Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 38 (2012) 1467 – 1478

Procedia Engineering

www.elsevier.com/locate/procedia

International Conference on Modeling Optimization and Computing

Frequent Itemset Generation using Double Hashing Technique

Jayalakshmi N[1],Vidhya V[2], Krishnamurthy M[3], Kannan A[4]

[1,3]*Department of Computer Applications,Sri Venkateswara College of Engg.,Sriperumbudur 602105,India*
[2]*Department of Computer Science and Engineering, Sri Venkateswara College of Engg.,Sriperumbudur 602105,India*
[4]*Department of Information Science and Technology, Anna University, Chennai600025, India*

Abstract

In data mining, frequent itemsets plays an important role which is used to identify the correlations among the fields of database.In this paper, we propose a new association rule mining algorithm called Double Hashing Based Frequent Itemsets, (DHBFI) in which hashing technology is used to store the database in vertical data format. This double hashing technique is mainly preferred for avoiding the major issues of hash collision and secondary clustering problem in frequent itemset generation. Hence this proposed hashing technique makes the computation easier, faster and more efficient.Also this algorithm eliminates unnecessary redundant scans in the database and candidate itemset generation which leads to less space and time complexity.

Keywords—Association Rule, Double Hashing, Frequent Itemset, Secondary Clustering, Hash Collision.

1. Introduction

Data mining, the extraction of hidden predictive information from large databases, is a powerful new technology with great potential to help companies focusing on the most important information in their data warehouses. Data mining tools predict future trends and behaviours, allowing businesses to make proactive, knowledge-driven decisions. The automated prospective analysis, offered by data mining move beyond the analysis of past events provided by retrospective tools, typically of decision support systems.

*Jayalakshmi N. Tel.:+91-9443458263.
E-mail address: njayalakshmi@svce.ac.in

1877-7058 © 2012 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
doi:10.1016/j.proeng.2012.06.181

Data mining tools can answer business questions that traditionally were too time consuming to resolve. They scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations.

1.1 Frequent ItemSets

The problem of association rule mining is defined as: Let I= $\{i_1, i2 \ldots i_n\}$ be a set of n binary attributes called items. Let D= $\{t_1, t_2 \ldots t_m\}$ be a set of transactions called the database. Each transaction in D has a unique transaction ID and contains a subset of the items in I. A rule is defined as an implication of the form X->Y where $X, Y \subseteq I$ and $X \cap Y = \emptyset$. The sets of items (itemsets) X and Y are called antecedent and consequent. Association rules are usually required to satisfy a user-specified minimum support and a user-specified minimum confidence at the same time. Association rule generation is usually involving two steps:

- First, minimum support is applied to find all frequent itemsets in database.
- Second, these frequent itemsets and the minimum confidence constraint are used to form rules.

To find the frequent item sets, efficient algorithm such as Apriori Algorithm is used.Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate generation), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found. Apriori uses breadth-first search and a tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length $k - 1$. Then it prunes the candidates which have an infrequent sub pattern.

Sequential pattern mining is essential in many applications, including computational biology, consumer behaviour analysis, web log analysis, etc. Although sequential patterns can tell what items are frequently to be purchased together and in what order, they cannot provide information about the time span between items for decision support. Previous studies dealing with this problem either set time constraints to restrict the patterns discovered or define time-intervals between two successive items to provide time information. Accordingly, the first approach falls short in providing clear time-interval information while the second cannot discover time-interval information between two non-successive items in a sequential pattern. To provide more time-related knowledge, Ya-Han Hu et al (2009) defined a new variant of time interval sequential patterns, called multi-time-interval sequential patterns, which can reveal the time-intervals between all pairs of items in a pattern. They developed two efficient algorithms, called the MI Apriori and MI-Prefix Span algorithms, to solve this problem. The experimental results show that the MI Prefix Span algorithm is faster than the MI-Apriori algorithm, but the MI-Apriori algorithm has better scalability in long sequence data.

In this paper, a new Double Hashing Based Frequent Itemset (DHBFI) generation algorithm of the vertical data format for the transactional database is proposed. In this,first the data is represented as an item and Transaction id set (Tidset) format. To avoid collision and primary clustering problem, quadratic probing technique is used. Primary clustering occurs with linear probing because the same linear pattern, if a bin is inside a cluster, then the next bin must also be in that cluster, or expand the cluster. Instead of searching forward in a linear fashion, searching forward using a quadratic function is done.

2 Literature Survey

There are many works in the literature that discuss about Association rules, hash based technique and Frequent Itemsets. The Association Rule mining raised by R. Agrawal is an important research in data mining field. His Apriori algorithm can discover meaningful itemsets and build association rules. However, a large number of candidate sets are generated and the database needs repeated scanning. In order to reduce the database scanning various studies were undergone. Further studies in data mining have presented many efficient algorithms for discovering association rules.

The discovery of association rules has been discussed in the past using two steps namely: Finding out all the frequent itemsets which are greater than or equal to user-specified minimum support threshold, generating association rules from frequent itemsets[8].The GRA (Gradational Reduction Approach)algorithm uses a hash based technique, which is similar to Hash Table to increase the access efficiency. This algorithm uses an infrequent item sets filtering mechanism to avoid generating a great deal of infrequent sub-itemsets of transaction records. It uses gradational reduction mechanism to reduce the database size which uses the frequent itemsets as the information of filtration mechanisms to erase the infrequent items from database at every phase. GRA algorithm can decrease a large number of non-frequent itemsets and increase the utility rate of memory [5].

The transaction-marked DHP algorithm (TMDHP) is used in mining frequent itemsets in pervasive computing. Each element of the itemsets and the transactions ID will be stored together in the hash table. We need to access database once and avoids producing a candidate itemsets[10]. Many algorithms focus on how to find the frequent itemsets quickly such as the Pincer-Search [6], FP-growth [3], ICI [4] and R. Agrawal and R. Srikant [2] . Among those algorithms, Pincer-Search combines both the bottom-up and top-down directions to reduce the number of candidate sets, and then increases the efficiency.

In addition, because FP-growth uses the structure of FP-tree to store the items of database, the algorithm only needs to scan the database two times without generating any candidate sets in mining frequent itemsets. Thus FP-growth can quickly finish the mining task. The HBMFI-LP method used hashing technology to store the database in vertical data format. To avoid hash collisions, linear probing technique was used. It generates the exact set of maximal frequent itemsets directly by removing all nonmaximal itemsets. In each level, maximal frequent itemsets is generated by using the frequent itemsets [9].

The hash based algorithm is used for candidate set generation. Here, a candidate 2-itemsets is generated.By using that, the next level candidates are generated and the drawback is the item which is not frequent is removed in the first level of candidate generation but it is not updated in the database. [7].Association rule mining algorithm is used for generating frequent itemsets in a single pass and it has two steps of partitioning and merging. Within each portion the Apriori approach is utilized to find the frequent itemset and at last the partitions are merged. The time taken for database scan is more than the time taken for candidate generation when the database size is large [1].

The SMM (Sparse-Matrix Mining) is used to find the frequent itemsets from sparse matrix. It uses a special linked list unit and two strategies to store data in matrix. It maps the database to binary sparse matrix (the binary form is 0s and 1s, here the data are stored in the form of binary only) and stores the compressed data into a linked list. When the transactions increases, its relative speed decreases. The reason is, after SMM produces frequent k-itemsets, all frequent k-itemsets and related transactions need to be kept in memory. When support is small, frequent 1-itemsets and frequent 2- itemsetss numbers are

very large, accessing these data will cause many times of page faults, and thus the calculating speed decreases [11].

An improved algorithm is used for mining the association rules in generating frequent k-itemsets. This new algorithm finds frequent itemsets directly and removes the subset that is not frequent, which based on the classical Apriori [12] instead of checking whether these candidates are frequent itemsets after generating new candidates. Many algorithms now focus on how to reduce the number of scanning database or the number of candidate sets.

3 Paper Organization

Section IV presents the different hashing techniques used for frequent itemset.Section V presents the proposed work for the system with the algorithm. Section VI deals with an example of the proposed work. Section VII deals with an experimental results and Section VIII provides the conclusions.

4 Hashing Techniques

4.1 Apriori Algorithm using Hashing

In Apriori algorithm, candidate item sets are generated to find the frequent itemsets. When candidate sets are generated, the algorithm needs to test their occurrence frequencies. This testing result in high frequency in querying, so tremendous amounts of resources will be utilized both in time and in space. An improved algorithm was proposed for generating frequent k-itemsets. Instead of judging whether these candidates are frequent item sets after generating new candidates, this new algorithm finds frequent itemsets based on the property that all subsets of a frequent item set must also be frequent. Through pruning candidate itemsets by the infrequent itemsets, we reduce the number of scanning and the redundancy. Hash data structures can be maintained to store databases. This makes easy in performing several tasks. As vertical data format is followed, support also need not be calculated separately. Minimum support threshold must be properly chosen such that it is not too high where we may lose some interesting itemsets or too low where unimportant itemsets are generated. It can be taken by the number of transactions in the tidset which is available as a count value in header node.

4.2 Hashing Techniques

A hash table or hash map is a data structure that uses a hash function to map identifying values, known as keys (e.g., a person's name), to their associated values (e.g., their telephone number). Thus, a hash table implements an associative array. The hash function is used to transform the key into the index (the *hash*) of an array element (the *slot* or *bucket*) where the corresponding value is to be sought. Ideally, the hash function should map each possible key to a unique slot index, but this ideal is rarely achievable in practice (unless the hash keys are fixed; i.e. new entries are never added to the table after it is created). Instead, most hash table designs assume that hash collisions—different keys that map to the same hash value—will occur and must be accommodated in some way. In a well-dimensioned hash table, the average cost (number of instructions) for each lookup is independent of the number of elements stored in the table. Many hash table designs also allow arbitrary insertions and deletions of key-value pairs, at constant average cost per operation.

4.2.1 Separate Chaining

In the strategy known as separate chaining, direct chaining, or simply chaining, each slot of the bucket array is a pointer to a linked list that contains the key-value pairs that hashed to the same location.

Lookup requires scanning the list for an entry with the given key. Insertion requires adding a new entry record to either end of the list belonging to the hashed slot. Deletion requires searching the list and removing the element.

4.2.2 Open Addressing

In another strategy, called open addressing, all entry records are stored in the bucket array itself. When a new entry has to be inserted, the buckets are examined, starting with the hashed to slot and proceeding in some probe sequence, until an unoccupied slot is found. When searching for an entry, the buckets are scanned in the same sequence, until either the target record is found, or an unused array slot is found, which indicates that there is no such key in the table. The name "open addressing" refers to the fact that the location ("address") of the item is not determined by its hash value. This method is also called closed hashing. Well-known probe sequences include:

- Linear probing, in which the interval between probes is fixed.
- Quadratic probing, in which the interval between probes is increased by adding the successive outputs of a quadratic polynomial to the starting value given by the original hash computation.
- Double hashing, in which the interval between probes is computed by another hash function.

4.2.3 Linear Probing

The linear hashing algorithm performs splits in a deterministic order, rather than splitting at a bucket that overflowed. The splits are performed in linear order (bucket 0 first, then bucket 1, then 2 …), and a split is performed when any bucket overflows. If the bucket that overflows is not the bucket that is split (which is the common case), overflow techniques such as chaining are used, but the common case is that few overflow buckets are needed.

Using linear hashing, the address space (number of buckets) increases linearly and is exactly as large as is needed. For any number of insertions, most of the overflow records are moved into primary buckets by splits, and thus the number of overflow records is small. Thus, we expect to find our data in one access most of the time.

4.2.4 Quadratic Probing

It is an open addressing method to handle overflows after a collision takes in some bucket of a hash table. Quadratic probing operates by taking the original hash value and adding successive values of an arbitrary quadratic polynomial to the starting value.

It is named quadratic probing because of the function it uses to resolve the collisions. The form of the equation is generally $f(k) = c_1k^2 + c_2k + c_3$. The function used might even be $f(k) = c_1k^2$ if c_2 and c_3 are taken as zero. Suppose a cell H is reached but is occupied, then the next sequence of cells to be examined would be $H + 1^2$, $H + 2^2$, $H + 3^2$, $H + 4^2$..., $H + k^2$. Linear Probing, instead, would examine the sequence $H + 1$, $H + 2$, $H + 3$, $H + 4$..., $H + k$. This would result in primary clustering and the larger the cluster grows, lesser will be the search efficiency for those items. Quadratic probing can be a more efficient algorithm in a closed hash table. Quadratic probing provides good memory caching because it preserves some locality of reference.

4.2.5 Double Hashing

Double hashing is a computer programming technique used in hash tables to resolve hash collisions, cases when two different values to be searched for producing the same hash key. It is a popular collision-resolution technique in open-addressed hash tables.

Like linear probing, it uses one hash value as a starting point and then repeatedly steps forward an interval until the desired value is located, an empty location is reached, or the entire table has been searched; but this interval is decided using a second, independent hash function (hence the name double hashing). Unlike linear probing and quadratic probing, the interval depends on the data, so that even values mapping to the same location have different bucket sequences; this minimizes repeated collisions and the effects of clustering. In other words, given independent hash functions h_1 and h_2, the jth location in the bucket sequence for value k in a hash table of size m is: h (k, j) = [h1 (k) +j*h2 (k)] mod m.

5 Proposed Work

Quadratic probing does indeed eliminate the primary clustering problem, it also places a restriction on the number of items that can be put in the table-the table must be less than half full. Double hashing is yet another method of generating a probing sequence. It requires two distinct hash functions (i) =i*g(k) where g is a second hash function .Probe sequence is,

0th probe = h (k) mod TableSize
1th probe = (h (k) + g(k)) mod TableSize
2th probe = (h(k) + 2*g(k)) mod TableSize
3th probe = (h(k) + 3*g(k)) mod TableSize
. . .
ith probe = (h(k) + i*g(k)) mod TableSize *(1)*

where h(k)=k mod TableSize and g(k)=1+(k mod (TableSize-1)) and TableSize should be prime.

5.1 Proposed System Architecture

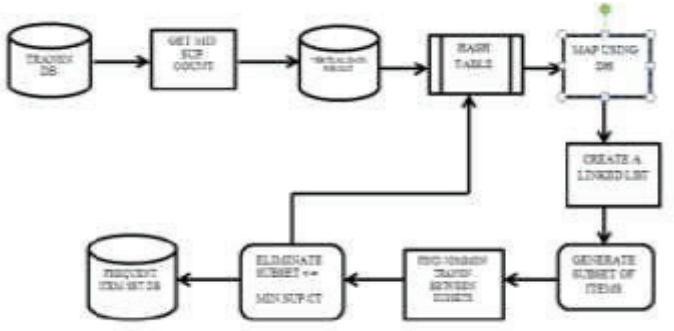

Fig 1 DHBFI Architecture

5.2 DHBFI Algorithm

Input: All transaction from the vertical hash table

Process logic: Finding the frequent itemsets
Output: Generating the frequent itemsets

```
begin
    m=0; k=0;
    Get the minimum support, min_sup;
    Generate the new database in (Items,Tidset) format
    For all Items I ε D do
         Increment m.
    n=2*m + 1
    Dk=D;
    do
    begin
          Make a hash table of size n.
          Map items to the buckets if collision
          occurs then use double hashing technique.
          Create a linked list for the kth level to
          maintain the transaction from the databse Dk.
          for all Items I ε Dk do
          begin
               for all Items I ε Dk do
                    Generate a subset of items.
          end
          Find common transaction between subsets in the kth level.
          Eliminate the subset <= min_sup.
          Dk = Items >=min_sup.
          Increment k.
     end until frequent itemset found
end
```

6 Example of the Proposed Work

Table 1.Initial Transaction Database

Tidset	Itemset
T1	Benoquin, Dialyte, Ibuprofen, Nutradrops
T2	Dialyte, Nutradrops
T3	Benoquin, Veetids
T4	Benoquin, Dialyte, Ibuprofen
T5	Benoquin, Nutradrops
T6	Dialyte, Ibuprofen
T7	Benoquin, Ibuprofen
T8	Dialyte, Nutradrops
T9	Benoquin, Dialyte
T10	Ibuprofen, Nutradrops

For our convenience, Let us replace these real time items Benoquin, Dialyte, Ibuprofen,Nutradrops and Veetids with I1, I2, I3, I4 and I5 respectively in tables.

Table 2 Transactional Database.

Tidset	Itemset
T1	I1, I2, I3, I4
T2	I2,I4
T3	I1,I5
T4	I1,I2,I3
T5	I1,I4
T6	I2,I3
T7	I1,I3
T8	I2,I4
T9	I1, I2
T10	I3, I4

By taking minimum support count = 3, the following table shows transaction set (Tidset) for all 5 items.

Table 3 Vertical Format of the Transactional Database

Tidset	Itemset
I1	T1, T3, T4, T5, T7, T9
I2	T1, T2, T4, T6, T8, T9
I3	T1, T4, T6, T7, T10
I4	T1, T2, T5, T8, T10
I5	T2, T3

The items in the transaction are hashed based on the hash function: h (k) = (order of item k) mod n. The n value is determined by using the formula 2m + 1 where m is the number of items in the database. The transaction in which I1 is present is connected in the form of linked list and the first node denotes the number of occurrences of the item in the transactions. It can be observed from Fig.2 that I1 is hashed to 1st location and it is determined using the hash function. Similarly all items are hashed into the hash table.

The cross symbol indicates the end of the items in the list. Here, the linked list is created based on the item set and not on the transactions because the transactions are more so that it occupies more memory and it is very difficult to access the items. There is a link between transactions in each itemsets.

The linked list is created for all levels of frequent itemset generation. In the next higher level, the item subsets become low and it is easy to find frequent itemsets of that level. The process continues until the exact frequent itemset is found.

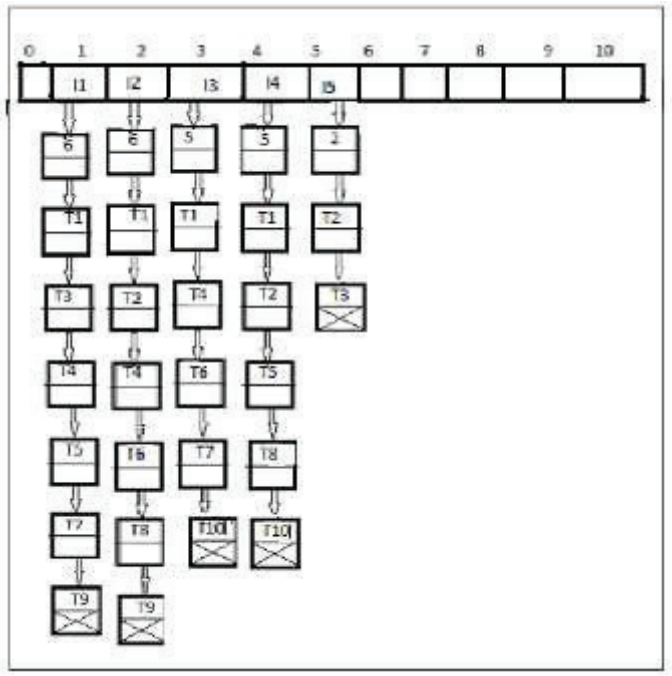

Fig.2 Hash table including links for the transactional database in the first level.

Table 4 Vertical Format of the Transactional Database in the Second Level.

Itemset	Tidset
{I1, I2}	T1, T4, T9
{I1, I3}	T1, T4, T7
{I1, I4}	T1, T5
{I2, I3}	T1, T4, T6
{I2, I4}	T1, T2, T8
{I3, I4}	T1, T10

The itemset in the second level are hashed based on the hash function, h (k) = ((order of X)*10 +order of Y) mod n. Here, the itemsets are mapped to 1, 2, 3, 1, 2, 1. Here, there is a collision for {I1, I2}{I2, I3} are mapped to 1 and {I1, I2} {I3, I4} are mapped to 1 and {I1, I3}{I2, I4} are mapped to 2. Double Hashing technique is used to overcome collision.

Let h (k) be a hash function that maps an element k to an integer in [0, m-1], where m is the size of the table. Let the ith probe position for a value k be given by the function $h\ (k, i)= (h(k)+i*h_p(k))\ mod\ m$,where $h_p(k)$ refers to second hash function called probing hash function. Then using DH technique, {I2, I3} is mapped to location 5,{I3, I4} is mapped to location 7,{I2, I4} is mapped to location 8. It better avoids the secondary clustering problem that occurs with quadratic probing.

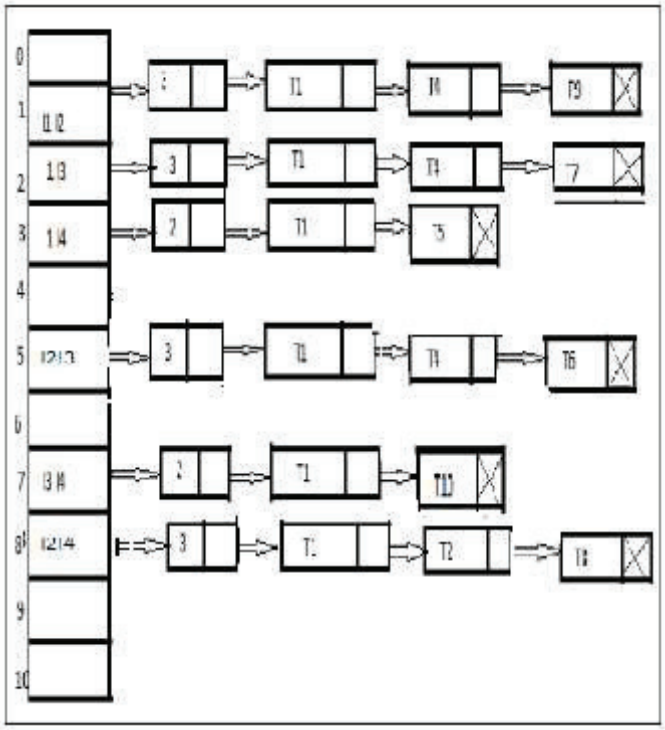

Fig.3 Hash table including links for the transactional database in the second level.

In the second level, itemsets {I1, I2}, {I1, I3}, {I2, I3},{I2, I4},{I3, I4} whose support counts are greater than or equal to 3 are said to be frequent itemsets are observed from the Table 4 and Fig.3. The 3-itemsets are generated from the frequent itemsets of second level.

Table 5 Vertical Format of the Transactional Database in the Third Level

Itemset	Tidset
{I1, I2, I3}	T1, T4
{I1, I2, I4}	T1
{I2, I3, I4}	T1

The itemsets in the third level are hashed based upon the hash function h(k)=((order of X)*100+(order of Y)*10+order of z)mod n). Using the hash function, {I1, I2, I3},{I1, I2, I4},{I2, I3,I4} are mapped to location 2,3,3. Here there is a collision for {I1, I2, I4} and {I2, I3, I4}. Then using quadratic probing technique, {I2, I3, I4} is mapped to location 4. The hash table for third level is in Fig 4.

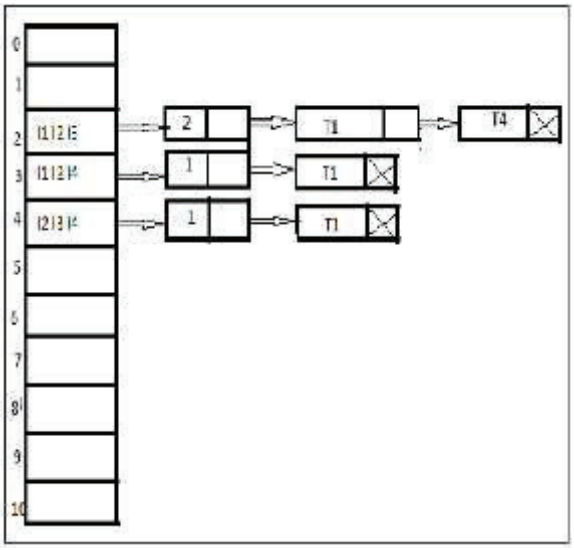

Fig. 4 Hash table including links for transactional database in the third level.

The proposed algorithm (DHBFI) performs better because FI is calculated in an efficient way. The structure of transactional database is in vertical data format. This makes easy to perform several tasks. In this format, support also need not be calculated separately. In this case, support is directly given by the number of transactions in the Tidset list of each FI or it can be obtained from the count value in the header node of the corresponding linked list. It is about 2 to 3 times faster than other hash based technique. It quickly finds an empty location in the hash table to map the items. The DHBFI performs better with large number of transactions and long itemsets.

7 Experimental Results

Fig.5 shows a comparison of results of HBFI - QP and DHBFI for various values of minimum support count thresholds. From the diagram it can be seen that the time taken for DHBFI is considerably reduced. In this method the time taken to hash itemsets in the vertical hash table is comparatively low. For various support counts, the time taken to find a frequent item set is less compared with quadratic probing.

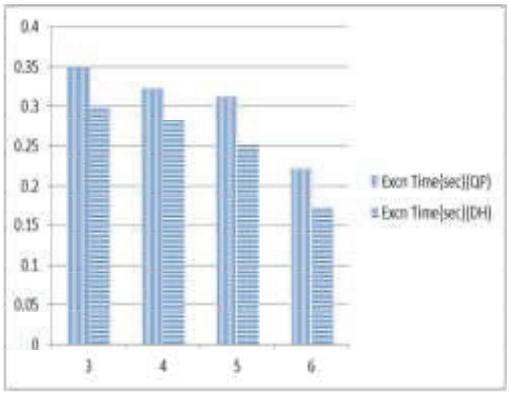

Fig.5 Time Comparison between HBFI-QP and DHBFI

8 Conclusion

Our experimental results demonstrate that DHBFI is better than HBFI-QP and other hash based methods because it efficiently maps the itemsets in the hash table and it also avoids both primary and secondary clustering problem. The vertical data format representation of the database makes easy manipulations on hash data structure. DHBFI does not use all the bins and hence the phenomenon of secondary clustering will not occur with double hashing technique.

References

[1] Krishnamurthy M, Kannan A, Baskaran R, and Deepalakshmi," Frequent itemset generation using hashing-quadratic probing technique",In European Journal of Scientific Research on volume 50 no.4,2011,pages 523 – 532.

[2]Agrawal R and Srikant R,"Fast algorithms for mining association rules in large databases", In Proceedings of the 20th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 1994,pages 487- 499.

[3] Han J, Pei J, and Yin Y," Mining frequent patterns without candidate Generation", In ACM SIGMOD Record, volume 29, , 2000, pages 1-12.

[4]Chien I.P, Huang J.P, and Kuo H.C,"An efficient incremental mining algorithm-ici", Journal of e-Business,2006, 8(3):393-413.

[5] Guo-Cheng Lan, Huang-Cheng Kuo, Jen-Peng Huang, Tzung-Pei Hong,"A decomposition approach for mining frequent itemsets",Decision Support Systems, 2007,2:605-608.

[6] Kedem.Z and Lin.D.I ,"Pincer-search: A new algorithm for discovering the maximum frequent set", Advances in Database Technology EDBT'98, 1998,pages 103-119.

[7] Chen M.S, and Park J.S, and Yu.P.S," An effective hash-based algorithm for mining association rules", In Proceedings of the 1995 ACM SIGMOD international conference on Management of data, 1995,pages 175-186.

[8]Agarwal A, Imielinski.T, and Swami.A"Mining association rules between sets of items in large databases", Proceeding in ACM SIGMOD,1993,pages 207-216.

[9]Balasubramanien.P, Rahman A.M.J.M.Z., and Krihsna P.V,"A hash based mining algorithm for maximal frequent item sets using linear probing", 2007.

[10]Chen. S, Fu. X, Su.J, Teng.S, Zhang.W,"An algorithm of mining frequent itemsets in pervasive computing", In Pervasive Computing and Applications, ICPCA 2008. Third International Conference on, volume 2,IEEE,2008, pages 559-563.

[11] Ji-Zhou.S and Xiao-Yan.Z,"Finding frequent item sets from sparse matrix", In Electronic Computer Technology, 2009 International Conference on, IEEE. 2009, pages 615-619.

[12]Yang.B,Liu.Y,"Research of an improved apriori algorithm in mining association rules", Journal of Computer Applications, 2007,pages 418-420.