ISSN No. 0976-5697

Volume 6, No. 6, July-August 2015

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A New Partition Based Association Rule Mining Algorithm for BigData

P.Prabakaran
Research Scholar,
Department of computer science,
Mass college of arts and science,
Kumbakonam, Tamilnadu, India.

Dr.K.RameshKumar, Ph.D.,
Research supervisor & Assistant professor
Department of computer science,
Mass college of arts and science,
Kumbakonam, Tamilnadu, India,

Abstract: Association Rule Mining is an important research area in the field of Data Mining especially in case of 'Sales transactions'. A number of algorithms have been presented in this regard. In this paper a comparison of PARTITION algorithm with CMA algorithm is presented after improving the PARTITION algorithm. In this study, randomized partitioning of database is done. The database is randomized so that real random data is available for better results. The randomized partitioning of database has been implemented in different tool, i.e., VB. Net, as compared to CMA, which uses MATLAB for randomization so as to achieve better performance and efficient results. In the end it has been proved with extensive experiments that although Randomized PARTITION algorithm takes two database scans as compared to CMA that takes single database scan, still it gives better results with more efficiency than CMA.

Keywords: Data Mining, Association Rule Mining, Randomized PARTITION Algorithm

I. INTRODUCTION

In the past few years the amount of data in business organizations has been increased to a greater extent, so the extraction of useful information from such databases is an uphill and challenging task. Data mining techniques can be applied in various fields like sales transaction, marketing, finance, insurance, medicine and fraud detection etc. Interesting association relationships are much helpful in taking good business decisions like how to increase sales or purchases process. Its basic aim is to analyze entire huge database to find the frequently occurring item sets together. For the rule generation it takes one or multiple scans of the whole database. Association rule mining can be best explained by a simple example of market basket analysis that basically determines which groups of products or items are sold or purchased together most frequently at the same time. Customer buying behavior can be best analyzed by this process, as it helps in finding association among different item sets. For example if customers buy soft drink and chips together mostly with in the same visit to the superstore then using this information for the next time the shop keeper will place the soft drink and chips in the nearer shelves. Such shelf arrangement of these items will increase the sales process in future visits to the store. Similarly if a person orders a birthday cake to some bakery, then he is likely to be purchasing some birthday candles as well. If the baker is aware of the association between candles and the birthday cake then he must be offering his customers to purchase candles from him, hence supporting his candle business as well. Market basket analysis is helpful for the retailers in the best adjustments of catalogue design and store layouts etc.

In this study an improved version of PARTITION algorithm is presented and it has been proved through various experiments that its performance is much better than previously implemented PARTITION algorithm as it reduces the memory usage as well as time. Its efficiency is also compared with previously implemented CMA algorithm and it has been proved that due to its efficient partitioning technique of database it gives better results than CMA in terms of time efficiency.

EXISTING RULE MINING ALGORITHMS

Association rule mining for the first time was introduced by Agrawal et. al. (1993). It has remained highly in use by the pundits and practitioners of the data mining for more research since its inception. Many algorithms have been discovered in this field. According to Han and Kamber (2001), association rule mining searches for the interesting relationships among the items in a given dataset. For example, if the support = 5%, confidence=70%. Association rule for the last discussed example of birthday cake and candle is:
Birthday cake => candles

Rule interestingness can be measured well by support and confidence. Support=5% shows that for the above discussed rule, the birthday cake and candles are bought together in all transactions. Confidence=70% means that 70% of the customers who buy a birthday cake also purchased birthday candles.

Association rule mining basic concept as illustrated by Han and Kamber (2001) is as follows: Let $I = \{i_1, i_2, \ldots, i_m\}$ be the set of items. Let D, the task-relevant data, is a set of database transactions where each transaction T is a set of items such that $T \subset I$. Each transaction is associated with an identifier TID. Let A be a set of items. A transaction T is said to contain A if and only if $A \subset T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, and $A \cap B = \varphi$. The rule $A \Rightarrow B$ holds in the transaction set D with support s, where s is the percentage of transactions in D that contain A U B (i.e., both A and B). This is taken to be the probability, P (AUB). The rule $A \Rightarrow B$ has confidence c in the transaction set D if c is a percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, P (B|A). That is,

$$\text{Support } (A \Rightarrow B) = P\,(AUB). \qquad (1)$$

Confidence $(A \Rightarrow B) = P(B|A)$. (2)

Rules that satisfy both a minimum support threshold (min_sup) and a minimum confidence threshold (min_conf) are called strong. By convention, we write support and confidence value so as to occur between 0% and 100%, rather than 0 to 1.0.

Association rule mining is a two-step process. Firstly, find all frequent itemsets: by definition, each of those itemsets will occur at least as frequently as a pre-determined minimum support count. Secondly, generate strong association rules from the frequent itemsets: by definition, these rules must satisfy minimum support and minimum confidence. The association rules are generated simply using the following formula:

If $\frac{\text{support}(\{Y, X\})}{\text{support}(\{X\})} \geq \text{min\_conf}$. (3)

Then $X \Rightarrow Y$ is a valid rule.

Here X is called the antecedent of the rule, whereas Y makes the consequent of the rule.

Many algorithms have been discovered in this field. The pioneer work in this area was presented by Agrawal and Srikant (1994). They discussed Apriori algorithm. Apriori is termed as the best base algorithm for all other subsequent algorithms. Apriori was an iterative algorithm which used complete bottom search to find out large 1-itemset in first pass. Next pass generated candidate itemsets and checked the support. The process repeated until all large itemsets were found. Other variants of Apriori, AprioriTid and AprioriHybrid were also discussed by them in the same paper. In Apriori for *n* iterations, *n* scans of the entire database were done. AprioriTid overcame this problem in later iterations, as it did not use the whole database for counting support after the first pass. Best features of both Apriori and AprioriTid were combined in AprioriHybrid which showed good performance than the other two in real applications.

DHP (Direct Hashing and Pruning) algorithm was presented by Park et. al. (1995), which was an extension of Apriori algorithm. It was confined to the generation of large itemsets, the step one of the mining association rules. The problem with this algorithm was that database pruning benefit was quite ambiguous.

Zaki (1999) presented the survey of parallel and distributed association rule mining algorithms. These algorithms were divided into groups according to the techniques utilized, database structure and search techniques etc. This paper also provided the design space of parallel and distributed algorithms either implemented on distributed or shared–memory architecture. The aim of this paper was to provide a reference for more research. It also described the challenges and problems in the field of association rule mining.

Savasere et. al. (1995) presented PARTITION algorithm that worked in two phases. Logical division of horizontal database into non-overlapping partitions was done. Then locally large itemsets were found for each partition. For each locally large itemset, the TIDs were stored in hash tree. Further potentially large itemsets were obtained by merging the locally large itemsets at the end of phase I. To find the globally large itemsets actual support of these itemsets was measured in phase II. For the available memory to uncover accurate no. of partitions was unsolvable by this algorithm.

Jamshaid et. al. (2007) implements CMA (Centralized Mining of Association-Rules) algorithm for association rule mining. In this algorithm the database was divided into logical non overlapping partitions and then DMA algorithm was applied on each partition to find local and global large itemsets. This algorithm took just a single database scan over each partition for the creation of large itemsets.

From the literature review it is concluded that despite the recent advances in association rule mining algorithms, there are still some problems like in large databases, scanning is much expensive and the resulting candidate itemsets are too large to fit in the aggregate memory. Data skew, data size, multiple scans and pruning techniques needs further study.

PROPOSED NEW RANDOMIZED PARTITION ALGORITHM

In this paper, Randomized PARTITION algorithm has been implemented. Logical non overlapping partitions are created with both random and non random data. Results of both randomized and non randomized partitions are compared to see the effect of data skew on both locally and globally large itemsets. Then the efficiency of Randomized PARTITION algorithm is compared with CMA.

PARTITION algorithm was previously implemented in Silicon Graphics Indy R4400SC workstation with a clock rate of 150 MHz and 32 Mbytes of main memory. The data resided on 1GB SCSI presented by Savasere et. al. (1995).

In this study Randomized PARTITION algorithm has been implemented in the same environment as CMA except the partitioning of database that has been done in a different tool (i.e. CMA has used .NET while Randomized PARTITION algorithm is using MATLAB) for achievement of better performance and efficient results. This change in partitioning technique increases the time efficiency of algorithm as compared to CMA. In today's world, time is an important factor. Along with the need of accurate results, time efficiency matters a lot. It is important to have better results in less time and minimum cost. Randomized PARTITION algorithm has also been improved by using count structure rather storing TIDs in Hash tree as was done in previously implemented PARTITION algorithm. This change reduces the high memory usage and time as compared to previously implemented PARTITION algorithm.

Synthetic database is used for this study which is the same database as was used by CMA. It is also assumed that transactions are in the form (TID, i_1, i_2, i_3). The items are assumed to be kept sorted in lexicographic order. Similar assumption is also made by Han and Kamber (2001).

Algorithm

```
//Read database sequentially
Read_Database
P=Create_Partitions (rand)
For x=1 to P
Begin
Generate_Candidate_Itemsets $C_i$ for $p_x$
For i=1 to k
Begin
```

```
//Generate local large itemsets $L_i$ for $p_i$ from $C_i$ and store items and their count in structure $S_i$
$S_i$=Gen_Local_Large_Itemset $L_i$
        End
//store value in global candidate structure
Merge local large itemsets to form global candidate itemsets $G_{i\_}C$
End
//Generate globally large itemsets $G_i$ from $G_{i\_}C$
$G_i$=Gen_Global_Large_itemset ($G_{i\_}C$)
```

Table 1: Notations Used

Notations	Definition
P_x	Partitions (x=1 to 4)
$C_{i\_}L$	Local candidate itemsets
L_i	Local large itemsets
S_i	Structure for storing local large itemsets along with their counts in particular partition p_i
Gi_C	Global candidate itemsets containing local large itemsets along with their counts
Gi	Globally large itemsets
*	Not possible

Working of Randomized Partition Algorithm

First of all database is read in sequential order. Then logical non overlapping randomized or non randomized partitions are created. Partition size is chosen in such a way that for a specific time a whole partition can easily reside in memory. Partition size should not be very small or very large, as small partitions are negatively affected by data skew and in large partitions for intermediate results processing buffer requirements can exceed the available space, so risk is involved in both cases. Candidate large 1 itemsets are generated for all partitions. Then local large 1 itemsets are generated containing items having their support greater than user defined minimum support within each partition and the candidate itemsets whose minimum support is less than user defined minimum support are pruned away. Here a count structure is used for large itemsets rather than storing TIDs in a Hash tree. Storage of TIDs in hash tree is wastage of time and memory that's why count structure is used for finding large itemsets directly. These large itemsets are then merged and stored in global candidate 1 itemset along with their counts for the generation of globally large itemsets. From large 1 itemsets, candidate large 2 itemsets are generated from which local large 2 itemsets are generated for each partition. The process continues upto locally large k itemset generation. Finally globally large itemsets are generated and global candidate itemsets having minimum support less than user defined minimum support are pruned away. Randomized PARTITION algorithm reduces the time complexity up to O (n) as compared to CMA whose time complexity is O (n^3). Experimental results proved that Randomized PARTITION is 3 times more efficient than CMA.

A. *Functions Of Randomized Partition Algorithm:*

Function [T] = Read_Database

Reads the entire database sequentially and returns the transactions in an array.

Function [P] = Create_Partitions (rand)

This function takes rand as argument. If rand=0 then sequential (non randomized) non overlapping partitions are created otherwise T is first randomized and then logical non overlapping partitions are created and an array of partition P is returned.

Function [LocalLargeItemset] = Generate_Large_Itemsets (p_i)

This function creates candidate itemsets for all partitions and then from these candidate itemsets local large itemsets are generated for each partition. This function returns locally large itemsets for each partition.

Function [GlobalCandidate] = Generate_Global_Candidate (LocalLargeItemsets)

This function takes LocalLargeItemsets as an argument, merge them and generates global candidate itemset.

Function [GlobalLargeItemset] = Generate_Global_Itemsets (GlobalCandidate)

This function takes GlobalCandidate as an argument and generates globally large itemsets RIP.

B. *Explanation of Randomized Partition Algorithm with Example:*

Performance of Randomized PARTITION algorithm can be best demonstrated by the case study. Following dataset was used for the case study: The dataset shown in Table II was randomized as shown in Table III to find out large itemsets. The results generated by this dataset are shown in Table IV. For testing the algorithm, many experiments were performed by taking both random and non random data. Number of items in the given dataset was 5.

Initially, the database was read sequentially and four logical non overlapping partitions were created. Then local and global itemsets for sequential database partitions were generated. In the second step, the database was randomized and again logical non overlapping partitions (P_1, P_2, P_3, P_4) were created. After partition creation, the Randomized PARTITION algorithm was applied to find locally large 1-itemsets for each partition. From this, candidate 2-itemset was generated and then locally large 2-itemset for each partition was created. This process continued unless up to k large itemsets were found. The algorithm stored counts of the locally large itemsets in a structure rather than using a hash tree for this process. This change reduced the high memory usage as compared to previously implemented PARTITION algorithm. Then large itemsets of all lengths were merged to produce global candidate itemsets from which the globally large itemsets of all lengths were generated. Table V shows the experimental results of randomized and non-randomized partitions with 100 transactions.

The comparison of results show that in non randomized or sequential partitions, there was data skew involved which caused outliers while randomization of partitions removed the outliers. Table VI shows the results of comparison of Randomized PARTITION algorithm with CMA. Experimental results show that although Randomized PARTITION algorithm takes two database scans for large itemset creation still its efficiency is better than CMA which takes one database scan for large itemset creation. The time complexity of partitioning technique as well as the rest of the algorithm is much better than CMA. Many experiments were performed using varying size of datasets. Table VII shows some of the experimental results with 20, 100, 1000, 10,000 transactions. Given dataset calculates upto large 3-itemsets. Randomized PARTITION algorithm can calculate upto k itemsets with other datasets. Randomized PARTITION algorithm was also compared with previously implemented PARTITION algorithm and it was observed that its efficiency is better in terms of memory usage and time as shown in Table VIII. Extensive experiments were performed with different datasets for large itemset generation upto k itemset to check the efficiency of algorithm.

Table 2: Sequential Dataset

Partitions			
P_1	P_2	P_3	P_4
T_1: A,B,C	T_{26}: A,D,E	T_{51}: A,E	T_{76}: A,C,D,E
T_2: A,B,D	T_{27}: A,B,C,D,E	T_{52}: A,B,E	T_{77}: B,C,E
T_3: B,D,E	T_{28}: C,D,E	T_{53}: B,D,E	T_{78}: C,D,E
T_4: A,B,E	T_{29}: B,C,D,E	T_{54}: B,C,E	T_{79}: B,C,D,E
T_5: A,C,D	T_{30}: A,D,E	T_{55}: C,D	T_{80}: A,D,E
T_6: A,C,E	T_{31}: A,B,E	T_{56}: A,D,E	T_{81}: C,D,E
T_7: A,C,D	T_{32}: D,E	T_{57}: A,D,E	T_{82}: B,D,E
T_8: B,D,E	T_{33}: A	T_{58}: A,C	T_{83}: A,B,C,E
T_9: C,D	T_{34}: C,D	T_{59}: A,B,C,E	T_{84}: B,C,E
T_{10}: C,D,E	T_{35}: C,D,E	T_{60}: A,B,C,D	T_{85}: B,C,D
T_{11}: A,C,D	T_{36}: B,C	T_{61}: A,C,D	T_{86}: C,D,E
T_{12}: A,B,E	T_{37}: A,C	T_{62}: A,C,D,E	T_{87}: B,E
T_{13}: B,D,E	T_{38}: A,C,D,E	T_{63}: D,E	T_{88}: A,B,E
T_{14}: D,E	T_{39}: D,E	T_{64}: C,D,E	T_{89}: A,B,C
T_{15}: C,D,E	T_{40}: B,D,E	T_{65}: A,B,D	T_{90}: A,C,E
T_{16}: A,D,E	T_{41}: A,B	T_{66}: B,D,E	T_{91}: B,C,E
T_{17}: A,B,C,D	T_{42}: A,C	T_{67}: C,D,E	T_{92}: B,D,E
T_{18}: B,C,E	T_{43}: A,B,C	T_{68}: A,D,E	T_{93}: A,C,E
T_{19}: A,C,E	T_{44}: B,C,D	T_{69}: B,D,E	T_{94}: B,C,E
T_{20}: B,C,D	T_{45}: D,E	T_{70}: A,C	T_{95}: D,E
T_{21}: A,B,D,E	T_{46}: B,C,D,E	T_{71}: A,B,C	T_{96}: C,E
T_{22}: C,D,E	T_{47}: C,D,E	T_{72}: A,C,D,E	T_{97}: B,D,E
T_{23}: B,E	T_{48}: D,E	T_{73}: B,C,D,E	T_{98}: A,B,D,E
T_{24}: A,C,D	T_{49}: C,D	T_{74}: A,D,E	T_{99}: D,E
T_{25}: B,D,E	T_{50}: A,D	T_{75}: B,C,D,E	T_{100}: A,C,D,E

Table 3: Randomized Dataset

Partitions			
P_1	P_2	P_3	P_4
T_{24}: A,C,D	T_{95}: D,E	T_{70}: A,C	T_{81}: A,C,D
T_{92}: B,D,E	T_{53}: B,D,E	T_{54}: B,C,E	T_{23}: B,E
T_{31}: A,B,E	T_{49}: C,D	T_{77}: B,C,E	T_{65}: A,B,D
T_8: B,D,E	T_{72}: A,C,D,E	T_{74}: A,D,E	T_9: C,D
T_{42}: A,C	T_{69}: B,D,E	T_{60}: A,B,C,D	T_{63}: D,E
T_{21}: A,B,D,E	T_{67}: C,D,E	T_{28}: C,D,E	T_{45}: D,E
T_{99}: D,E	T_{22}: C,D,E	T_3: B,D,E	T_{41}: A,B
T_{91}: B,C,E	T_{75}: B,C,D,E	T_{11}: A,C,D	T_{73}: B,C,D,E
T_{25}: B,D,E	T_{57}: A,D,E	T_{80}: A,D,E	T_{37}: A,C
T_{55}: C,D	T_{44}: B.C.D	T_{64}: C,D,E	T_{59}: A,B,C.E
T_{85}: B,C,D	T_{16}: A,D,E	T_{66}: B,D,E	T_{58}: A,C
T_{15}: C,D,E	T_{19}: A,C,E	T_{96}: C,E	T_{89}: A,B,C
T_{50}: A,D	T_{36}: B,C	T_{40}: B,D,E	T_{84}: B,C,E
T_{51}: A,E	T_{48}: D,E	T_{43}: A,B,C	T_5: A,C,D
T_{27}:A,B,C,D,E	T_{10}: C,D,E	T_{52}: A,B,E	T_{20}: B,C,D
T_{30}: A,D,E	T_{78}: C,D,E	T_{79}: B,C,D,E	T_{93}: A,C,E
T_{94}: B,C,E	T_{33}: A	T_{56}: A,D,E	T_{62}: A,C,D,E
T_{39}: D,E	T_7: A,C,D	T_{76}: A,C,D,E	T_{18}: B,C,E
T_{26}: A,D,E	T_{35}: C,D,E	T_{47}: C,D,E	T_{13}: B,D,E
T_2: A,B,D	T_{98}: A,B,D,E	T_{71}: A,B,C	T_{34}: C,D
T_{88}: A,B,E	T_4: A,B,E	T_{90}: A,C,E	T_{17}: A,B,C,D
T_{87}: B,E	T_{61}: A,C,D	T_{14}: D,E	T_1: A,B,C
T_{29}: B,C,D,E	T_{46}: B.C.D.E	$T_{32:}$ D.E	T_{82}: B,D,E
T_{97}: B,D,E	T_{83}: A,B,C,E	T_6: A,B,C	T_{86}: C,D,E
T_{68}: A,D,E	T_{38}: A,C,D,E	T_{12}: A,B,E	T_{100}:A,C,D.E

Table 4: Results of Local and Global Itemset Generation

Local Large itemsets (L_i)				Global Large Itemset (G_i)
P_1 Item = Count	P_2 Item = Count	P_3 Item = Count	P_4 Item = Count	G_i Item = Count
L_1:B=14 D=18, E=19 L_2:* L_3:* L_4:* L_5:*	L_1:C=16 D=19, E=19 L_2:DE=17 L_3:* L_4:* L_5:*	L_1:C=15 D=15, E=20 L_2:* L_3:* L_4:* L_5:*	L_1:C=18 D=15 E=14 L_2:* L_3:* L_4:* L_5:*	G_1:D=52 E=72 G_2:* G_3:* G_4:* G_5:*

Table 5: Comparison of number of large itemsets of randomized and non randomized datasets

Sequential		Randomized 1		Randomized 2		Randomized 3	
ΣL_i P	G_i	ΣL_i P	G_i	ΣL_i P	G_i	ΣL_i P	G_i
L_1=14 L_2= 1 L_3= * L_4= * L_5= *	G_1=3 G_2=* G_3=* G_4=* G_5=*	L_1=12 L_2= 1 L_3= * L_4= * L_5= *	G_1=2 G_2=* G_3=* G_4=* G_5=*	L_1=12 L_2= 1 L_3= * L_4= * L_5= *	G_1=2 G_2=* G_3=* G_4=* G_5=*	L_1=11 L_2= 2 L_3= * L_4= * L_5= *	G_1=2 G_2=* G_3=* G_4=* G_5=*

Table 6: Comparison of Randomized Partition with CMA

CMA	Randomized PARTITION
Time complexity of Partitioning database=O(n^3) Time complexity of algorithm after partitioning=O(n^3)	Time complexity of Partitioning database=O(n) Time complexity of algorithm after partitioning=O(n)

Table 7: Comparison of number of large itemsets with variable size of data

Datasets		No. of Local Large itemsets				Sum	G_i	
Trans	m_sup	P_1	P_2	P_3	P_4	ΣL_i	m_ sup	G_i
20	3	L_1=2 L_2=1 L_3=* L_4=* L_5=*	L_1=5 L_2=1 L_3=* L_4=* L_5=*	L_1=3 L_2=2 L_3=* L_4=* L_5=*	L_1=4 L_2=5 L_3=2 L_4=* L_5=*	14 9 2 * *	10	G_1=2 G_2=1 G_3=* G_4=* G_5=*
100	13	L_1=3 L_2=* L_3=* L_4=* L_5=*	L_1=3 L_2=1 L_3=* L_4=* L_5=*	L_1=4 L_2=1 L_3=* L_4=* L_5=*	L_1=4 L_2=1 L_3=* L_4=* L_5=*	14 3 * * *	50	G_1=3 G_2=* G_3=* G_4=* G_5=*
1000	250	L_1=4 L_2=1 L_3=* L_4=* L_5=*	L_1=4 L_2=1 L_3=* L_4=* L_5=*	L_1=4 L_2=1 L_3=* L_4=* L_5=*	L_1=4 L_2=1 L_3=* L_4=* L_5=*	16 4 * * *	500	G_1=4 G_2=1 G_3=* G_4=* G_5=*
1,0000	2500	L_1=4 L_2=1 L_3=* L_4=* L_5=*	L_1=4 L_2=1 L_3=* L_4=* L_5=*	L_1=4 L_2=1 L_3=* L_4=* L_5=*	L_1=3 L_2=1 L_3=* L_4=* L_5=*	15 4 * * *	5000	G_1=3 G_2=1 G_3=* G_4=* G_5=*

Table 8: Comparison of previous partition with randomized partition algorithm

Previous PARTITION	Randomized PARTITION
Memory usage=increased Hash tree storage	Memory usage=decreased Structure with count

IV. CONCLUSION

This improved version of Randomized PARTITION algorithm is more efficient than CMA as its partitioning technique takes much less time than the one used by CMA. Its time complexity is O (n) much better than the time complexity of CMA that is O (n^3). Its efficiency is greater than CMA due to its changed partitioning technique. This version of Randomized PARTITION algorithm is also more efficient than previously implemented PARTITION algorithm as it reduces the memory usage by simply storing counts for each large itemset instead of storing the TIDs in hash tree. This has also increased the time efficiency of Randomized PARTITION algorithm as compare to previously implemented PARTITION algorithm. In future we have a plan to implement this algorithm in parallel environment for better efficiency.

V. REFERENCES

[1] Han, J. and M. Kamber, 2001,Data Mining: concepts and Techniques. Kaufmann Publishers, San Francisco, Calfornia, USA, 2001.

[2] Agrawal, R., T.Imielinski, and A. Swami, 1993. Mining Association Rules between Sets of Items in Large Databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington, D.C.USA, May 1993, pp.207-216.

[3] Agarwal, R. and R.Srikant, 1994. Fast algorithms For Mining Association Rules. In Proceddings of the 20th International Conference on Very Large Data Bases,

Santiago de Chile, Chile, September 1994, edited by J.B.Bocca, M.Jarke, and C.Zaniolo, “Very Large Data Bases”, Morgan Kaufmann Publishers, San Francisco, CA, USA, pp. 487-499.

[4] Park. J.Soo, M. Chen and P.S.Yu, 1995. An Effective Hash-Based Algorithm for Mining Association Rules. In Proceedings of the ACM SIGMOD International Conference on Management of Datga, May 1995, pp.175-186.

[5] Zaki, M.J.1999. Parallel and Distributed Association Mining: A Survey. In EEE Concurrency Journal, Special issue on Parallel Mechanisms for Data Mining, Vol.7, No.4, December 1999, pp.14-25.

[6] Savasere, A.E. Omiecinski and S.Navathe, 1995. An Efficient Algorithm for Mining Association Rules in Lagre Databases, In Proc. Of the 21st Int’l VLDB Conference, Zurich, Switzerland, September 1995, pp.432-444.

[7] Jamshaid, S., Z. Jalil, M,S.H.Khiyal and M. Saeed, 2007. Association Rule Mining in Centralized Databases, Information technology Journal, Vol.6, Issue 2, 2007, pp.181.