ISSN:0975-9646
Mamatha Nadikota et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (4) , 2011, 1448-1452

Hashing and Pipelining Techniques for Association Rule Mining

Mamatha Nadikota, Satya P Kumar Somayajula,Dr. C. P. V. N. J. Mohan Rao
CSE Department,Avanthi College of Engg &Tech ,Tamaram,Visakhapatnam,A,P..,India

***Abstract:*Apriori is a classic algorithm for learning association rules. It is designed to operate on databases containing transactions .In This The candidate itemsets and a database is loaded into the hardware. But capacity of the hardware architecture is fixed. As number of candidate itemsets or the number of items in the database is larger than the hardware capacity. So That the items are loaded into the hardware separately, Due To this The time complexity is more to load candidate itemsets or database items into the hardware is in proportion to the number of candidate itemsets multiplied by the number of items .in the database. Increase Of candidate itemsets and a large database would create a performance blockage. we propose a Hash-based and Pipelined (HAPPI) architecture for hardware-enhanced association rule mining. By Using the pipeline methodology to compare itemsets with the database and gather useful information for reducing the number of candidate itemsets and items in the database concurrently. To find frequent itemsets the database is fed into the hardware, candidate itemsets are compared with the items in the database. At the same time, trimming information is collected from each transaction. Next itemsets are generated from transactions and hashed into a hash table. The useful trimming information and the hash table enable us to reduce the number of items in the database and the number of candidate itemsets. So that we can effectively reduce the frequency of loading the database into the hardware. Hashing And Pipelining solves the Performance bottleneck problem in a priori-based hardware schemes.**

INTRODUCTION

Association mining was introduced by it has emerged as a prominent research area. The association mining problem also referred to as the *market basket* problem can be formally defined as follows. Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items as $S = \{s_1, s_2, \ldots, s_m\}$ be a set of transactions, where each transaction $s_i \in S$ is a set of items that is $s_i \subseteq I$. An *association rule* denoted by $X \Rightarrow Y$, where $X, Y \subset I$ and $X \cap Y = \Phi$, describes the existence of a relationship between the two itemsets X and Y.Several measures have been introduced to define the *strength* of the relationship between itemsets X and Y such as support, confidence, and interest. The definitions of these measures, from a probabilistic model are given below.

Support $(X \Rightarrow Y) = P(X, Y)$, or the percentage of transactions in the database that contain both X and Y.

Confidence $(X \Rightarrow Y) = P(X, Y) / P(X)$, or the percentage of transactions containing Y in transactions those contain X.

Interest($X \Rightarrow Y) = P(X, Y) / P(X)P(Y)$ represents a test of statistical independence.

Boolean Association Mining

Given a set of items $I = \{i_1, i_2, \ldots, i_n\}$, a transaction t is defined as a subset of items such that $t \in 2^I$, where $2^I = \{\emptyset, \{i_1\}, \{i_2\}, \ldots, \{i_n\}, \{i_1, i_2\}, \ldots, \{i_1, i_2, \ldots, i_n\}\}$. In reality, not all possible transactions might occur. For example, transaction $t = \emptyset$ is excluded.

Let $T \subseteq 2^I$ be a given set of transactions $\{t_1, t_2, \ldots, t_m\}$. Every transaction $t \in T$ has an assigned weight w'(t). Several possible weights could be considered,

w'(t) =	1, for all transactions $t \in T$.
w'(t) =	f(t), where f(t) is the frequency of transaction t, for all transactions $t \in T$, i.e., how many times the transaction t was repeated in our database.
w'(t) =	|t| * g(t) for all transactions $t \in T$, where |t| is the cardinality of t, and g(t) could be either one of the weight functions w'(t)'s defined in (i) and (ii). In this case, longer transactions get higher weight.
w'(t)=	v(t) * f(t) for all transactions $t \in T$, where v(t) could be the sum of the prices or profits of those items in t.

The weights w's are normalized to

$$w(t) = \frac{w'(t)}{\sum_{\forall t' \in T} w'(t')}, \text{ and } \sum_{\forall t \in T} w(t) = 1$$

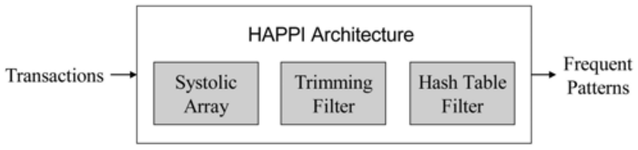


Fig. 1. System architecture.

As shown in Fig. 1, there are three hardware modules in our system. First, when the database is fed into the hardware, the candidate itemsets are compared with the items in the database by the systolic array. Candidate itemsets that have a higher frequency than the minimum support value are viewed as frequent itemsets. Second, we determine the frequency that each item occurs in the candidate itemsets in the transactions at the same time. These frequencies are called trimming information. From this information, infrequent items in the transactions can be eliminated since they are not useful in generating frequent itemsets through the trimming filter. Third, we generate item sets from transactions and hash them into the hash table, which is then used to filter out unnecessary candidate itemsets. After the hardware compares candidate itemsets with the items in the database, the trimming information is collected and the hash table is built. The useful information helps us to reduce the number of items in the database and the number of candidate itemsets. Based on the trimming information, items are trimmed if their corresponding occurrence frequencies are not larger than the length of the current candidate itemsets. In addition, after the candidate itemsets are generated by merging frequent subitemsets, they are sent to the hash table filter. If the number of itemsets in the corresponding bucket of the hash table is less than the minimum support, the candidate itemsets are pruned. As such, HAPPI solves the bottleneck problem mentioned earlier by the cooperation of these three hardware modules. To achieve these goals, we devise the following five procedures in the HAPPI architecture: support counting, transaction trimming, hash table building, candidate generation, and candidate pruning. Moreover, we derive several formulas to decide the optimal design in order to reduce the overhead induced by the pipeline scheme and the ideal number of hardware modules to achieve the best utilization.

2. RELATED WORKS

We implemented a systolic array with several hardware cells to speed up the Apriori algorithm. Each cell performs an ALU (larger than, smaller than, or equal to) operation, which compares the incoming item with items in the memory of the cell. This operation generates frequent itemsets by comparing candidate item-sets with the items in the database. Since all the cells can execute their own operations simultaneously, the performance of the architecture is better than that of a single processor. However, the number of cells in the systolic array is fixed. If the number of candidate itemsets is larger than the number of hardware cells, the pattern matching procedure has to be separated into many rounds. It is infeasible to load candidate itemsets and the database into the hardware for multiple times. the performance is only about four times faster than some software algorithms. Hence, there is much room to improve the execution time.

3 HAPPI ARCHITECTURE

Apriori-based hardware schemes have to load candidate itemsets and the database into the hardware to execute the comparison process. Too many candidate itemsets and a huge database would cause a performance bottleneck. To solve this problem, we propose the HAPPI architecture to deal with efficient hardware-enhanced association rule mining. We incorporate the pipeline methodology into the HAPPI architecture to perform pattern matching and collect useful information to reduce the number of candidate itemsets and items in the database simultaneously. In this way, HAPPI effectively solves the bottleneck problem the pipeline scheme of the HAPPI architecture is presented. The transaction trimming scheme is given in Section 4.3. Then, we describe the hardware design of hash table filter in Section 4.4. Finally, we derive some properties for performance evaluation in Section 4.5.

4 SYSTEM ARCHITECTURE

The HAPPI architecture consists of a systolic array, a trimming filter, and a hash table filter. There are several hardware cells in the systolic array. Each cell can perform the comparison operation. Based on the comparison results, the cells update the support counters of candidate itemsets and the occurrence frequencies of items in the trimming information. A trimming filter then removes infrequent items in the transactions according to the trimming information. In addition, we build a hash table by hashing itemsets generated by each transaction. The hash table filter then prunes unsuitable candidate itemsets.

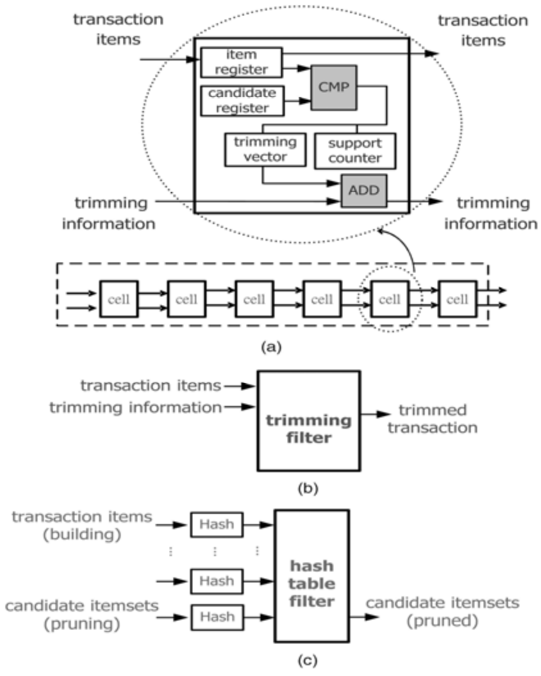
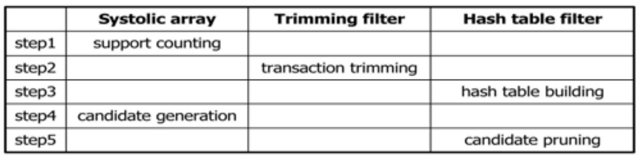


Fig. 2. The HAPPI architecture: (a) systolic array, (b) trimming filter, and c) hash table filter.



	Systolic array	Trimming filter	Hash table filter
step1	support counting		
step2		transaction trimming	
step3			hash table building
step4	candidate generation		
step5			candidate pruning

Fig :3The procedure flow of one round.

To find frequent k-itemsets and generate candidate (k+l)-itemsets efficiently, we devise five procedures in the HAPPI architecture using the three hardware modules: the systolic array, the trimming filter, and the hash table filter. The procedures are support counting, transaction trimming, hash table building, candidate generation, and candidate pruning. The work flow is shown in Fig. 4 The support counting procedure finds frequent itemsets by comparing candidate itemsets with transactions in the database. By loading candidate k-itemsets and streaming transactions into the systolic array, the frequencies that candidate itemsets occur in the transactions can be determined. Note that if the number of candidate itemsets is larger than the number of hardware cells in the systolic array, the candidate itemsets are separated into several groups. Some of the candidate itemsets are loaded into the hardware cells and the database is fed into the systolic array. Afterward, the other candidate itemsets are loaded into the systolic array one by

one. To complete the comparison with all the candidate itemsets, the database has to be examined several times. To reduce the overhead of repeated loading, we design two additional hardware modules, namely, a trimming filter and a hash table filter. Infrequent items in the database are eliminated by the trimming filter, and the number of candidate itemsets is reduced by the hash table filter. Therefore, the time required for support counting procedure can be effectively reduced.
After all the candidate k-itemsets have been compared with the transactions, their frequencies are sent back to the system. The frequent k-itemsets can be obtained from the candidate k-itemsets whose occurrence frequencies are larger than the minimum support. While the transactions are being compared with the candidate itemsets, the corresponding trimming information is collected. The occurrence frequency of each item, which is contained in the candidate itemsets in the transactions, is recorded and updated to the trimming information. After comparing candidate itemsets with the database, the trimming information is collected. The occurrence frequencies and the corresponding transactions are then transmitted to the trimming filter, and infrequent items are trimmed from the transactions according to the occurrence frequencies in the trimming information. Then, the hash table building procedure generates (k+1)-itemsets from the trimmed transactions. These (k+l)-itemsets are hashed into the hash table for processing. Next, the candidate generation procedure is also executed by the systolic array. The frequent k-itemsets are fed into the systolic array for comparison with other frequent k-itemsets. The candidate (k+l)-itemsets are generated by the systolic injection . techniques The candidate pruning procedure uses the hash table to filter candidate (k+l)-itemsets that are not possible to be frequent itemsets. Then, the procedure reverts to the support counting procedure. The pruned candidate (k+l)-itemsets are loaded into the systolic array for comparison with transactions that have been trimmed already. The above five processes are executed repeatedly until all frequent itemsets have been found.

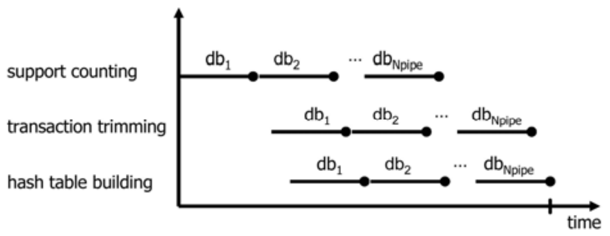


Fig. 4 A diagram of the pipeline procedures

4.1 Pipeline Design

We observe that the transaction trimming and the hash table building procedures are blocked by the support counting procedure. The transaction trimming procedure has to obtain trimming information to execute the trimming process. However, this process cannot be completed until the support counting procedure compares all the transactions with all the candidate itemsets. In addition, the hash table building procedure has to get the trimmed transactions from the trimming filter after all the transactions have been trimmed. This problem can be resolved by applying the pipeline scheme, which utilize the three hardware modules simultaneously in the HAPPI framework. First, we divide the database into Npipe parts. One part of the transactions in the database is streamed into the systolic array and the support counting process is performed on all candidate itemsets. After comparing the transactions with all the candidate itemsets, the transactions and their trimming information are passed to the trimming filter first. The systolic array then Processes the next group of transactions. After items have been trimmed from a transaction by the trimming filter, the transaction is passed to the hash table filter, as shown in Fig. 6, and the trimming filter can deal with the next transaction. In this way, all the hardware modules can be utilized simultaneously. Although the pipelined architecture improves the system’s performance, it increases the computational overhead because of multiple times of loading candidate itemsets into the systolic array

4.2 Transaction Trimming

While the support counting procedure is being executed, the whole database is streamed into the systolic array. However, not all the transactions are useful for generating frequent itemsets. Therefore, we filter out items in the transactions according to Theorem 2 so that the database is reduced. In the HAPPI architecture, the trimming information records the frequency of each item in a transaction that appears in the candidate itemsets. The support counting and trimming information collecting operations are similar since they all need to compare candidate itemsets with transactions. Therefore, in addition to transactions in the database, their corresponding trimming information is also fed into the systolic array in another pipe, while the support counting process is being executed. As shown in Fig. 7, a trimming vector is embedded in each hardware cell of the systolic array to record items that are matched with candidate

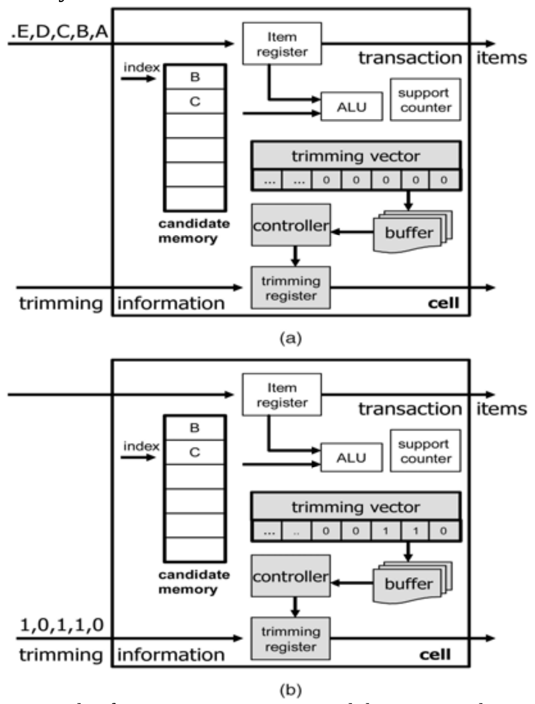


Fig 5. An example of streaming a transaction and the corresponding trimming information into the cell. (a) Stream a transaction into the cell. (b) Stream trimming information into the cell.itemsets.

The ith flag in the trimming vector is set to true if the ith item in the transaction matches the candidate itemset. After comparing the candidate itemset with all the items in a transaction, if the candidate itemset is a subset of the transaction, the incoming corresponding trimming information will be accumulated according to the trimming vector. Since transactions and trimming information are input in different pipes, support counters and trimming information can be updated simultaneously in a hardware cell. In Fig. 7a, the candidate itemset < BC > is stored in the candidate memory, and a transaction (A; B; C; D; E) is about to be fed into the cell. The resultant trimming vector after comparing < BC > with all the items in the transaction is shown in Fig. 7b. Because items B and C are matched with the candidate itemset, the trimming vector becomes < 0; 1; 1; 0; 0 > . Meanwhile, the corresponding trimming information is fed into the trimming register, and the trimming information is updated from < 0; 1; 1; 0; 1 >to < 0; 2; 2; 0; 1 > . After passing through the systolic array, transactions and their corresponding trimming information are passed to the trimming filter. The filter trims off items whose frequencies are less than k. As the example in Fig. 8 shows, the trimming information of the transaction (A; B; C; D; E) is < 2; 2; 2; 1; 2 > and the current k is 2. Therefore, the item D should be trimmed. The new transaction becomes {A; B; C; D}. In this way, the size of the database can be reduced. The trimmed transactions are sent to the hash table filter module for hash table building.

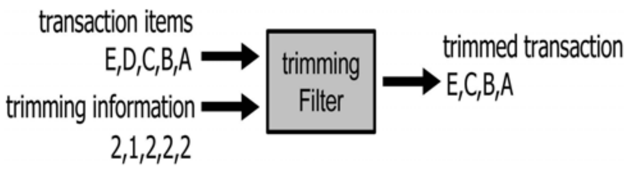


Fig. 6 The trimming filter

4.3Hash Table Filtering

To build a hardware hash table filter, we use a hash value generator and hash table updating module. The former generates all the k-itemset combinations of the transactions and puts the k-itemsets into the hash function to create the corresponding hash values. As shown in Fig. 9, the hash value generator comprises a transaction memory, a state machine, an index array, and a hash function. The transaction memory stores all the items of a transaction. The state machine is the controller that generates control signals of different lengths (k= 2; 3 . . .Þ flexibly. Then, the control signals are fed into the index array. To generate a k-itemset, the first k entries in the index array are utilized. The values in the index array are the indices of the transaction memory. The item selected by the ith entry of the index array is the ith item in a k-itemset. By changing the values in the index array, the state machine can generate different combinations of k-itemsets from the transaction. The procedure starts by loading a transaction into the transaction memory. Then, the values in the index array are reset, and the state machine starts to generate control signals. The values in the index array are changed by the different states. Each item in the generated itemset is passed to the hash function through the multiplexer. The hash function takes some bits from the incoming k-itemsets to calculate the hash values.

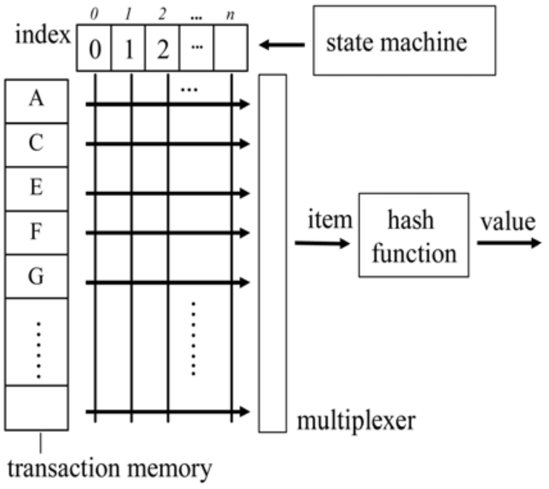


Fig. 7 The hash value generator

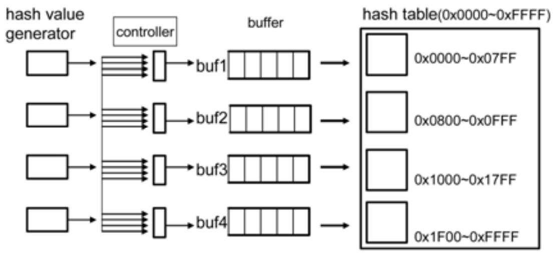


Fig8 The parallel hash table building module.

Consider the example in Fig 8. We assume the current k is 3. The first three index entries in the index array are used in this case. The transaction fA; C; E; F; Gg is loaded into the transaction memory. The values in the index array are initiated to 0, 1, and 2, respectively, so that the first itemset generated is < ACE > . Then, the state machine changes the values in the index array. The following numbers in the index array will be < 0; 1; 3 > , < 0; 1; 4 > , < 0; 2; 3 > , < 0; 2; 4 > , to name a few. Therefore, the corresponding itemsets are < ACF > , < ACG > , < AEF > , < AEG > , and so on. The hash values generated by the hash value generator are passed to the hash table updating module. To speed up the process of hash table building, we utilize N parallel hash value generators so that the hash values can be generated simultaneously. In addition, the hash table is divided into several parts to increase the throughput of hash table building. Each part of the hash table contains a range of hash values, and the controller passes the incoming hash values to the buffer they belong to. These hash values are taken as indexes of the hash table to accumulate the values in the table. There are four parallel hash value generators. The size of the whole hash table is 65,536, and it is divided into four parts. Thus, the range of each part is 16,384. If the incoming hash value is 5, it belongs to the first part of the hash table. The controller would pass the value to buffer 1. If there are parallel accesses to the hash table at the same time, only one access can be executed. The others will be delayed and be handled as soon as possible. The delayed itemsets are stored in the buffer temporally. Whenever the access port of hash table is free, the delayed itemsets are put into the hash table. After all the candidate k-itemsets have been generated, they are pruned by the hash table filter. Each candidate item set is hashed by the hash function. By querying the number of itemsets in the bucket with the corresponding hash value, the candidate item set is pruned if the number of itemsets in the bucket does not meet the minimum support criteria. Therefore, the number of the candidate itemsets can be reduced effectively with the help of the hash table filter.

5. Performance Evaluation

Initially, we conduct experiments to evaluate the performance of several schemes in the HAPPI architecture and DC. The testing data sets are T10I4D100 with different numbers of items in the database. The minimum support is set to 0.5 percent. As shown in Fig. 11, the four different schemes are

1. The DC scheme,
2. The systolic array with a trimming filter,
3. The combined scheme made up of the systolic array, The trimming filter and the hash table filter, and
4. The overall HAPPI architecture with the pipeline design

CONCLUSION

We have proposed the hashing and Pipelining technique for Association rule mining.we apply the pipelining methodology in The HAAPI architecture to compare itemsets with the database and collect useful information to reduce the number of candidate itemsets and items in the database concurrently.HAPPI can reduce infrequent items in the transactions tions and reduce the size of the database progressively by utilizing the trimming filter.Next HAPPI can effectively eliminate infrequent candidate itemsets with the help of the hash table filter.Finally Hashing And Pipelining solves the Performance bottleneck problem and acquire good scalability.

References:

[1]R. Agarwal, C. Aggarwal, and V. Prasad, “A Tree Projection Algorithm for Generation of Frequent Itemsets,” *J. Parallel and Distributed Computing,* 2000.

[2] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” *Proc. 20th Int'l Conf. Very Large Databases(VLDB),* 1994.

[3] Z.K. Baker and V.K. Prasanna, “Efficient Hardware Data Mining with the Apriori Algorithm on FPGAS,” *Proc. 13th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM),* 2005.

[4] S. Cong, J. Han, J. Hoeflinger, and D. Padua, “A Sampling-Based Framework for Parallel Data Mining,” *Proc. 10th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP '05),* June 2005.

Authors Biography

Mamatha.N received M,Sc degree in computer science from Andhra university in 2007 .where she is currently working towards M.tech degree in computer science. Avanthi College of Engg & Tech, Tamaram, Visakhapatnam, A.P., India.

Satya P Kumar Somayajula, working as Asst. Professor, in CSE Department, Avanthi College of Engg & Tech, Tamaram, Visakhapatnam, A.P., India. He has received his M.Sc(Physics) from Andhra University, Visakhapatnam and M.Tech (CST) from Gandhi Institute of Technology And Management University (GITAM), Visakhapatnam, A.P., INDIA. His research areas include Image processing, network security and neural networks.

Dr. C.P.V.N.J Mohan Rao is Professor in the Department of Computer Science and Engineering and Principal of Avanthi Institute of Engineering & Technology - Narsipatnam. He did his PhD from Andhra University and his research interests include Image Processing, Networks, Information security, Data Mining and Software Engineering. He has guided more than 50 M.Tech Projects and currently guiding four research scholars for Ph.D. He received many honors and he has been the member for many expert committees, member of many professional bodies and Resource person for various organizations.