Efficient Adaptive Frequent Pattern Mining Techniques for Market Analysis in Sequential and Parallel Systems

Sherly Kuriakose[1] and Raju Nedunchezhian[2]
[1]Department of Information Technology, Rajagiri School of Engineering and Technology, India
[2]Department of Computer Science and Engineering, Coimbatore Institute of Technology, India

Abstract: *The classical applications of Association Rule Mining (ARM) are market analysis, network traffic analysis, and web log analysis where strategic decisions are made by analyzing the frequent itemsets from a large pool of data. Datasets in such domains are constantly updated and as they require an efficient Frequent Pattern Mining (FPM) algorithm which is capable of extracting the required information. Several incremental algorithms have been proposed to generate frequent patterns, but they are ineffective with very large datasets and do not provide the user interaction to adjust the minimum support value. This paper first presents an efficient interactive sequential FPM algorithm that uses the knowledge gained in the previous mining steps to incrementally mine the updated database with fewer complexities. Then to further reduce the time complexity it proposes an efficient interactive and incremental parallel mining algorithm. It also prepares incremental frequent patterns, without generating local frequent itemsets with less communication and synchronization overheads.*

Keywords: *Association rule, frequent pattern mining, interactive mining, incremental mining, parallel mining.*

Received June 30, 2014; accepted August 31, 2014

1. Introduction

Evolution of technology and globalization create the subsequent acceleration of information flow. Thus, extraction of knowledge from the large pool of information is becoming a very difficult task. The rapid advancement in electronic commerce increases online transactions every year. Organizations store their ever-increasing day-to-day transactional details in their transaction databases. Data mining prepares models by analyzing the hidden relationships among stored data and deals with the problems that arise with large data repositories.

The classical application of Association Rule Mining (ARM) is market-basket analysis that has been used to predict customer purchasing/spending behavior by analyzing the frequent itemsets in a large pool of transactions. Frequent patterns are a set of all subsets of items that frequently appear together in a dataset. Frequent Pattern Mining (FPM) plays a key role to obtain associations and correlations among items in a large transactional dataset [2]. As the amount of transactions increases it becomes very difficult to determine the frequent patterns with less time and space complexities. Scalability is one of the main requirements of an FPM algorithm. Some of the algorithms address the space complexity problem of very large database using partitioned database approach [14]. Partitioned algorithms generate all possible large itemsets from each partition in a sequential manner in the first scan, which may contain false positives (globally infrequent). During the second scan they remove the false positives and the global frequent sets are generated. The occurrence of false positives may lead to space complexity and time complexity for very large database. Thus, sequential algorithms [2, 7, 12] can provide scalability and very good performance up to a certain database size limit. Hence, parallel mining approaches [1, 5, 6] are required to provide scalability in massive data stores in an efficient manner.

As the day to day transaction details get added to the transactional database, database becomes dynamic and incremental updating of frequent patterns is required. Also there is possibility for change in the customer's purchase behavior due to the change in life style as well as addition of new customers. Thus to reflect the current status of database, old patterns must be removed and new patterns might appear. Many of the FPM algorithms [2, 7, 12, 14] prepare static patterns and use them for long term predictions; but those may not be capable to accommodate the behavioral changes in the incremental database. Thus dynamic algorithms [3, 4, 8, 9] that are capable of incremental and interactive mining with less computational cost are essential for an incremental database. The essential key feature of an incremental algorithm is to reuse the previously mined information and combine this information with the new data to incrementally update the frequent itemsets without rescanning the entire database.

The objective of this research is to develop FPM algorithms which are efficient, scalable, faster and dynamic to support behavioral changes using both sequential and parallel approaches. Sherly [15] proposes Interactive and Adaptive Partitioned Incremental (IAPI) FPM (IAPI Quad-Filter) algorithm for incremental FPM in large databases to solve the space and computational complexity. But it requires more than two database scan (equivalent to the number of frequent items), thus the data fetching time is fairly high. This paper proposes two incremental FPM algorithms (modified IAPI versions) capable of generating large itemsets from an incremental massive data store in two database scans and incrementally update the frequent itemsets without rescanning the entire database using sequential and parallel approaches with a system having fairly good storage and computing capability.

1.1. Overview of the Proposed Algorithms

IAPI type of algorithms use a database partitioning approach to produce frequent itemsets without generating local frequent itemsets. In these approaches, 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 logically divides the dataset into small sized non-overlapping horizontal partitions of user specified sizes so that each partition can be accommodated in the main memory. To reduce the computational cost, I/O overhead as well as space complexity, each Frequent Item (FI) transaction group is collected separately and four level filtering is done to remove infrequent items [15].

Unlike Apriori [2] in IAPI, the number of transaction to be compared and their length both get reduced in finding higher frequent itemsets. This method is capable to incrementally update the database to accommodate the customer behavioural changes. IAPI also provides the user with the facility to interactively adjust the minimum support value as per one's own convenience. To find the higher frequent itemsets, IAPI Quad-filter collects each FI transaction groups separately one after the other, thus the number of database scans required is the number of frequent 1-itemsets, which is fairly high. Thus this approach is best suited with systems having low memory capacity. To improve the performance, this paper proposes another algorithm Faster-IAPI, which generates frequent patterns in two database scans only. It collects all the FI transaction groups simultaneously in one database scan; but the down side is that it requires more memory (multiple memory buffers) to hold the different FI transaction groups. Then the higher frequent itemsets of each transaction groups are obtained sequentially.

Speed of operation in very large dataset can be further improved using parallel mining approach. Thus a second algorithm Parallel-IAPI is suggested for shared memory multi processor systems. This method finds the higher frequent itemsets of all frequent 1-itemsets simultaneously using parallel processors. In this approach each Local Processor (LP) finds the local partition count of each item and sends it to Master Processor (MP) to obtain their global count. MP identifies the frequent 1-itemsets and sends the count of all frequent items to each LP and assigns them to obtain higher frequent itemsets of each item. LP collect the transaction groups of assigned FI with their co-occurring items (items having count greater than it) and find their higher frequent itemsets concurrently using the IAPI approach. The main attraction of this approach is that each LP work independently for higher frequent itemset generation; thus there is very less communication overhead. Figures 1 and 2 describes the frequent itemset generation procedures of Faster-IAPI and Parallel-IAPI respectively.

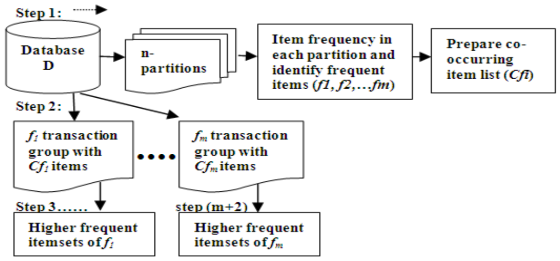

Step 1:
Database D
n-partitions
Item frequency in each partition and identify frequent items (f1, f2,…fm)
Prepare co-occurring item list (Cfi)
Step 2:
f1 transaction group with Cf1 items
fm transaction group with Cfm items
Step 3……
step (m+2)
Higher frequent itemsets of f1
Higher frequent itemsets of fm

Figure 1. Functional block diagram of Faster-IAPI.

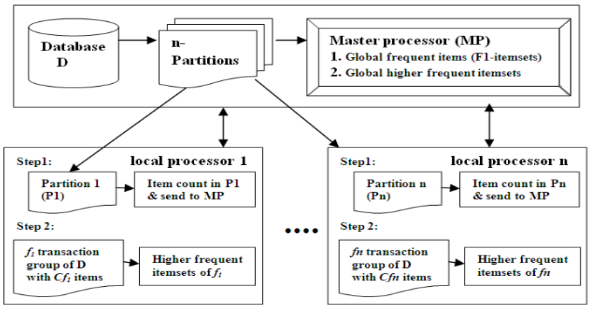

Database D
n-Partitions
Master processor (MP)
1. Global frequent items (F1-itemsets)
2. Global higher frequent itemsets
Step1:
local processor 1
Partition 1 (P1)
Item count in P1 & send to MP
Step 2:
f1 transaction group of D with Cf1 items
Higher frequent itemsets of f1
Step1:
local processor n
Partition n (Pn)
Item count in Pn & send to MP
Step 2:
fn transaction group of D with Cfn items
Higher frequent itemsets of fn

Figure 2. Functional block diagram of parallel-IAPI.

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

2. Related Works

The popular algorithm Apriori [2] forms the foundation for static frequent pattern mining. The major problem of Apriori is that it has to read the entire database in every pass, although many items and transactions are no longer needed in later passes. It generates candidate itemsets iteratively, which makes the computational cost very high. Instead of using generate and test paradigm of Apriori, FP-tree approaches [7, 8, 10] encode the dataset using a compact tree structure and directly extracts the frequent itemsets from this structure. Thus tree approaches outperform Apriori-like approaches by generating frequent patterns without producing candidate sets. But it has to generate conditional pattern bases and sub-conditional pattern tree recursively.

To obtain frequent sets from very large datasets with low memory utilization, [14] suggests a partitioning algorithm which generates frequent itemsets in two database scans. During the first scan, it identifies the local frequent item-sets from each partition and in the second scan it estimates the global frequent sets. This algorithm is highly dependent on the heterogeneity of the database and may generate too many independent local frequent itemsets. To analyze the problem of market basket data, [4] presents an algorithm DIC which uses fewer passes over the data than classical algorithms to find the frequent itemsets. It provides the flexibility to add and delete counted itemsets on the fly. Downside of DIC is that it is sensitive to the homogeneity of data.

An interactive mining algorithm, Continuous Association Rule Mining Algorithm (CARMA) [9] requires two database scans to produce large itemsets. CARMA provides a lower and upper bound for its support for each set. Thus, the user can interactively adjust the support and confidence at any time. A dynamic algorithm CanTree [10] 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. A novel tree structure called CP-tree [16], which creates incremental frequent patterns with the support of interactive mining in one database scan. First phase inserts transactions into CP-tree and second phase 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. To reduce the time of restructuring a new prefix tree structure proposed in [8]. An Incremental Mining Binary Tree (IMBT) algorithm is presented in [19] in which each node of the tree represents one of all the possible combinations of items in the entire dataset. It processes a transaction at a time and record the possible itemsets in the respective nodes, thus reduces the processing and I/O time but requires more memory to keep all combinations of items in the database. To reduce the search space and model size in evolving database, YAMI (YAMI is derived from the names of the Authors) [18] a dynamic ARM algorithm is developed. It uses a shocking interestingness measure as a constraint to discover rules that are interesting for the user.

A potential solution for improving the performance and scalability in FPM from very large database is to parallelize the mining algorithms. An algorithm Parallel Data Mining (PDM) is proposed [13] for parallel mining which is an adaptation of the Direct Hashing and Pruning (DHP) algorithm [12] in the distributed environment. In PDM each node computes the globally large itemsets by exchanging the support counts of the candidate sets. Downside of this is that $O(n^2)$ messages are required for support count exchange among n nodes for each candidate set. Another algorithm Count Distribution (CD) [1], which is an adaptation of the Apriori algorithm, is proposed for the same parallel mining environment. This algorithm also has the similar problems. A tree-partition algorithm for parallel mining of frequent patterns on shared-memory structures is presented in [5]. It builds one FP-Tree of the entire database, then partitions it into several independent parts and distributes them to different threads. This approach uses a Master/Slave Model. The parallel implementation of Apriori algorithm based on MapReduce framework [11] is suggested for processing huge datasets using a large number of computers. But these parallel algorithms are not suitable for incremental database. An incremental, interactive and parallel mining technique for shared memory multiprocessor system is designed in [17] for incremental mining. This approach is based on the adaptive tidlist interval distribution technique, which continuously assigns partitions of the tidlist among the different processors. A parallel IMBT structure is proposed in [3] to enumerate the support count of each itemset in an efficient way after the new transactions are added or deleted.

3. Proposed Algorithms

3.1. Problem Definition

Let D be a database with N number of transactions. Let I be the item domain, $\{I_1, I_2, ..., I_q\}$. The problem is to identify all interesting frequent patterns in an interactive and incremental manner with fewer complexities. 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 n 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)≥Sh and Nearly frequent if $Sl \leq$ support (X)≤ Sh. Pn represents a partition number at which a *NFset* has been last updated. Let $F_{1\text{-}itemset}$ be the frequent 1-item domain, $F_{1\text{-}itemset}=\{f_1, f_2, ..., f_m\}$in the ascending order of occurrence count. Each FI is associated with a co-occurring item set list Cf, $\{Cf_1, Cf_2, ..., Cf_{m-1}\}$ refers to the list of items to be considered along with each FI to find the higher frequent itemsets, where as Cf_1 be the co-occurring item list of FI f_1 i.e., $Cf_{1=}\{f_2, f_3,..., f_m\}$. In general: $Cf_i = \{\{f_{i+1}, f_{i+2}, ..., f_m\} | \text{frequency}(f_{i+1}, f_{i+2},\dots, f_m) \geq \text{frequency}(f_i)\}$ i.e., $Cf_1 \supset Cf_2 \supset, ..., \supset Cf_{m-1,}$ it indicates that as the frequency of occurrence is more the number of co-occurring items considered for frequent itemset mining get reduced.

3.2. Faster-IAPI Algorithm

This algorithm has all the features of IAPI Quad-filter and 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 patterns to accommodate the behavioural changes. The last phase provides the facility to interactively adjust minimum support value as per the user's requirement. In the first scan, Faster IAPI adopts the same method of IAPI Quad-filter and generates frequent 1-itemsets. To generate higher frequent itemsets, do the second scan of database and collect only frequent items from each transaction. Then, form separate transaction set groups of each FI and keep only the corresponding co-occurring items of each transaction group in separate buffers as shown in Figure 3. Also eliminate the similar transaction entry in each transaction group (compress the transaction set) by recording the occurrence count, then find the frequent items in the selected group and eliminate others. Higher frequent itemsets are obtained from each buffer sequentially using IAPI Quad-filter approach as shown in Figure 3. Frequent itemset generation steps of Faster-IAPI algorithm are given below.

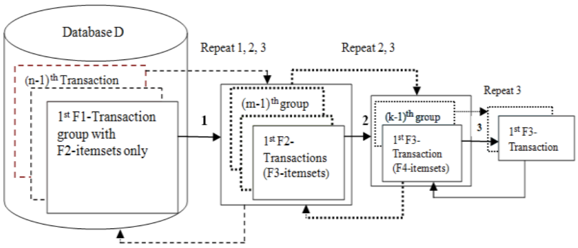

Database D
(n-1)th Transaction
1st F1-Transaction group with F2-itemsets only
Repeat 1, 2, 3
Repeat 2, 3
Repeat 3
1
2
3
(m-1)th group
1st F2-Transactions (F3-itemsets)
(k-1)th group
1st F3-Transaction (F4-itemsets)
1st F3-Transaction

Figure 3. Frequent set generation.

3.2.1. Faster-IAPI Frequent Item Set Generation Steps (Phase 1)

Input:

- D: Transaction database contain N transactions (T_1, T_2, …, T_N), horizontally partition D into n non-overlapping partitions (P_1, P_2, …, P_n) and sort the items of each transaction in the order of item code.
- Sl: Low *minimum support* value
- Sh: User *selected* min. support ($Sh > Sl$)

Output:

Complete set of frequent itemsets

```
1. For each partition do
   Read each transaction and find frequency $flocal(i)$
      for each item $i$;
2. Identify frequent 1-itemsets $F_{\mathbf{1\text{-}itemset}} = \{i \mid \sum flocal(i) \geq Sh$ for each item $i\}$
3. Sort $F_{1\text{-}itemset}$ in ascending order $F_{1\text{-}sorted} = \{f_1, f_2, \dots, f_m\}$
4. Prepare co-occurring item set list $Cf_i = \{\{f_{i+1}, f_{i+2}, \dots, f_m\}|\text{frequency}(_{i+1}, f_{i+2}, \dots, f_m) \geq \text{frequency}(f_i)\}$ for each $f_i$
5. Read each transaction of $D$ and do
Collect transactions contain each $f_i$ in to separate
   buffers and remove items that are not in the $Cf_i$ list
   from each transaction.
6. For each $f_i$-transaction group:
      • Find frequency of each $Cf_i$ item in the selected $f_i$-transaction group
      • If frequency($Cf_i$ ) $\geq Sl$
      • $F_2$-itemset= $\{Cf_i\}$ for each $Cf_i$ item
      • Else remove $Cf_i$ from the selected buffer
      • Sort $F_2$-itemset in ascending order
      • To obtain higher frequent itemsets of $F_2$
        for each item in $F_{2\text{-}itemset}$ do
        $F_{itemset}$ =Higher-frequentItemset-Generate ($f_i$-transactions(Buffer$_1$), $F_{2\text{-}itemset}$ );
        If support($F_{itemset}$) $\geq Sh$ then
          $Fset = F_{itemset}$
          Else $NFset = F_{itemset}$
```

Procedure Higher-frequentItemset-Generate (f_i-transactions (Buffer$_1$), Fn) //Fn_p:p^{th} item of $F_{n\text{-}itemset}$:

```
1. Collect $fi$-transactions contain $Fn_p$ to a new temporary buffer$_n$ and remove items having count $\leq Fn_p$ from each transaction in the buffer$_n$, $p$ initialized to 0
2. Find frequency of each item in the selected $Fn_p$ transaction group
3. $F_{(n+1)\text{-}itemset} = \{Fn_{(p+k)}|\text{frequency}(Fn_{(p+k)}) \geq Sl\}$for each $Fn_{(p+k)}$ item where $k$=1 to ($m$-$p$)
4. else remove $Fn_{(p+k)}$ from the $Fn_p$ transactions
5. Sort $F_{(n+1)\text{-}itemset}$ in ascending order
6. To obtain higher frequent itemsets of $f_i$ do
7. if ($F_{(n+1)\text{-}itemset} \neq \Phi$) then
```

```
    $n$=$n$+1 & Repeat above steps
8.  else if $p$< size$F_{n\text{-itemset}}$) then
        $p$=$p$+1 & repeat above steps
9.  else remove buffer$_n$ content $n$=$n$-1, $p$=1
10. if $n \geq 2$ repeat above steps
11. else return $F_{itemset}$
```

3.2.2. Working of Faster IAPI

The working principle of algorithm is illustrated with a sample dataset having 10 transactions with 5 distinct items {*p*, *q*, *r*, *s*, *t*} shown in Table 1. It is divided into two horizontal partitions of size 5 transactions in each. Scan each transaction and count of each item in both partitions is recorded separately, which is given in Table 2. Then, calculate total count of each item and identify the frequent items (items with count greater than or equal to minimum support i.e., 50% of total transaction count). Next step is to find the higher frequent itemsets of each frequent item. This approach reduces the database scan and the computational complexity by grouping transaction containing each FI in to separate buffers as shown in Table 4 during the second database scan. To reduce the space complexity while forming the frequent transaction group, each group needs to consider only the items that are having count greater than its own count which is shown as co-occurring items list in Table 3. Higher frequent itemset generation steps of frequent items *q* and *p* are illustrated in Figures 4 and 5 respectively. To facilitate incremental mining this algorithm uses two minimum support values *Sh* and *Sl* and considers itemsets having support greater than *Sh* (50%) as frequent itemsets (*FSets*) and *Sl* (30%) as nearly frequent itemsets (*NFsets*). *Fsets* and *NFsets* obtained from each transaction group are recorded separately into Tables 5 and 6 respectively.

Table 1. Dataset.

Tid	Transactions	partition
1	r, s, t	P1
2	p, q, s, t	
3	r, t	
4	p, s, t	
5	p, r, s, t	
6	p, q, r, s	P2
7	r, s	
8	p, r, s	
9	p, r, s, t	
10	q, s, t	

Table 2. Item Count.

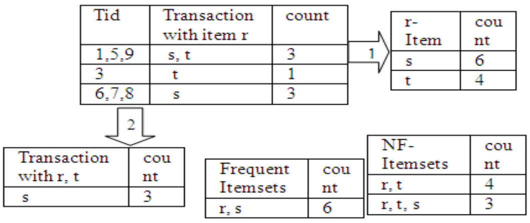
Item	P1	P2
p	3	3
q	1	2
r	3	4
s	4	5
t	5	2

Table 3. Co-items list.

Item	Co-Items
p	r, s, t
r	s, t
t	s

Table 4. Buffers.

Tid	Transactions	count
Buffer p		
2, 4	s, t	2
5,9	r, s, t	2
6,8	r, s	2
Buffer r		
1,5,9	s, t	3
3	t	1
6,7,8	**s**	3
Buffer t		
1,2,4,5,9,10	s	6



Tid	Transaction with item r	count
1,5,9	s, t	3
3	t	1
6,7,8	s	3

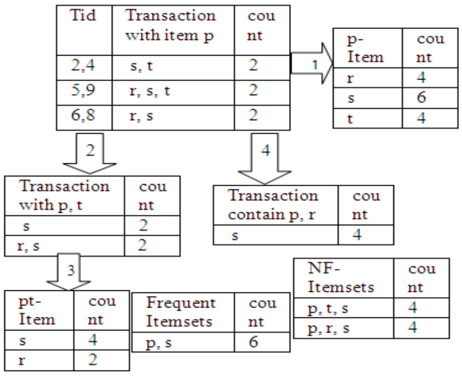
r-Item	count
s	6
t	4

Transaction with r, t	count
s	3

Frequent Itemsets	count
r, s	6

NF-Itemsets	count
r, t	4
r, t, s	3

Figure 4. Buffer r frequent set generation steps.



Tid	Transaction with item p	count
2,4	s, t	2
5,9	r, s, t	2
6,8	r, s	2

p-Item	count
r	4
s	6
t	4

Transaction with p, t	count
s	2
r, s	2

Transaction contain p, r	count
s	4

pt-Item	count
s	4
r	2

Frequent Itemsets	count
p, s	6

NF-Itemsets	count
p, t, s	4
p, r, s	4

Figure 5. Buffer p frequent set generation steps.

Table 5. Frequent itemsets.

F-itemset	count
p, s	6
r, s	6
t, s	6

Table 6. Nearly frequentsets.

NF-itemset	count
p, t	4
r, t	4
p, r, s	4
p, t, s	4
r, t, s	3

3.2.3. Incremental Mining (Phase 2)

This algorithm is capable to accommodate newly arrived transactions and update the existing frequent itemsets without rescanning the entire datasets by utilizing *NFsets*. Incremental mining process is illustrated using the example given in Tables 7, 8, 9, 10 and 11. The newly added transaction set is recorded in Table 7 and count of items in the new partition is calculated and added with the existing count to update the frequent items, Table 8. FI transaction groups of the newly added partition are collected to separate buffers Table 9 and find their higher frequent itemsets in the new partition. Then update the count of the existing *Fset* belongs to each group Table 10. If any existing frequent item/itemsets is found to be infrequent, remove it from the *Fset* list and include it with the *NFset* list if it's support≥*Sl* along with the current partition number Table 11. If any new frequent itemset obtained from new partition, obtain its global count using *NFset.* If not obtained from *NFset,* then conduct a possibility test using Equation 1 and if possible to be frequent find their global count from the old transaction set.

Table 7. New transactions.

Tid	Transactions
11	r, s, t
12	q, s
13	p, q, r, t
14	q, s, t
15	p, r, s, t

Table 8. Updated item count.

Item	P1	P2	P3	total
p	3	3	2	8
q	1	2	3	6
r	3	4	3	10
s	4	5	4	13
t	5	2	4	11

Table 9. New buffers.

Tid	Transactions
Buffer p	
13	r, t
15	r, s, t
Buffer r	
11	s, t
13	t
15	s, t
Buffer t	
11	s
14	s
15	s

Table 10. Updated frequent sets.

Updated Fitemset	New count
r, s	8
t, s	9

Table 11. Updated NFsets.

NF-itemset	New count
p, t, s	5
r, t, s	5
p, r, s	5
p, r	6
p, s	7
p, t	6
r, t	7

$$S*Z+(Pc-Pi-1)(U*Z-1)+L*Pi*Z-1 \geq Pc*U*Z \qquad (1)$$

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.

3.2.4. Accommodation of Behavioural Changes (Phase 3)

After a certain period, the behaviour/purchase patterns may change. Thus to reflect the behavior change older transactions (partitions) must be removed as shown in Table 12. 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. To update the frequent itemsets first update the count of each item by deducting the count of each item in that partition from the total count as shown in Table 13. If any existing FI becomes infrequent, remove it from the frequent itemset and if any new FI occurred, find its higher frequent itemsets by rescanning the remaining partitions. Updated item count recorded in Table 13 shows that a new FI *q* is generated. Now update the co-item list as shown in Table 14 and find the higher frequent itemsets of the newly formed FI *q* by collecting transactions containing item *q* from the entire dataset into buffer *q* as shown in Table 15. Next to update the count of *Fset* and *NFset,* find their count in the removed partition buffers as shown in Table 17 and deduct them from the previous count. Finally update the frequent and nearly frequent itemset list based on the updated count as shown in Tables 16 and 18.

Table 12. Updated dataset.

Tid	Transactions	Partition
6	p, q, r, s	P2
7	r, s	
8	p, r, s	
9	p, r, s, t	
10	q, s, t	
11	r, s, t	P3
12	q, s	
13	p, q, r, t	
14	q, s, t	
15	p, r, s, t	

Table 13. Updated Item count.

Item	P2	P3	total
p	3	2	5
q	2	3	5
r	4	3	7
s	5	4	9
t	2	4	6

Table 14. Updated co-item list.

Item	Co-Items
p	r, s, t
r	s, t
t	s
q	p, r, s, t

Table 15. New buffer q.

Tid	Transactions with item q
6	p, q, r, s
10,14	q, s, t
12	q, s
13	p, q, r, t

Table 16. Updated fset.

Updated Fitemset	New count
r, s	6
t, s	5
p, r	5

Table 17. Removed Buffers.

Tid	Transactions
Buffer p	
2,4	t, s
5	r, t, s
Buffer r	
1,5	t, s
3	t
Buffer t	
1,2,4,5	s

Table 18.Updated NF-sets.

NF-itemset	New count
p, t, r	3
r, t, s	3
p, r, s	4
p, s	4
p, t	3
r, t	4
q, s	3

3.2.5. Interactive Mining (Phase 4)

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. Interactive mining provides the user to interactively adjust minimum support value.

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.
3. If the Partition number (*Pn*) of any of itemset in 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 Fsets.
4. 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

3.3. Parallel IAPI Algorithm

Parallel IAPI algorithm generates higher frequent itemsets of n FI in parallel manner using multiple processors. Thus, time requirement for higher frequent itemset generation get reduced to 1/nth of frequent itemset generation time of Faster IAPI. In this approach count of each item in each partition are obtained by n LP simultaneously and send them to the MP. Then MP calculates their global count and identifies the frequent items (support≥*Sh*). Also prepare a co-occurring item list, *Cf* for each frequent item. For higher frequent itemset generation MP assigns separate LPs for each frequent item, and then MP rescans the database and broadcasts each transaction to all LPs. Further there is no communication required between LP and MP. LP collect the transactions containing the assigned FI and select only the co-occurring items of the assigned fi from each transaction for higher frequent itemset generation using IAPI approach. Parallel IAPI requires no communication among LPs, thus very less communication overhead. Parallel IAPI Algorithm steps of each phase are given in following sections.

3.3.1. Working of Parallel IAPI

The working of parallel IAPI can be illustrated using the same sample dataset given in Table 1. Consider that there are three processors *Pr*1 and *Pr*2 and *Pr*3 and *Pr*1 and *Pr*2 are considered as *LP* and *Pr*3 as *MP*. In Phase 1 during the first database scan *Pr*1 and *Pr*2 calculate the item count concurrently and submit to Pr3 to find global frequent itemsets as shown in Table 2. Then, *Pr*3 generates co-occurring item list of each item as shown in Table 3 and assign *Pr*1 and *Pr*2 to find the higher itemsets of frequent items *p* and *r*. *Pr*3 scans database second time and send each transaction to both the processors for the buffer storage as shown in Table 4. Then, higher frequent itemsets of items *r* and *p* are generated in parallel as shown in Figures 4 and 5. Item *t* has only one co-item, thus its count can be directly obtained from buffer *t*. All *Fset* and *NFset* generated are consolidated at processor Pr3 as shown in Tables 5 and 6).

In incremental mining (Phase 2) the count of each item in the newly added partition as shown in Table 7 is calculated at *Pr*3 and added with the previous count as shown in Table 8 to identify the present frequent items. FI buffers in the newly added partition as shown in Table 9 are generated and frequency of the higher frequent itemsets is obtained by *Pr*1 and *Pr*2. Higher frequent itemsets count in the new partition is added with the previous count by *Pr*3 to update the frequency of the existing *Fset* and *NFset* as shown in Tables 10 and 11. New patterns may get generated on adding new transactions made by the new customers as well due to the change in purchase behavior. Thus to reflect the pattern changes Phase 3 old transactions may be removed as shown in Table 12 and update the frequent patterns. Frequent pattern updating procedure is same as that used in Faster IAPI. Count of individual items in the removing partition is calculated by *Pr3*. Further the frequency of the existing frequent itemsets in the removed partition is obtained by the LP as shown in Table 17 and is deducted from the previous count by MP. Higher frequent itemsets of newly created frequent items (*F1*$_{new}$) is obtained by scanning the remaining partitions by *Pr*3 in coordination with *Pr*1 a *Pr* nd 2 as shown in Table 15 and update the *Fset* and *NFset* as shown in Tables 16 and 18.

3.3.2. Parallel IAPI Frequent Itemset Generation Steps (Phase 1)

Input:

- *D*: Transaction database contain *N* transactions (T_1, T_2, ... T_N), horizontally partition *D* into *n* non-overlapping partitions (P_1, P_2, ..., P_n) and sort the items of each transaction in the order of item code.
- *Sl:* Low *minimum* support value.
- *Sh:* User selected minimum support (*Sh* > *Sl*).

Output:

Complete set of frequent itemsets:

1. For each *LP* do
   Read local partition, find local
   frequency *flocal*(*i*) and send to *MP* for each item *i*;
2. In MP
   - $F_{1\text{-}itemset}=\{i \mid \sum flocal(i) \geq Sh \text{ for each } i\}$
   - Sort $F_{1\text{-}itemset}$ in ascending order $F_{1\text{-}sorted}=\{f_1, f_2, \ldots, f_m\}$.
   - Prepare co-occurring itemset list
     $Cf_i=\{f_{i+1}, f_{i+2}, \ldots, f_m\}$
     $|\text{count}(f_{i+1}, f_{i+2}, \ldots, f_m) \geq \text{count}(f_i)\}$ for each f_i and send to LP(i)
   - Read each transaction of *D* and send to *LP*.
3. For each LP do
   - Collect transactions contain FI f_i in to buffer_1 and remove items that are not in the Cf_i list from each transaction.
   - Find frequency of each Cf_i item in the selected f_i-transaction group
   - If frequency(Cf_i) ≥ *Sl*
     $F_{2\text{-}itemset}=\{Cf_i\}$ for each Cf_i item
   - Else remove Cf_i from the selected buffer.
   - Sort $F_{2\text{-}itemset}$ in ascending order.
   - To obtain higher frequent itemsets of f_i.
     $F_{itemset}$ = Higher-frequentItemset-Generate (f_i-transactions(Buffer_i), $F_{2\text{-}itemset}$);
   - Send $F_{itemset}$ to the MP.
4. In MP

   Fset = $F_{itemset}$ | (support ≥ *Sh*)
   Else *NFset* = $F_{itemset}$

3.3.3. Incremental Mining Steps (Phase 2)

1. In MP
   - Read each transaction in the new partition (P_{new}) and update the count of each item *i*.
   - $n = n+1$, update $F_{1\text{-}itemset}$.
   - If $F1_{new}$ then set $Cf_{new} \supset Cf_1$ and update existing *Cf*.
   - Collect transactions contain $F1_{new}$ item from all the partitions and find its higher frequent itemsets.
2. In each LP do
   - Read each transaction of P_{new} and follow the same procedure of higher frequent itemset generation.
   - Send all frequent itemsets to the MP.
3. In MP
   - Update existing *Fset* and *NFset*.
   - If any of the existing *Fset* is not updated rescan P_{new} and update it.
   - If any existing *Fset* become infrequent shift to *NFset* list, similarly any existing *NFset*s become frequent do vice versa.
   - If $Fset_{new}$ in P_{new}, conduct possibility test and if possible to be frequent find its global count by rescanning the previous partitions.

3.3.4. Accommodation of Behavioural Changes (Phase 3)

1. Remove partition P1 and update each item count *TCount*(*i*)=*Count*(*i*)–*P*1*Count*(*i*) for each *i* and identify the current frequent items;
2. If $F1_{new}$ then set $Cf_{new} \supset Cf_1$ and update existing *Cf*.
3. Collect transactions contain $F1_{new}$ item from all the remaining partitions and find its higher frequent itemsets.
4. Find the count of existing *Fset* and *NFset* in the removing partition and deduct it from existing count.
5. If any existing *Fset* become infrequent shift it to *NFset* list, similarly any existing *NFset*s become frequent do vice versa.

3.3.5. Interactive Mining (Phase 4)

Pn: Last updated partition number of *NFset*.
Pc: Current partition number.
Sh_{new}: Newly set minimum support by user.

1. If $Sh_{new} > Sh$

   For each *Fset*
   If Support(*Fset*)<Sh_{new} shift to *NFset*
2. If $Sh_{new} < Sh$
   - For each *NFset*
     If *Pn* < *Pc* then for P_{n+1} to *Pc* find the support(*NFset*)
     if support(*NFset*) ≥ Sh_{new} shift to *Fset*
   - For each *i* if count(*i*) ≥Sh_{new} shift to $F_{1\text{-}itemset}$ then

     a. If $F1_{new}$ then set $Cf_{new} \supset Cf_1$ and update existing *Cf*.

b. Collect transactions contain $F1_{new}$ item from all the partitions and find its higher frequent itemsets.

4. Experimental Results and Performance Analysis

Functionalities and effectiveness of the proposed IAPI algorithms 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 (i3), 3.2 GHz CPU having 4 GB RAM with Microsoft Windows 7 OS 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.

Experimental results show that execution time is directly proportional to the size of the dataset when the minimum support value remains constant as shown in Figure 6. Faster IAPI requires only two database scan for frequent itemset generation where as IAPI Quad-filter requires $(n+1)$ database scans if there are n frequent 1-itemsets. Thus initial pattern creation time of Faster IAPI is less compared with IAPI Quad filter. Updating of the frequent sets on addition of new data and deletion of old data requires less time compared with the initial pattern creation time. From the test results it is observed that updating time is related with the heterogeneity of the data, i.e., if new frequent 1-itemset generated, then it requires entire database scan, else previous information can be used and require less time (5%-30% of the initial pattern creation time) depends on the number of FI sets as shown in Figure 7. It is also observed that, when the support threshold reduces, the number of frequent items increases, thus execution time required is more. 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 and the memory requirement of both sequential and parallel IAPI directly depend on the number of frequent items in the dataset. Graphical representations of the test results of IAPI are shown in Figures 6, 7 and 8.

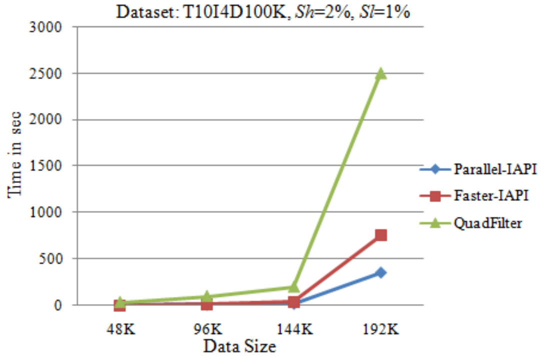


Figure 6. Execution time comparison.

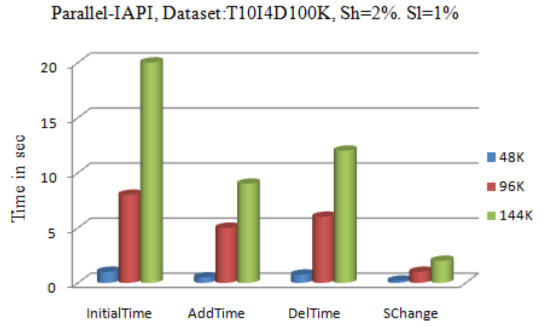


Figure 7. Partition add, delete, support change time comparisons.

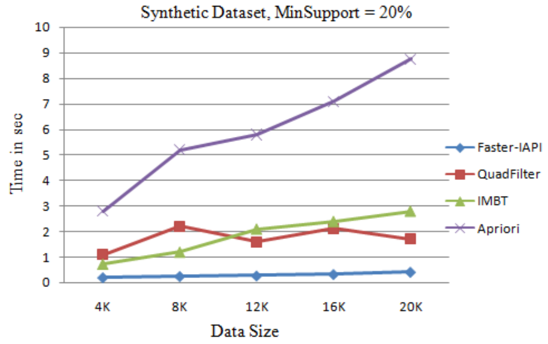


Figure 8. Comparison of FPM algorithms.

4.1. Performance Comparison

Performance of Faster-IAPI is compared with popular algorithms: Apriori, Partition Algorithm, DIC, CanTree and IMBT using T10I4D100K dataset and a synthetic dataset. Speed of execution of Faster-IAPI is faster and the memory requirement of IAPI is lesser than other algorithms as shown in Figure 8. Though partition algorithm is designed for very large dataset, due to large number of independent local frequent sets generated for dense datasets, it requires more memory and computational delay; thus there is limitation in dataset size. For updating the frequent patterns, it is required to keep all local frequent sets 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. Since IAPI is not generating any local frequent sets, it is suitable for both dense and sparse datasets.

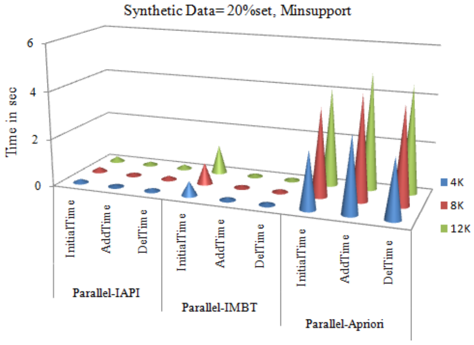


Figure 9. Parallel FPM comparisons.

DIC requires less time for updating the frequent patterns on addition of new datasets and the removal of old transactions 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. Subset generation introduces more computational cost and requires 40 to 50 times 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 2 to 3 times more memory than IAPI. CanTree requires re-construction of FP tree for each FI for every addition, deletion as well as the support change cases, which may lead to more computational delay. It is observed that IMBT tree requires more time to create and more memory to store the entire tree. Thus it may not be suitable for datasets having more number of distinct items. This approach does not need not to predetermine the minimum support threshold and scans the database only once. IMBT requires less time to update the frequent sets on addition and deletion of data than IAPI algorithms. Performance of Parallel-IAPI is tested with two parallel algorithms Parallel-IMBT and parallel-Apriori using multi-threading and multi-processing systems as shown in Figure 9. Parallel-IAPI generates frequent itemsets in less time and requires less memory compared with Parallel-IMBT and Parallel-Apriori.

5. Conclusions

To extract knowledge from the real life databases, efficient incremental and interactive FPM approaches of very large databases are required. This study proposes incremental and interactive mining algorithms with partitioning approach, designed to obtain frequent patterns from a very large sized dataset in sequential and parallel manner. This approach generates frequent sets without generating candidate sets/local frequent itemsets in two database scans with simple data structures. It combines the features of various algorithms such as Apriori, FP-Growth, CARMA and Partitioning algorithm to obtain frequent itemsets. The length and the number of transaction to be compared at each level of higher frequent sets get reduced due to four level filtering approaches. Thus, 30% to 50% of data comparisons reduction is achieved at each level (n-itemsets to (n+1)-itemsets). This approach uses two bounds (low, high) for minimum support to incrementally update the frequent set without rescanning the entire dataset. This study illustrates that the proposed methods are capable to prepare more accurate user spending profile and market analysis with less time and space complexities compared with the existing techniques.

References

[1] Agrawal R. and Shafer J., *Parallel Mining of Association Rules: Design, Implementation, and Experience*, IBM Research Report, 1996.

[2] Agrawal R. and Srikant R., “Fast Algorithms for Mining Association Rules,” *in Proceeding of International Conference Very Large Databases*, San Francisco, pp. 487-499, 1994.

[3] Bhadane C., Shah K., and Vispute P., “An Efficient Parallel Approach for Frequent Itemset Mining of Incremental Data,” *International Journal of Scientific and Engineering Research*, vol. 3, no. 2, pp. 1-5, 2012.

[4] Brin S., Motwani R., Ullman J., and Tsur S., “Dynamic Itemset Counting and Implication Rules for Market Basket Data,” *in Proceeding of ACM SIGMOD International Conference on Management of Data*, New York, pp. 255-264, 1997.

[5] Chen D., Lai C., Hu W., Chen W., Zhang W., and Zhen W., “Tree Partition Based Parallel Frequent Pattern Mining on Shared Memory Systems,” *in Proceeding of 20th International Conference on Parallel and Distributed Processing Symposium*, Rhodes Island, pp. 1-8, 2006.

[6] Cheung D., Ng T., Fu A., and Fu Y., “Efficient Mining of Association Rules in Distributed Databases,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 911-922, 1996.

[7] Han J., Pei J., and Yin Y., “Mining Frequent Patterns without Candidate Generation,” *in Proceeding of ACM SIGMOD International Conference on Management of Data*, New York, pp. 1-12, 2000.

[8] Hamedanian M., Nadimi M., and Naderi M., “An Efficient Prefix Tree for Incremental Frequent Pattern Mining,” *International Journal of Information and Communication Technology Research*, vol. 3, no. 2, pp. 49-55, 2013.

[9] Hidber C., “Online Association Rule Mining,” *in Proceeding of the ACM SIGMOD International Conference on Management of Data*, Philadelphia, pp.145-0156, 1999.

[10] Leung C., Khan Q., Quamrul I., Li Z., and Hoque T., “CanTree: A Canonical-Order Tree for Incremental Frequent-Pattern Mining,” *Knowledge and Information Systems*, vol. 11, no. 3, pp. 287-311, 2007.

[11] Li N., Zeng L., He Q., and Shi Z., “Parallel Implementation of Apriori Algorithm Based on MapReduce,” *International Journal of Networked and Distributed Computing*, vol. 1, no. 2, pp. 89-96, 2013.

[12] Park J., Chen M., and Yu P., “An Effective Hash-Based Algorith for Mining Association Rules,” *in*

Proceeding of PYOC ACM-SIGMOD International Conference Management of Data, New York, pp.175-186, 1995.

[13] Park J., Chen M., and Yu P., "Efficient Parallel Data Mining for Association Rules," *in Proceeding of International Conference Information and Knowledge Management*, Baltimore, pp. 31-36, 1995.

[14] Savasere A., Omiecinski E., and Navathe S., "An Efficient Algorithm for Mining Assocation Rules in Large Databases," *in Proceeding of International Conference Very Large Databases*, San Francisco, pp. 432-444, 1995.

[15] Sherly K., Nedunchezhian R., and Rajalakshmi M., "IAPI Quad-Filter: An Interactive and Adaptive Partitioned Approach For Incremental Frequent Pattern Mining," *Journal Of Theoretical and Applied Information Technology*, vol. 63, no. 1, pp. 147-157, 2014.

[16] Tanbeer S., Ahmed C., Jeong B., and Lee Y., "Efficient Single-Pass Frequent Pattern Mining Using a Prefix-Tree," *Information Science*, vol. 179, no. 5, pp. 559-583, 2008.

[17] Veloso A., Meira W., Carvalho M., Parthasarathy S., and Zaki M., "Parallel, Incremental and Interactive Mining for Frequent Itemsets in Evolving Databases," *in Proceeding of International Workshop High Performance Data Mining*, New York, pp. 1-10, 2003.

[18] Yafi E., Al-Hegami A., Alam A., and Biswas R., "YAMI-Incremental Mining of Interesting Association Patterns," *The International Arab Journal of Information Technology*, vol. 9, no. 6, pp. 504-510, 2012.

[19] Yang C. and Yang D., "IMBT-A Binary Tree for Efficient Support Counting of Incremental Data Mining," *in Proceeding of International Conference on Computational Science and Engineering IEEE Computer Society*, Vancouver, pp. 324-329, 2009.

Sherly Kuriakose received her B.E (Electronics & Communication) degree in 1990, M.Tech (Information Technology) degree in 2004 and Ph.D (Computer Science and Engineering) in 2015. Presently she is working as Associate Professor in Rajagiri School of Engineering, Ernakulam. She also worked as Head of Department of Information Technology at Toc H Institute of Science & Technology, Arakunnam. She has more than 25 years of academic experience. Her research interests are Network security, Knowledge Discovery in Databases, Distributed Database Systems and Parallel Processing.

Raju Nedunchezhian is the Professor in Coimbatore Institute of Technology, TamilNadu. Prior to this, he was Principal of Sri Ranganathar Institute of Engineering and Technology, Coimbatore and Vice-principal of KIT-Kalaignarkarunanidhi Institute of Technology, Coimbatore. He also worked as Research Coordinator of the Institute and Head of Computer Science and Engineering Department (PG) at Sri Ramakrishna Engineering College, Coimbatore. He has more than 25 years of experience in research and teaching. He obtained his BE(Computer Science and Engineering) degree in 1991, ME(Computer Science and Engineering) degree in 1997 and Ph.D(Computer Science and Engineering) in 2007. He has guided many UG, PG, M.Phil and Ph. D scholars. Currently, he is research guide for many Ph.D scholars of the Anna University, Coimbatore, and Bharathiar University. His research interests include knowledge discovery and data mining, Soft Computing, distributed computing, Information Privacy and security, Video processing and Software Engineering. He has published many research papers in national/international conferences and journals. He is a Life member of Advanced Computing and Communication Society and ISTE.