction reduction optimization strategy means to generate *k+1* frequent itemset scan the database and delete all infrequent k-itemset from all transactional database, delete the transaction whose length is less than *k+1* directly [30].

The proposed parallel transaction reduction MapReduce Apriori (TRMR-Apriori) algorithm is based on pruned optimization techniques to optimize the generation of frequent itemsets and meanwhile, transaction reduction technique is to shorten the transaction database. All of these techniques such as transaction reduction, pruning, and parallel implementation on Hadoop-MapReduce will effectively decrease the system overhead and improve the execution time, improve the efficiency of earlier existing well known sequential ARM algorithm as well as parallel algorithm based on Hadoop-MapReduce. The proposed TRMR-Apriori algorithm flow chart is illustrated in Figure 1.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

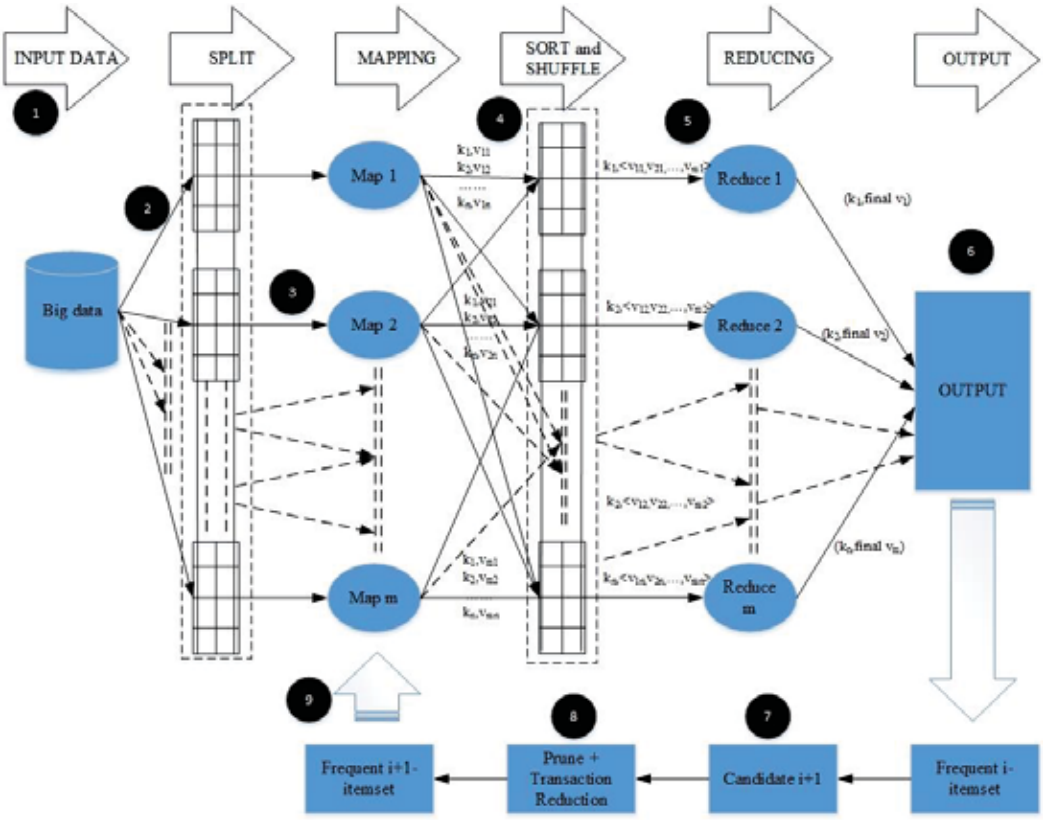


Figure 1. TRMR-Apriori flow chart

Step 1: The dataset (DB) pre-processed <key, value> format which contains information about particular transaction id and items.

Step 2: For processing parallel scan, partition the complete dataset (DB) into number of available processors $DB = \{DB_1, DB_2, DB_3, \dots \dots, DB_n\}$ where *n* is number of processors available to distribute the dataset.

Step 3: The split chunk of the data file fed to the available mapper. The Map function reads one line a time and assigns each item a key and the value associated with the key to find the frequent 1-itemset.

TRMR-Apriori mapper algorithm
Map (key=Tid, value=itemset in transaction t_i*)*
{
//split itemset based on space
String items{} = itemset.split(" ")
for all transaction $t_i \in D_i$ *{*
    for all item $i \in t_i$ *{*
        output $< i, 1 >$*;*
    }
}

Step 4: Shuffle and sort takes place in between map and reduce steps. It is responsible for ordering the <key, value> pairs from map function.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

Step 5: The reducer's job is to process the data that come after shuffle and sort and intermediate results means candidate itemset store in HDFS.

TRMR-Apriori reducer algorithm
Reduce (key=item, value=count)
for all key x *{ /* initial x.count=0*/*
for all value y *in* $x's$ *value list {*
$x.count+=y$;
}
if $x.count >= min\_sup${
output$< x, x.count >$; */*collected in* L_1 */*
}
}

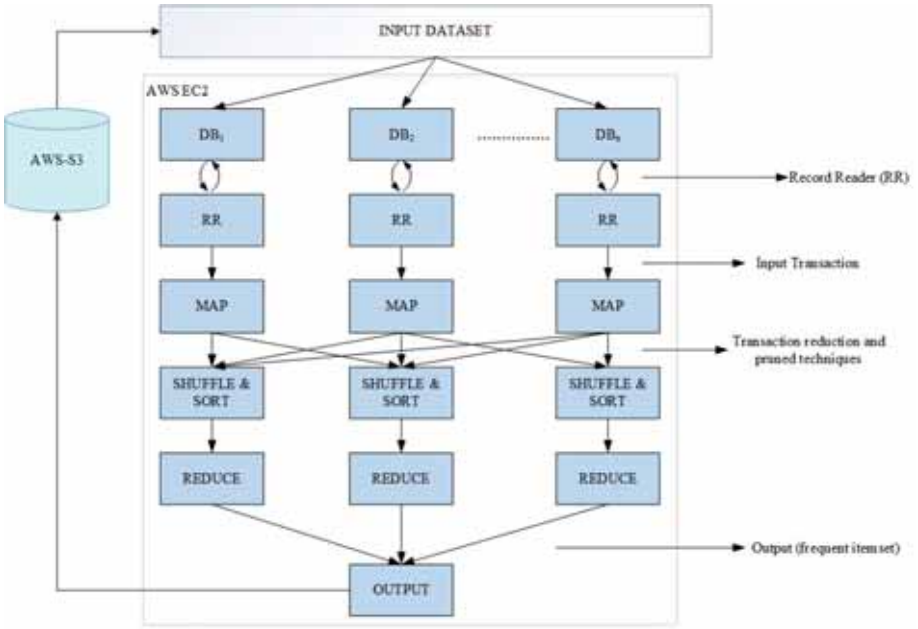
Step 6: Compare the intermediate results with predefined minimum support value and find the all 1-frequent itemset and store in the distributed cache.

Step 7: Ready to find the 2-candidate itemset.

Step 8: Implement our proposed transaction reduction methods with all split transaction that are contain the transaction length less than with next candidate itemset so can reduce the multiple scanning time and also pruned all non-frequent itemset transaction also should eliminated from the whole sub divided transaction. Finally we can increase the execution time performance from pruning and transaction reduction method to extract whole frequent itemset from given massive data.

Step 9: Repeat the process to find all frequent itemset efficiently

Detail architecture diagram of TRMR-Apriori algorithm using Hadoop-MapReduce on AWS presented in Figure 2.



* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

2.1.1 TRMR-Apriori pseudo code

Input: transaction database partition D_i
TRMR-Apriori mapper algorithm
Map (key=Tid, value=itemset in transaction t_i)
{
//split itemset based on space
String items{} = itemset.split(" ")
for all transaction $t_i \in D_i$ {
    for all item $i \in t_i$ {
        output $< i, 1 >$;
    }
}
TRMR-Apriori reducer algorithm
Reduce (key=item, value=count)
for all key x { /* initial x.count=0*/
    for all value y in $x's$ value list {
    $x.count +=$ y;
    }
    if $x.count >= \min\_sup${
    output$< x, x.count >$; /*collected in L_1 */
    }
}
Input: transaction database partition D_i and L_{k-1} $(k >= 2)$
TRMR-Apriori mapper algorithm
Map (key=Tid, value=itemset in transaction t_i)
read L_{k-1} from DistributedCache;
for $(k = 2; L_{k-1} \neq \emptyset; k++)$
{
for $C_k = apriori - gen(L_{k-1}, \min\_sup)$;
for all transaction $t_i \in D_i$ {

$C_t = subset(C_k, t_i)$;
for all candidate $c \in C_t$
$L_k = \{c \in C_k \mid c.count \geq \min\_sup\}$;
If $(k \geq 2)$
emit_transaction_value$(D, L_k.L_{k-1})$;
emit_transaction_row(D, L_k);
{
Output$< c, 1 >$;
}
    }
}
TRMR-Apriori reducer algorithm
Reduce (key=item, value=count)
for all key x { /* initial x.count=0*/
    for all value y in $x's$ value list {

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

$x.count+=$ y;
}
if $x.count >= \ \min\_sup$
output$< x, x.count >$; /*collected in L_k */
}
Procedure emit_transaction_value (D_i: Database; L_k: frequent k-itemsets; L_{k-1}: frequent k-1 itemsets, tv: transaction_value)
for all itemset $i \in L_{k-1} and\ i \notin L_k$
{
for all transaction $t \in D$
{
for all $tv \in t$
{
if $(tv = i)$
update *tv*=null;
}}}

Procedure emit_transaction_row (D_i: Database; L_k: frequent k-itemsets; tr: transaction_row)
for all transaction $t \in D$
{
for all $tv \in t$
{
$if(tv \neq null\ and\ tv \neq 0)$
{
$tr.count++;$
}}
$if(tr.count < k)$
{
delete tr;

2.2 TRMR – Apriori working Scenario

Assume that Table 1 be a transaction database *D* to briefly describe how the proposed transaction reduction MapReduce algorithm works. This transaction database of seven transactions, min_sup= 2/7 or 28%, and three map nodes.

Table 1. Sample transaction dataset

TID	Items
t1	Mango, Banana
t2	Orange, Apple
t3	Mango, Banana, Durian
t4	Mango, Durian
t5	Banana, Orange, Durian
t6	Orange, Durian
t7	Durian

Phase one: Given transaction database *D* divided into three sub dataset as T_1 consist *t1* and *t2*, T_2 with *t3*, *t4*, and T_3 with *t5*, *t6*, and *t7*. Now three subdivided dataset are distributed to three map nodes that is T_1, T_2, and T_3 to M_1, M_2, and M_3 respectively. So, we able to obtain 1 candidate items C_{m1} with itemset pair $< item, count >$ on the map nodes

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

M_1, M_2, and M_3. M_1 handles transaction database T_1 by outputting $< Mango, 1 >, < Banana, 1 >, < Orange, 1 >, \ and \ < Apple, 1 >$. M_2 handles transaction database T_2 by generating output as $< Mango, 2 >, < Banana, 1 >, and \ < Durian, 2 >$. M_3 handles transaction database T_3 by outputting $< Banana, 1 >, < orange, 2 >, < Durian, 3 >$

$$C_{11} = \{< Mango, 1 >, < Banana, 1 >, < Orange, 1 >, < Apple, 1 >\}$$
$$C_{21} = \{< Mango, 2 >, < Banana, 1 >, < Durian, 2 >\}$$
$$C_{31} = \{< Banana, 1 >, < Orange, 2 >, < Durian, 3 >\}$$

From the all map nodes, reducer sums up candidate 1 item pairs C_1 and find the 1-frequent item pair L_1 that $C_1 >= \ min\_sup$ value.

$$C_1 = \{[Mango, 3], [Banana, 3], [Orange, 3], [Apple, 1], [Durian, 5]\}$$

$[Apple, 1]$, not greater than equal to min_sup. Apple is not frequent itemset anymore, so we can eliminate the transaction which consist Apple item by using theorem 1. Finally we can generate 1-frequent itemset L_1 as follows:

$$L_1 = \{[Mango, 3], [Banana, 3], [Orange, 3], [Durian, 5]\}$$

The Figure 3 illustrates the phase 1 process

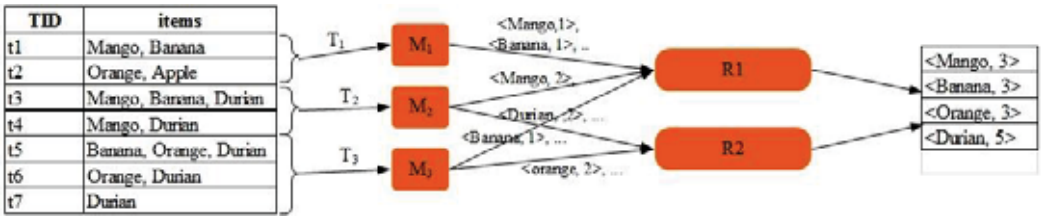


Figure 3. TRMR-Apriori phase one process

Phase two: check the transaction before generating next candidate itemset if number of transaction is less than candidate itemset then remove the particular transactions. Here *t7* has only one transaction this is less than to generate 2-candidate itemset. So *t7* eliminate before generating 2-candidate and frequent itemset and transaction *t2* consist *Apple* item set, we already found *Apple* is not frequent itemset so we can eliminate item *Apple* from the transaction *t2* so now *t2* has only one transaction this is less than 2-candidate itemset. So *t2* eliminate before generating 2-candidate itemset too. In this phase, T_1 consist *t1,* T_2 with *t3*, and *t4*, and T_3 with *t5*, and *t6*, and each mapper reads L_1, means L_1 maps to M_1, M_2, and M_3, thus

$$L_{11} = \{Mango, Banana\}$$
$$L_{21} = \{Orange\}$$
$$L_{31} = \{Durian\}$$

We can generate C_{m2} by using Apriori property as joining and sorting in hash tree, where the duplicate itemsets are removed.

$$C_{m2} = L_1 \bowtie L_{m1}$$

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

$$C_{12} = \begin{Bmatrix} < Mango, Banana >, < Mango, Orange >, \\ < Mango, Durian >, < Banana, Orange >, \\ < Banana, Durian > \end{Bmatrix}$$

$$C_{22} = \{ < Mango, Orange >, < Banana, Orange >, < Orange, Durian > \}$$

$$C_{32} = \{ < Mango, Durian >, < Banana, Durian >, < Orange, Durian > \}$$

$$C_2 = \begin{Bmatrix} [Mango, Banana], [Mango, Orange], \\ [Mango, Durian], [Banana, Orange], \\ [Banana, Durian], [Orange, Durian] \end{Bmatrix}$$

From the all map nodes, reducer sums up candidate 2 item pairs C_2 and find the 2-frequent item pair L_2 that C_2>= min_sup value.

$$C_2 = \begin{Bmatrix} [Mango\ Banana, 2], [Mango\ Orange, 0], [Mango\ Durian, 2], \\ [Banana\ Orange, 1], \\ [Banana\ Durian, 2], [Orange\ Durian, 2] \end{Bmatrix}$$

From the above result the two itemset $[Mango\ Orange, 0], [Banana\ Orange, 1]$ are not greater than equal to min_sup value. We can generate 2-frequent itemset L_2

$$L_2 = \begin{Bmatrix} [Mango\ Banana, 2], [Mango\ Durian, 2], \\ [Banana\ Durian, 2], \\ [Orange\ Durian, 2] \end{Bmatrix}$$

Figure 4 illustrates the above procedure in graphical view.

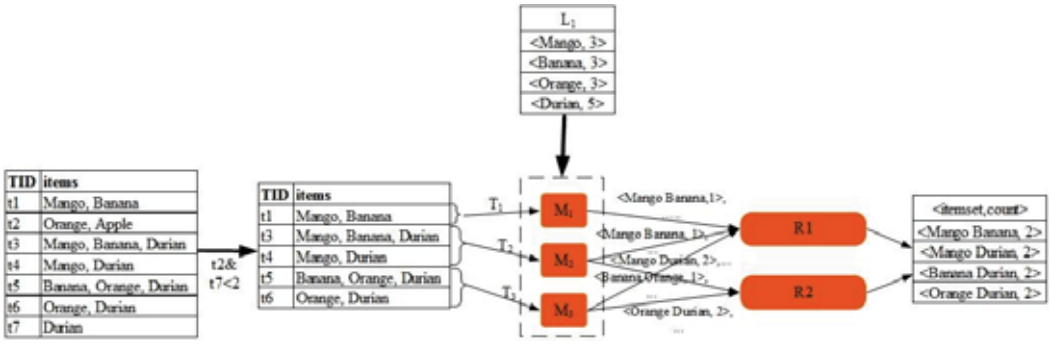


Figure 4. TRMR-Apriori find the frequent 2-itemset

Phase three: in this phase we check the transaction before generating next candidate itemset if number of transaction is less than candidate itemset then remove the particular transactions. Here *t1, t2, t4*, and *t6* has only two transactions this is less than to generate 3 – candidate itemset. So *t1, t2, t4*, and *t6* removed before generating 3- candidate and frequent itemset. In this phase, T_1 no transaction, T_2 with *t3*, and T_3 with *t5*, and each mapper reads L_2, means L_2 maps to M_1, M_2, and M_3, thus

$$L_{12} = \{Mango\ Banana, Mango\ Durian\}$$

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

$$L_{22} = \{Banana\ Durian\}$$
$$L_{32} = \{Orange\ Durian\}$$

We can generate C_{m3} by using Apriori property as joining and sorting in hash tree, where the duplicate itemsets are removed.

$$C_{m3} = L_2 \bowtie L_{m2}$$
$$C_{13} = \{< Mango, Banana, Durian >, < Mango, Orange, Durian >, < Banana, Orange, Durian >\}$$
$$C_{23} = \{< Mango, Banana, Durian >, < Banana, Orange, Durian >\}$$
$$C_{33} = \{< Mango, Orange, Durian >, < Banana, Orange, Durian >\}$$

As the time size k is greater than or equal 3, prune step takes the part to remove non frequent itemset that violates Apriori property that is all nonempty subset of frequent items must be also frequent. When using prune step $C_{13}, C_{23}, and\ C_{33}$ becomes as follows:

$$C_{13} = \{< Mango, Banana, Durian >,\}$$
$$C_{23} = \{< Mango, Banana, Durian >\}$$
$$C_{33} = \{\}$$

Above itemsets $< Mango, Orange, Durian >, < Banana, Orange, Durian >$ removes from C_{13} because $< Mango, Orange >, < Banana, Orange >$ are not member of L_2, $< Banana, Orange, Durian >$ removes from C_{23} because $< Banana, Orange >$ are not member of L_2, and $< Mango, Orange, Durian >, < Banana, Orange, Durian >$ removes from C_{33} because $< Mango, Orange >, < Banana, Orange >$ are not member of L_2. Finally C_3 becomes

$$C_3 = \{< Mango, Banana, Durian >\}$$

From the all map nodes, reducer sums up candidate 3 item pairs C_3 and find the 3-frequent item pair L_3 that $C_3 >= min\_sup$ value.

$$C_3 = \{[Mango\ Banana\ Durian, 1]\}$$

C_3 does not have the value greater than min_sup value since L_3 is empty and algorithm terminated.

$$L_3 = \{\}$$

Therefore, we can generate $L_1\ and\ L_2$ efficiently using the proposed transaction reduction parallel Apriori algorithm on Hadoop-MapReduce framework.

$$L_1 = \begin{Bmatrix} [Mango, 3], [Banana, 3], \\ [Orange, 3], [Durian, 5] \end{Bmatrix}$$

$$L_2 = \begin{Bmatrix} [Mango\ Banana, 2], [Mango\ Durian, 2], \\ [Banana\ Durian, 2], \\ [Orange\ Durian, 2] \end{Bmatrix}$$

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

Figure 5 illustrates the above procedure in graphical view

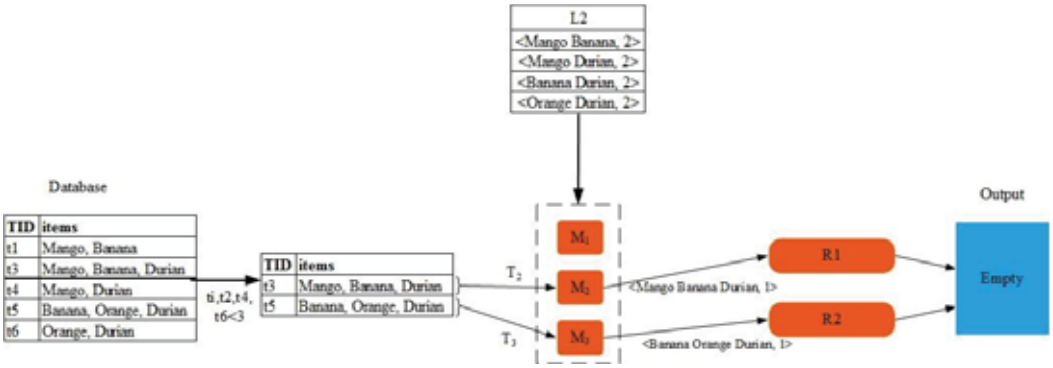


Figure 5. TRMR-Apriori find the frequent 3-itemset

2.3 TESTING CLOUD ENVIRONMENT AND DATASETS

In this study, we used Amazon web as a cloud service provider to evaluate the proposed algorithm performance on cloud. Elastic computing cloud (EC2), simple storage service (S3), and elastic map reduce (EMR) services have been used to analyse the performance of the proposed transaction reduction Apriori parallel algorithm on AWS cloud platform and we tested existing sequential versions of ARM algorithm on Amazon EC2 using SPMF open-source data mining library.

2.3.1 SPMF

SPMF is an open-source data mining library for frequent pattern mining. It was developed under the GPL v3 license and written java programming language. It has 93 data mining algorithms for sequential pattern mining, association rule mining, itemset mining, sequential rule mining, and clustering. SPMF can be used as a standalone program with a simple user interface or from the command line.

In this thesis, we used SPMF in command line prompt on Amazon EC2 instance to evaluate the performance of selected sequential algorithms with selected benchmark dataset.

2.3.2 Amazon EC2

Amazon EC2 which is also known as Elastic Compute Cloud provides resizable computing capacity in the AWS cloud platform. It eliminates the need to invest in computing hardware up front which saves money but also allows applications to be developed and deployed faster. Amazon EC2 can uses to launch a virtual machine and can configure all of the associated security and network settings. Amazon EC2 instances will automatically scale-up to add capacity to handle user needs. Amazon EC2 environment typically called instances. There are wide range of instance types that have varying combinations of CPU power, amount of memory, storage size, and networking capacity.

Amazon EC2 also provides pre-configured templates for user instances known as Amazon Machine Images (AMI). These AMI templates can include just an operating system like Windows or Linux they can include a wide range of components such as operating system, and pre-installed software packages. Amazon EC2 instances range start from small “micro” instances for small jobs to high performance “X-large” instances for like data warehousing.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

2.3.3 Amazon S3

Amazon Simple Storage Service also known as S3 is safe, secure, and highly scalable object storage in the cloud. It can be used to store and retrieve any amount of data anytime from anywhere on the web. Amazon S3 is a very affordable solution for hosting data on the web since user only pay for the storage in bandwidth user use. Amazon S3 users can also use S3 to securely store their own information and backup their own critical date offsite. Amazon S3 is designed for more durability, and very reliable. Amazon S3 storage data as objects and objects restored within folders that are called buckets to store an objects in Amazon S3. If user want to store the files on S3 buckets, they can upload that file with set of permission on the objects so it is should be private.

2.3.4 Amazon EMR

Amazon elastic map reduce gives the ability to process huge amount of data such as GB, TB, and more without the need to buy or manage hardware. It offers the ability to easily launch, scale up/down, and manage apache Hadoop clusters on Amazon web services, and can quickly leverage the large ecosystem of tools and applications in the Hadoop ecosystem. Amazon EMR provides a managed framework to easily create, customize, and use Hadoop clusters created with Amazon EC2 instances.

Amazon EMR also has tight integration with other Amazon web services like Amazon S3 where can store input and output data for Amazon EMR cluster analysis. Amazon EMR takes much of the complexity out of managing cluster of Amazon EC2 instances. Amazon EMR takes care to setting up hardware, installing, tuning, and tuning Hadoop. So user can focus on the analysis of data instead of managing infrastructure and software. Amazon EMR can easily expand or shrink the cluster by adding and removing Amazon EC2 nodes according to the user needs. If data store in Amazon S3, it can be access by multiple Amazon EMR clusters to run separate workload simultaneously.

User can speed up their innovation and experimentation results using Amazon EMR service and they can pay only for their usage. It is easy to use a verity of customize efficient programs in the Hadoop ecosystem on specific cluster such as MapReduce, Apache Hive, and Pig. Amazon EMR can access via AWS management console, command line, or SDK.

We deployed the proposed TRMR-Apriori algorithm, selected parallel Map-Reduced based Apriori algorithm jar files on Amazon EC2 for evaluating the performance by increasing the number of nodes, dataset size, and varying min_sup value in the set up. The data input files were saved on S3. Amazon S3 is a data storage service. Transfer between S3 and Amazon EC2 is free. This makes use of S3 attractive for Hadoop users who run clusters on EC2. The details related like path and permission to S3 is configured. The output data are also written back in the bucket of S3 at the end. The temporary data is written in the HDFS files so transaction reduction techniques can apply to find the frequent itemset effectively.

Amazon EMR takes care of provisioning a Hadoop cluster, running the job flow, terminating the job flow, moving the data between Amazon EC2 and Amazon S3, and optimizing Hadoop. Amazon EMR removes most of the difficulties associated with the Hadoop configuration like setting up the hardware and networking required by the Hadoop cluster, such as monitoring the setup, configuring Hadoop, and executing the job flow.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

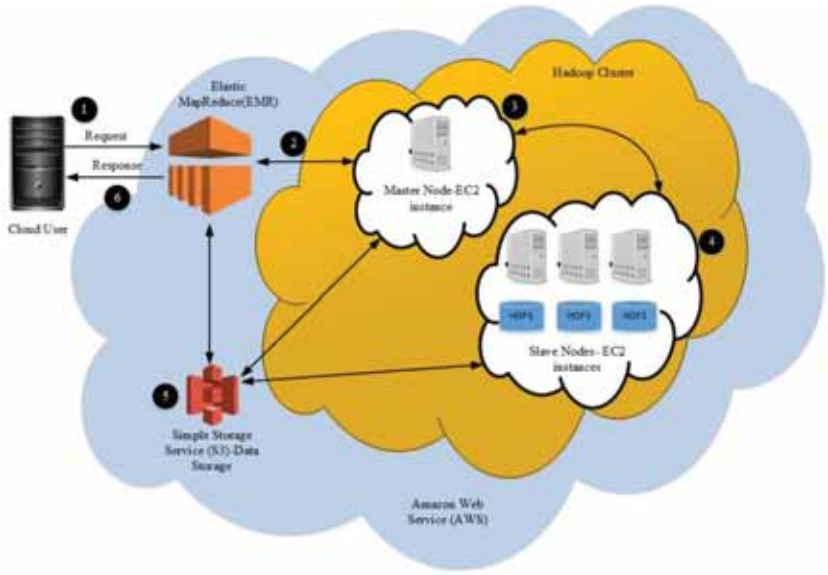


Figure 6. Hadoop job flow on the AWS-EC2

Hadoop on the cloud can be shown in Figure 6. The request is sent to the EMR model, to start the job with all the details required for the job, such as path to S3 etc. The Hadoop cluster with master and slave instances is created. The Hadoop cluster works on the job and finishes the job. The temporary created during the run of job files can be stored in HDFS or S3. Storing in S3 may not be the right choice for all the cases as it may add to communication overhead. The output after the job is stored in S3. Only the error or fatal messages are written on the screen during the entire run of job. Once the job is complete response is sent indicating the completion of the job.

2.4 DATASET DETAILS

In order to prove the effectiveness and efficiency of our proposed TRMR-Apriori algorithm, the experiment is done using well-known datasets which are available online. Those selected datasets are generating frequent itemsets, and commonly used in many pervious research experiments [20,25,26,31]. We have selected five benchmark dataset to evaluate the performance of proposed method comparing with selected existing sequential ARM algorithms. Those selected dataset details are as follows in Table 2 and Figure 7 illustrates the sample snapshot of accidents dataset:

Table 2. Selected dataset details

Dataset	Total instances	Total attributes
Connect	64,557	42
Retail	88,162	72
Accidents	340,184	45
Kosarak	990,002	625
Webdocs	1,692,082	71,472

For experiments T40I10D100K dataset which has been generated by IBM's Quest Synthetic Data Generator is used and it was obtained from FIMI repository. It has 100000 transactions; each transaction contains 40 items in average.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

For the T40I10D100K, we have replicated it to 2fold, 4fold, 8fold, and 16fold to increase the number of transaction and size and we denote them T40I10D200K, T40I10D400K, T40I10D800K, T40I10D1.6M respectively. Figure 8 illustrates the sample snapshot of T40I10D100K dataset.

All the experiments were performed in AWS cloud environment. We used EC2, S3, and EMR services to evaluate the proposed TRMR-Apriori algorithm performance with existing selected sequential versions of algorithm as well as existing parallel Apriori Hadoop-MapReduce algorithms. The details of EC2 instance are the type named “t2.medium” that contain 4GB memory, 2unit EC2 compute units (ECU)- one ECU compute unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor, Elastic block store, and 2 cores. We run the sequential algorithm on t2.medium EC2 instance.

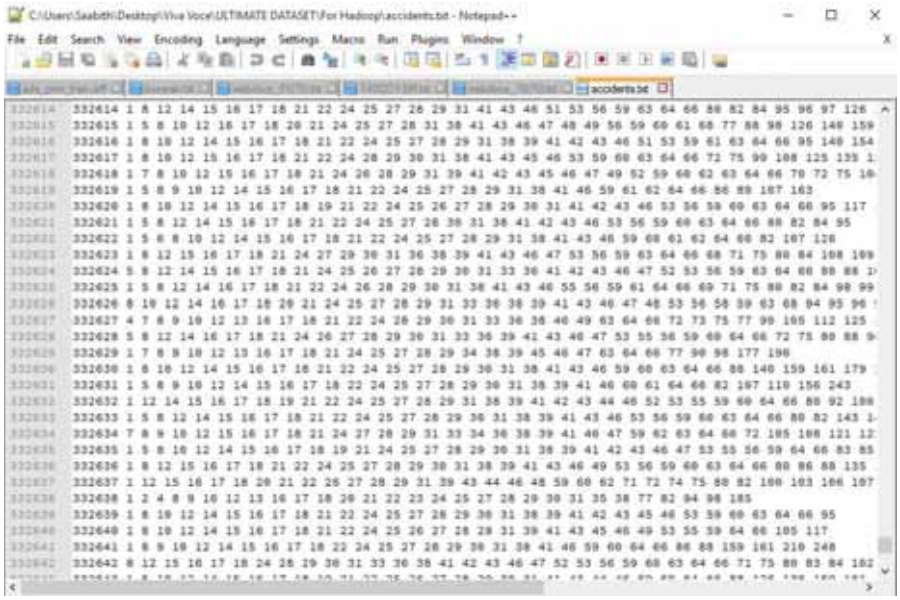


Figure 7. Accidents dataset sample snapshot

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

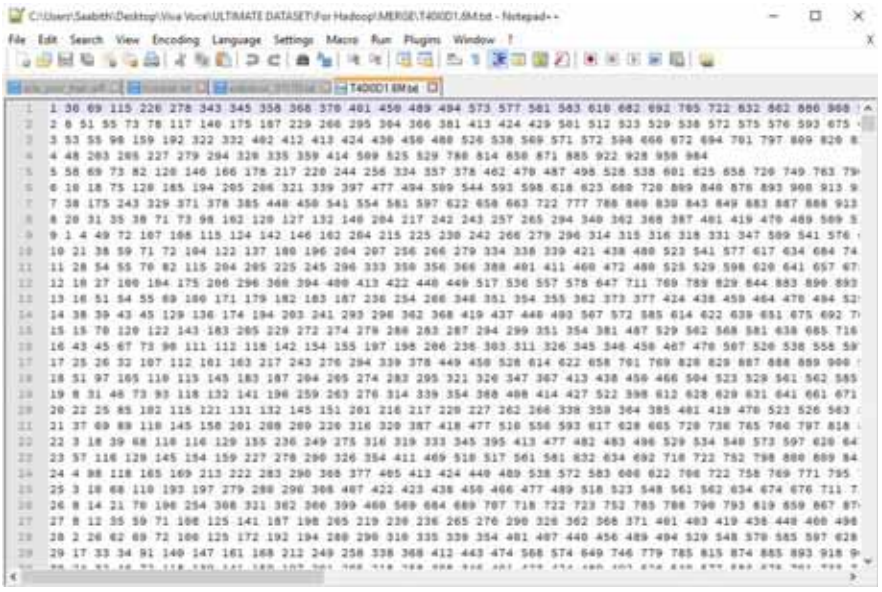


Figure 8. T40I10D100K dataset sample snapshot

TRMR-Apriori and Existing parallel algorithm were run on EMR. We created Hadoop cluster using same type of EC2 instance, S3 storage to store the data, Hadoop version 1.0.3, and HDFS.

3. RESULTS AND DISCUSSION

Experiments that have been carried out to verify the performance of the proposed parallel transaction reduction Apriori algorithm on Hadoop-MapReduce. The results of the experiments are presented in tables as well as appropriate graphs. The description and analysis about the performance improvement in the proposed method are compared against the performance of existing well known sequential algorithm as well as parallel Apriori on Hadoop-MapReduce problem researches.

The results are divided into two categories, one comparison of the proposed method against sequential algorithms in cloud environment. The second category is the comparison of the proposed method against some parallel Apriori algorithm which were implemented on Hadoop-MapReduce in unique environment.

3.1 TRMR-Apriori comparison results with sequential algorithms

Figure 9 illustrates the EC2 logon screen with Amazon Linux AMI. We have uploaded the selected dataset on it and customized SPMF-open source data mining library uploaded as named “p68509.jar”.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

Table 3 summarises the sequential Apriori algorithm results for Retail dataset. The table also shows that we found the frequent itemset varying the value of min_sup. We have tested every experiment three times for more accuracy and calculated the average execution time in millisecond.

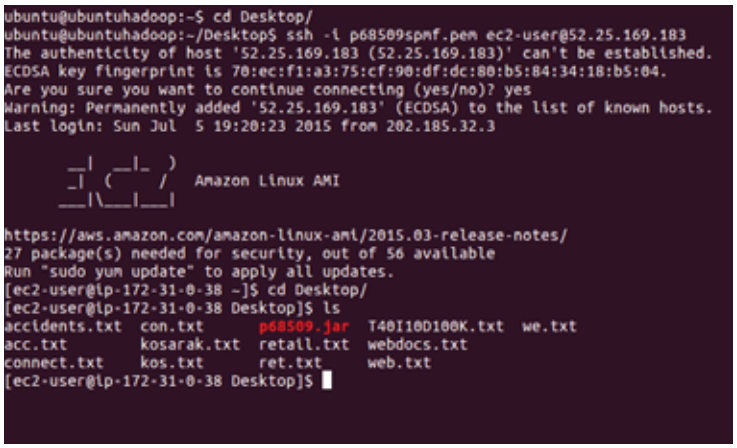


Figure 9.AWS-EC2 logon screen

Table 3. Sequential Apriori algorithm results using Retail dataset

Min_Sup	frequent_ itemset_count	Test 01	Test 02	Test 03	Avg Time(ms)
0.5	1	377	371	382	377
0.4	2	392	386	378	385
0.3	3	410	394	398	401
0.2	3	399	402	407	403
0.1	9	411	417	411	413
0.08	13	417	411	406	411
0.06	15	416	420	418	418
0.04	18	424	422	420	422
0.02	55	579	592	634	602
0.01	159	2555	2536	2543	2545
0.008	243	4886	4898	4910	4898

Similarly, Tables 4, 5, and 6 summarises the sequential AprioriTid, Eclat, and FP-Growth algorithm results for Retail dataset respectively. Those tables also shows that we found the frequent itemset varying the value of min_sup. We have tested every experiment three times for more accuracy and calculated the average execution time in millisecond.

Table 4. Sequential AprioriTid algorithm results using Retail dataset

Min_Sup	frequent_ itemset_count	Test 01	Test 02	Test 03	Avg Time(ms)
0.5	1	849	815	865	843
0.4	2	893	869	858	873
0.3	3	890	888	890	889
0.2	3	892	881	875	883
0.1	9	909	919	926	918

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

0.08	13	918	920	919	919
0.06	15	933	928	916	926
0.04	18	939	935	936	937
0.02	55	1111	1188	1165	1155
0.01	159	1600	1572	1530	1567
0.008	243	1848	1846	1840	1845

Table 5. Sequential Eclat algorithm results using Retail dataset

Min_Sup	frequent_ temset_count	Test 01	Test 02	Test 03	Avg Time(ms)
0.5	1	2462	2405	2334	2400
0.4	2	2325	2373	2318	2339
0.3	3	2369	2332	2363	2355
0.2	3	2375	2321	2489	2395
0.1	9	2408	2345	2378	2377
0.08	13	2389	2380	2432	2400
0.06	15	2355	2343	2364	2354
0.04	18	2392	2462	2387	2414
0.02	55	2383	2368	2323	2358
0.01	159	2417	2517	2388	2441
0.008	243	2386	2428	2385	2400

Table 6. Sequential FP-Growth algorithm results using Retail dataset

Min_Sup	frequent_itemset_count	Test 01	Test 02	Test 03	Avg Time(ms)
0.5	1	500	515	514	510
0.4	2	527	531	521	526
0.3	3	500	538	542	527
0.2	3	524	528	550	534
0.1	9	568	562	560	563
0.08	13	549	546	555	550
0.06	15	551	578	561	563
0.04	18	557	571	584	571
0.02	55	594	600	593	596
0.01	159	728	721	750	733
0.008	243	831	816	814	820

Table 7 summarises the proposed TRMR- Apriori algorithm results for Retail dataset using only one node. The table also shows that we found the frequent itemset varying the value of min_sup. We have tested every experiment three times for more accuracy and calculated the average execution time in millisecond.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

Table 7. TRMR-Apriori algorithm results using Retail dataset

Min_Sup	frequent_ itemset_count	Test 01	Test 02	Test 03	Avg Time(ms)
0.5	1	256	249	259	255
0.4	2	272	273	269	271
0.3	3	287	281	285	284
0.2	3	306	299	299	301
0.1	9	321	329	327	326
0.08	13	354	349	340	348
0.06	15	365	362	360	362
0.04	18	379	384	401	388
0.02	55	401	410	396	402
0.01	159	434	432	438	435
0.008	243	441	444	462	449

Table 8 summarises the above tables (3, 4, 5, 6, and 7) to compare the performance of the proposed TRMR-Apriori algorithm with existing sequential algorithm on unique cloud environment. The retail dataset results show that the Apriori algorithm is better than other three sequential algorithms until min_sup=0.04 because extracting number of frequent itemset is less and beyond that min_sup value, FP-growth is better than Apriori algorithm but our proposed TRMR-Apriori algorithm is outperform than the other four sequential algorithms when varying min_sup values. Figure 10 also illustrates the graphical analysis view for more prominent.

Table 8. Retail dataset result summarization for all algorithms

Min_Sup	Apriori	AprioriTid	Eclat	FP-Growth	TRMR-Apriori
0.5	377	843	2400	510	255
0.4	385	873	2339	526	271
0.3	401	889	2355	527	284
0.2	403	883	2395	534	301
0.1	413	918	2377	563	326
0.08	411	919	2400	550	348
0.06	418	926	2354	563	362
0.04	422	937	2414	571	388
0.02	602	1155	2358	596	402
0.01	2545	1567	2441	733	432
0.008	4898	1845	2400	820	449

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

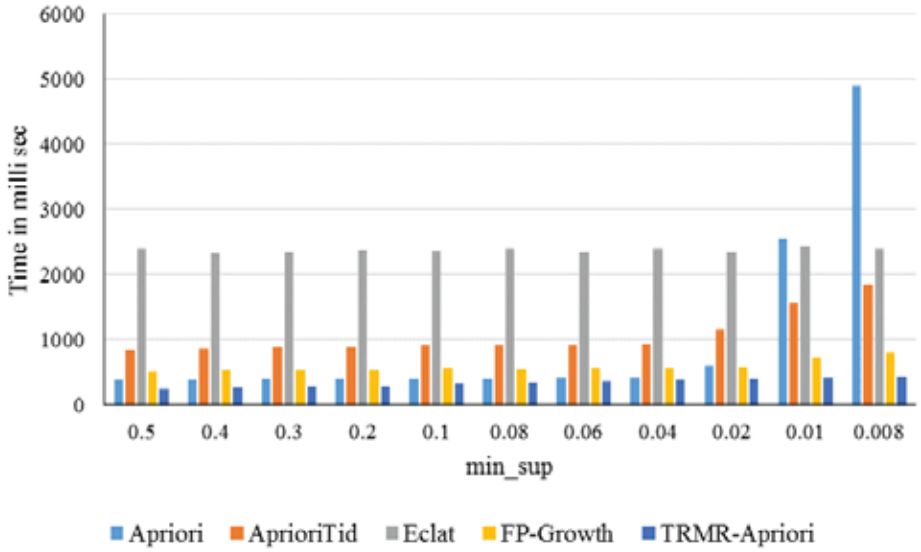


Figure 10. Retail dataset result summarization for all algorithms

3.2 TRMR-Apriori performance results increasing number of nodes

The performance of the proposed TRMR Apriori algorithm over different dataset such that connect dataset that is more dense dataset and number of frequent itemset is very high and webdocs dataset is massive dataset on multi-node Hadoop cloud are depicted in Table 9 and 10, and Figure 11 and 12. We found that when applying the proposed TRMR Apriori algorithm on multiple Hadoop nodes. We found that when applying the proposed TRMR-Apriori algorithm on multiple Hadoop nodes, the performance over the connect dataset is better than webdocs dataset.

The reduction of execution time is more evident when increasing the number of nodes. So proposed algorithm also proves very good scalability when tested on the scale of 1 node to 8 node cluster. So we concluded the performance also depends on the number of nodes in the Hadoop cluster. In our experiments, with the increase of the number of nodes in the cluster, the performance improves.

Table 9. TRMR-Apriori connect dataset results with different nodes.

# of Nodes| Connects	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6	0.55	0.5
1Node	386	414	497	775	1485	3721	8114	16074	32241	64426
2Node	179	221	310	615	753	1936	4315	9256	17589	36258
4Node	96	124	172	521	503	1032	2856	5689	9254	20489
8Node	53	65	96	142	222	396	924	1495	5062	9329

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

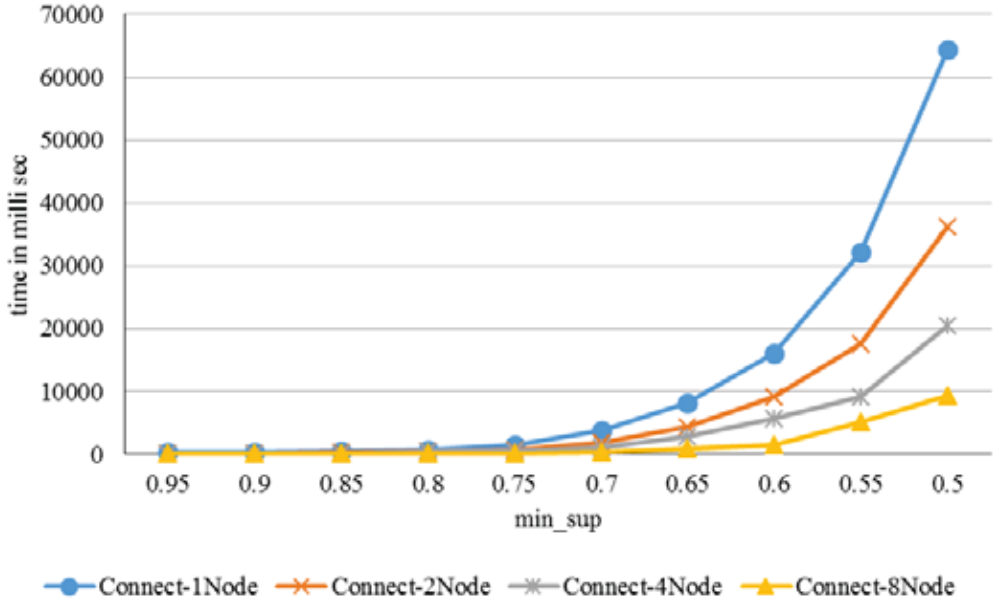
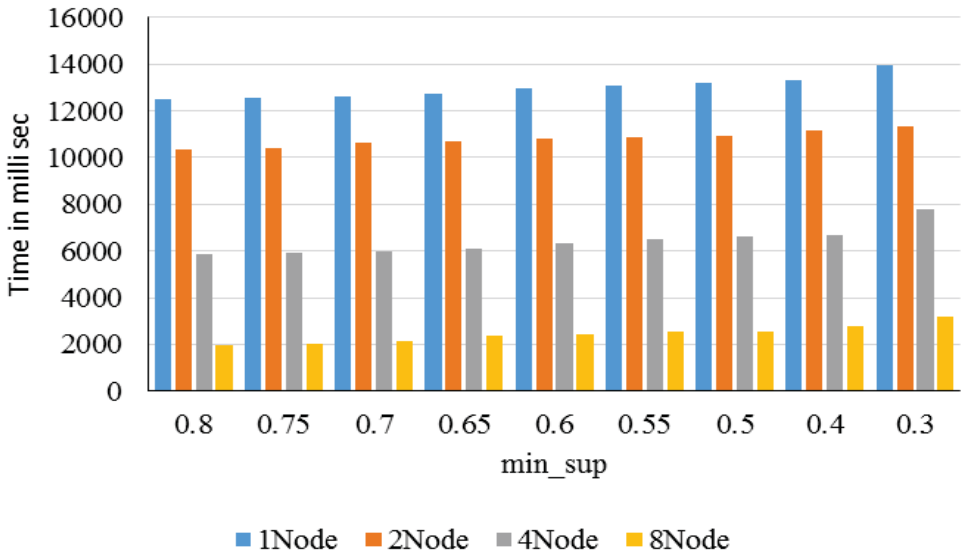


Figure 11. TRMR-Apriori connect dataset results with different nodes.

Table 10. TRMR-Apriori webdocs dataset results with different nodes.

# of Node| Webdocs	0.8	0.75	0.7	0.65	0.6	0.55	0.5	0.4	0.3
1Node	12483	12559	12638	12714	12942	13110	13226	13313	13979
2Node	10342	10423	10651	10703	10825	10853	10936	11172	11329
4Node	5853	5906	5994	6126	6327	6528	6594	6683	7812
8Node	1963	2007	2121	2358	2459	2536	2549	2793	3212



* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

Figure 12. TRMR-Apriori webdocs dataset results with different nodes.

3.3 TRMR-Apriori compared with existing parallel

The comparison of the performance of the proposed TRMR Apriori algorithm with well-established existing parallel Hadoop-MapReduce algorithms such as PApriori [26], MRApriori [25], and Modified Apriori [20] over increasing the T40I10100K dataset transaction from 100K to 1.6M on multi-node Hadoop cloud with fixed min_sup value=0.001 are depicted in Table 11, 12, 13, and 14 and graphical view illustrates in Figure 13, 14, 15 and 16.

We found that the proposed TRMR Apriori algorithm when compared with existing parallel algorithms the proposed algorithm performance was good when increasing the number of transaction size as well as increasing the number of nodes in Hadoop clusters. The proposed TRMR Apriori algorithm works more efficiently comparing with some existing parallel Hadoop-MapReduce algorithms.

A. Comparison with 1 Node

Table 11. TRMR-Apriori comparison with existing parallel-1Node

1-Node	T40I10 100K	T40I10 200K	T40I10 400K	T40I10 800K	T40I10 1.6M
TRMR-Apriori	389	736	1098	1875	3156
Papriori	496	896	1326	2135	3339
MRApriori	529	958	1369	2231	3489
Modified Apriori	689	1126	1589	2314	3579

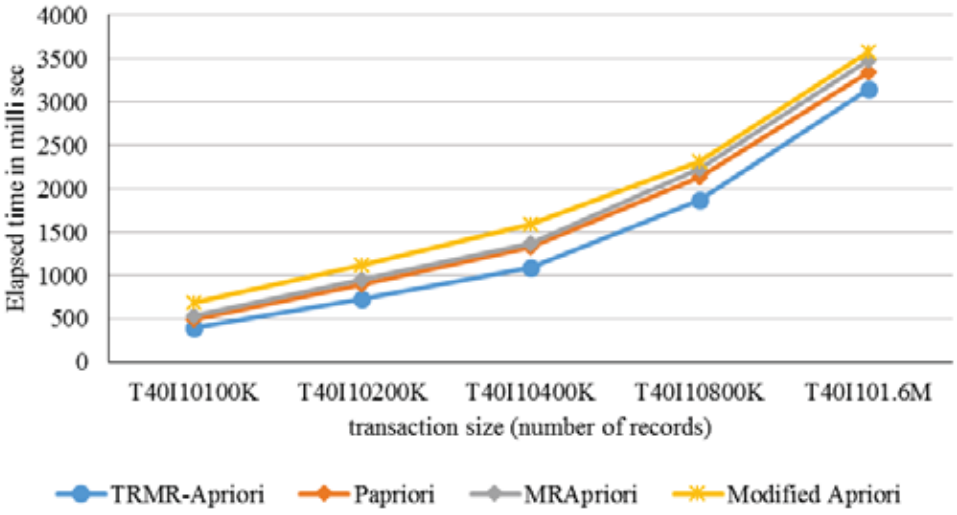


Figure 13. TRMR-Apriori comparison with existing parallel-1 Node

B. Comparison with 2 Node

Table 12. TRMR-Apriori comparison with existing parallel-2 Node

2-Node	T40I10 100K	T40I10 200K	T40I10 400K	T40I10 800K	T40I10 1.6M
TRMR-Apriori	350	610	833	1311	2048

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

Papriori	460	762	1175	1752	2541
MRApriori	488	811	1190	1869	2655
Modified Apriori	602	960	1321	1903	2890

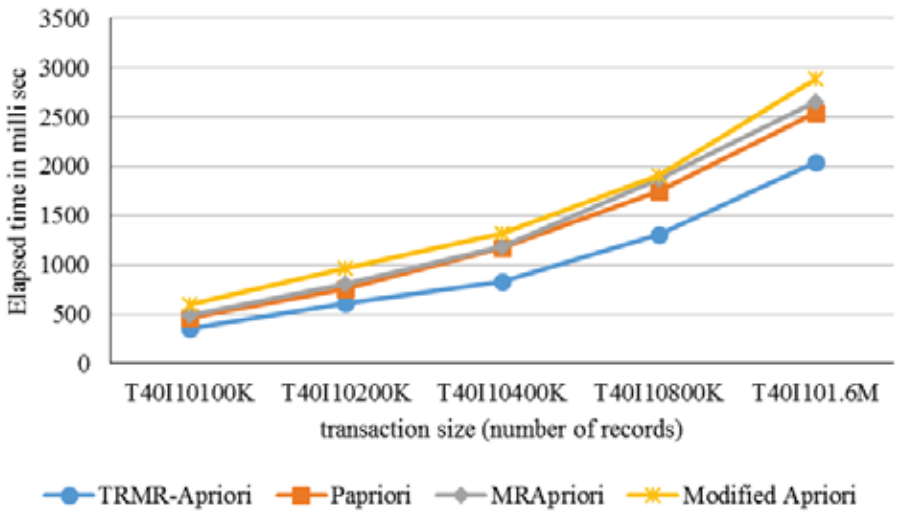


Figure 14. TRMR-Apriori comparison with existing parallel-2 Node

C. Comparison with 4 Node

Table 13. TRMR-Apriori comparison with existing parallel-4 Node

4-Node	T40I10 100K	T40I10 200K	T40I10 400K	T40I10 800K	T40I10 1.6M
TRMR-Apriori	321	529	709	1120	1624
Papriori	413	614	965	1421	2122
MRApriori	430	744	1021	1556	2233
Modified Apriori	512	831	1123	1753	2410

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

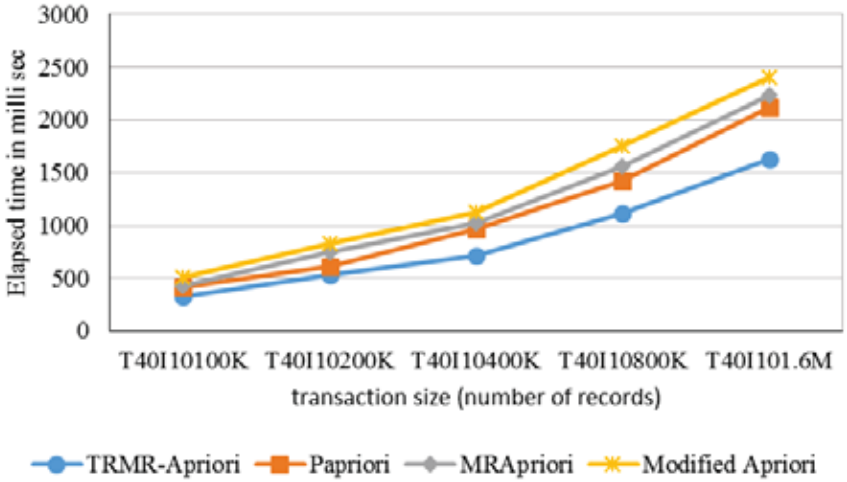


Figure 15. TRMR-Apriori comparison with existing parallel-4 Node

D. Comparison with 8 Node

Table 14. TRMR-Apriori comparison with existing parallel-8 Node

8-Node	T40I10 100K	T40I10 200K	T40I10 400K	T40I10 800K	T40I10 1.6M
TRMR-Apriori	290	411	512	689	863
PApriori	359	526	668	863	1147
MRApriori	398	633	821	1122	1396
Modified Apriori	452	711	937	1221	1556

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

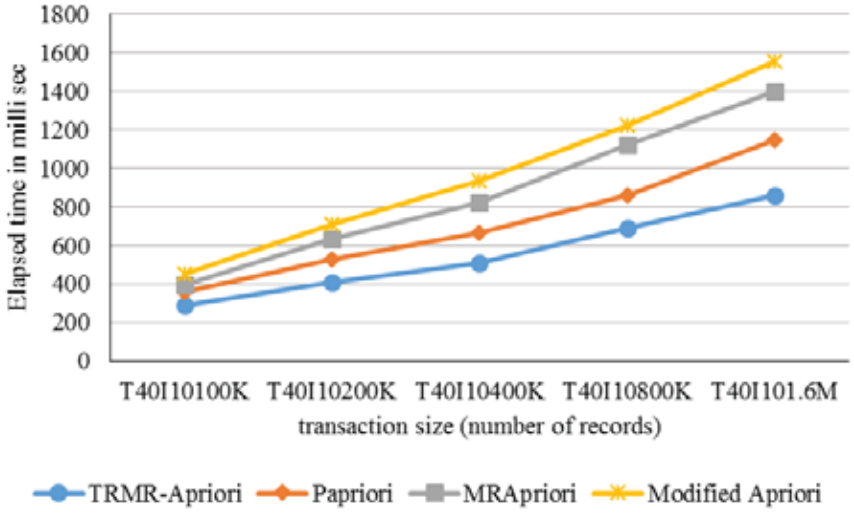


Figure 16. TRMR-Apriori comparison with existing parallel-8 Node

4. CONCLUSION

This study attempted to examine the performance of extracting frequent itemset from massive dataset. The main aim of this study was discover whole frequent itemset from huge dataset on cloud environment using Hadoop-MapReduce parallel techniques. An algorithm proposed to extract entire frequent itemset using transaction reduction technique that is reduce unessential transaction values and transactions from the dataset in parallel manner.

The transaction reduction is one of the best way to find the frequent itemset from massive dataset with efficiently. Our experimental results also exhibit the performance of our proposed parallel transaction reduction algorithm compared with existing sequential as well as parallel algorithms on cloud environment.

The proposed Transaction Reduction MapReduce Apriori (TRMR-Apriori) algorithm was tested on Amazon Web Service (AWS) cloud environment.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

Moreover, the experiments results show that the proposed TRMR-Apriori algorithm was implemented to extract entire frequent itemsets from big dataset with effectively and efficiently by comparing existing sequential Apriori algorithms as well as parallel Hadoop-MapReduce Apriori algorithms on unique environment.

This study mainly proposed an Apriori parallel algorithm using transaction reduction method. The proposed algorithm was experimented using Hadoop-MapReduce techniques on cloud environment. The implications of finding in the context of the research objectives addressed by this study are discussed in the following.

To achieve the first objective, we tested selected existing well established sequential frequent itemset mining algorithms in association rule mining such as Apriori, AprioriTid, Eclat, and FP-Growth on cloud environment. In this test we analyzed the performance varying the minimum support (min_sup) value using selected all benchmark dataset. We was analyzed with existing Apriori parallel algorithms which based on the Hadoop-MapReduce platform with different conditions such as varying min_sup value, increase number of nodes, and replicated the transaction size to evaluate the existing algorithms performance.

To achieve the second objective, the proposed TRMR-Apriori algorithm was compared their performance with experimental results which were achieved by objective one such as existing well established sequential frequent itemset mining algorithms in association rule mining (Apriori, AprioriTid, Eclat, and FP-Growth). In this test we analyzed and compared the performance varying the minimum support (min_sup) value using selected all benchmark dataset.

The proposed TRMR-Apriori algorithm was tested in different conditions such as varying min_sup value, increase number of nodes, and replicated the transaction size to evaluate the proposed algorithm performance. The proposed TRMR-Apriori algorithm was analyzed with existing Apriori parallel algorithms which based on the Hadoop-MapReduce platform.

We implemented our proposed transaction reduction MapReduce algorithm (TRMR-Apriori) on Amazon web service (AWS) to evaluate the performance. We used EC2, S3, EMR services from AWS and uploaded selected and pre- processed bench mark dataset to EC2 and S3. We also used SPMF open source data mining library for the experimental results of existing versions of sequential Apriori algorithm such as Apriori, AprioriTid, FP-Growth, and Eclat on cloud environment.

REFERENCES

1. Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. 1996. The Kdd Process for Extracting Useful Knowledge from Volumes of Data. *Communications of the ACM* 39(11): 27-34.
2. Agrawal, R. & Srikant, R. 1994. Fast Algorithms for Mining Association Rules. *Proceeding 20th international conference on very large data bases, VLDB*, 487-499.
3. Han, J. & Kamber, M. 2006. *Data Mining, Southeast Asia Edition: Concepts and Techniques*. Morgan kaufmann.
4. Patel, A. B., Birla, M. & Nair, U. 2012. Addressing Big Data Problem Using Hadoop and Map Reduce. *Engineering (NUiCONE), 2012 Nirma University International Conference on*, 1-5.
5. Park, J. S., Chen, M.-S. & Yu, P. S. 1995. Efficient Parallel Data Mining for Association Rules. *Proceedings of the fourth international conference on Information and knowledge management*, 31-36.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

6. Singh, J., Ram, H. & Sodhi, D. J. 2013. Improving Efficiency of Apriori Algorithm Using Transaction Reduction. *International Journal of Scientific and Research Publications* 3(1): 1-4.
7. Thevar, R. E. & Krishnamoorthy, R. 2008. A New Approach of Modified Transaction Reduction Algorithm for Mining Frequent Itemset. *Computer and Information Technology, 2008. ICCIT 2008. 11th International Conference on*, 1-6.
8. Yu, H., Wen, J., Wang, H. & Jun, L. 2011. An Improved Apriori Algorithm Based on the Boolean Matrix and Hadoop. *Procedia Engineering* 15(1827-1831).
9. Zaki, M. J. 1999. Parallel and Distributed Association Mining: A Survey. *IEEE concurrency* 7(4): 14-25.
10. Toivonen, H. 1996. Sampling Large Databases for Association Rules. *VLDB*, 134-145.
11. Brin, S., Motwani, R., Ullman, J. D. & Tsur, S. 1997. Dynamic Itemset Counting and Implication Rules for Market Basket Data. *ACM SIGMOD Record*, 255-264.
12. Zaki, M. J. & Gouda, K. 2003. Fast Vertical Mining Using Diffsets. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 326-335.
13. Zheng, Z., Kohavi, R. & Mason, L. 2001. Real World Performance of Association Rule Algorithms. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 401-406.
14. Agrawal, R., Imieliński, T. & Swami, A. 1993. Mining Association Rules between Sets of Items in Large Databases. *ACM SIGMOD Record*, 207-216.
15. Chen, Ming-Syan, Jiawei Han, and Philip S. Yu. "Data mining: an overview from a database perspective." *Knowledge and data Engineering, IEEE Transactions on 8.6* (1996): 866-883.
16. Agrawal, R. & Shafer, J. C. 1996. Parallel Mining of Association Rules. *IEEE Transactions on Knowledge and Data Engineering* 8(6): 962-969.
17. Kovacs, F. & Illes, J. 2013. Frequent Itemset Mining on Hadoop. *Computational Cybernetics (ICCC), 2013 IEEE 9th International Conference on*, 241-245.
18. Li, L. & Zhang, M. 2011. The Strategy of Mining Association Rule Based on Cloud Computing. *Business Computing and Global Informatization (BCGIN), 2011 International Conference on*, 475-478.
19. Modgi, M. P. & Vaghela, D. 2014. Mining Distributed Frequent Itemset with Hadoop.
20. Li, J., Roy, P., Khan, S. U., Wang, L. & Bai, Y. 2012. Data Mining Using Clouds: An Experimental Implementation of Apriori over Mapreduce. *12th International Conference on Scalable Computing and Communications (ScalCom)*
21. Oruganti, S., Ding, Q. & Tabrizi, N. 2013. Exploring Hadoop as a Platform for Distributed Association Rule Mining. *FUTURE COMPUTING 2013, The Fifth International Conference on Future Computational Technologies and Applications*, 62-67.
22. enqi, W., Qiang, L., 2013 Algorithm for Map/Reduce-based association rules data mining, *International Conference on Information Science and Computer Applications (ISCA)* 334-339.
23. Dhamdhere Jyoti, L. & Deshpande Kiran, B. 2014. A Novel Methodology of Frequent Itemset Mining on Hadoop. *International Journal of Emerging Technology and Advanced Engineering* 4(7): 851-859.
24. Woo, J. & Xu, Y. 2011. Market Basket Analysis Algorithm with Map/Reduce of Cloud Computing. *The 2011 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2011), Las Vegas*.
25. Yahya, O., Hegazy, O. & Ezat, E. 2012. An Efficient Implementation of Apriori Algorithm Based on Hadoop-Mapreduce Model. *International Journal of Reviews in Computing* 12.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.

26. Li, N., Zeng, L., He, Q. & Shi, Z. 2012. Parallel Implementation of Apriori Algorithm Based on Mapreduce. *Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on*, 236-241.
27. Ezhilvathani, A. & Raja, K. 2013. Implementation of Parallel Apriori Algorithm on Hadoop Cluster.
28. Qureshi, Z. & Bansal, S. 2014. Improving Apriori Algorithm to Get Better Performance with Cloud Computing.
29. Itkar, S. A. & Kulkarni, U. V. 2013. Distributed Algorithm for Frequent Pattern Mining Using Hadoopmap Reduce Framework. *International Conference on Advances in Civil Engineering, AETACE*. 15-24.
30. Chen, Z., Cai, S., Song, Q. & Zhu, C. 2011. An Improved Apriori Algorithm Based on Pruning Optimization and Transaction Reduction. *Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011 2nd International Conference on*, 1908-1911.
31. Moens, S., Aksehirli, E. & Goethals, B. 2013. Frequent Itemset Mining for Big Data. *Big Data, 2013 IEEE International Conference on*, 111-118.

* Tel.: +xx xx 265xxxxx; fax: +xx aa 462xxxxx.
E-mail address: xyz@abc.com.