Journal of Theoretical and Applied Information Technology
10th May 2014. Vol. 63 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

IAPI QUAD-FILTER: AN INTERACTIVE AND ADAPTIVE PARTITIONED APPROACH FOR INCREMENTAL FREQUENT PATTERN MINING

[1]SHERLY K.K, [2]Dr. R. NEDUNCHEZHIAN, [3]Dr. M. RAJALAKSHMI

[1]Assoc. Prof., Dept. of Information Technology, Toc H Institute of Science & Technology, Ernakulam
[2]Principal, Sri Ranganathar Institute of Engineering & Technology, Coimbatore, India
[3]Associate Professor, Department of Computer Science, Coimbatore Institute of Technology, Coimbatore
E-mail: [1]sherly.shilu@gmail.com, [2]rajuchezhian@gmail.com, [3]raji_nav@yahoo.com

ABSTRACT

Association rule mining has been proposed for market basket analysis and to predict customer purchasing/spending behaviour by analyzing the frequent itemsets in a large pool of transactions. Finding frequent itemsets from a very large and dynamic dataset is a time consuming process. Several sequential algorithms have contributed to frequent pattern generation. Most of them face problems of time and space complexities and do not support incremental mining to accommodate change in customer purchase behaviour. To reduce these complexities researchers propose partitioned and parallel approaches; but they are compromising on anyone of these. An interactive and adaptive partitioned incremental mining algorithm with four level filtering approaches for frequent pattern mining is proposed here. It prepares incremental frequent patterns, without generating local frequent itemsets in less time and space complexities and is efficiently applicable to both sequential and parallel mining.

Keywords— *Frequent Pattern Mining; Association Rule; Partitioned Database; Parallel Mining; Interactive Mining; Incremental Mining*

1. INTRODUCTION

In the present scenario plenty of data and information are available globally. The explosive growth of electronic commerce increases online transactions every year. Organizations store their ever-increasing day-to-day transactional details in their transaction databases. Data mining techniques can be used for retrieving knowledge from the available data and information. It prepares models by analyzing the hidden relationships among stored data. Association rule mining (ARM) is one of the data mining tasks, which has been applied for different real-life applications: [1] and [2].

Association rules identify the set of items that are most often purchased with another set of items in a transaction database. Frequent pattern mining is an important task to obtain associations or correlations among items in a large dataset which in turn becomes a time consuming process. Frequent patterns are itemsets that appear frequently in a dataset. The main operation in the frequent pattern mining process is computing the occurrence frequency of the interesting set of items and identifying the subsets of items that frequently occur in many database transactions. Association rules derived from frequent patterns enable behavioural analysis, marketing policies, web log analysis, decision making and even in medical diagnosis and fraud detection [8] and [14].

As the amount of transactions increases it becomes very difficult to determine the frequent patterns with fewer complexities. Thus efficient techniques are required to find association rules from very large databases. A potential solution for improving the performance and providing scalability is to parallelize the mining algorithms. But computational cost of parallel mining is fairly high compared with sequential mining. A significant number of parallel and distributed FP mining algorithms have been proposed [3]. Most of these approaches generate static frequent patterns and suffer excessive communication overhead. Static approaches do not provide the support to accommodate the day to day transactions. Usage of static patterns for long term prediction of frequent patterns for a dynamically growing database is not advisable. Thus algorithms support incremental mining is suitable for dynamic databases to accommodate the pattern changes. Dynamically growing databases may become very large after a certain period of time. There is a possibility for change in behaviour due to the change in life style. Thus very old transactions may be removed from the database to accommodate the new behavioural patterns. Different users may require different

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

support values for different applications, finding an appropriate support value is a challenging task. It is better to provide the user with the facility to interactively adjust the support value as per their requirement. Hence an algorithm capable to handle very large database with incremental and interactive data mining capability is required for dynamic databases.

The objective of this research is to propose a method which can solve these shortcomings and generate frequent patterns which are efficient, scalable, faster and interactive with the capability to accommodate behaviour changes with less CPU and I/O overheads. To overcome these limitations a new partitioned approach is proposed which generates frequent itemsets without generating any local frequent itemsets. This algorithm is designed for dealing with the problems of space complexity, time complexity and computational cost faced by the researchers in frequent pattern mining of very large database and provides incremental mining as well as interactive mining to accommodate the change in behavioural patterns.

1.1 Overview

The proposed frequent pattern mining algorithm, Incremental and Adaptive Partitioned Incremental Mining Four-level filter (IAPI Quad-filter) deal with the problems and difficulties of association rule mining in very large datasets. This algorithm uses a database partitioning approach to produce frequent itemsets without generating local frequent itemsets. In this approach, transaction items are pre-processed and arranged according to the item code, thus individual item counting and count comparisons are made faster. Rather than fixing single minimum support value IAPI Quad-filter uses a range of support values (***low, high***) for making the dynamic and the interactive mining faster. It divides the dataset into small sized non-overlapping partitions of user specified sizes so that each partition can be accommodated in the main memory. To reduce the computational cost as well as I/O overhead and space complexity while finding the frequent itemsets, each frequent item transaction group is collected separately and four level filtering is done to remove infrequent items, which is shown in Figure 1. First, it removes the global infrequent items from the selected transactions, then items which have count less than the count of the selected frequent item are removed from each transaction. Also similar transactions of the resultant transaction sets are removed, after recording their occurrence count. Then to reduce the computational cost further, items which are found to be infrequent in the selected group are also removed and considered only the frequent ones to get the frequent itemsets.

In Apriori when the length of the itemsets increased, the number of candidate sets gets reduced accordingly, but the size and the number of transactions to be compared remains same. But in IAPI-Quad-filter the number of transaction to be compared and their size also get reduced in finding higher frequent itemsets. This method is supported by incremental database features and is also adaptive to the customer behavioural changes. IAPI also provides the user with the facility to interactively adjust the minimum support value as per one's own conveniences. Thus, this algorithm performs better than the existing algorithms with respect to scalability, speed and efficiency. The main attraction of this approach is that it generates the frequent itemsets without producing any local frequent itemsets; thus generation of false frequent itemsets are eliminated.

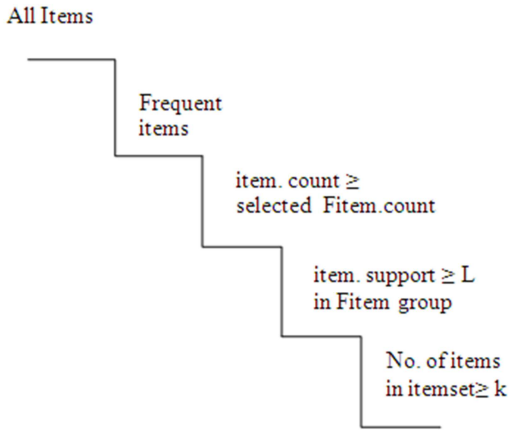

Figure 1 Four-level filtering of IAPI

This paper is organized as follows. Section 2 describes related work on different frequent pattern mining algorithms. Section 3 presents the basic terminologies associated with the proposed method. Section 4 gives the details of the various phases of the proposed algorithm and their functionalities are described using sample data. Section 5 gives the details of experiments conducted and performance analysis. Section 6 concludes the paper.

2. RELATED WORK

Several algorithms have been developed by researchers to find the static frequent patterns from small and medium sized databases. The popular algorithm Apriori R. Agrawal et al. [1] forms the foundation for static frequent pattern mining. Apriori-like approaches prepare frequent pattern by generating candidate sets. The major problem of Apriori is that it has to read the entire database in every pass, although many items and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

transactions are no longer needed in later passes. It generates candidate item sets iteratively. Thus its computational cost is very high. Eclat, M.J.Zaki. [4] is basically a depth-first search algorithm uses vertical database layout and uses intersection based approach to compute the support of an itemset. First scan of the database builds the TID_set of each single item. The advantage of Eclat over Apriori is that there is no need to scan the database to find the support of (k+1) itemset, because the TID_set of each k-itemset carries the complete information required for counting support and it also uses the Diffset technique. However, the TID_set can be long, taking more memory space as well as computation time for intersecting the long sets. The hybrid algorithm, C.Borgelt. [5] performs best when Apriori was switched to Eclat after the second iteration.

To improve the efficiency of the mining process, Han et al. [6] proposed a tree structure based algorithm FP-growth that constructs a frequent pattern tree which generates frequent patterns in two database scans without generating candidate sets. It shows an outstanding improvement over Apriori. The disadvantage of FP-growth is that it has to generate conditional pattern bases and sub- conditional pattern tree recursively. Addition of new transactions may require reconstruction of the FP-tree. M. El-Hajj et al. [7] proposed another approach COFI tree that uses a compact memory based data structure. For each frequent item, a relatively small independent tree is built summarizing co-occurrences and a clever pruning reduces the search space drastically. Finally, a simple and non-recursive mining process reduces the memory requirements as minimum candidacy generation and counting is needed to generate all relevant frequent patterns. Qian Wan et al. [9] use a compact tree structure, called CT-tree, to compress the original transactional data. This allows the CT-Apriori algorithm, to generate frequent patterns quickly by skipping the initial database scan and reducing a great amount of I/O time per database scan. It reduces the storage space requirement and mining time. These are static algorithms.

To obtain frequent sets from very large datasets with low memory space utilization in two database scan, researchers propose partitioning algorithms. Ashok Savasere et al. [10] suggest a partitioning algorithm, which splits the given database into a number of small segments. During the first scan on the database, it identifies the local frequent itemsets of individual segments. At the end of the first scan, these local large itemsets are merged to generate a set of all potential large itemsets. In the second scan, the actual support for these itemsets are generated and the large itemsets are identified. This algorithm is highly dependent on the heterogeneity of the database; it may generate too many independent local frequent itemsets and they may not fit in the memory. To analyze the problem of market basket data, Sergey Brin et al. [11] present an algorithm DIC which uses fewer passes over the data than classical algorithms to find the frequent itemsets. DIC provides the flexibility to add and delete itemsets on the fly. DIC checks all the subsets of each transaction to find frequent sets which is expensive both in time and space compared to partition algorithm. DIC can accommodate the behavioural changes and interactive mining with less time complexity.

CARMA proposed by C.Hidber [12] requires two database scans to produce all large itemsets. During the first scan, the algorithm continuously constructs a lattice of all large item-sets. For each set in the lattice CARMA provides a lower bound and an upper bound for its support. The user is free to adjust support and confidence at any time. During the second scan, the algorithm determines the precise support of each set in the lattice and continuously removes all small itemsets. Carson Kai-Sang et al. [13] propose a dynamic algorithm CanTree which facilitates incremental mining as well as interactive mining. In this approach, the items in each transaction are arranged in a canonical order and the entire transactions are stored in a tree structure with one database scan. The construction of CanTree is independent of the threshold values. Thus, interactive mining is possible without rescanning the entire database. S.K Tanbeer et al. [15] present a novel tree structure called CP-tree, which creates efficient frequent patterns with interactive and incremental mining functionalities in one database scan. It has two phases, insertion phase that inserts transactions into CP-tree and tree reconstructing phase that rearranges the items according to the frequency order. Since items are arranged in the ascending order, CP-tree has less number of nodes compared to CanTree. But tree reconstruction introduces additional computations. M. Hamedanian et al. [16] have proposed a new prefix tree structure to reduce the time of restructuring.

3. BASIC TERMINOLOGIES

The database $\boldsymbol{D}$ consists of variable length transactions $\boldsymbol{T}$ and these transactions have itemsets $\{X,Y,Z,\ldots\}$. Let $I = \{I_1, I_2, I_3, \ldots I_m\}$ be a set of m

Journal of Theoretical and Applied Information Technology
10th May 2014. Vol. 63 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

distinct items and transaction T is a set of items such that $T \subseteq I$. Let X and Y are sets of items. An itemset with k elements is called k-itemsets. Database D is a multiset of subsets of I, $T \in D$.
Association rules are used to discover the associations among items in a transactional database. Such rules can be used to classify customers according to the purchase pattern, product placement, target marketing and so on. It is a relationship between two or more items of the form $X \rightarrow Y$ where $X,Y \subseteq I$ and $X \cap Y = \Phi$. Such a rule reveals that the transactions in the database containing items in X tend to contain in items Y. Support of an itemset X in a transaction sequence is the fraction of all transactions containing the itemset i.e the frequency occurrence of X in D. Support S of an association rule $X \rightarrow Y$ in the transaction set D is the percentage of transactions in D that contain items both X and Y.

$$S(X) = \frac{|X|}{|D|} \quad \ldots\ldots\ldots\ldots \quad (1)$$

$$S(X,Y) = \frac{|X \cup Y|}{|D|} \quad \ldots\ldots\ldots\ldots \quad (2)$$

Confidence C of an association rule $X \rightarrow Y$ in the transaction set D is the conditional probability that a transaction having X also contains Y. This rule holds in D with confidence C if $C\%$ of transactions in D that contain X also contain Y.

$$C(X \rightarrow Y) = \frac{S(X \cup Y)}{S(X)} \quad \ldots\ldots\ldots(3)$$

An itemset is considered as frequent or large if its support is greater than or equal to the user defined support threshold otherwise it is small. To generate strong association rule, the support and confidence of the rule should satisfy a user specified minimum support and confidence.

4. PROPOSED METHOD

4.1 Problem Definition

Let D be a database with N number of transactions. Let I be the item domain, $\{I_1,I_2,\ldots..I_m\}$. The problem is to identify all interesting frequent patterns in an interactive and incremental manner. A partition $P \subseteq D$ of the database refers to any subset of the transactions contained in the database D. Initially the database D is logically partitioned into r non-overlapping partitions of size Z, i.e. $Pi \cap Pj = \Phi$, $i \neq j$. Two minimum support values used here are: Sl, Sh namely, lower minimum support value and upper minimum support value. It creates two category itemsets: Frequent ($Fset$), Nearly Frequent ($NFset$). Itemset X is Frequent if support(X) $\geq Sh$ and Nearly frequent if $Sl \leq$support (X) $\leq Sh$. Pn represents a partition number at which a $NFset$ has been last updated. Let f be the frequent item domain, $\{f_1,f_2,\ldots..f_n\}$ in the ascending order of occurrence count. Each frequent item is associated with a co-occurring itemset list Cf, $\{Cf_1,Cf_2,\ldots Cf_{n-1}\}$ refers to the subset of frequent items, where as Cf_1 be the co-occurring item list of frequent item f_1, $\{f_2,f_3,\ldots..f_n\}$, $Cf_2=\{f_3,f_4,\ldots..f_n\}$. $Cf_1 \supset Cf_2 \supset \ldots\ldots \supset Cf_{n-1}$, it indicates that as the frequency of occurrence is more the number of co-occurring items considered for frequent itemset mining get reduced.

4.2 Algorithm Functionalities

This algorithm has four phases. The first phase generates the frequent itemsets from the large history database. The second phase accommodates the newly arrived transactions to the existing set and updates the frequent itemsets to provide incremental mining. The third phase removes the old transactions after a preset time period and modifies the pattern as per the changes occurred in the dataset to accommodate the behavioural changes. The last phase provides the facility to interactively adjust minimum support value as per the user's requirement. Counting strategy adopted in the proposed approach to determine the itemset support is horizontal counting.

The proposed method filters out the infrequent items at four levels before it searches for the frequent itemsets. At the first level, it removes the global infrequent items, at the second level, it eliminates items whose support count less than the selected frequent item, at the third level, it selects the itemsets which have length k or more to obtain the frequent k-itemsets and at the fourth level, it considers only the items which are frequent in a set where a frequent item belongs to. IAPI combines the features of Apriori, Partition, CanTree, COFI tree, CARMA and record filter approaches and avoids the tree construction complexities.

Property 1: I2AP provides two bounds (Sl, Sh) for minimum support to reduce search delay to update the frequent itemsets in incremental mining and interactive mining.
Property 1.1: The itemsets with support value in between Sl and Sh are stored as nearly frequent sets with the partition number last updated.
Property 1.2: Nearly frequent sets are updated only if they appear as a frequent set in the newly added partition or if the preset minimum support value Sh is decremented by the user.

Journal of Theoretical and Applied Information Technology
10th May 2014. Vol. 63 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Property 2: To reduce the memory utilization and searching delay in finding the frequent itemsets, each frequent item transaction groups are collected separately and infrequent items in the selected group are filtered out at four levels.

Property 3: Each frequent item has a co-occurring itemset list ***Cf*** for finding the frequent itemsets, the same set of items is considered while adding new transactions.

Property 3.1: Co-occurring items list of each frequent item is decided at the time of initial frequent pattern generation, based on their support count value and is updated only on every removal of old partitions.

4.2.1 Frequent Itemsets Generation

There are two steps in the first phase; the first step is to find the frequent items in the entire dataset and the second step is to find the frequent itemsets among the identified frequent items. The phase I functional details are shown in figure 2.

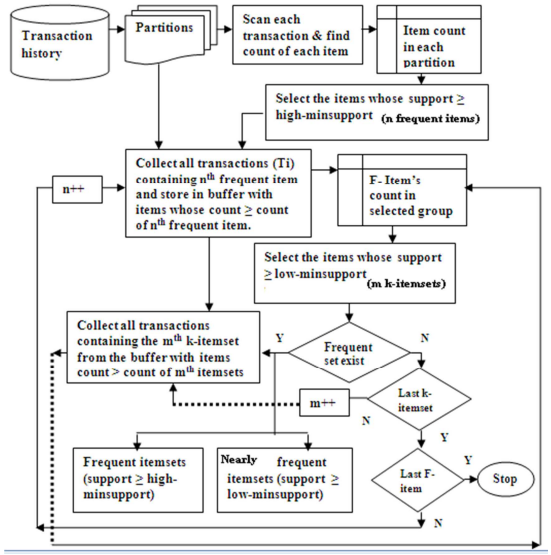

Figure 2 Functional block diagram of phase I

For finding the frequent items ***f***, first divide the database into convenient size of non-overlapping partitions and from each partition find the count of individual items and record their count in each partition separately in an item count table. Then find the frequent items which meet the minimum support count (***Sh***) in the entire dataset during the first database scan. Initially, items in each transaction are arranged according to the item code, which is a fixed global ordering and a counter is set up for each item and initialized to zero. Increment the count of items as per their occurrence in each transaction and record their count in each partition separately. Then their cumulative count gives the global support for the items. Support calculation is done by scanning transaction one by one, and increasing the counter of the itemset if it is a subset of the transaction. Figure 3 illustrates the first step of phase I, that is global frequent items generation using a sample dataset having 15 transactions with 14 distinct items and these transactions are divided into 3 partitions with 5 transactions each. IAPI Quad-filter uses two preset minimum values (***Sl, Sh***) to facilitate incremental and interactive mining [property 1]. The given example sets low, high values as 25% and 40% respectively. Here among 14 items, 7 are found to be globally frequent (support ≥ 40%).

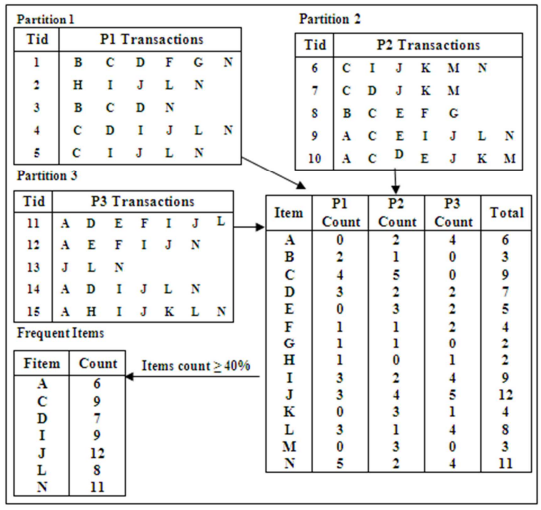

Figure 3 Frequent Item Generations

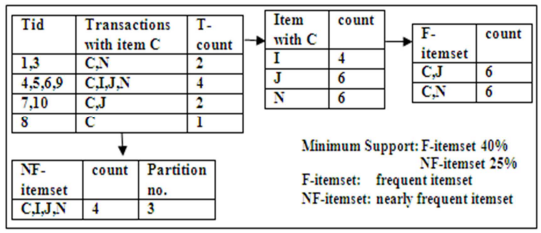

Figure 4: Frequent itemset generation of item C

Table 1: Co-occurring Items List

Item	Co-occurring Items	count
A	D,L,C,I,N,J	6
D	L,C,I,N,J	7
L	C,I,N,J	8
C	I,N,J	9
I	N,J	9
N	J	11

To find the frequent itemsets first collect all the transactions that contain the first frequent item f_1 from the entire database and store only the items whose count is greater than the selected frequent item in memory buffer[property 2]. Keep a list of items ***Cf*** which are considered along with each frequent item for finding the frequent itemsets.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The same sets of items are considered together for finding the frequent itemsets on addition of new set of transactions [property 3]. Figure 4 shows the frequent itemset generation steps of item ***C*** by grouping the transactions containing item ***C*** into a buffer. To reduce the buffer size and computational cost of finding frequent itemsets, it collects only 3 items {***I,J,N***}, where their count is greater than or equal to the count of item ***C*** which is 9. These three items are recorded in the co-occurring list of item ***C*** shown in Table 1. Then record the count of each item in that transaction group and remove the infrequent ones (support < 25%) from the selected transaction set, which gives the count of frequent and nearly frequent 2-itemsets.

Tid	Transactions with item I	T-count
2.4.5.6.9, 12,14,15	I,J,N	8
11	I,J	1

Item with I	count
J	9
N	8

F-itemset	count
I,J	9
I,J,N	8

Figure 5: Frequent itemset generation of item I

Now to find the frequent 3-itemsets, select the set of transactions containing the first frequent 2-itemset {***C,I***} from the buffer by eliminating the items having count less than the count of item ***I*** in the 2-itemset group and identify the items having support ≥ ***Sl***. The same procedure is used for finding the higher frequent itemsets (4-itemset, 5-itemset and so on). For finding the support of k-itemsets it is necessary to collect only the transactions containing at least k items. Similar steps are used for other 2-itemsets {***C,N***}of item ***C*** to get their co-occurring higher frequent itemsets. Repeat the same for all other frequent items {***I,J,N***} identified in the first scan and find all possible frequent itemsets of the whole dataset (Figure 5). Keep the list of frequent and nearly frequent itemsets separately. There may be chances of new frequent itemsets to occur, on adding new transaction sets to the existing one. Thus ***NFsets*** helps to avoid the rescanning of the whole dataset, while adding new partitions and updates the frequent patterns with less time compared to the existing methods[property 1.1]. Updating the ***NFsets*** on addition of every partition requires more time. Thus ***NFsets*** are updated only if they appear as a frequent set in the newly added partition. To keep track of how many newly added partitions are to be scanned for updating the ***NFset***, every ***NFset*** is tagged with the lastly updated partition number ***Pn*** [property 1.2].

4.2.1.1 IAPI Quad-filter Algorithm -Phase I

1. Partition database into ***r*** non-overlapping partitions of convenient size (***Z***) to accommodate in memory. Fix two minimum support values, ***Sh*** as the user selected minimum support and ***Sl*** to accommodate dynamic support (***Sl*** < ***Sh***).
2. Find the count of each item in each partition, then calculate their total count and identify frequent 1-itemsets (support ≥ ***Sh***) and initialize item length ***k***=1.
3. Prepare a co-occurring item list, ***Cf*** for each frequent item, i.e. items having support greater than or equal to the support of that item which is considered for the frequent itemset selection.
4. Select each frequent k-itemset and find its co-occurring frequent items from the set of transactions containing the selected frequent itemset by removing the items having count less than the count of that itemset from each transaction.
5. Remove the items having count less than the ***low*** minimum support count in the selected group and identify the (k+1)-itemsets with support ≥ ***Sl***, then increment ***k***.
6. Repeat the steps 4, 5 for higher itemsets till no higher frequent sets are obtained, then decrement ***k.***
7. Repeat the steps 4,5 and 6 for all k-itemsets of the selected frequent item and identify the frequent itemsets (support ≥ ***Sh***) and nearly frequent itemsets (support between ***Sl*** and ***Sh***) containing the selected frequent item say f_I; also record the current partition number (***Pn***) along with each nearly frequent itemsets.
8. Repeat the same for the remaining frequent items.

4.2.1.2 Memory and CPU overhead Management

To reduce the delay in finding the occurrence count of each item in the first scan and the filtering delay in the frequent itemset generation, pre-processing is done to arrange the items in each transaction according to the order of item codes. Unlike Apriori, this approach avoids the searching of entire dataset to obtain the occurrence count of each itemset and thus reduces the CPU overhead and the number of database accesses. To optimize the memory utilization, co-occurring frequent itemsets of each frequent item is obtained by collecting all transactions containing these frequent items separately in to memory buffer. Further four level filtering is done to reduce the size of the selected transactions.

It is observed that as the count of frequent item increases, the number of transactions collected

Journal of Theoretical and Applied Information Technology
10th May 2014. Vol. 63 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

together to identify the frequent itemset also increases. At the same time, the number of items in each transaction gets reduced due to filtering approach. It also shows that rather than searching in the entire database it needs to search only in the buffer for obtaining higher frequent itemsets. To reduce the searching delay in incremental and interactive mining, nearly frequent itemsets count is not getting updated on every new transaction set addition. These are updated only if they are found to be frequent in the newly added partitions and if the user wishes to interactively reduce the preset minimum support value (***Sh***); here the partition number indicates the last partition that the itemset had counted before and only the remaining partitions are to be searched to obtain its support count.

4.2.2 Incremental Mining

This method accommodates the newly arrived transactions and updates the frequent patterns with less computational delay. To reduce the computational cost, it is suggested to include transactions at the end of the day or store the newly arrived transactions separately in a temporary partition and after a partition becomes full, add it to the existing partitions. Then find the count of each item in the newly added partition and update the count in the existing list. Also update the count of existing frequent itemsets. If any existing frequent item/itemsets is found to be infrequent, remove it from the ***Fset*** list. Then include it with the ***NFset*** list if it's support $\geq$ ***Sl*** along with the current partition number. Further find the count of its subset and if found to be frequent include it in the ***Fset***. Then find the new frequent itemsets having support $\geq$ ***Sh*** in the new partition, if any, then find its count in other partitions by searching it in the ***NFset*** list. If not obtained, conduct a possibility test using equation 4. If there is possibility to be frequent, search its count in the entire dataset. If found to be frequent include it in the ***Fset*** list, else if nearly frequent, include in ***NFset*** list else discard it. At the end of day (or partition become full), find the frequent supersets of the newly generated frequent items if any, by collecting all the transactions containing the new item from the entire dataset.

$$S * Z + (Pc - Pi - 1)(U * Z - 1) + L * Pi * Z - 1 \geq Pc * U * Z \quad \text{..... (4)}$$

Where ***Pi*** – no. of partitions used for initial pattern generation, ***Pc*** – current partition no.
Z– Partition size, ***L*** - Lower minimum support,
U - Upper minimum support,
S - new ***Fset*** support in new partition

Table 2: Newly Added Partition

Tid	Partition4 Added
16	D E I J L N
17	E I J L N
18	D E J K M
19	D E J K M
20	N O

Table 2 shows the sample partition added to the existing database and the updated count of individual items are shown in table 3. In the given example, a new frequent item ***E*** is generated on addition of the new partition 4. Thus to find the frequent supersets of item ***E,*** collect all transactions containing item ***E*** and follow the same frequent itemset generation procedure. Also update the count of existing itemsets. Figure 6 shows the frequent itemset updating procedure of item ***I***.

Table 3: Updated Item Count

Item	P1-Count	P2-Count	P3-Count	P4-Count	Total
A	0	2	4	0	6
B	2	1	0	0	3
C	4	5	0	0	9
D	3	2	2	3	10
E	0	3	2	4	9
F	1	1	2	0	4
G	1	1	0	0	2
H	1	0	1	0	2
I	3	2	4	2	11
J	3	4	5	4	16
K	0	3	1	2	6
L	3	1	4	2	10
M	0	3	0	2	5
N	5	2	4	3	14

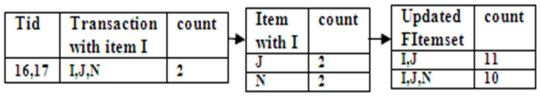

Figure 6: Frequent Itemset updating procedure on adding new partition (Item I)

4.2.2.1 IAPI Quad-filter Algorithm (Phase II) - Incremental Mining

1. Find each item count in the new partition, then update its total count and identify the frequent items.
2. If any new frequent item is generated, find its frequent supersets. Consider the existing frequent items and other new frequent items having count greater than the selected one as co-occurring items ***Cf***.
3. Include the co-occurring item list ***Cf*** of each new frequent item to the existing list. Also include the newly formed frequent and nearly frequent itemset to the existing list.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. Find the frequent itemsets in the new partition by considering only the items listed in ***Cf*** and update their total count. Also update the count of other existing ***Fset***s and if their support < ***Sh,*** shift them to the nearly frequent set list and record the current partition number.
5. If any new frequent itemset is obtained from the new partition, find its total count by referring to the ***NFset*** list.
6. If not found in the ***NFset*** list, conduct a possibility test using equation 4 and if there is a possibility to be frequent, scan the entire old transactions containing the selected frequent item and find its total count. If its support ≥ ***Sh,*** include it in the ***Fset*** list else if support ≥ ***Sl,*** keep it in ***NFset*** list along with the current partition number (***Pn***). If there is no possibility to be frequent, discard it.

4.2.3 Accommodation of Behavioural Changes (Phase III)

After a certain period, if the behaviour/purchase pattern changes, older transactions (partitions) are removed. Then update the existing frequent patterns. When the older partitions are removed, the count of each frequent itemsets in the partition, where the removal happened is obtained and their count is deducted from the frequent set list. Also count of each item in that partition which is deducted from the total count. After the partition removal the occurrence frequency of each item may change and due to the heterogeneity of the dataset, there is a chance of new frequent itemsets to occur with the existing frequent items. Thus a new co-occurring item list ***Cf*** is prepared and generates the new ***Fset*** and ***NFset*** lists using the same frequent itemset generation procedure [property 3.1]. The partitioning technique used in the proposed method helps to update the item count without rescanning the entire database.

1. Count of each item in the old partition which is removed is deducted from the total count and identifies the frequent items in the remaining dataset.
2. Find the ***Fset***s and ***NFset***s of each frequent item from the remaining partitions using the frequent pattern mining procedures stated above.

4.2.4 Interactive mining (Phase IV)

Interactive mining provides the user to interactively adjust minimum support value. Finding an appropriate support value for a dataset is a challenging task. It is better to provide the users with the facility to change the support value as per their requirements.

1. When user increase the preset ***Sh*** value, choose the itemsets with support ≥ new support as ***Fset*** from the existing frequent set and shift others to the ***NFset*** list and record their partition number (***Pn***) as last partition number.
2. When the user reduces the ***Sh*** value, select the itemsets having support ≥ new support from the existing ***NFset*** and include it along with the existing ***Fset*** list. If the partition number (***Pn***) of the ***NFset*** is less than the current partition number (***Pc***), then find their count in the remaining partitions (P_{n+1},P_{n+2},…***Pc***) to get the total count and identify the newly formed ***Fset***s.
3. Also choose the items having count greater than or equal to the newly set support count as frequent items and find their frequent supersets from the entire database.

5. EXPERIMENTAL SETUP AND PERFORMANCE ANALYSIS

Functionalities and effectiveness of the proposed algorithm IAPI Quad-filter were tested with market basket datasets T10I4D100K prepared by IBM Almaden Quest research group and a Synthetic dataset. This algorithm is developed and tested on Intel(R) core (TM) 2 Duo, 2.1 GHz CPU having 1.2 GHz, 3 GB RAM with Microsoft Windows XP using NetBeans IDE 7.0.0 and MySQL Server 4.1. Execution time and memory utilization are compared for various support threshold values with differently sized partitions as well as with different number of partitions in both datasets (Figure 7).

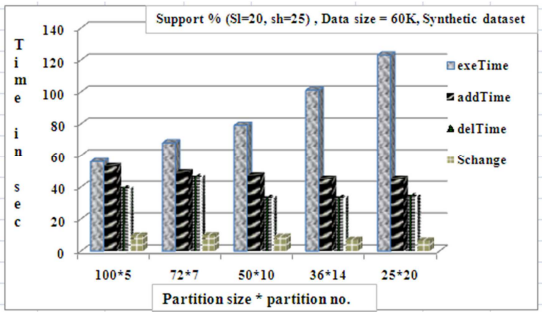

Figure 7: Time comparison of IAPI in initial pattern generation, addition and deletion, support change using synthetic dataset

Experimental results show that execution time increases, when the total size of the dataset increases with constant minimum support (Figure 8). Initial pattern creation time required is more if

Journal of Theoretical and Applied Information Technology
10th May 2014. Vol. 63 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the number of partitions is more, but at the same time new partition addition and deletion of old partitions require less time if the partition size is small with constant dataset size, i.e updating time depends on the size of the data added/deleted. Test results given in figure 8 shows that addition of 20% of data requires about 55% to 60% of the initial pattern creation time, 10% of data takes 25% to 30% of the initial pattern creation time. It is observed that, when the support threshold reduces, the number of frequent items increases, thus execution time required is more (Figure 9). Due to heterogeneity of dataset there are chances of reducing the number of frequent items, even though the dataset size increases. The test results illustrate that the execution time of IAPI Quad-filter directly depends on the number of frequent items in the dataset. Graphical representations of the test results of IAPI are shown in Figures 7, 8 and 9.

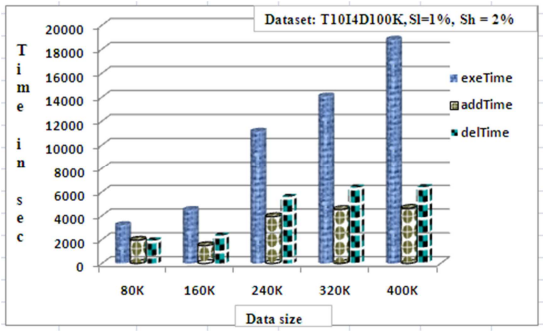

Figure 8: Frequent pattern generation time comparisons of IAPI with T10I4D100K

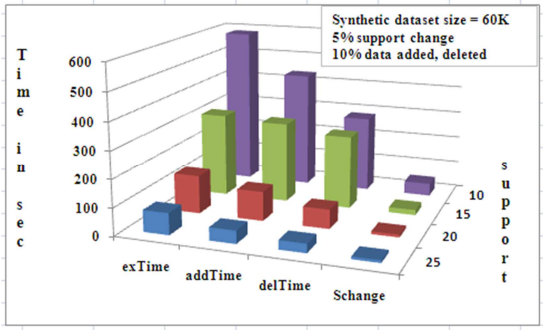

Figure 9: Frequent pattern generation time of IAPI with different support values

5.1 Performance Comparison

Performance of IAPI Quad-Filter algorithm is compared with three popular algorithms: Partition Algorithm, DIC and CanTree using T10I4D100K dataset. Speed of execution of IAPI Quad-filter is ten times faster than Partition algorithm and twenty times faster than DIC. The memory requirement of IAPI Quad-filter is around five to six times lesser than Partition algorithm and twenty to thirty times less compared to DIC (Figure 10 and 11). Though partition algorithm is designed for very large dataset due to large number of independent local frequent itemsets generated for dense datasets, it requires more memory; thus there is limitation in dataset size. Since IAPI Quad-filter is not generating any local frequent sets, it is suitable for both dense and sparse datasets. A large number of frequent sets require more computations; thus execution time required is more in partition algorithm due to the occurrence of large number of local frequent itemsets. Partition algorithm creates static patterns and for incrementally updating the frequent patterns, it is required to keep all local frequent itemsets in memory, also if the user wishes to change the support value, the rescanning of the entire database is required for updating the frequent sets.

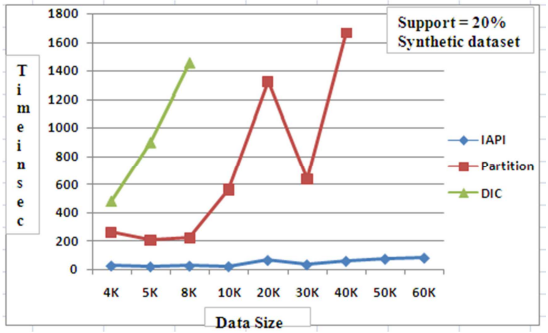

Figure 10: Execution time comparisons of IAPI, Partition and DIC with different data size

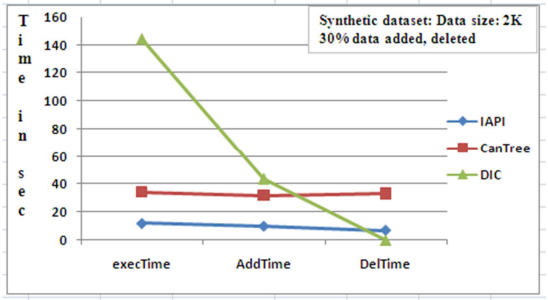

Figure 11 Time comparison of IAPI, CanTree and DIC in initial pattern creation, addition and deletion of partitions using synthetic dataset

DIC is developed for dynamic frequent set generation, thus it requires four to ten times less time for incrementally updating the database, compared with the initial frequent set generation. It is observed that, in DIC, removal of old transaction requires five to ten times less time compared with IAPI. But the main difficulty with DIC is that it has to keep the count of all possible subsets in the entire transaction set; thus it consumes more memory and hence not suitable for very large data sets. Subset generation introduces more computational cost and requires 10 to 20 times

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

more time than IAPI for the initial frequent set generation. CanTree is suitable for both incremental and interactive mining. It keeps the entire transactions in the CanTree for preparing frequent itemsets; thus it requires two to three times more memory than IAPI. CanTree requires re-construction of FP tree for each frequent item for every addition, deletion as well as the support change cases, which may lead to more computational delay.

6. CONCLUSION

Most of the real life databases are very large and dynamic. Thus it is essential to have an approach which has the flexibility to handle larger dynamic dataset efficiently and effectively. This study proposes an incremental and interactive mining algorithm with partitioning approach, designed to obtain frequent patterns from a very large sized dataset without generating local frequent itemsets. Most of the frequent pattern mining algorithms have complex structures and requires more database scans. This approach combines the features of various algorithms such as Apriori, CanTree, CARMA and Partitioning algorithm to obtain frequent itemsets with no complex calculations or data structures using small memory space in short time duration. Unlike Apriori, IAPI Quad-filter avoids the searching of entire dataset to obtain the occurrence count of each itemsets. The size and the number of transaction to be compared are further reduced with higher itemsets, due to four level filtering approaches. Thus 50% to 60% of memory space reduction and 30% to 40% of data comparisons reduction are achieved by this approach. This approach uses two bounds (***low***, ***high***) for minimum support and generates two category itemsets: frequent itemsets having support greater than ***high*** minimum support value and nearly frequent itemsets having support in between ***low*** and ***high***. Partitioning approach avoids the first database scan while deleting the old partitions and the nearly frequent itemsets help to reduce search delay for updating the frequent itemsets in incremental mining and interactive mining. The most attractive feature of IAPI is that this can be applied on both sequential and parallel mining approaches. Thus the proposed method can prepare more accurate user spending profile in short time period with less space and computational complexity.

REFERENCES

[1] R. Agrawal, T. Imielienski, and A. Swami, 'Mining Association Rules between Sets of Items in Large Databases'. *Proceedings of the Management of Data Conference*, ACM Press, New York, NY, USA, 1993, pp. 207–216.

[2] R. Agrawal, R. Srikant, 'Fast algorithms for mining generalized association rules', *Proceedings of the 20th International Conference on Very Large Databases*, 1994, pp. 487–499.

[3] David Wai-Lok Cheung, Vincent T. Y. Ng, Ada Wai-Chee Fu, Yongjian Fu. 'Efficient Mining of Association Rules in Distributed Databases', *IEEE Trans. Knowl. Data Eng.* Vol. 8 No. 6, pp. 911-922.

[4] M.J.Zaki. (2000), 'Scalable Algorithms for Association Mining', *IEEE Transactions on Knowledge and Data Engineering, 1996,* Vol. 12, pp. 372-390.

[5] C.Borgelt, 'Efficient Implementations of Apriori and Eclat', *Proceedings of 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations. CEUR Workshop Proceedings 90,* Aachen, Germany. *2003.*

[6] J. Han, J. Pei, and Y. Yin. (2000), 'Mining frequent patterns without candidate generation', *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 1–12.

[7] M. El-Hajj and O. R. Zaiane, 'COFI approach for mining frequent itemsets revisited', *Proceedings of the 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, New York, 2004, pp. 70–75.

[8] Chuang-Cheng Chiu and Chieh-Yuan Tsai. 'A Web Services-Based Collaborative Scheme for Credit Card Fraud Detection', *Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04),* pp. 177-181,28-31 march 2004.

[9] Qian Wan and Aijun An. (2005), 'Compact Transaction Database for Efficient Frequent Pattern Mining', *IEEE international conference on Granular Computing*, Vol. 2, pp. 652-659.

[10] Ashok Savasere Edward Omiecinski Shamkant Navathe. (1995), 'An Efficient Algorithm for Mining Association Rules in Large Databases', *Proceedings of the 21st VLDB Conference*, pp. 432−444.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[11] Sergey Brin, Rajeev Motwani, Jeffrey D Ullman and Shallom Tsur. (1997), ' Dynamic itemset counting and implication rules for market basket data', *Proceedings of ACM SIGMOD International conference on Management of data*, Vol. 26, No. 2 of SIGMOD record, pp. 255-264.

[12] C. Hidber. (1999), "Online association rule mining", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 145–156.

[13] Leung C.K., Khan, Quamrul I., Li Zhan., Hoque, T. (2007), 'CanTree: A Canonical-Order Tree for Incremental Frequent-Pattern Mining', *Knowledge and Information Systems*, Vol. 11, No. 3, pp. 287–311.

[14] Jianyun Xu, Andrew H. Sung and Qingzhong Liu., 'Behaviour Mining for Fraud Detection', *Journal of Research and Practice in Information Technology*, Vol. 39, No. 1, p.p 3-18, February 2007.

[15] S. K Tanbeer, C. F Ahmed, Byeong-Soo Jeong, Young-Koo Lee. (2008), 'Efficient single-pass frequent pattern mining using a prefix-tree', *Information Science*, Vol. 179, pp. 559–583.

[16] Mohadeseh Hamedanian, Mohammad Nadimi, Mohammad Naderi., ' An Efficient Prefix Tree for Incremental Frequent Pattern Mining', International Journal of Information and Communication Technology Research, Vol. 3 No. 2, pp. 49-55, February 2013.