

Parallel Mining of Association Rules on heterogeneous cluster of workstations

Shipra Agrawal

M.E., Dept of Computer Science and Automation, IISC, Bangalore, India

shipra@csa.iisc.ernet.in

Abstract

Discovery of Association rules is an important data mining task. Several algorithms for association rule mining have been proposed in literature. Since the databases to be mined are often very large (measured in gigabytes and even terabytes), parallel algorithms are required. This project designs, implements and analyzes a simple asynchronous algorithm with dynamic load balancing based on Viper Algorithm for association rule mining. It shows the benefit of the asynchronous and dynamic load balancing algorithm with increased heterogeneity among the workstations.

1. Problem Definition

All extant algorithms for Parallel Mining of Association Rules use only a static load balancing scheme based on the initial data decomposition, and they assume a homogeneous, dedicated environment. Static load balancing initially partitions work among the processors using some heuristics; no subsequent data or computation movement is available to correct load imbalances.

This is far from reality. A typical parallel database server has multiple users and transient loads. This calls for an investigation of dynamic load-balancing schemes. Dynamic load balancing is also crucial in a heterogeneous environment. Such an environment might include meta-clusters and super-clusters, with machines ranging from ordinary workstations to supercomputers.

This project is an attempt to deal with the problem of Parallel mining of association rules in such a heterogeneous environment. It assumes no prior knowledge of relative processing speeds at various workstations involved. It assumes an environment where the mining process may be running on the background of other tasks, so that effective processing speeds of processors may even vary with time. Hence no prior distribution of workloads can be done based on some heuristic function (static load balancing), and the load balancing is essentially required to be dynamic.

2. Related Work

The main limitation of algorithms like Count-distribute[3] proposed previously is that it requires the processors to synchronize at the end of each pass. This approach does work well when the processors are of equal capability. But in a heterogeneous environment like ours it may not work well, as observed in this work.

The asynchronous algorithms like Candidate distribution [3] and New Parallel algorithms proposed by Mohammad Zaki [5] take the approach of partitioning the candidates and data among the processors after a few passes in such a way that that workload is divided approximately equally and processors work independent of each other for subsequent passes. Thus they are based on static load balancing and assume a homogeneous environment.

Our approach differs from the above in the way that we do not distribute the work equally among due to heterogeneous nature of workstations, moreover we cannot do a prior distribution of work as no knowledge of relative capabilities is assumed.

3. Algorithm

To meet the above requirements of heterogeneous environment, we design and implemented an asynchronous dynamic load balanced parallel version of the algorithm Viper[2]. The algorithm takes a simple approach for maximizing the flexibility in dynamic load balancing. Initially the tuples of the database exist distributed among the processors.

Pseudo code for Parallel Viper

Begin

/*Synchronous Phase*/

1. Generate 1-itemset support counts locally at each processor as in Viper, synchronize and count distribute to determine L1 , write snakelets for L1
2. Generate 2-itemset support counts locally, synchronize and count distribute to determine L2.

/*Communication phase*/

3. Communicate L1 snakelets between the processors. At end of the phase each processor has full L1 snakes.

/*Asynchronous phase*/

4. Cluster the itemsets of L2 into equivalence classes at each processor
5. Group the equivalence classes further into class groups each group consisting skip number of equivalence classes in sorted order.
6. At each processor P_i process class group $i, (i+n), (i+2n) \dots$ one by one ,until there are no more unprocessed class groups.

/*Complete and offer phase*/

7. On completion of work , P_i sends a 'DONE P_i ' message to all the other processors.
8. while there is an unfinished processor P_j
begin
 P_i sends an OFFER to the unfinished processor P_j
 P_j sends an unprocessed group no, if any, back to P_i and marks it processed
 P_i process the class group on behalf of P_j
end

/*Processing at all processors complete*/

4. Performance Study

The database used in our experiments was synthetically generated using the technique described in [1]. The parameters used in the synthetic generator and their default values are described in Table 2.

Parameter Symbol	Parameter Meaning	Value
N	No of items	1000
T	Mean Transaction Length	10
L	No. of frequent itemsets	2000
I	Mean frequent itemset length	4
D	No. of transactions	0.8 M

Table 2: Database Parameter Table

Our experiments were conducted on 4 Intel Pentium III (Coppermine) machines. To incorporate the effect of heterogeneity other processes were run in background.

4.1 Experiment: Comparison with Count-distribute on Viper

In our experiment, we evaluated the performance of our Asynchronous Algorithm and Count-distribute algorithm for parallelization of Viper for the above database. The results of this experiment are shown in Figure 1. These results correspond to $minsupport = .33\%$, $skip=16$. The values on X axis denote the no of processes running in background on one of the processors, that is amount of heterogeneity. It can be seen that our algorithm shows a better performance than Count-distribute with increased heterogeneity.

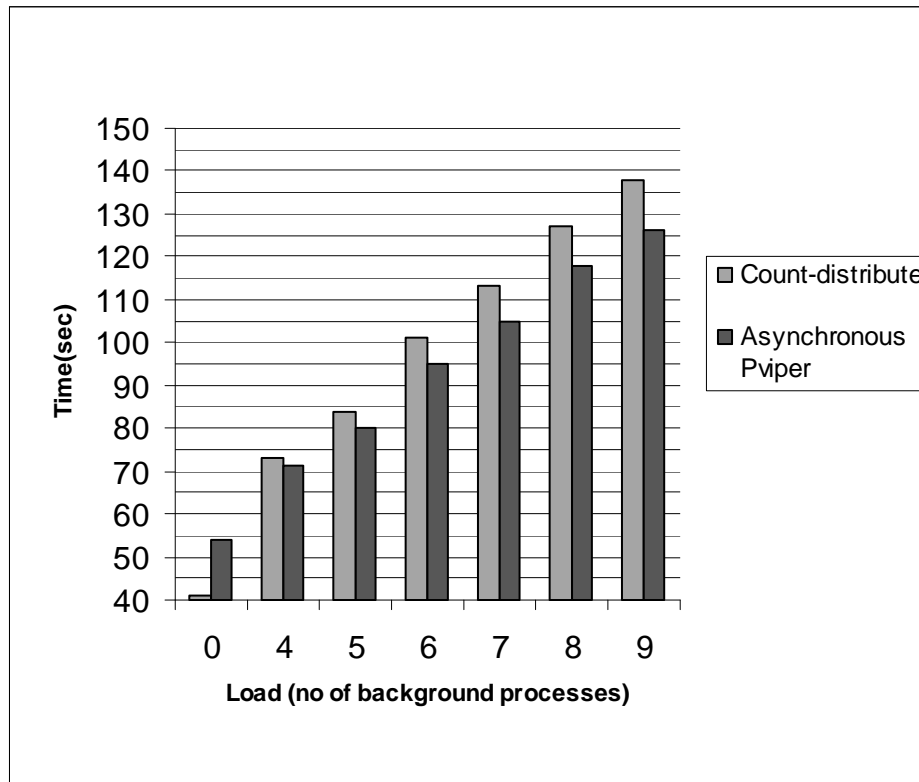


Figure 1 : Comparison of Count-distribute and Asynchronous PViper

5. Analysis of the effect of various features of the Algorithm

5.1 Count-distribute for Pass 1 and Pass 2

For equivalence class clustering, 2-itemset support counts are required. We generate them using count distribute in first two passes which requires synchronizing at the end of these passes. This forms a performance bottleneck for the new algorithm.

5.2 Communication step

The algorithm depends on communication of F_1 snakelets between the processors in the Communication phase which seems to be a large performance bottleneck. But the cost of this is somewhat mitigated due to following facts:

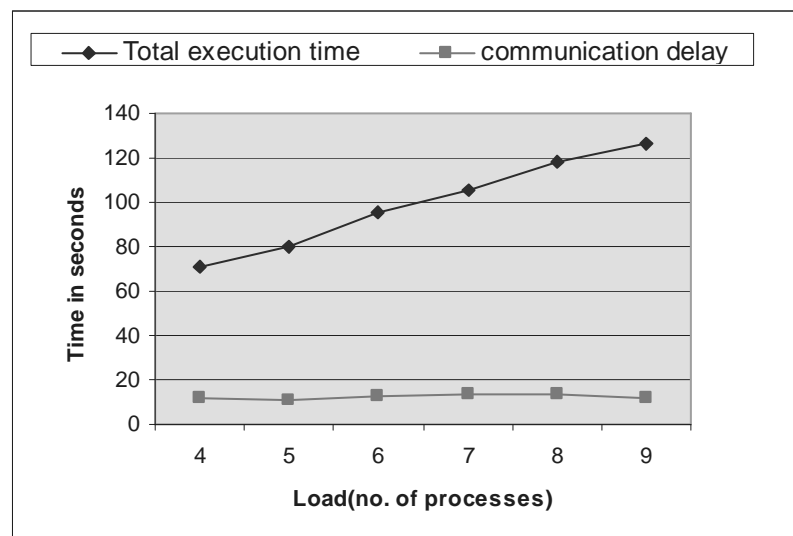
5.2.1 Smaller size of snakelets

Snakes are 'compressed tid vectors' and hence the communication cost for snakelets is less as compared to original data. Typically in our experiments for 39 MB of data, the size of snakelets was approximately 2.8MB.

5.2.2 Increase in communication delay not proportional to increase in total mining time

Additionally, when individual processors were loaded heavily by other running processes in background, the processing time was increased, but the communication delay was observed not to increase in a proportional manner. This can be because decrease in processing capability increases the writing and reading time and not the communication delay. Moreover communication is done once, whereas the decrease in processing speed of a processor effects processing time of all the passes on the slow processor. Also in an heterogeneous environment the fast processors can still communicate at fast speeds.

Graph 2 demonstrates this effect. As one of the processor was loaded more and more the execution time of the system increases but the communication delay shows minor variation.



5.3 Vertical Partitioning of Work - Effect of skip parameter

The algorithm divides the work among the processors through candidate distribution. Each processor processes only the candidates belonging to equivalence class groups selected by it, and this selection is mutually exclusive.

But the total work to be done by all the processors in Parallel Viper may be more than when Viper is run on a single processor sequentially for the whole database even if the communication cost is ignored. This is due to the following reasons:

5.3.1 Incomplete Pruning

Pruning of candidates can be done only on the candidates belonging to same equivalence class group.

5.3.2 Multiple accesses of common snakes

The snakes which are common to multiple equivalence class groups are accessed once for each such group, whereas in case of Sequential Viper every snake is read most once.

These effects are even more when the class groups are small and contain less number of equivalence classes, i.e. skip is decreased.

Hence increasing skip gives less execution time on individual processors. Thus we may intend to increase the skip to maximum possible, i.e. $(\text{total no of classes})/(\text{no of processors})$, i.e. process all the classes on one processor as a single class group.

But in an heterogeneous environment, this won't allow one processor to take over the work of other processor when its work is finished.

Our experiments were performed with constant skip=16. It was observed that in case one of the processors is made relatively very slow (more than 5-6 processes in background), the slow processor processed only a single class and after that was offered help by other processors. Hence the total time was equal to the time taken by the slow processor to process the single class, which is more when skip is large. Much more performance improvement could be observed in case of high heterogeneity had the skip been smaller. Thus for more flexibility in dynamic load balancing, skip is required to be small.

Hence this parameter needs to be tuned.

6. Conclusion

In this project, we addressed the problem of parallel mining of association rules in a heterogeneous cluster of workstations where no prior knowledge of relative capabilities is assumed. It was an attempt to exploit vertical mining's attractive feature of supporting asynchrony in the counting process. Our experimental results demonstrate that the new dynamically load balanced asynchronous algorithm outperforms synchronous count-distribute algorithm as the heterogeneity of system increases, despite of the higher communication cost involved.

In future work, tuning of skip parameter can be done to improve the performance. Detailed performance analysis of the algorithm for transient loads also needs to be done.

7. References

- [1] R. Agrawal et al., Fast Discovery of Association Rules, *Advances in Knowledge Discovery and Data Mining*, U. Fayyad et al., eds., AAAI Press, Menlo Park, Calif., 1996, pp. 307-328.
- [2] P. Shenoy, J. R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa and D. Shah, Turbo-charging Vertical Mining of Large Databases. In *ACM SIGMOD Intl. Conf. Management of Data*, May 2000
- [3] R. Agrawal and J. Shafer, Parallel Mining of Association Rules, *IEEE Trans. Knowledge and Data Eng.*, Vol. 8, No. 6, Dec. 1996, pp. 962-969.
- [4] M.J. Zaki et al., New Algorithms for Fast Discovery of Association Rules, *Proc. 3rd Int'l Conf. Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., 1997, pp. 283-286.
- [5] M.J. Zaki et al., Parallel Algorithms for Fast Discovery of Association Rules, *Data Mining and Knowledge Discovery: An Int'l J.*, Vol. 1, No. 4, Dec. 1997, pp. 343-373.
- [6] M. J. Zaki, Parallel and Distributed Association Mining: A Survey, *IEEE Concurrency'99*
- [7] Scheduling of Parallel Applications on Heterogeneous Workstation clusters, *Proc. PDCS'96, the ISCA 9th International Conference on Parallel and Distributed Computing Systems*, pp. 330-337, September 25-27, 1996, Dijon, France