Performance Analysis of Parallel Mining for Association Rules on Heterogeneous System

Rakhi Garg[1], P.K.Mishra[2]
[1]Computer Science Section, MMV, Banaras Hindu University,
[2]Department of Computer Science, Banaras Hindu University,
Varanasi-221005, India
{rgarg, mishra}@bhu.ac.in

Abstract**:** Association Rule Mining plays an important role in predicting business trends those can occur in near future because it finds the hidden relationships among items in the transactions. Several sequential algorithms have been developed for finding maximal frequent itemsets and generating association rules. Due to advent of high storage devices large database can be stored. Parallel algorithms are very promising to mine these huge databases. Par-MaxClique, a parallel association rule mining algorithm is developed, uses static load balancing. In this paper we propose a simple parallel algorithm for association rule mining on heterogeneous system with dynamic load balancing based on Par-MaxClique algorithm. We compare our algorithm with the existing one for homogeneous environment and observed that the execution time gets reduced dramatically**.**

Keywords: Parallel association rule mining, heterogeneous system, Par-MaxClique algorithm

1. Introduction

Most of the parallel association rule mining algorithm developed so far uses static load balancing for homogeneous systems [12]. In static load balancing the job is initially partitioned among the homogeneous processors using some heuristics. There is no data movement among the processors during execution.

Moreover, if we apply the parallel algorithm developed for homogeneous system to heterogeneous environment, it will again leads to significant performance deterioration [1]. Since in homogeneous system there is an equal distribution of job among the processors of the same speed, uses static load balancing technique whereas heterogeneous system has processors of different speeds in which one completes job earlier than the other due to speed mismatch [4]. The high speed processor executes the assigned job quickly and sits idle while low speed processor is still busy with the assigned job that degrades the performance of the system. To utilize system processors efficiently and enhance the performance we design an algorithm that during execution checks the load of the processor and on the basis of which it moves the job from heavy loaded processor to least loaded one so that no processor sits idle till the completion of the whole jobs in a system.

In our algorithm, initially, the same number of jobs assigned to all the processors in a cluster by the scheduler using the same heuristics as in the homogeneous system. Since the processing speeds of the processors in the cluster are different so algorithm first finds out the fastest processor in the cluster and also computes the execution time to complete the execution of all the jobs assigned to it. After that it will compute the total number of complete and incomplete jobs of all the processors in the system and maintain load value of each processor in the cluster at the scheduler end i.e. the host. Then the load value of processors in a cluster are compared and the job is moved from the heavy loaded processor to the least loaded one and thus balances load dynamically in a cluster. A linked list containing the load values of all processors in a cluster are maintained at the scheduler end that gets updated during the completion of all the jobs assigned to the fastest processor. In this way load balancing becomes dynamic and involves data movement among the processor only when there is no communication overhead to enhance the performance of the system.

Section 2 briefly explains Par-MaxClique algorithm and focuses on the related work done. In section 3 we explain the functioning of the algorithm designed by us. Our experimental study is presented in section 4 and our conclusion in section 5.

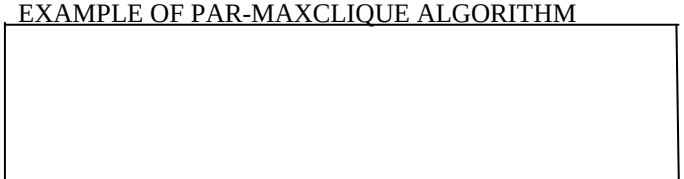
2. Par-MaxClique Algorithm & Related Work

2.1 Par-MaxClique Algorithm

M. Zaki, Parathasarathy, Oghihara and Li [2] developed Par-MaxClique algorithm that gives more accurate frequent itemsets. It uses clique clustering which is more accurate than equivalence class clustering [2],[7]. Here, the database is vertically partitioned and hybrid search is applied on it to generate the longest frequent itemsets by using the (L_2) frequent 2-itemsets and some non frequent itemsets. The items are organized in a subset lattice search space, which is decomposed into small independent chunks or sub-lattices, which can be solved in memory. Efficient lattice traversal techniques are used, which quickly identify all the frequent itemsets via simple tid-list intersections [2].

Basically Par-MaxClique algorithm is divided into three phases i.e. initialization phase, asynchronous phase and final reduction phase [2],[7]. It generates clusters from L_2 using uniform hypergraph cliques and partition the clusters and the tid-list among the processors in the very first phase called the initialization phase. After that in the next phase called the asynchronous phase, the frequent itemsets are computed independently by each processors from the cliques assigned to it. Finally, the last phase i.e. the reduction phase produces the aggregate results and outputs the associations between the frequent itemsets.

EXAMPLE OF PAR-MAXCLIQUE ALGORITHM



Let database contains A,C,D,T and W four itemsets and 6 transactions are:-

Transactions	A	C	D	T	W
T1	1	1	0	1	1
T2	0	1	1	0	1
T3	1	1	0	1	1
T4	1	1	1	0	1
T5	1	1	1	1	1
T6	0	1	1	1	0

Tid-list is computed as: T(A) = {1,3,4,5}; T(C)={1,2,3,4,5,6}; T(D)={2,4,5,6} and T(W)={1,2,3,4,5}. During the initialization phase the tid-list is communicated among the processors and support counts for 2-itemsets are read. e.g. support count for AC ={1,3,4,5} = 4 which is counted by the intersection of the tid list of A and C. Similarly the support counts of AD, AT, AW, CD, CT, CW, DT, DW and TW are 2,3,4,3,4,4,3,2,3 and 3 respectively. Let us assume that minimum support = 3 so AD and DT will be discarded.

Frequent 2- itemsets are :-
AC,AT,AW,CD,CT,CW,DW,TW

Equivalence classes are:-
[A]: C T W
[C]: D T W
[D]: W
[T]: W

By applying the hypergraph clique for clustering to L2, the set of potential maximal cliques per equivalence class are generated.

Generated Maximal cliques per class:-
[A]: ACTW, ACW, ATW, ACT
[C]: CDW, CTW
Maximal cliques for equivalence class A

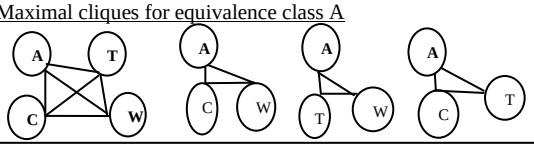


Figure 1: Equivalence class and Uniform Clique clustering [10]

Here, two cliques and equivalence class are generated which are distributed on the processors to achieve equal load balancing. Each processor independently computes the maximal frequent itemsets which are used in association rule generation in the last phase of the algorithm.

Par-MaxClique algorithm uses the static load balancing technique with some heuristics for equal balance among the processors in the homogeneous system. This is far from reality because a database server has multiple systems with different configurations and speeds. If this algorithm is used there then it will degrade the performance of the system. This demands the dynamic load balancing schemes.

We have developed an algorithm for parallel mining of the association rules for such heterogeneous system that uses dynamic load balancing technique and enhances the system performance by reducing the execution time.

2.2 Related Work

Several parallel algorithms for association rules have been proposed in the literature. The most known parallel algorithm is Count Distribution (CD), Data distribution and Candidate Distribution; proposed by Rakesh Agrawal and J. Shafer [4], [5], [6]. Among these CD is the most promising one which minimizes the communication overheads but utilize memory less efficiently than DD.

The FDM and FPM algorithms are the enhanced versions of CD [3]. In FDM, two rounds of the communications are required in each iteration one for computing the global support and the other for broadcasting the frequent itemsets. FPM is more efficient than FDM in communication which broadcast local supports to all processors which is determined by the candidate size as in CD [3]. Thus for small minimum support, the communication cost could be very high at some passes where the candidate set is large.

Par-MaxEclat and Par-MaxClique is parallel MFI (maximal frequent itemsets) mining algorithm proposed which are parallel versions of MaxEclat and MaxClique respectively. These algorithms distribute over the processor in the system the cluster of the generated potential maximal frequent itemsets. These algorithms are implemented on dedicated homogeneous system which uses static load balancing technique. Par-MaxClique algorithm outperforms CD algorithm because it utilizes the aggregate memory of the parallel system, decouples the processors right in the beginning by repartitioning the database so that each processor can compute independently, use vertical database layout which clusters the transactions containing an itemset into tid-list without scanning the database and computes the frequent itemsets by simple intersections on two tid-lists without having an overhead of maintaining complex data structures[2].

Problem here is that although Par-MaxClique algorithm outperforms but it has limitation that it is only implemented for homogeneous system that uses static load balancing technique. It won't take care of fault tolerance i.e. what happens if one of the processor in the system fails, how the jobs assigned to it gets executed and also what happens in the case of heterogeneous system which have processors with speed mismatch. If it used in heterogeneous system with no check on the load factor of the processor maintained during execution phase then it might happen the processors with high speed may sit idle after completing the execution of all the jobs assigned to it while others with less speed are still involved in the processing work. This won't utilize the processor to their maximum extent. Hence an algorithm which uses dynamic load balancing technique is needed for proper utilization of all the processors in the cluster.

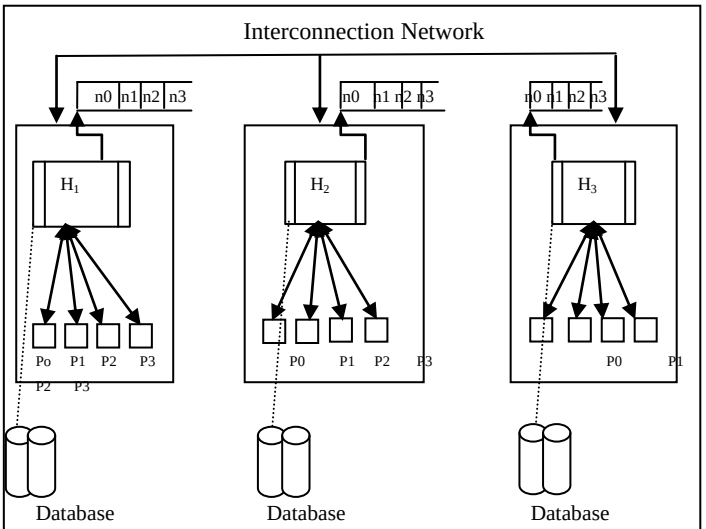
Load Balancing FP-Tree (LFP-tree) algorithm is proposed by Kun-Ming Yu, Jiayi Zhou and Wei Chen Hsiao based on FP-tree structure that divides the item set for mining by evaluating the tree's width and depth and proposed a simple and trusty calculate formulation for loading degree [8]. But it has limitation of maintaining the complex tree structure.

Masaru Kitsuregawa and Takahilus Shintani, Masahisa Tamura and Iko Pramudiono, proposed Parallel Data Mining on large scale PC Cluster, the dynamic load balancing methods for association rule mining for heterogeneous system [9] which uses candidate migration and transaction migration. Initially if load is not balanced after candidate migration then it applies the transaction migration which is costly but more effective for strong imbalance.

3. Proposed Algorithm

In our algorithm, the hypercliques of frequent 2-itemsets which are considered as jobs are equally divided among the processors of the system for having equal load balance as done in the homogeneous system. During execution we find out the processor which has completed the execution of all the jobs assigned to it i.e. the fastest processor of the cluster. Then we arrange the processors in the decreasing order according to their respective speeds and compute the number of complete and incomplete jobs of every processor at a time when fastest processor have completed the execution of all jobs assigned to it. After that the scheduler queue which is

maintained at the host to which numbers of processors are attached and contains the load value i.e. the number of incomplete jobs of every attached processor gets updated. After that the data is moved from heavy loaded i.e. the slowest processor to the least loaded one i.e. the fastest processor in the cluster only if the remaining execution time of the job assigned to the slowest processor is more than that of its execution time at the fastest processor. This takes care of communication overhead. Since we have distributed the hypercliques among the processors in the cluster for generating the maximal frequent itemsets (MFI), it might happen that the fastest processor have generated it at a time when others are involved in generating MFI from one of the cliques from the cluster of cliques assigned to it. In that case the remaining untouched cliques from the list of a given processor will move to the fastest processor for computation. This will engage all the processors of various speeds in the cluster which cannot be done by adopting the algorithm designed for the homogeneous system. In this way every processor in the cluster gets utilized to its maximum extent and also reduces the total execution time. e.g. Consider a case where (frequent 2-itemsets) L_2 = {12, 13, 14, 15, 23, 24, 25, 34, 35, 45} and two Processors P_o and P_1; where P_o is faster than P_1. For having equal load balance the clique of [1] get assigned to P_0 while [2] and [3] get assigned to P_1. It might happen that P_0 have generated all the MFI from clique [1] at a time when P_1 is busy in generating from [2] and [3] remained untouched. In that case [3] gets moved to P_0.



H_1, H_2 and H_3 are the host where the scheduler queue is maintained. n0, n1, n2 and n3 are the load factor of P_0, P_1, P_2 and P_3. Here, the database of tid-list is initially equally partitioned and the support count of all L_2 are available.
Each processor say P_0, P_1, P_2 and P_3 computes the maximal frequent itemsets from the cluster assigned to them.

Figure 2: Working of algorithm in Heterogeneous system

Table 1: Pseudo code for parallel association rule mining algorithm for heterogeneous environment

Begin
/* Initialization Phase */
1. Generate L_2 from 2-itemset support counts
2. Generate clusters from L_2 using uniform hypergraph cliques
3. Partition clusters among the processors
4. Scan local database partition
5. Transmit relevant tid-list to other processors
6. Received tid-list from other processors.
7. First, we compute the job queue and linked list of each processor and scheduler respectively. Initially, all processor have the same load value since jobs are equally distributed among the processors as in Par-MaxClique algorithm for homogeneous system.

/* Asynchronous Phase */
8. For each assigned cluster C_2, compute Frequent Itemsets
9. During execution, each processor updates its job queue and the linked list at the scheduler is also gets updated accordingly.

/* Communication OR Complete and offer Phase */
10. If job queue of all processors are empty then stop
11. else
The scheduler compares the load value of all the processors within the cluster and if any difference is found then perform the following :-
(i) Job from heavy loaded processor say P_i is taken and gets assigned to least loaded processor say P_j.
(ii) Job queues of the P_i and P_j are adjusted accordingly.
(iii) The link list at the scheduler is also adjusted accordingly.
12. Go to asynchronous phase i.e. step 8.

/* processing completes at each processor and then moves to reduction phase that involves 13.*/
13. Aggregate Results and Output Associations
14. STOP

4. Analysis of proposed algorithm

We have designed a simulator in C language that reads number of processor in the system, there processing speed and the number of jobs to be executed by the system. The execution time of each of the job is randomly generated. The major difference between the homogeneous and heterogeneous system is observed at the Communication OR Complete and offer Phase of the proposed algorithm where dynamic allocation of jobs are done in heterogeneous system and static in case of homogenous system. Initially our simulator distributes the jobs equally among all the processors in the system so that work load remains same at every processor and then computes the actual execution time of each processor as well as the computation time of all the jobs assigned to it for doing dynamic allocation. In the case of heterogeneous system the actual execution time of each processor is different whereas it remains same in the case of homogeneous system. So, simulator will list out the total number of incomplete jobs allocated to each processor at the time when the fastest processor has completed the execution of all the jobs assigned to it. On the basis of that the entry at the scheduler that keeps track of the work load factor of each of the processor in the system will be updated. After that simulator compares the remaining execution time of the incomplete jobs of each processor with its execution time at the fastest processor and if it is more then only the data movement will be done from that processor to the fastest processor otherwise not. In this way the fastest processor is not overloaded and this process repeats till all the jobs complete its execution. By doing so it will also take care of fault tolerance because if any of the processor is not completing its execution then after comparing it with the processor arranged in the decreasing order of their processing speed the job will be assigned to the one that involves in processing. Not only this, it also does the equal distribution of jobs among the processors in the system while

doing dynamic allocation so that no processor sits idle. We can observe it very well in figure 7.

We have executed algorithm in heterogeneous system having four processors with processing speeds 2.2GHz, 3.2GHz, 3.6GHz and 3.8GHz for the number of jobs executed ranging between 200 and 15,000 and also the same in homogeneous system with four processors with processing speed 2.2GHz, 3.2GHz, 3.6GHz and 3.8GHz respectively and obtain results shown in figure 3,4,5 and 6.

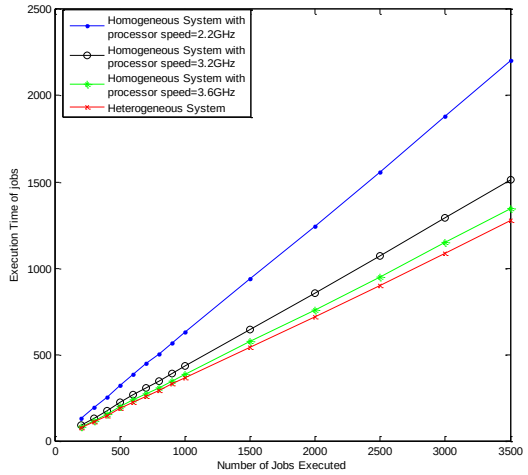


Figure 3

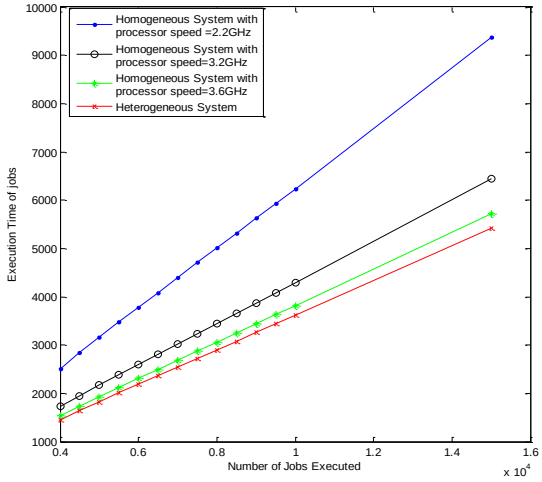


Figure 4

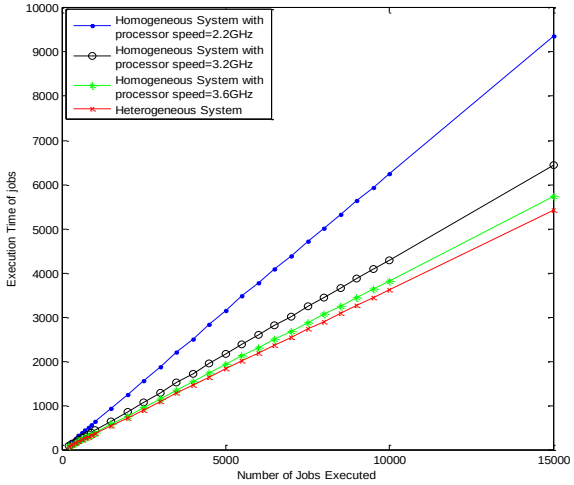


Figure 5

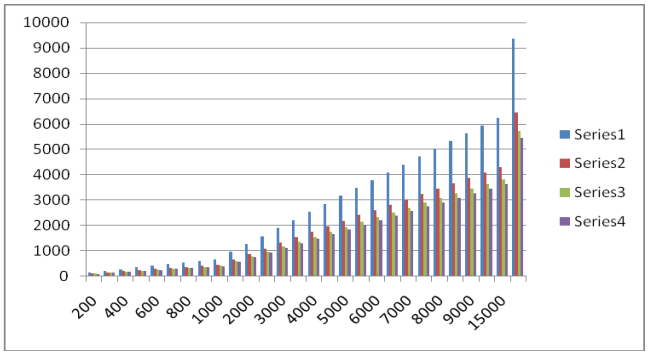


Figure 6

In figure 6, Series1, Series2 and Series3 represents homogeneous system having 4 processors with processing speed 2.2GHz, 3.2GHz and 3.8GHz respectively, Series 4 showing heterogeous system having 4 processors with processing speed 2.2GHz, 3.2GHz, 3.6GHz and 3.8GHz.

It is observed from figure 2 and 3 that the execution time reduces dramatically when the number of jobs increases above 10000 as compared to it range between 200 and 4000. It means that as the number of jobs increases the execution time reduces. Thus, we can say that the performance of the heterogeneous system which uses dynamic load balancing is much better than that of homogeneous system that uses static load balancing technique. Moreover, we can save cost also by having some low speed processors because instead of having all high speed processors, the same performance can be achieved by having a combination of low and high speed processors in the cluster. Not only is this it also seen that the high performance is obtained in the heterogeneous system as compared to the homogeneous with the involvement of the same number of processors in the data movement which is shown in figure 7.

5. Conclusion

In this algorithm, we have reduced the execution time. Moreover, the same performance can be achieved by the heterogeneous system with a combination of low and high speed processors as in homogeneous system with same number of high speed processors. It means we can utilize the low speed processors also for having the desired

performance. Hence the cost of having all high speed processors can be saved. Not only this, our algorithm also takes care of fault tolerance in the cluster. Also, it has good features of the Par-MaxClique parallel association rule mining algorithm for homogeneous environment which outperforms count distribution, data distribution and candidate distribution algorithm for parallel association rule mining. It enhances the performance of the heterogeneous system by having dynamic load balancing techniques.

In future, we try to perform dynamic load balancing in between the clusters. If the graph is too dense and if support decreases and transaction size increases it will affect the edge density and leads dense graph resulting in large cliques with significant overlap among them. We will try to handle this problem in our future work.

6. References

[1] Masahisa Tamura and Masaru Kitsuregawa, "Dynamic Load Balancing for Parallel Association Rule Mining on Heterogeneous PC Cluster System", Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, pp. 163, 1999.

[2] Mohammed J. Zaki, Srinivasan Parthasarthy, Mithsunori Ogihara and Wei Li, "Parallel Algorithms for Discovery of Association Rules", Data Mining and knowledge Discovery, © Kluwer Academic Publishers, pp. 360, 364, 1997.

[3] Soon M. Chung, Congnan Luo, "Efficient mining of maximal frequent itemsets from databases on a cluster of workstations", © Springer-Verlag London Limited 2007 pp. 359-391, Published online: 12 December 2007.

[4] M. J. Zaki, "Parallel and Distributed Association Mining: A Survey", Concurrency, IEEE, Volume 7, Issue 4, pp. 2-3, 10-13, Oct-Dec. 1999.

[5] R. Agrawal and J. Shafer, "Parallel mining association rules", IEEE Transaction On Knowledge and Data Engineering, Volume 8, Issue 6, 8(6): pp. 962-969, December 1996.

[6] E-H. Han, G. Karypis and Vipin Kumar, "Scalable parallel data mining for association rules", Proceedings of ACM SIGMOD Conf. Management of Data, pp. 279-284, May 1997.

[7] Mohammed J. Zaki, "Parallel and Distributed Data Mining: An Introduction", C.-T. Ho (Eds.): Large-Scale Parallel Data Mining © Springer-Verlag Berlin Heidelberg, LNAI 1759, pp. 9, 2000.

[8] Kun-Ming Yu, Jiayi Zhou and Wei Chen Hsiao, "Load Balancing Approach Parallel Algorithm for Frequent Pattern Mining", V. Malyshkin (Ed.): PaCT 2007. © Springer-Verlag Berlin Heidelberg. LNCS 4671, pp. 623–631, 2007.

[9] Masaru Kitsuregawa and Takahilus Shintani, Masahisa Tamura and Iko Pramudiono, "Parallel Data Mining on large scale PC Cluster", H. Lu and A. Zhou (Eds.): WAIM 2000, © Springer-Verlag Berlin Heidelberg, LNCS 1846, pp. 15–26, 2000.

[10] Rakhi Garg and P. K. Mishra ,"Parallel Association Rule Mining on Heterogeneous system", research papers published in an International Journal of Computer Application (0975 – 8887) , Volume-1, No. -14, pp. 87-91; Feb 2010.

[11] Jochen Hipp, Ulrich Gauntzer, Gholamreza Nakhaeizadeh,"Algorithms for Association Rule Mining-A General Survey and Comparison", SIGKDD explorations copyright © 2000, ACM SIGKDD, Volume 2, Issue 1, pp. 58-61, July 2000.