Scalable Parallel Mining for Frequent Patterns from Dense Datasets Using a Cluster of PCs

Amit Rudra
School of Information Systems
Curtin University of Technology
Kent Street, Bentley, 6102, Australia
rudraa@cbs.curtin.edu.au

Raj P. Gopalan, Yudho Giri Sucahyo
Department of Computing
Curtin University of Technology
Kent Street, Bentley, 6102, Australia
{raj, sucahyoy}@cs.curtin.edu.au

***Abstract* – Parallelizing computation intensive data mining tasks provides an excellent means of meeting their ever-increasing processing needs. Connecting several off-the-shelf PCs in a cluster is an effective and efficient way to harness their combined computing power for parallel processing. This paper describes the design and implementation of a parallel data mining algorithm on a PC cluster. The architecture of the system and its implementation are described first. Then the performance results for different configurations of processors using various dense databases are presented and compared with theoretically optimal performance.**

I. INTRODUCTION

Recent advances in parallel processing using affordable PCs are challenging the notion of increasing computing power by making ever faster and more efficient processors. Using low-cost commodity hardware (e.g. PCs), fast LAN (e.g. Myrinet, fast-Ethernet) and software systems like UNIX, MPI parallel environment, scalable systems can be built to an affordable budget [1]. The scalability of such systems is attractive to problems like association rule mining, where finding frequent associations between items in transactional data requires enormous processing time. Apart from market basket data, other application areas of data mining include DNA sequences, telephone call analysis etc. Some of these applications involve large dense databases that normally require a large amount of time to mine.

The current algorithms for association rules mining significantly slow down for dense datasets. As the support level of frequent patterns is reduced, the processing time required increases almost exponentially. We have several possible options to improve the performance: use a faster processor, make the algorithm more efficient, or mine in parallel. While the performance improvement from using a faster processor is limited by cost and availability, there are limits to improving an algorithm since the problem is intractable in the worst case. Parallel processing is attractive as it can deliver improved performance using the existing processors and suitably modified algorithm designs. Though, several algorithms have been developed since the early work in the area [2] to reduce the number of passes over the database, more research is still needed particularly to deal with dense datasets. As pointed out in an excellent survey in [3], parallel mining is an attractive option, but work in this area has been limited. Since that publication, some researchers have ventured further but much remains to be done [4][5][6][7]. For example, the implementation of association rule mining on shared-memory cluster machines was studied in [7]. However, there has been hardly any research done in this problem domain using a cluster of PCs.

The recently introduced FP-growth algorithm [8] is significantly more efficient than the Apriori algorithm for mining association rules [9]. The pattern growth approach has been improved further using a compact representation of transaction data in [10]. In this paper, we study the performance gains from implementing a parallel algorithm based on the approach in [10] on the shared nothing environment of a PC cluster. It will be interesting to find out the scale up and speed up of the algorithm when an increasing number of nodes are used for data mining and the limitations, if any.

Mining frequent itemsets is the most expensive step in association rules mining. Thus, distributing this task evenly over the cluster nodes is important for speed up. We use a round robin scheme for allocating projections of transactions to various nodes. For instance, the first node is allocated the first projection, the second node the second, and so on. Since the number of nodes in a cluster is likely to be less than the number of projections, a node will have several projections assigned to it. As detailed later, these projections can be mined independently, requiring almost no communication amongst nodes to exchange data or information during the mining phase.

The data parallel mining program was executed on a PC cluster to study its performance on dense datasets, as well as to measure the speed up in relation to the number of nodes. The scalability and speedup were tested using both small and large dense datasets. The performance results show the scale up and speed up achieved for different number of nodes.

Rest of the paper is arranged as follows. Section II defines relevant terms used in the PC cluster environment, association rules mining, and describes the pattern growth algorithm based on the compact transaction tree. In Section III, the data distribution scheme and the parallel algorithm are presented. The performance results are described in Section IV. The conclusion and some pointers to further work are given in Section V.

II. TERMS AND DEFINITIONS

We now describe some of the terms useful for understanding clusters and association rules mining, as well as the algorithm for mining frequent itemsets.

A. A Cluster of PCs

A cluster of PCs consists of several PCs typically connected by a fast LAN [11]. As shown in Fig. 1, each PC in the cluster, called a node, has its own copy of the OS, cluster management software (e.g. MPI, PVM) and application programs. Application programs are executed in

parallel on each node and communicate with one another using cluster management primitives.

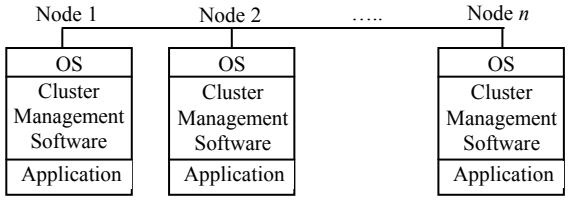


Fig.1 A typical cluster of PCs

B. Support and Confidence of Itemsets

The definitions below are adapted from [2]. Let $I=\{i_1,i_2,\ldots,i_n\}$ be a set of items, and T_D be a set of transactions, where $T \subseteq I$. Each transaction is identified by a *tid*. Let A and B each be a set of one or more items. An association rule is an expression of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$ and $A \cap B = \varnothing$. A is termed the body of the rule and B the head. For example, if 60% of transactions that purchase items *bread* also purchase *milk* and 5% of all transactions contain both; then the *support* of itemset {*bread, milk*} is 5% and the *confidence* of the rule $A \Rightarrow B$ is 60%. For an itemset to be termed *large* its *support* must be ≥ *support threshold* which is specified by the user. Otherwise, it is termed *small* or *infrequent*. An association with the *confidence* ≥ *confidence threshold* is considered as a valid association rule.

C. Dense and Sparse Datasets

Consider a list of eight transactions as shown in Fig. 2. We can denote the presence or absence of an item in a transaction in the binary form (1 for presence, else 0). Then a dataset is termed dense if the number of 1s is more than the number of 0s, otherwise, it is termed as sparse. In such a representation, finding the support for an item is to simply count the number of 1s for that item.

Tid	Items
1	1 2 3 5
2	2 3 4 5
3	1 2 4 5
4	3 4 5
5	3 4 5
6	1 2 3 4 5
7	1 2 3 5
8	1 2 4 5

⇨

1	2	3	4	5
1	1	1	0	1
0	1	1	1	1
1	1	0	1	1
0	0	1	1	1
0	0	1	1	1
1	1	1	1	1
1	1	1	0	1
1	1	0	1	1

Fig.2 The Transaction Database

D. Mining Frequent Itemsets from a Transaction Dataset

An efficient algorithm named CT-ITL for mining frequent patterns from a transaction dataset was described in [10]. We have adapted this algorithm for mining at each node of the cluster. The algorithm has three main steps: 1. Build a compact transaction tree to compress the transaction data; 2. Map the compressed data to an array based data structure named ItemTrans-Link; 3. Mine frequent itemsets of two or more items.

Step 1. Building a Compact Transaction Tree for a Dataset

Using the representation described in [10], transactions are stored in a compact tree that requires only about half the number of nodes of prefix trees used by various well-known algorithms such as FP-Growth [8]. The compression achieved by the compact tree reduces the cost of traversing the transactions in the mining phase.

In order to build the compact transaction tree only the 1-freq items entered in the ItemTable are considered. Each node of the tree will contain a 1-freq item and a set of counts indicating the number of transactions that contain subsets of items in the path from the root. Fig. 3 shows ItemTable of the database in Fig. 2, and the compact tree of only transactions that begin with item 1.

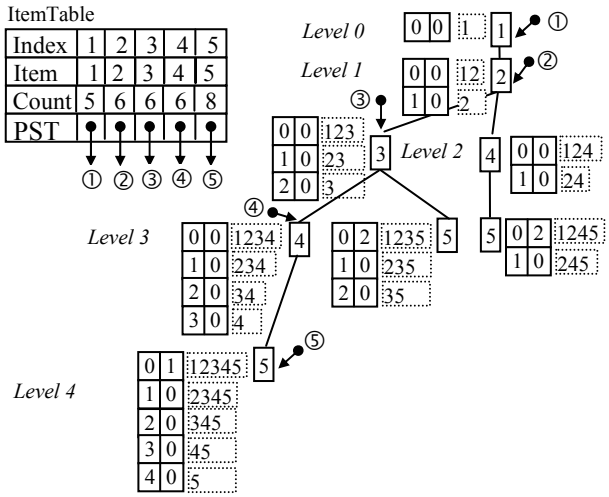
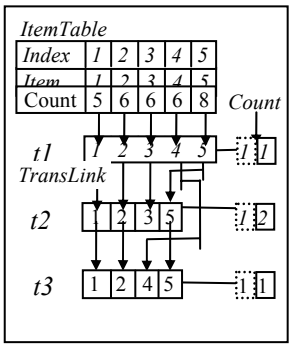


Fig.3 Compact Transaction Tree

Step 2. Mapping the Compact Tree to Item-TransLink (ITL)

For computing the support of itemsets, group intersection is used in this algorithm. For faster intersection operations, transaction groups in the tree are mapped to a data structure called Item-Trans Link (ITL) that has features of both vertical and horizontal data layouts [10]. Fig. 4 shows the result of mapping the tree in Fig. 3 to ITL. (Refer to [10] for a detailed description of ITL structure.)



(a) The Item-TransLink (ITL) Data Structure

Prefix	TempList (count)	Patterns (count)
1	3 (3), 4 (3), 2 (5), 5 (5)	1 (5), 1 2 (5), 1 5 (5)
1 2	3 (3), 4 (3), 5(5)	1 2 5 (5)

(b) Mining Frequent Patterns Recursively (support of the pattern shown in brackets)

Fig.4 The ITL Data Structure and Mining Frequent Patterns

Step 3. Mining Frequent Patterns

Each item in the ItemTable is used as a starting point to mine all longer frequent patterns for which it is a prefix. For

example, starting with item 1, the vertical link is followed to get all other items that occur together with item 1. These items with their support counts and transaction-count lists are entered in a table named TempList as in Fig. 4b (transaction lists are not shown in the figure). For prefix 1, items {2, 5} are frequent (count ≥ 5). Therefore, 1 2, and 1 5 are frequent item sets. After generating the 2-frequent-itemsets for prefix 1, the associated group lists of items in the TempList are used to recursively generate frequent sets of 3 items, 4 items and so on by intersecting the transaction-count lists. For example, the transaction-count list of 1 2 is intersected with that of 1 5 to generate the frequent itemset 1 2 5. At the end of the recursive calls for item 1, all frequent patterns that contain item 1 would be generated: 1 (5), 1 2 (5), 1 5 (5), 1 2 5 (5).

III. PC CLUSTERS AND MINING OF LARGE DENSE DATASETS IN PARALLEL

As outlined in an earlier section, large dense datasets are quite common in data mining applications. Our algorithm is especially suited for mining such large dense datasets. For mining such datasets we perform the following three steps: First, a partitioning scheme for creation of parallel projections; Second, an allocation scheme for load balancing the mining activity on each node of the cluster; Third, efficient memory based mining of all the projections allocated to each node of the cluster.

In the following subsections we first discuss the data-partitioning scheme, followed by the master-slave scheme where a coordinator distributes tasks to the participant nodes for mining frequent itemsets in parallel. The algorithm described in Section II.D is used for mining each projection at a node.

A. Partitioning Scheme for the Transaction Database

The database is partitioned into a set of projections that can be mined independently. For each item *i* in a transaction, all items that occur after *i* (in lexicographic order, using an in-memory sorting) are added to projection *i*. Consequently, a given transaction could be present in many projections. If there are *M* distinct items, *M-1* projections are formed. For example, if we have items 2 3 4 5 in a transaction, we put a transaction 2 3 4 5 in the projection for item 2, a transaction 3 4 5 in the projection for item 3, and a transaction 4 5 in the projection for item 4. A sample database and its projections are shown in Fig. 5.

The total disk space requirement of all the projections together will be higher than that of the original transaction database. However, this scheme allows the entire database to be partitioned into parallel projections for allocation to the nodes. Once distributed, the partitions can be for mined many times with different parameter settings (support and confidence).

A very important task for parallelizing schemes to work effectively is to make sure that the load on each node of a cluster has roughly equal amount of computational load. In our case we follow a round robin scheme for allocation of projections to various nodes of the cluster. The scheme for this task is shown in Fig. 6.

B. Parallel Mining in a PC Cluster Environment

This section describes the framework for mining frequent patterns using the multiple nodes of a cluster of PCs. We based it on a master-slave scheme using Message Passing Interface (MPI) calls [12] for distributing the tasks to various nodes of the PC cluster. Here, the coordinator node (master) assigns mining tasks to the participant nodes (slaves). The parallel program running on each participant node carries out the following tasks:

1. Disk I/O: Handles the opening, reading and finally, closing of the projections allocated to it (in the earlier step described in Section III.A). An earlier step called *makepart* creates these projections which are already present on the server disk of the cluster. The I/O system utilizes NFS file management system to perform this task.
2. Mining of projections for frequent itemsets: The next task is to mine the projections allocated to the cluster node for itemsets that are frequent as described in Section II.D. The main advantage here is that these projections can be mined independently of each other. Thus this important activity at each node is carried out independent of other nodes. Once a mining task has been allocated to a node by the coordinator, there is no need for further communication during the execution.
3. Creating datasets of frequent itemsets: Mining activity on each node of the cluster, results in frequent itemsets. These datasets are written to the cluster server.

Fig. 7 outlines the algorithm used for parallel execution of tasks at all nodes.

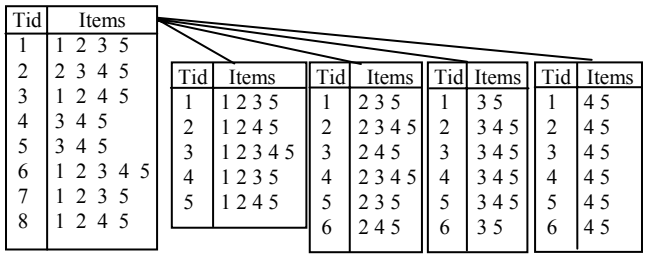


Fig.5 Parallel Projection Example

```
Start with first node
Allocate first projection to first node
Find next node
While not the last projection do
   Allocate next projection to next node
   Find next node
   If all nodes finished
       Move back to first node
   End if
End do
```

Fig.6 Scheme for allocation of parallel projections to nodes

```
MPI_Initialize
Find my rank
If coordinator
   Start timing work
   Distribute work to participants
Else if participant
   Mine allocated dataset projections
End if
Time end of work
Finalize work
```

Fig.7 Scheme for distributing the work to different nodes

IV. PERFORMANCE STUDY

We implemented our parallel algorithm described above on a cluster of Pentium PCs. Our main objective was to study the performance gain in mining dense datasets on the PC cluster as these datasets require highly compute intensive tasks at low support levels. First, the performance of the parallel program was studied on small databases at low support levels. Our partitioning scheme facilitates the transformation of sequential mining into a data parallel task by creating parallel projections, which can be allocated to various nodes of the cluster. The purpose of distributing a data parallel task to a varying number of nodes is to get a steady linear speed up. Subsequently, we tested the scalability of our algorithm using a very large dense dataset.

The parallel mining program was written in C++ and executed on a cluster of 8 processors of Pentium III-800 MHz, 256 MB RAM, 10 GB HD, running FreeBSD. The PCs were connected by a 100 Mbps fast-Ethernet link. MPI routines were used for communication among the nodes. In our study, the virtual processor feature of MPI was used to increase the number of processors above 8 (up to 20).

We tested our algorithm on two small dense datasets and a large dense dataset. The small datasets were downloaded from Irvine Machine Learning Database Repository [13]. They included Chess (3196 transactions, 75 items, average transaction 37 items) and Connect-4 (67557 transactions, 129 items, average transaction 43 items). A dataset containing 1 million transactions with characteristics similar to Connect-4 was generated using a synthetic data generator [14]. The density of the dataset is 67%. A support of 60% was used.

Fig. 8 shows the speed up achieved by executing our parallel mining program using various configurations of processors, on Connect-4, Chess and 1M datasets. On Connect-4 (support 40), the results show that the speed up is consistent. A reasonable speed up was achieved when the configuration was changed from 2 processors to 4, to 6, to 8 and then to 12 processors. Beyond that, no further speed up was achieved. On Chess (support 40), there was a slight speed up when the configuration of processors is varied from 2, to 4, to 6. No further speed up was achieved beyond 6 processors. On the very large dataset of 1 million transactions, the speed up was consistent.

In all cases, the time taken for a sequential mining (i.e. using a single processor) is considered as the unit of comparison. This time is then divided by the maximum time taken for mining on a given number of nodes. The time at each node includes the fixed time of setting up the task and the variable time of mining the given set of projections. Utilizing additional processors, the variable time for mining is shared among more processors, but the fixed time required remains unchanged. Hence, adding more processors beyond a certain number does not significantly reduce the run time. For the large dense dataset of 1M transactions, the increase in speed up is small when the number of processors is increased. On all the charts in Fig. 8 we have also plotted the average time taken on each node. The average time indicates the maximum speed up that can be achieved for the given configuration, dataset and algorithm. Since the scale up of average time per node is almost linear, it is evident that some reorganization of the data distribution or the partitioning scheme may benefit in avoiding the wide variations in time taken by the nodes.

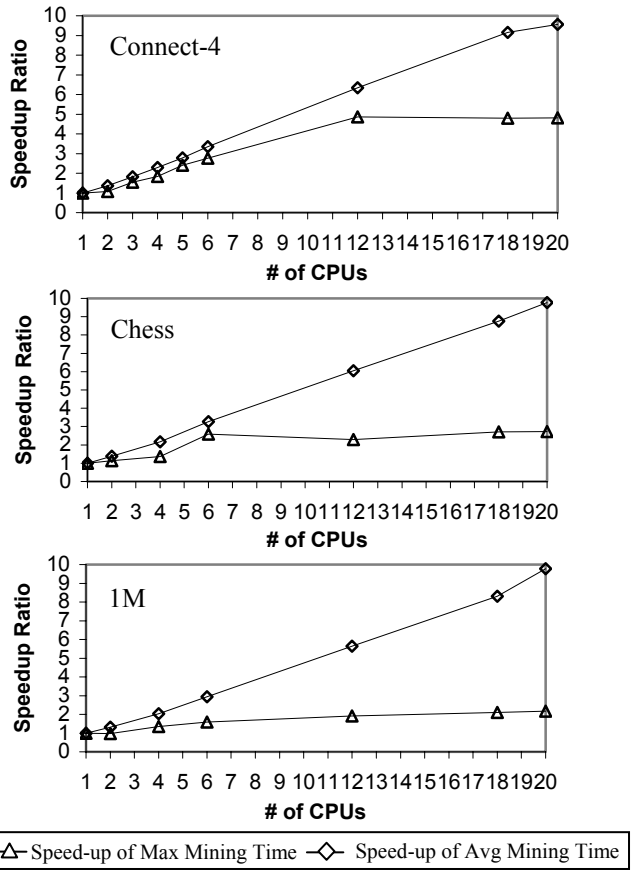


Fig.8 Speedup on Connect-4 (Support 40), Chess (Support 40), and 1 millions (Support 60) datasets

V. CONCLUSIONS

We have described the implementation of a parallel algorithm for mining frequent patterns from dense datasets on a cluster of PCs and discussed its performance characteristics. An efficient sequential pattern growth algorithm named CT-ITL [10] that was known to outperform FP-Growth and Apriori on dense datasets, was adapted for mining at the individual nodes. The datasets were partitioned into several projections to enable parallelization, and a round robin allocation scheme was used to distribute the projections to the nodes of a PC cluster. The mining algorithm was tested on both small and large dense datasets. The speed up achieved was reasonable and it improved with the number of processors in the cluster configuration to some extent.

However, the experiments also showed that there is scope for further improvements by better load balancing. It may be possible to design alternative partitioning and allocation schemes that show better effect on performance. We also plan to study further, the factors that influence the scale up and speed up of data mining on clusters.

VI. REFERENCES

[1] J. E'Silva and R. Buyya, "Parallel Programming Models and Paradigms", *High Performance Cluster Computing.* vol. 2, 1999, pp 4-27.

[2] R. Agrawal, T. Imielinski and A. Swami, "Mining Association Rules between Sets of Items in Large Databases", *in ACM SIGMOD*, Washington DC. 1993.

[3] M.J. Zaki, "Parallel and distributed association mining: A survey", *IEEE Concurrency (Special Issue on Data Mining)*. October/December 1999, pp 14-25.

[4] R. Agrawal and J.C. Shafer, "Parallel Mining of Association Rules", *IEEE Transactions on Knowledge and Data Engineering,* vol 8(6), December 1999, pp 962-969.

[5] E-H. Han, G. Karypis and V. Kumar, "Scalable Parallel Data Mining for Association Rules", *IEEE Transactions on Knowledge and Data Engineering,* vol 12(3), May/June 2000, pp 337-352.

[6] R. Jin and G. Agrawal, "An Efficient Association Mining Implementation of Cluster of SMPs". *in Workshop on Parallel and Distributed Data Mining (PDDM),* 2001.

[7] M.J. Zaki, "Parallel Sequence Mining on Shared-Memory Machines", *Journal of Parallel and Distributed Computing*, vol 61(3), March 2001, pp 401-426.

[8] J. Han, J. Pei, Y. Yin and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-pattern Tree Approach", *Data Mining and Knowledge Discovery: An International Journal,* Kluwer Academic Publishers, The Netherlands, 2003.

[9] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules", *in 20th Int. Conf. on VLDB*, Santiago, Chile. 1994.

[10] R.P. Gopalan and Y.G. Sucahyo, "Improving the Efficiency of Frequent Pattern Mining by Compact Data Structure Design", *in 4th Int. Conf. on Intelligent Data Engineering and Automated Learning (IDEAL)*, Hong Kong, 2003.

[11] M. Baker and R. Buyya, "Cluster Computing: The Commodity Supercomputing", *Software-Practice and Experience,* vol 1(1), Jan 1999, pp 1-26.

[12] W. Gropp, E. Lusk and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface,* 2nd ed, MIT Press, Cambridge, MA; 1999.

[13] http://www.ics.uci.edu/~mlearn/ MLRepository.html

[14] http://www.almaden.ibm.com/software/quest/