An Efficient Load Balancing Multi-core Frequent Patterns Mining Algorithm

Kun-Ming Yu, Shu-Hao Wu
Department of Computer Science and Information Engineering, Chung Hua University
Hsinchu, Taiwan
yu@chu.edu.tw, wushuhaur@gmail.com

***Abstract*—Mining frequent pattern from transactional database is an important problem in data mining. Many methods have been proposed to solve this problem. However, the computation time still increase significantly while the data size grows. Therefore, parallel computing is a good strategy to solve this problem. Researchers have proposed various parallel and distributed algorithms on cluster system, grid system. However, the construction and maintenance cost is pretty high. In this paper, a multi-core load balancing frequent pattern mining algorithm is presented. The main goal of the proposed algorithm is to reduce the massive duplicated candidates generated in previous method. In order to verify the performance, we also implemented the proposed algorithm as well as previous methods for comparison. The experimental results showed that our method could reduce the computation time dramatically with more threads. Moreover, we could observe that the workload was equally dispatched to each computing unit.**

Keywords: multi-core; frequent pattern mining; load balancing; association rules

I. INTRODUCTION

In the digital era, much of the information is large and fast output. To discover the important information is much harder than before. Data mining is a useful technique which could help people to find and discover hidden rules from data. Applying the results of data mining in the planning of company's strategy could effectively increase the profit and reduce the risks.

There are many techniques had been proposed to discover the essential information from transaction-oriented database for mining association rules, time series, classification, etc…. However, the performance is the key issue while mining large data set. Many methods also were proposed to improve the efficiency [5, 6, 7]. On the other hand, the technological advances in hardware architectures are from single core, multi-core, cluster system [8, 9] to cloud system, and graphical processing units (GPU) [13]. These improvements could provide more computation power.

This work is partially supported by the National Science Council NSC 99-2221-E-216 -012 -MY2

In order to achieve the better performance in multi-core computing system, the traditional data mining algorithm must be re-designed in order to take the advantages of multi-core hardware architectures. Firstly, to avoid duplicated computation tasks and CPU idle time are critical issue. In this paper, three strategies were proposed to solve these problems to achieve better performance in multi-core system. The main concept is to avoid duplicated candidate in candidate generation stage. Moreover, in order to verify the performance of proposed algorithm, we also implemented our proposed algorithm and compared with other algorithms.

This paper is organized as follows: in Section 2, Apriori, AprioriTid algorithms are described. Then, the proposed algorithm is introduced in Section 3 and Section 4 illustrates our experimental results. Finally, the conclusion is given in Section 5.

II. RELATED WORKS

Most of classical data mining algorithms were designed for single-core computer architectures; however, the computation time is pretty long while the data set is large. Therefore, various methods based on distributed computing system were proposed to improve the performance. Many parallel algorithms have been proposed to speed up the computation time of data mining algorithm, but most of them are designed for computer clusters or grid systems.

In cluster system, multiple computers on the network were linked to share the computation and storage resources. But it is hard to balance the workload of each node, and the build and maintenance cost is too expensive for small or medium business. In recent years, GPU was a popular parallel architecture for high performance computing as well as data mining [13]. However, the memory access cost is large, and the memory size is small. Therefore, effectively using the memory is the key to utilize the power of GPU.

Apriori algorithm [1, 2] and Frequent-Pattern Tree (FP-tree) algorithm [3] are two commonly used techniques for data mining. Apriori algorithm is simple, easy to understand, and easy to implement although it requires repeatedly scanning of database. However, Apriori algorithm is more suitable for parallel processing than FP-tree method since balancing the load while parallelizing

978-0-7695-4600-1/11 $26.00 © 2011 IEEE
DOI 10.1109/TrustCom.2011.192

tree structures has been a difficult issue of FP-tree [10]. However, Apriori algorithm has the fast computation acceleration problem when data set size increases. While mining the database, some important information may be removed with high threshold value encourages low threshold pattern mining which costs much computing time.

Apriori algorithm is the representative association rule mining algorithm which was proposed by Agrawal in 1994. First of all, a threshold is given; the threshold is the number of frequent of candidate itemsets appeared in the database. When the threshold was set, Apriori used it to filter the candidate itemsets. In the initial stage, the candidate itemsets is counted from each item in database. If the support is higher than threshold, the itemsets were selected. In the next stage, new candidate itemsets are merged and generated. Repeat above steps, we could find all of the frequent patterns that the support is greater than given threshold.

Another important concept is Transaction Identification (TID) [10]. In order to count how many times of the itemset appeared in the database, we need to go through whole database. Therefore, we can use TID to shorten the time of database scan process by creating the TID table. In the TID table, the key is ItemID, and the value is TID. Therefore, we could quickly compute the number of itemset appeared in the database.

There were many multi-thread strategies was proposed to balance the loading [11, 12, 17, 18]. Ye and Chiang [18] proposed a parallel-distributed algorithm based on the Trie tree. Their algorithm distributes workloads according to the Trie tree to balance workload and speed-up the computation. However, the items are distributed to the nodes only based on the first level of the Trie tree. This may cause the sizes of candidate itemsets (workloads) among processors significantly varying. Moreover, this method also requires a database to be scanned many times.

Chia [15] also proposed a work-balanced data mining algorithm. It reduces database scanning time by pruning the candidate itemsets generation to speed up mining association rule. The algorithm will not generate duplicated candidate itemsets. It could save the candidate itemsets verification time.

III. PROPOSED ALGORITHM

At present time, most of computers is multi-core architecture, but the traditional data mining algorithm couldn't use effectively of the multi-core; therefore, we present the Multi-Core Apriori Transaction Identifiers algorithm (MATI) in the paper. The key issue of MATI is how to enhance the efficiency of Apriori on the multi-core architecture.

The process of Apriori algorithm can be divided into 2 parts: (1) generate candidate itemsets and (2) check the candidate itemsets is frequent or not. In previous works [9, 10], we realize that part (2) can be completed in a short time. Therefore, we focus on how to reduce the scan range on database to enhance efficiency for candidate generating (part 1). In MATI, we proposed two strategies to reducing the processing time of generating candidate itemsets.

A. Itemset Block

In the process of generating candidate itemsets in Apriori algorithm, it used two frequent itemsets of current level, say k-itemset, to merge them to produce candidate (k+1)-itemset, and then to filter the generated candidate (k+1)-itemsets is frequent or not. It is not necessary to generate all possible (k+1)-itemsets' candidates, we only need to merge these useful frequent itemsets.

We divide frequent itemsets into multiple blocks, all frequent itemsets with the same first item are put into the same block, and each block uses the same index. We call them as itemset block. The candidate (k+1)-itemsets are generated in same block only; these frequent itemsets will not existed in different block.

B. Task Dispatches

MATI algorithm can reduce unnecessary burdens by avoiding assign the same task to different cores. More importantly, the strategy can be more effective to balance the work load in the system.

The itemset has the same index in the itemset block. The frequent itemsets in the same itemset block will be generated on the same core in MATI. It avoids same index frequent itemsets of data distributed on different core.

MATI composes of two functions; the first one is similar to Apriori algorithm. It processes roughly the same as Apriori, but it used TID table to check the candidate itemsets is frequent or not. The second one (merging) is performing the process of two frequent k-itemsets merged into candidate (k+1)-itemsets.

```
MATI: main function
Structure a TID table and scan database
T = a transaction database over the set of items
K = 1; C_k = T; // K is merge round, C_k is candidate itemsets in K round
while C_k number > 1 do
   allocated candidate itemsets to each thread
   foreach thread do
      scan TID table
      if candidate itemset count > min support
         candidate itemset push F_k
//F_k is frequent itemsets in K round
      end
   end
   collect each thread's frequent itemsets
   allocated frequent itemsets to each thread
   foreach thread do
      Merge_function
   end
   collect each thread's candidate itemsets
   k++
end
```

```
Merge function
if k>1
    foreach frequent_itemset do
        index_item = frequent_itemset.item1
        M_frequent_itemset = next frequent_itemset
      while index_item == M_frequent_itemset.item1 do
          if
frequent_itemset.item1==M_frequent_itemset.item1,…,
frequent_itemset.itemk-1==M_frequent_itemset.itemk-1        &&
frequent_itemset.itemk<M_frequent_itemset.itemk
     frequent_itemset merged with M_frequent_itemset
            else
              break;
              end
        M_frequent_itemset = next M_frequent_itemset
        end
    end
else
  foreach frequent_itemset do
    M_frequent_itemset = next frequent_itemset
     while M_frequent_itemset != NULL do
       frequent_itemset merged with M_frequent_itemset
       M_frequent_itemset = next M_frequent_itemset
       end
   end
end
```

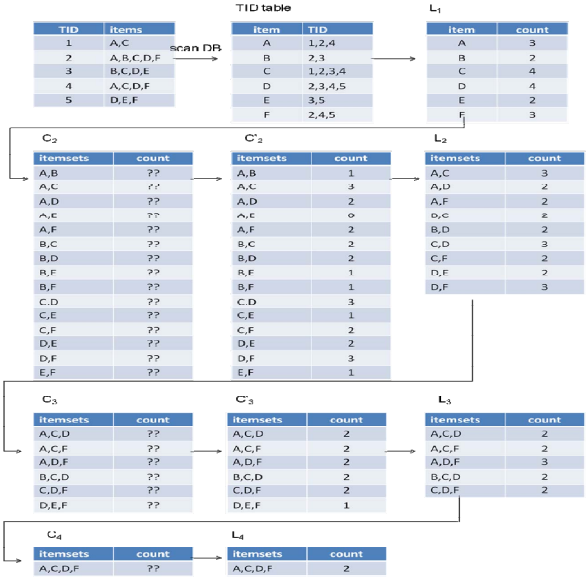

Figure 1. An execution example of MATI.

Figure 1 illustrates the procedure of MATI, in the example, database has five transactions and five items, items from A to F, and the support is set to 40%. When MATI executes, it scans the database and builds the corresponding TID table. When the TID table is created, MATI can check the candidate itemsets rapidly from TID table. In figure 1, the L1 denotes the number of occurrence of 1-itemset in database, and all 1-itemset are frequent in the example. Then, MATI used L1 frequent item to generate table C_2, which contains only name of possible candidate 2-itemset. After C_2 was generated, MATI checks the TID table to fill in the number of occurrence in the table and then produce C_2'. Moreover, MATI prune the candidate itemsets, if candidate itemsets' support is less than the minimum support, the result is shown as L_2. After finishing generated L_2, we can obtain four itemset blocks, such as, itemset block 1 = {AC, AD, AF}, itemset block 2 = {BC, BD}, itemset block 3 = {CD, CF}, itemset block 4 = {DE, DF}. When merging {AC}, {AD}, {AF} is proceeded, the first items is the same, therefore, MATI combined them into {ACD}, {ACF}, {ADF} in L_3. MATI repeats above process until non-frequent itemsets is found. L_4 is the final result in the example.

IV. EXPERIMENTAL RESULTS

In order to validate the efficiency of MATI, we implemented it in a multi-core architecture and compared the performance with the algorithm proposed by Chai [15]. The specification of simulation environment is as the followings: Intel(R) Core(TM) i5 CPU 650 @ 3.20GHz equips with 4G memory and 4MB cache memory. The experiment data is generated by IBM data generator [4].

In the experiments, we compared execution time against different size of dataset, transaction numbers, average transaction length, different threshold, and thread numbers. In experiment, para_TID denoted paralleled version of Apriori algorithm with TID; para_Chai denoted paralleled version of Chai algorithm.

The computation time is shown on Figure 2 when using the same data and threshold but different number of threads. MATI only takes 80 seconds when the number of thread is 2, while para-Chai takes more than 60000 seconds. When the number of threads increases, the execution time decreased to a stable state. In figure 2, we can observe that MATI performs best.

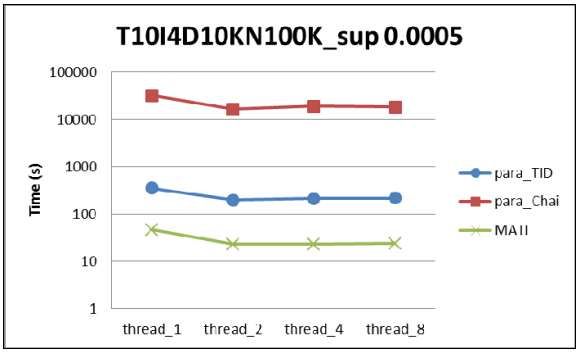

Figure 2. The execution time of different number of threads. (Sup=0.0005)

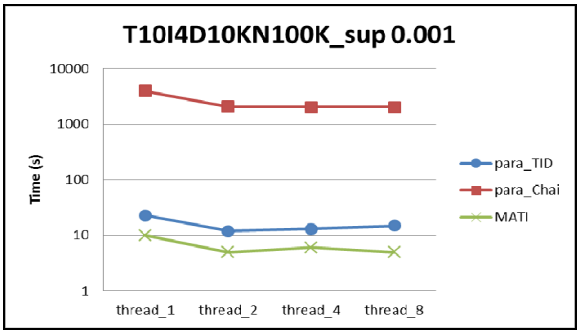

Figure 3. The execution time of different number of threads (Sup=0.001)

Figure 3 depicts the execution time when support is set to 0.001. We can see that MATI still has the smallest executing time, but the execution time of para_TID is lower compared with Figure 2. The reason is the threshold becomes higher lead to the number of frequent itemsets reduced. From figure 4, we can observe that the execution time of para_Chai decreased significantly when the value of support increased. However, MATI still has better performance than others.

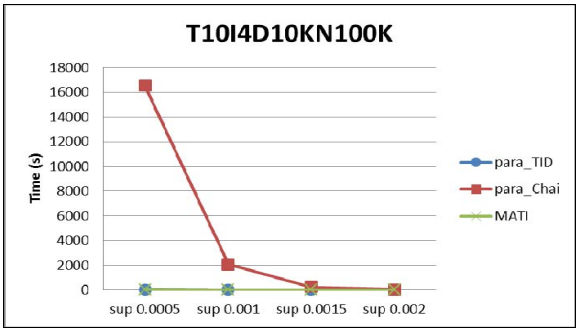

Figure 4. The execution time of different support.

For the workload balance issue, we can observe in figure 5. Since MATI is capable to generate uniform number of candidates in each thread, the workload is evenly distributed in each core compared with para-Chai. Figure 5 is the execution result of dataset T10I4D10KN100K, threshold is 0.0005, eight threads, and generated in level 2 (2-itemset).

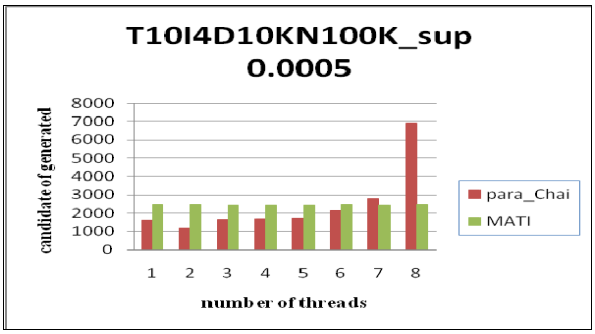

Figure 5. Number of candidate items in 2-itemsets.

From the experimental results, we can conclude that MATI has more effective compared with para-Chai on multi-core architecture. MATI has 22.85 times of speed-up ratio compared with para_TID.

V. CONCLUSIONS

Mining frequent patters from transactional database is an important problem in data mining research. Many sequential or parallel methods had been proposed to solve this problem. However, the execution time increases quickly if the data set grows. Therefore, in this paper, we proposed a multi-core based parallel frequent pattern mining algorithm, MATI. The experimental results showed that MATI could reduce the computation time if more cores are used. We also observed that the workload was balanced among each thread in MATI.

REFERENGES

[1] Agawal, R., Imilinski, T., and Swami, A. "Mining Association Rules between Sets of Items in Large Database," Proceeding of the 1993 ACM SIGMOD International Conference on Management of Data, Vol. 22, Issue 2, June 1993, pp. 207-216.

[2] Agrawal, R. and Srikant, R. "Fast Algorithms for Mining Association Rules," Proceeding of 20th international conference on Very Large Database, 1994, pp. 487-499.

[3] A. Javed, and A. Khokhar, "Frequent pattern mining on message passing multiprocessor systems," *Distributed and Parallel Databases,* vol. 16, no. 3, 2004, pp. 321-334.

[4] Almaden, "I. Quest synthetic data generation code," http://www.almaden.ibm.com/cs/quest/syndata.html

[5] A. M. J. Md. Zubair Rahman and P. Balasubramanie, "An Efficient Algorithm for Mining Maximal Frequent Item Sets," Journal of Computer Science 4 ISSN 1549-3636, 2008, pp.638-645.

[6] Bodon, F. "A fast Apriori implementation," Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, 2003.

[7] G. Liu, H. Lu, Y. Xu, and J. X. Yu, "Ascending Frequency Ordered Prefix-tree: Efficient Mining of Frequent Patterns," In Proc. 8th Int. Conf. Database Systems for Advanced Applications, 2003, pp. 65-72.

[8] Jiayi Zhou, Kun-Ming Yu* and Bin-Chang Wu, "Parallel Frequent Patterns Mining Algorithm on GPU", IEEE International Conference on Systems Man and Cybernetics (SMC2010), 2010, pp. 435-440.

[9] Jiayi Zhou and Kun-Ming Yu, "Tidset-Based Parallel FP-tree Algorithm for the Frequent Pattern Mining Problem on PC Clusters," Proceeding of 3rd international conference on grid and pervasive computing, 2008, pp. 18-28.

[10] K.-M. Yu, J. Zhou, T.-P. Hong, "A Load-Balanced Distributed Parallel Mining Algorithm," *Expert Systems with Applications,* vol. 37, no. 3, 2009, pp. 2486-2494.

[11] Kun-Ming Yu, Jiayi Zhou, Chun-Yuan Lin, and Chuan Yi Tang, "Efficient Parallel Branch-and-Bound Algorithm for Constructing Minimum Ultrametric Trees," Journal of Parallel and Distributed Computing, Vol 69, Issue 11, November 2009, pp. 905-914.

[12] Kun-Ming Yu, Jiayi Zhou and Wei Chen Hsiao, "Load Balancing Approach Parallel Algorithm for Frequent Pattern Mining," Parallel Computing Technologies, 2007, pp. 623-631.

[13] Rabenseifner, R., Hager, G., Jost, G. "Hybrid MPI/OpenMP

Parallel Programming on Clusters of Multi-Core SMP Nodes," Euromicro International Conference on Parallel, Distributed and Network-based Processing, May 2009, pp.427-436.

[14] Ravi, V.T., Agrawal, G., "Performance Issues in Parallelizing Data-Intensive Applications on a Multi-core Cluster" IEEE/ACM International Symposium on Cluster Computing and the Grid, June 2009, pp.308-315.

[15] Sheng Chai, Jia Yang, Yang Cheng, "The Research of Improved Apriori Algorithm for Mining Association Rules," IEEE International Conference on Communication Technology, December 2008, pp.513-516.

[16] S. Prakash, R. M. S. Parvathi, "An Enhanced Scaling Apriori for Association Rule Mining Efficiency," European Journal of Scientific Research ISSN 1450-246X, Vol.39, No.2, 2010, pp.257-264.

[17] Yan Zhang, Jing Chen, "AVI: Based on the vertical and intersection operation of the improved Apriori algorithm," International Conference on Future Computer and Communication, June 2010, pp.V2-718-721.

[18] Ye, Y. and Chiang, C.C. "A Parallel Apriori Algorithm for Frequent Itemsets Mining," Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications, pp. 87-94.