Information Technology Journal 6 (2): 174-181, 2007
ISSN 1812-5638
© 2007 Asian Network for Scientific Information

Association Rule Mining in Centralized Databases

Saleha Jamshaid, Zakia Jalil, Malik Sikander Hayat Khiyal and Muhammad Imran Saeed
Faculty of Applied Sciences, International Islamic University, Islamabad, Pakistan

Abstract: Mining of Association Rules between the items of a huge centralized Database is very interesting and important research area. Its importance becomes more significant in case of sales transaction. There are a number of algorithms working on this specialized research area. The algorithm presented in this study, (Centralized Mining of Association-Rules), CMA is more efficient than the previous existing algorithms, as it not only reduces the overhead of frequent disk I/Os and the CPU cost, but it also reduces the database scan to the half. The CMA algorithm, presented in this study, basically takes the best features of two state-of-the-art algorithms in the area, i.e., the technique of PARTITION algorithm of centralized database area is taken for partitioning the huge Database and then, the DMA algorithm of distributed database environment is applied on each partition. The large itemsets to be found at the end of operation at each partition are to be merged together and then the actual set of large itemsets is finally created.

Key words: Data warehousing, data mining, association rules

INTRODUCTION

With the growing competition in retail industry, it has been observed that along with other factors, proper placement of different items together at shelves is also a major factor to increase the sales of a store. Because some items have special sort of relationships among each other, which can never be targeted out simply with the help of Entity-Relationship Diagram (ERD) or some mathematical formulae, as these relationships are not the casual relationships. To represent these relationships between data items, association rules are used. For example, if a person buys a computer, he is likely to be buying some software CDS as well. If we offer our customers free Operating System installation and place some antivirus CDS on the prominent and nearest shelves to the counter, the customer couldn't stop himself from noticing the importance of the antivirus software for his system's protection as well. And if we offer this antivirus to him with some special concession package, he will definitely be induced to buy those as well. It means that not only we are running our computer business with distinction from our competitors, but also establishing a side business of CD shop. This is what we call Association between two items; the Association between computer and CDS and the association between computers and the virus-guard software. And all such business decisions are supported well with the help of finding out these Associations. And to find these Associations, we use Association Rules.

Association rule Mining is an important research area in Databases. It usually involves huge amount of data glut, out of which, useful information is to be extracted in the form of association rules. The task of Mining association rules is to analyze the entire Database and find out the association rules for different sets of items. It requires multiple Database scans for the rule generation.

The algorithm presented in this study CMA, generates large itemsets by only taking a single scan of the database after the creation of the candidate sets. It divides the centralized database into a number of partitions, which are not taken sequentially from the database, but the partitions are created by taking random tuples from the database and then collecting them together in different partitions. Each partition is loaded into the memory one by one and then the large itemsets are created from each partition. At the end, the large itemsets of all the partitions are collected and then examined that whether these are actually large itemsets in the entire database or not. Within each partition, the large itemsets are created by taking a single database scan after creation of the candidate sets, thus minimizing the disk I/Os to the half, as that of the PARTITION Algorithm presented by Savarse *et al.* (1995). The database used for the experiments of CMA Algorithm is synthetic data, created exclusively for this study.

BASIC CONCEPT

In the present study we give the basic description of this problem area, which is mostly based on the description given by Agarawal *et al.* (1993). The problem

Corresponding Author: Malik Sikander Hayat Khiyal, Faculty of Applied Sciences, International Islamic University, Islamabad, Pakistan

of association rule mining is tackled in many research papers. Many algorithms are devised to solve this problem. From the literature, it has been observed that problem can be divided into two sub problems:

- Find all frequent/large itemsets
- Generate strong association rules from these frequent/large itemsets (Han, 2001).

Here, by large itemsets, we mean the items that frequently occur together in different transactions. We define a threshold number of occurrences for a given itemset, if the number of occurrences is greater than that threshold value, then the item is a larger one, otherwise a small one. It is done by scanning the entire Database and calculating the number of occurrences of each itemset. This individual number of occurrences of an itemset is called the support of that itemset. And the threshold value of support, specified by the user is called the minimum_support. Like support another feature used is called confidence, which actually determines the strength of the rule. To understand it lets take an example. In order to measure the confidence of two items, say burger and the drink, the number of times the burger appears in the transaction, so is the drink. This is called the confidence between two items. It can have any value, 60 and 70% etc. association rules are the implication of the form $X \Rightarrow Y$. Here X is called the *antecedent* and Y is called *consequent* of the rule. The rule $X \Rightarrow Y$ has confidence c in the transaction set D if c is the percentage of the transaction in D containing X that also contain Y (Han, 2001).

The literature surveyed and the study with different algorithms, has shown that in order to have an efficient algorithm for the association rule mining, it is required to emphasize a lot over the generation of frequent/large itemsets, the first sub problem. The quicker the algorithm at step1, the efficient will be the association rule Mining process. So other present study is confined with the step1 only.

The functionality, followed so far for generation of large itemsets is as follows:

Let $I = \{i_1, i_2, \ldots, i_m\}$ be the set of items, DB be the transactional database, T the set of items such that $T \subset I$. So the rule $X \Rightarrow Y$ means $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \varphi$. As mentioned before, it is the step1 which determines the efficiency of the algorithm as the database is scanned for many times during this step. Firstly, the items are taken from the database, then their support is counted from the database, then 2-itemsets are created and up to k-itemset creation (k-itemset is an itemset of size k), the database is scanned again and again, i.e., the same is done for the every iteration. As it is obvious that the mining of association rules is not done on a small database, rather it could be on a huge database, or a data warehouse, or some distributed database with multiple nodes. So this multiple scan of database physically means a lot. Excessive study has been done on this area.

An earlier study in this area is done by Agrawal *et al*. (1993), in which AIS algorithm is presented. This algorithm scans the database to create the candidate sets of frequently occurring itemsets. The second database scan counts the support of these candidate sets. The candidates generated from a transaction are added to the set of candidate itemsets for the pass, or the counts of the corresponding entries are increased if they were created by an earlier transaction. The problem with this algorithm is that, it is confined to only the single consequent rule generation and creates larger candidate set.

An algorithm SETM is presented by Houtsma and Swami (1995). It uses SQL for computing large itemsets. Candidate itemsets and the corresponding TIDs are saved together and are sorted and aggregated at the end of pass to determine the support count. The small candidate itemsets are pruned out. Then the set of the candidate set is again sorted on the basis of TIDs for next pass. SETM also uses the single consequent rule generation technique. It also generates large candidate itemsets. For each candidate itemset, the candidate generated has many entries as the number of transactions in which the candidates are present. Also to count the support for candidate itemsets, the candidate set is in wrong order and needs to be sorted on TIDs. After counting and pruning out small candidate itemstes, the resulting set of large items needs another sort on TID, to be used in the subsequent passes.

The pioneer work is presented by Agrawal and Srikant (1993) in which notorious Apriori algorithm is presented. All other subsequent algorithms are the adaptation of Apriori to some extents. This algorithm counts item occurrences from the database to determine large 1-itemset in first pass. In the next pass, the algorithm first generates candidate itemsets, using apriori-gen () and then checks the support count. It stores the candidate itemsets in a special structure, the Hash Tree. The candidate itemsets generated by this algorithm is smaller than that created by SETM and AIS. Also, unlike the AIS and SETM, it generates multiple consequent association rules. Another version of Apriori, AprioriTid, is also presented in this study, which does not use the database for support counting after the first scan, instead it uses the pair of itemset and it's TIDs for this purpose. It works efficiently in later passes. AprioriHybrid is another variant of the same algorithms which uses Apriori in earlier iterations and AprioriTid in later ones, to enjoy more benefits from both the algorithms. In Apriori, the problem is that the database is scanned entirely for each pass, so it means for 10 iterations, 10 scans of the entire database. For AprioriTid algorithm, its performance is not better than Apriori's in initial stages, as there are too many candidate k-itemsets to be tracked during the early stages of the process. The challenge in AprioriHybrid is to determine the switch over point between the two algorithms.

Partition algorithm, presented by Savasere *et al.* (1995), takes two database scans. Firstly, for generation of potentially large itemsets and store it as a set, which is the superset of all the large itemsets. Secondly, to measure the support of these itemsets and storing them in their respective counters created.

The working of this algorithm is divided in two phases. In phase I, the database is logically divided into non-overlapping partitions, which are considered one by one and large itemsets for each partition are generated and at the end all itemsets are merged to form the set of all potentially large itemsets. The phase II generates the actual support of these itemsets, to identify large itemsets. The database is read once in phase I and once in phase II. The small itemsets are pruned out. For each itemset, is associated its sorted TID list. To count the support of all itemsets in a partition, this algorithm divides the cardinality of TID list by the total number of transactions in that partition. Initially, the tidlists for 1-itemsets are generated directly by reading the partition. The TID list for a candidate k-itemset, is generated by joining the TID lists of the two (k-1)-itemsets that were used to generate the candidate k-itemset. The main problem with PARTITION algorithm is to find out the accurate number of partitions for the given memory.

The mining of association rules in distributed environment is discussed by Cheung *et al.* (1996). An algorithm Distributed Mining of Association rules, (DMA) is presented, which is an adaptation of Apriori in parallel systems. In DMA, to generate candidate itemsets, apriori-gen() is not applied directly, rather it is applied in such a way that it minimizes the candidate set to a greater extend than in the case of direct application of apriori-gen(). It uses polling site technique to determine heavy itemsets, after local pruning. In the whole procedure, the database partition at each site is scanned only once for calculating the support count and then those counts are stored in Hash Tree. This Hash Tree contains the support counts of both the heavy itemsets at that site and heavy itemsets at some other site. The later are stored in order to entertain the polling requests made by the remote sites so one database scan is done to compute the support counts of itemsets and are stored in the Hash Tree and retrieved from that Hash Tree when required. This optimizes the database scanning required for count exchange.

The literature proves that the generation of large itemsets is the main problem in generating the association rules in large databases. It involves many problems like the candidate sets created for generation of large itemsets are very large; the database is supposed to be scanned again and again in order to create candidate sets and to generate their support counts. Also the pruning of the itemsets from the candidate sets, that are not large, is also a problem.

MATERIALS AND METHODS

The algorithm DMA requires only one database scan at each site in order to calculate the heavy itemsets. So far, the best work done on centralized databases, the Partition algorithm, achieves the same with two database scans. So to apply DMA algorithm on each partition (obviously excluding the concept of polling site, which is specific for distributed databases), the large itemsets could be found with a single database scan.

So the major task is to divide the database into a number of partitions. These partitions are logical and non-overlapping, i.e., the transactions found in one partition should not also be present in other partition. Similarly, the partitions are not to be taken sequentially from the centralized database, these are to be created by taking the transactions randomly from the database to avoid the outliers (i.e., itemsets caused due to data skewness). For example, the October earthquake of Pakistan caused a drastic demand for tents. Per month demands of the tents for the affected areas were about 200,000 and the situation remained so throughout the winters. By using the sequential approach for partitioning the centralized database, if we take the data of a quarter of the year as one partition, then according to the 4th partition of 2005's data, tent will emerge as a large itemset, which in fact is not. As a contrary, the annual sale of tents in Pakistan, before the October earthquake, is 25 tents per year. This is the outlier, caused by the data skewness of the abnormal circumstances. And if we take the transactions randomly from the database to build different partitions, the October-Dec'05's tragic condition's transactions will be distributed throughout the database, so no problem of outliers will emerge. This task is achieved with the specially devised function, the Random (), to randomly take transactions from the database and group them together in partitions. Each partition is supposed to contain D_i number of transactions, after which the counter is initialized and then selected transactions are to be stored in second partition and so on. Then each partition is brought into memory one by one. For each partition, candidate 1-itemset is created, from which, large 1-itemset is generated. Then candidate 2-itemset is generated using apriori-gen () and from that candidate set, large 2-itemset is created and so on up to k-itemsets. Same is done with each partition.

Dataset: The functionality of the CMA algorithm has been tested on extensive experiments. The dataset used in these experiments is synthetic data, created exclusively for these experiments. Each tupple of the database is of the form <TID> <A, B, C, D,...> where TID is the transaction identifier, which is the primary key to uniquely identify each tupple and the literals represent the retail store's items bought together in each transaction. For example, take a transaction <T1004> <A, C, D> for example, where A represents butter, C represents bread and D represents milk, so we can say that transaction number 1004 represents the purchase of breakfast items by a customer during his visit to the store. The size of the database used is 100, 000 transactions.

Random function: To calculate large itemsets, CMA Algorithm partitions the database randomly, i.e., for each partition, transactions are picked from the database randomly. It is also possible to avoid this overhead of randomizing the partitions by picking up sequential partitions, i.e., if the size of the database is 20 MB, then take first five MB data in partition 1, next five MB data as partition 2, next five MB data as partition 3 and the last five MBs as partition 4. But the problem with sequential partitioning is that there might be some item that are excessively bought during some specific time period, under some specific abnormal circumstances, which actually are not the large itemsets, but within a partition containing those transactions of that particular time period, it appear as large itemsets. Such situations cause into wasted efforts for considering something as large itemsets, which factually are not. Let's take October 8th earthquake example once again. During those 3-4 months, sweaters, jackets and blankets were extensively purchased by the people of the saved areas, in order to gift those to the people of devastated areas, to help them fight the extreme climatic conditions. Under normal circumstances, these items are not heavily bought items, because people usually buy two or three sweaters during the whole winters. And blankets are the least purchased items, as those are purchased after some five years. So, by taking sequential partitions, the partition consisting of transactions of the last quarter of the year 2005 will result blankets, sweaters and jackets as heavily bought items, which actually are not and also no other partition will approve those as the heavy itemsets. So it means we are wasting our resources on wrong candidates, in other words, false positives. To overcome these problems caused by the sequential partitioning, random partitions are used. We devised a random function, which randomly picks transactions and store them in partitions and at the end, equal-sized, n number of partitions are created. These partitions are random, equal-sized and non-overlapping. By non-overlapping, we mean that a transaction once picked for a particular partition, have 0% probability to be picked up again for any other partition. The functionality of random function is tested over a number of experiments, with varying number of transactions different datasets and it proved its performance in every case.

Decreased disk I/Os: Earlier, algorithms were taking multiple database scans in order to calculate the support counts of the candidate itemsets, to approve them as large itemsets or not. Which in large databases, means a lot of extra work to be done. In CMA algorithm, we are using a formula (as is used by Cheung *et al.* (1996) in DMA algorithm) to check the support of an itemset. To calculate locally large itemsets, $s * D_i$ is used, in which s is the support of the itemset, set by the user (50% in these experiments) and D_i is the size of each partition. For global support count, $s * D$ is used, where s is same support, while D is the size of entire central database.

The size of the D_i is selected by keeping in view the hardware requirements of the system in use, so that each partition fits well in the memory, keeping enough space for the L1_List, L_List. And other Operating System processes to run, simultaneously. After the partitioning, each partition is to be loaded into the memory and then the DMA algorithm (with due amendments) is to be applied on it. As the database to be used in this study is a centralized database, so there is no need of polling site technique or polling requests etc. Similarly, only one set of support count is to be generated, instead of the two sets generated in DMA. The database to be used for the experiments is the synthetic data, created exclusively for this study. The functionality of our devised CMA algorithm is presented in the Fig. 1. When a partition is loaded into the memory, the partition is scanned and supports for the candidate itemsets are counted and stored in the L1_List. To find out the large itemsets of the partition, each itemset is checked for the following condition:

```
If $X.sup_i \geq s * D_i$ then
Add(X, $X.sup_i$, partition#) into L_List
```

Where D_i is the number of transactions in ith partition. The itemsets not meeting this condition are pruned away to avoid extra processing. In the next iteration, the function apriori_gen () is applied on the large itemsets created in the previous iteration and so on, till either no new itemset is added to the L_List, or no itemsets are there to be the input for the apriori_gen () in the next iteration. At the end of calculations in all the partitions, the itemsets in the G_List are now checked again, this time with the D, the number of transactions in the entire database. The condition is:

```
For all $X \in$ L_List
        {$X.sup = \sum^{n}_{i=1} X.sup_i$;
        If $X.sup \geq s * D$ then
        {Insert X into L_List AND
        Prune away rest of the itemsets;
        }
    }
```

So only the actually large itemsets are left in the L_List.
CMA algorithm is presented in Fig. 1.

```
Partition = Random (Centralized DB); //n number of
partitions are generated
P = partition-database (D)
N = number of partitions
For i = 1 to n begin
  Read-in-partition ($pi \in P$) // load partition
    If k = 1 then
        Scan $DB_i$ to compute L1_List;
    Else
        {
$C_k = U^{n}_{i=1} C^{i}_{k} = U^{n}_{i=1}$ Apriori_gen
        (L1_List$^{i}_{k-1}$)
    Scan $DB_i$ for support and put in $C_k$;
        }
//calculation of locally large itemset
for all $X \in C_k$ do
        {
        If $X.sup_i \geq s * D_i$ then
        Add(X, $X.sup_i$, partition#) into
        L_List;
        }
//calculation of actually large itemset
For all $X \in$ L_List
        {
        $X.sup = \sum^{n}_{i=1} X.sup_i$;
        If $X.sup \geq s * D$ then
                {
                Insert X into L_List AND
                Prune away rest of the itemsets;
                }
        }
```

Fig. 1 CMA-Algorithm

Example: To illustrate the performance of CMA algorithm, we give in the following section an example, in which at first Partition algorithm is applied on the central database, which divides the database into three distinct, non-overlapping and random partitions, and then calculates the locally large k-itemsets, and finally computes globally large itemsets of size 1,2,…,k. In the second part of the example, CMA Algorithm is applied on the same database, which also divides the central database into three distinct, non-overlapping and random partitions and then calculates locally large itemsets by bringing each partition one by one into memory. Globally large itemsets are calculated at the end. Instead of counting the itemsets in the transactions of the actual database, CMA Algorithm uses the formula of $s*D_i$ (as is used by Cheung *et al.* (1996) for counting support within each partition, where s is the support of the itemset, set by the user, like 30, 60 and 75% etc. (50% in this example) and D_i is the size of each partition database (3 in this example). For global support count, it uses the formula, s * D, where D is the size of the entire database (12 in this example). The dataset is presented below:

TID	Itemset
T1	A,B,C
T2	A,B,D
T3	A,D,E
T4	A,B,D
T5	B,C,D
T6	A,B.E
T7	A,D,E
T8	B,C,E
T9	A,B,C
T10	A,C,D
T11	A,D,E
T12	C,D,E

First, Partition algorithm is applied to the database, which works on random partitions, so the original database, given in dataset presentation, is divided into three distinct, non-overlapping and random partitions, which are presented in Table 1.

It then generates the candidate 1-itemset and then counts its support. The support taken in this example is 50%. So the large itemsets are created and stored along with their TIDs, as shown in Table 2.

Then candidate 2-itemset is generated by multiplying large 1-itemset with itself, like $l_1 * l_1$.The candidate 2-itemsets of partition 1 are {AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}. Then the TIDs of the transactions containing these 2-itemsets are stored against each 2-itemsets. The number of TIDs against each itemset shows its occurrence in the partition. For example, in partition 1, AB is present in T2, T6 and T9, so it means it has occurrence of 3 out of 4 transactions, i.e., 3/4 = 0.75, which means 75% transactions in partition 1 contains AB, which

Table 1: Three Partitions created after dividing the database

Partition 1		Partition 2		Partition 3	
TID	Itemset	TID	Itemset	TID	Itemset
T2	A,B,D	T1	A,B,C	T3	A,D,E
T6	A,B,E	T4	A,B,D	T5	B,C,D
T9	A,B,C	T7	A,D,E	T8	B,C,E
T12	C,D,E	T11	A,D,E	T10	A,C,D

Table 2: Candidate 1-itemsets of partition 1

TID	1-itemset
T2	A
T2	B
T2	D
T6	A
T6	B
T6	E
T9	A
T9	B
T9	C
T12	C
T12	D
T12	E

is greater than 50%, so AB is a locally large 2-itemset in partition 1. The supports calculated for each candidate 2-itemset is given below:

AB = {T2,T6,T9}, so support is AB = 3/4 = 0.75
AC = {T9}, so support is AC =1/4 = 0.25
AD = {T2}, so support is AD =1/4 = 0.25
AE = {T2}, so support is AE =1/4 = 0.25
BC = {T9}, so support is BC =1/4 = 0.25
BD = {T2}, so support is BD = 1/4 = 0.25
BE = {T6}, so support is BE = 1/4 = 0.25
CD = {T12}, so support is CD = 1/4 = 0.25
CE = {0}, so support is CE = 0
DE = {T12}, so support is DE = 1/4 = 0.25

So the large 2-itemsets in Partition 2 are {AB, AD and AE, DE}. From this we will have candidate 3-itemset of {ABD, ADE, ABE, BDE}. But as only 1 large 2-itemset is present in partition 1, so no candidate 3-itemset is possible. Same is repeated with partition 2 and partition 3, which are illustrated in Table 3 and 4.

Partition = 2

Candidate 2-itemset= $l_1 * l_1$ ={AB,AC, AD, AE, BC, BD, BE,CD, CE, DE}

AB = {T1T,4}, so the support is= AB= 2/4 = 0.5
AC = {T1}, so the support is=1/4 = 0.25
AD = {T4,T7,T11}, so the support is= 3/4 = 0.75
AE = {T7,T11}, so the support is= 2/4 = 0.5
BC = {T1}, so the support is=1/4 = 0.25
BD = {T4}, so the support is=1/4 = 0.25
BE = {0}, so the support is = 0
CD = {0}, so the support is = 0
CE = {0}, so the support is = 0
DE = {T7,T11}, so the support is= 2/4 = 0.5

Table 3: Candidate 1-itemset of partition 2

TID	1-itemset
T1	A
T1	B
T1	C
T4	A
T4	B
T4	D
T7	A
T7	D
T7	E
T11	A
T11	D
T11	E

Table 4: Candidate 1-itemset of partition 2

TID	1-itemset
T3	A
T3	D
T3	E
T5	B
T5	C
T5	D
T8	B
T8	C
T8	E
T10	A
T10	C
T10	D

Candidate 3-itemsets are

ABD = {T4}, so the support is = 1/4 = 0.25
ADE = {T7,T11}, so the support is = 2/4 = 0.5
ABE = {0}, so the support is = 0
BDE = {0}, so the support is = 0

So at the end we have only ADE as large 3-itemset in Partition 2.

Partition = 3

Supports of the 1-itemsets are

A = 4, B = 2, C = 1, D = 3, E = 2

Candidate 2-itemset = {AB, AC, AD, AE, BC, BD, BE, CD,CE, DE }

AB = {0}, so the support is = 0
AC = {T10}, so the support is = 1/4 = 0.25
AD = {T3,T10}, so the support is = 2/4 = 0.5
AE = {T3}, so the support is = 1/4 = 0.25
BC = {T5,T8}, so the support is = 2/4 = 0.5
BD = {T5}, so the support is = 1/4 = 0.25
BE = {T8}, so the support is = 1/4 = 0.25
CD = {T5,T10}, so the support is = 2/4 = 0.5
CE = {T8}, so the support is =1/4 = 0.25
DE = {T3}, so the support is =1/4 = 0.25

As we have only one large 2-itemset so no candidate 3-itemset is possible.

Now, we combine the large itemsets of all the partitions and merge them in a single global set (Table 5-7).

Table 5: Globally large 1-itemsets

TID	1-itemset
1	A
1	B
1	C
2	A
2	B
2	D
3	A
3	D
3	E
4	A
4	B
4	D
5	B
5	C
5	D
6	A
6	B
6	E
7	A
7	D
7	E
8	B
8	C
8	E
9	A
9	B
9	C
10	A
10	C
10	D
11	A
11	D
11	E
12	C
12	D
12	E

Table 6: Globally large 2-itemsets

TID	2-itemset
T1,T2,T4	AB
T2,T3,T4	AD
T2,T4	BD
T5,T8	BC
T6,T7	AE
T6,T8	BE
T9,T10	AC
T10,T11	AD
T10,T12	CD
T11,T12	DE

Table 7: Globally large 3-itemsets

TID	3-itemset
T2,T4	ABD

Supports of the 1-itemsets are

A = 9, B = 7, C = 6, D = 7, E = 6

Supports of the large 2-itemsets are

AB = 7, AD = 2, BD = 1, BC = 1, AE = 1, BE = 1, AC = 1, AD = 1, CD = 1, DE = 1.

And the support of the 3-itemset is 1, so no globally large 3-itemset is present in the database taken.

Now, we apply CMA Algorithm on the same database, which also works by randomizing the database and divides the database into three distinct, non-overlapping and random partitions, which are presented in Table 8.

We consider in this example, s = 50%. So the candidate 1-itemsets are presented in Table 9.

In Partition 1 the large 1-itemsets are {A, B, C, D, E}. In Partition 2 the large 1-itemsets are {A, B, D, E}. While in Partition 3 the large 1-itemsets are {A, B, C, D, E}.

The candidate 2-itemsets in Partition 1 are {AB, AC, AD, AE, BC, BD, BE, CD,CE, DE}, in Partition 2 are {AB, AD, AE, BD, BE, DE} and in Partition 3, are {AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}. The large 2-itemsets of three partitions are presented in Table 10.

As shown in the Table 10, large 2-itemset in Partition 1 is {AB}, in Partition 2 are {AB, AD, AE, DE} and in

Table 8: 3 Partitions created by CMA

Partition 1		Partition 2		Partition 3	
TID	Itemset	TID	Itemset	TID	Itemset
T2	A,B,D	T4	A,B,D	T3	A,D,E
T9	A,B,C	T11	A,D,E	T5	B,C,D
T6	A,B,E	T1	A,B,C	T8	B,C,E
T12	C,D,E	T7	A,D,E	T10	A,C,D

Table 9: Candidate 1-itemsets

Partition 1		Partition 2		Partition 3	
Candidate 1-Itemset	Local count (X.support $\geq s*D_i$)	Candidate 1-Itemset	Local count (X.support $\geq s*D_i$)	Candidate 1-Itemset	Local count (X.support $\geq s*D_i$)
A	3≥0.5*4	A	4≥0.5*4	A	2≥0.5*4
B	3≥0.5*4	B	2≥0.5*4	B	2≥0.5*4
C	2≥0.5*4	C	1≥0.5*4	C	3≥0.5*4
D	2≥0.5*4	D	3≥0.5*4	D	3≥0.5*4
E	2≥0.5*4	E	2≥0.5*4	E	2≥0.5*4

Table 10: Large 2-itemsets and their supports

P1		P2		P3	
CI_2^1	$X.sup^1$	CI_2^2	$X.sup^2$	CI_2^3	$X.sup^3$
AB	3	AB	2	AB	0
AC	1	AD	3	AC	1
AD	1	AE	2	AD	2
AE	1	BD	1	AE	1
BC	1	BE	0	BC	2
BD	1	DE	2	BD	1
BE	1			BE	1
CD	1			CD	2
CE	1			CE	1
DE	1			DE	1

Table 11: Large 3-itemsets and their supports

P2		P3	
CI_3^2	$X.sup^2$	CI_3^3	$X.sup^3$
ABD	1	ABC	0
ADE	2	ABD	0
ABE	0	ACD	1
		BCD	1

Table 12: Global Support count of candidate itemsets

Locally large candidate set	X.sup[1]	X.sup[2]	X.sup[3]	X.Sup ≥s*D
A	3	4	2	9≥6
B	3	2	2	7≥6
C	2	1	3	6≥6
D	2	3	3	8≥6
E	2	2	2	6≥6
AB	3	2	0	5<6
AC	1	0	1	2<6
AD	1	3	2	6≥6
AE	1	2	1	4<6
BC	1	0	2	3<6
BD	1	1	1	3<6
BE	1	0	1	2<6
CD	1	0	2	3<6
CE	1	0	1	2<6
DE	1	2	1	4<6

Partition 3 are {AD, BC, CD}. Table 10 shows that no candidate 3-itemset in Partition 1 is possible. Whereas in Partition 2 the candidate 3-itemset are {ABD, ADE, ABE} and in Partition 3 {ABC, ABD, ACD, BCD}.

The support counts of candidate 3-itemsets of three partitions are given in the Table 11.

So the only large 3-itemset found is {ADE}: In order to check whether the locally large itemsets are actually large in database or not, we check each locally large itemset globally with the help of formula X.Sup≥ s*D, where D is the size of the entire database, which in this case is 12.

From the Table 12, the only globally large 2-itemset is {AD} and sine we have a single globally large 2-itemset so no candidate 3-itemset is possible and the process stops here.

CONCLUSIONS

Previously, the database was used to be scanned again and again for the generation of candidate itemsets and then the actually large itemsets, which in the case of huge database means a lot. Which resulted in higher CPU costs and large number of I/Os as well. CMA algorithm presented, not only reduces it, but also decreases the database scanning with the percentage of 50%, which is an achievement. It also results in the efficient processing speed and hence association rules could easily be generated from them. The database scans taken by the different algorithms are given in Fig. 2.

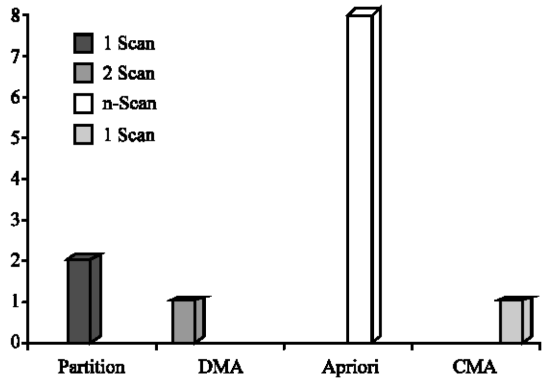

Fig. 2: Database scans taken by different algorithms

We have implemented CMA Algorithm, but used only with our synthetic dataset, created exclusively for the study. In future we will compare PARTITION and our designed CMA Algorithm with different datasets and systems to conduct a comparative study.

REFERENCES

Agarawal, R., T. Imielinski and A. Swami, 1993. Mining association rules Between Sets of item in large databases. Proceedings of the ACM International Conference on Management of Data, pp: 207-216.

Agarawal, R. and R. Srikant, 1994. Fast algorithms for mining association rules. Proceedings of the International Very large Databases Conference, pp: 487-499.

Cheung, D.W., V.T. Ng, A.W. Fu and Y. Fu, 1996. Efficient mining of association rules in distributed databases. IEEE Trans. on Know. Data Engineering, pp: 911-921.

Han, J.M., Kamber and Simon Fraser University, 2001. Data Mining: Concepts and Techniques, Kaufmann Publishers.

Houtsma, M. and A. Swami, 1995. Set-Oriented mining for association rules in relational databases. Proceedings of the IEEE Intl. Conference on Data Engineering, pp: 25-32.

Savasere, A., E. Omiecinski and S. Navathe, 1995. An Efficient algorithm for mining association rules in large databases. Proceedings of the 21st VLDB Conference, Zurich, Switzerland, pp: 432-443.