Proceedings of the
38th Southeastern Symposium on System Theory
Tennessee Technological University
Cookeville, TN, USA, March 5-7, 2006

Parallel Implementation of Association Rule in Data Mining

Sussan Einakian, IEEE Member, University of Alabama in Huntsville
M. Ghanbari, *IEEE Member,* Department of Computer Science Alabama A&M University

Abstract This paper discusses parallel Data Mining architecture for large volume of data which eventually scanning billions of rows of data per record. Here we compare the different parallel algorithms for Association Rule Mining and discuss the advantages and disadvantages of each method. We also compare the computational time of serial and parallel algorithms for Association Rule Mining.

I. Introduction

Data Mining is a technology that combines traditional data analysis methods with complicated algorithms for processing big volume of data.

Rapid advancement of IT technology has resulted accumulation of tremendous amount of data for organization and therefore extracting needed information from huge amount of data has been a big challenge for researchers.

In the process of computing the frequency of the occurrences of an interesting subset of items in the database of transactions (called Candidate) process time always is a big factor of attention. To prune the exponentially large amounts of candidates, common algorithms, which are most frequently used, are those dealing with Candidate having user defined minimum support. In these situations, even with pruning, finding all association rules requires large computation power and need a lot of memory.

A traditional computer has a single processor for executing a task. One way of increasing the computational speed is by using multiple processors within a single computer (multiprocessor) or alternatively multiple computers, operating together on a single problem, depending upon the problem and the amount of parallelism in the problem. What make parallel computing timeless is the continual improvements in the execution speed of processors. Therefore, using parallel processors and different algorithm of implementation of Association Rules is always a plus.

In general, a non-trivial parallel algorithm may include some or all of the following:

- Identify portion of work that can be performed concurrently.
- Mapping the concurrent pieces of work onto multiple processors running in parallel.
- Distribute the input, output, and intermediate data associate with program.
- Managing accesses to data shared by multiple processors.
- Synchronizing the processors at various stage of the parallel program execution.

In the context of parallel algorithm design, processes are logical computing agent that performs tasks.
Processors are the hardware units that physically perform computations. Therefore, here, like [3] we suggest the following distinction.

$$\left\{\begin{array}{l}\text{1) Task Parallel}\\ \text{2) Data Parallel}\end{array}\right.$$

Treating processes and processors separately is useful when designing parallel programs for hardware that supports multiple programming paradigms.

1) Task parallel algorithm split the performed computation into a set of tasks for concurrent execution defined by task dependency graph. The task parallel approaches can further be divided into two groups:

a) Recursive decomposition for including concurrency in those problems that could use divide and conquer. In this method, problem is solved by first dividing it into a set of independent subprograms. Each one of those subprograms is solved by recursively applying a similar division into smaller subprograms followed by a combination of their results.

b) Based on task queue that dynamically assigns the small portions of the computations to a processor whenever it becomes available.

2) Data Parallel, distribute the data over the available processors. Data parallel approach also could be divided into two sections:

a) A partitioned based on records will assign non-overlapping sets of records to each of the processors.

b) A partitioning attributes will assign the set of process. Attribute based is relying on the observation of the problem which could be set of elementary functions. If attribute are distributed over multiple processors, then those function could be executed in parallel. Good distribution will result in load balancing.

II. Association Rule

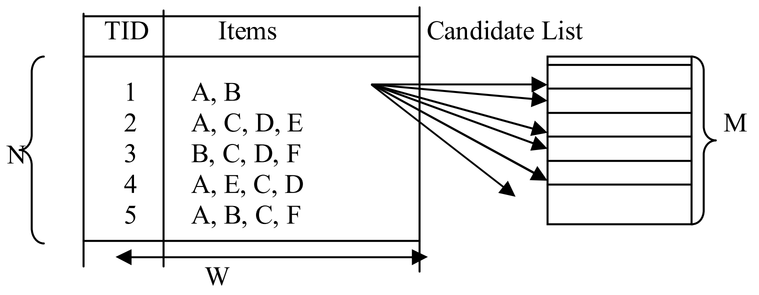
Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction
This type of mining is useful for transactional data like market basket application. By association rule mining they can analyze the data to learn purchase behavior of customer.
An insinuation of form x -> y where x and y are item sets.
Example, buying {bread, butter}->{milk}.
Two important measures that help to find good rules are support and confidence.
Support (s): Fraction of transactions that contain both X and Y.

Confidence (c): Measures how often items in Y appear in transactions that contain X.
Given a set of transactions T, the goal of association rule mining is to find all rules having
-support ≥ *minimum support* threshold
-confidence ≥ *minimum confidence* threshold
Our model supposed to automatically find all interesting rules which is a large task because of the volume of data involved. One approach to solve this problem is:

Brut Force:

- List all possible association rules
- Compute the support and confidence for each rule
- Prune rules that fail the minimum support and minimum confidence thresholds.

Here is an example that can explain this approach.



TID	Items
1	A, B
2	A, C, D, E
3	B, C, D, F
4	A, E, C, D
5	A, B, C, F

Each itemset is a candidate frequent itemset; count the support of each candidate by scanning the database. If x : {C, D} and y : {E} then the support and confidence of x -> y will be :

S (x -> y) = δ(x U y) / N for N = 5 ➔ S (x -> y) = 2/5 = .4
C (x -> y) = δ(x U y) / δ(x) ➔ C (x -> y) = 2/3 = .67

Each transaction matches to every candidate then the complexity is ~ O(NMW). On the other hand the total number of itemsets is 2^d. Therfore the total number of possible rules would be:

$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right] = 3^d - 2^{d+1} + 1$$

Too many rules can be extracted from a data set with “d” items. Thus for 6 items we will have 602 rules ($R = 3^6 - 2^{6+1} + 1 = 602$). There are too many rules so we somehow need to decrease them. There are different ways to decrease the number of rules like:

- Reduce the number of candidates (M) since complete search is $M=2^d$.
- Use pruning techniques to reduce M
- Reduce the number of transactions (N) as the size of itemset increases.
- Reduce the number of comparisons (NM) by using efficient data structures to store the candidates or transactions. So there is no need to match every candidate against every transaction.

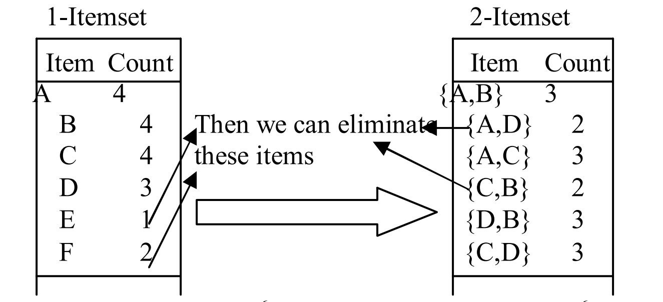
Second approach is Apriori Principal:

The first association rule algorithm that uses support base pruning to control the growth of candidate itemsets is the Apriori Algorithm. If an itemset is frequent, then all of its subsets must also be frequent. Apriori principle follows the property of support measure that shows:

- Support of an itemset never exceeds the support of its own subsets

$$\forall X, Y: X \subseteq Y \quad s(X) \geq s(Y)$$

As an example; in Market Basket Transaction, suppose we have 6 items and Minimum Support = 3. The support count of these items are in table 1 :



1-Itemset

Item	Count
A	4
B	4
C	4
D	3
E	1
F	2

2-Itemset

Item	Count
{A,B}	3
{A,D}	2
{A,C}	3
{C,B}	2
{D,B}	3
{C,D}	3

Table1(# of candidate $\binom{6}{1}$=6) Table2(# of candidate $\binom{6}{2}$=15)

And 3-Itemset is {A, B, C} count = 3, {A, B, D} < 3, and {A, D, C} < 3 (# of candidate $\binom{6}{3}$=20

With all subset the result is: 6+15+20=41

But after using pruning the result is: $\binom{6}{1} + \binom{4}{2} + 1 = 13$

Pseudocode For frequent Item Set [2]

```
Assume that:
$C_k$      : set of candidate k-itemset
$F_k$      : set of frequent k-itemset
Minsup : Minimum Support
$\sigma(\{i\})$ : Support Count or number of transaction contains {i}
T      : Set of Transaction
I      : Itemset
C      : Subset of I
For k = 1
$F_k = \{ i \mid i \in I \;\&\; \sigma(\{i\}) >= Minsup\}$ // Fin all frequent
                                        // 1-itemset
Repeat
  k=k+1
  $C_k$ = apriori_gen($F_{k-1}$)  // Generate candidate itemset
  For each trans $t \in T$ do
    $C_t$ = Subset ($C_{k,}$t)  // Identify all candidates belong to t
    For each candidate itemset $c \in C_t$ do
      $\sigma(c) = \sigma(c) + 1$  // increment support Count
    end For
  end For
  $F_k = \{ C \mid C \in C_k \;\&\; \sigma(C) >= Minsup\}$  //Extract the
                                        //  frequent k-itemset
Until $F_k = 0$
Result = $\cup F_k$
```

The union of the frequent itemset is the frequent itemset, which generates the association rules. Counting the candidate itemset is the most expensive step in computation. One-way of improvement is using the candidate hash tree to increase the speed of computation.

Parallel Algorithms

1) Data Mining Server architecture (DMS)

One method is record based partitioning that call DMS (Data Mining Server) [3] implementation of data mining. Client/Server or DMS is a term to describe computing model for development of computerized systems. This model is based on distribution of functions between two types of independent and autonomous processes, server and client. A **Client** is any process that request specific services from server processes. A **Server** is a process that provides requested services for Clients. Cline/Server Systems will be classified as a 3-tier. In a 3-tier Client/Server System, the client requests are handled by intermediate server (Manager), which coordinates the execution of the clients request with subordinate servers. Each server consists of the following elements [3] (Factory, Engine, Cache, and File Server).
The DMS's architecture is pictured below:

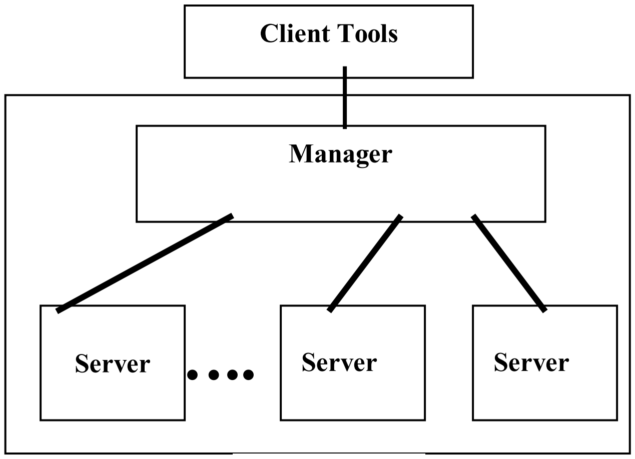
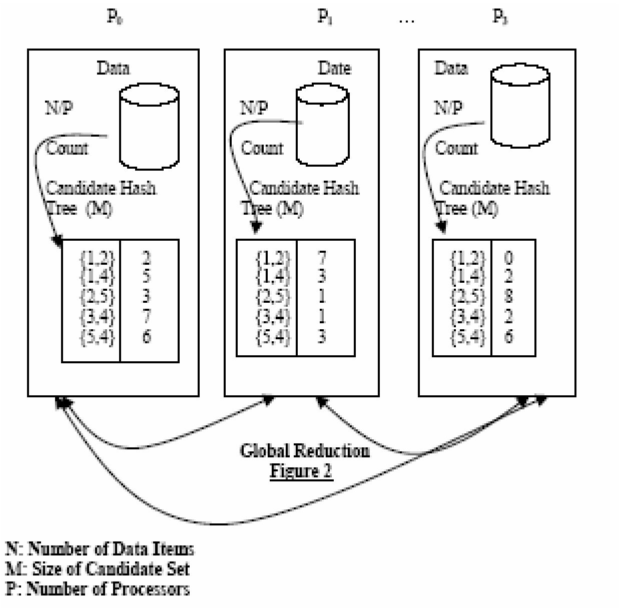


Figure 1

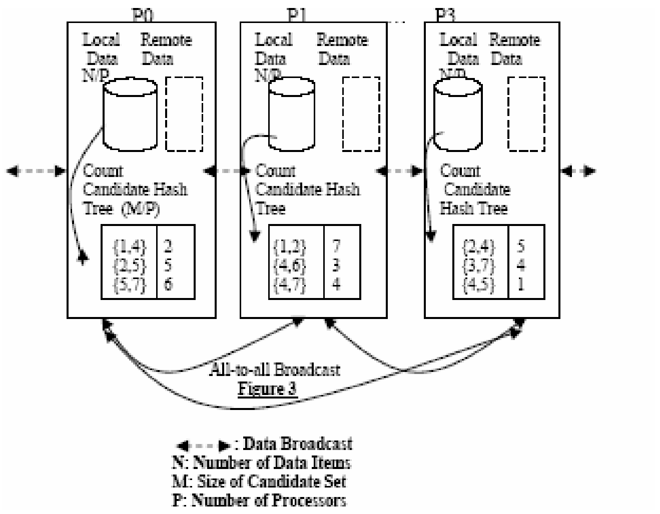
2) Count Distribution (CD) Algorithm:

In CD, each processor by building the entire hash tree that corresponds to all the candidates computes the count of all the candidates showing up in transactions that stored locally. The global count is computed by summing up the count of each processor using a global reduction operation. The Count Distribution method scales with the data size, but does not scale with main memory usage. In this case, each processor can compute the count independently of others and need to communicate with other processors only once at the end of the computation. Therefore, it scales linearly with the number of transactions. [9] This algorithm has big advantages when the hash trees can fit into the main memory of each processor. This algorithm is more useful for a small number of items.
Below you can see a sample of the Count Distribution Algorithm.



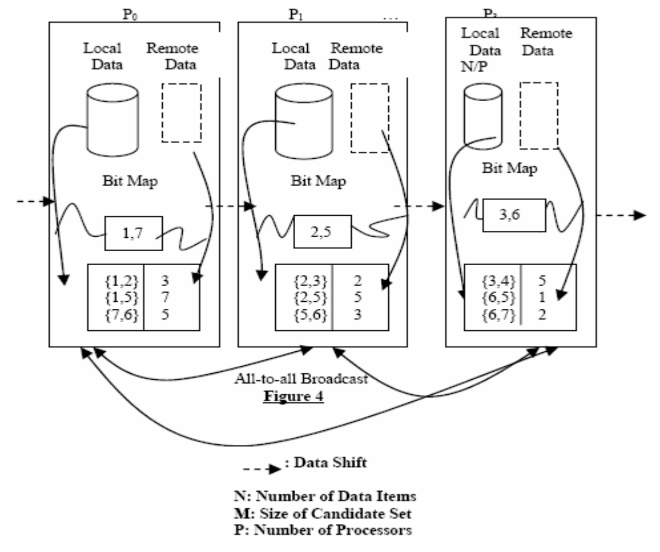
3) Data Distribution (DD) Algorithm:

Data Distribution algorithm [9] partitions the candidate itemset between the processors. Then each processor scans part of the transactions assigned to the other processors as well as its own local part of transactions. Each processor assigns P buffers then in processor P_i the i^{th} buffer is used for storing local transactions and the remaining buffers are used for transactions from other processors (For 4 processor, in processor 2, 2^{nd} buffer for local transactions and 1^{st}, 3^{rd}, 4^{th} buffers store transactions from other processors 1, 3, and 4). The buffer is processed by processor and updates the count of its own candidate subset then each processor has a different set of candidates in candidate hash tree. If this buffer is related to local transactions it is sent to all other processors by asynchronous sends. Otherwise, if the buffer is related to other processor it cleared and an asynchronous receives request is issued to that processor. This process continues till every processor has processed all the transactions. The computed count of each processors candidate itemset, is sent to every other processors by an all-to-all broadcast operation. This algorithm will cover the memory problem of the Count Distribution algorithm. In this algorithm a problem happens when the number of processors increases. The communication pattern and redundant work creates a problem for the Data Distribution algorithm.
Below you can see the sample of DD algorithm:



4) Intelligent Data Distribution (IDD) Algorithm:

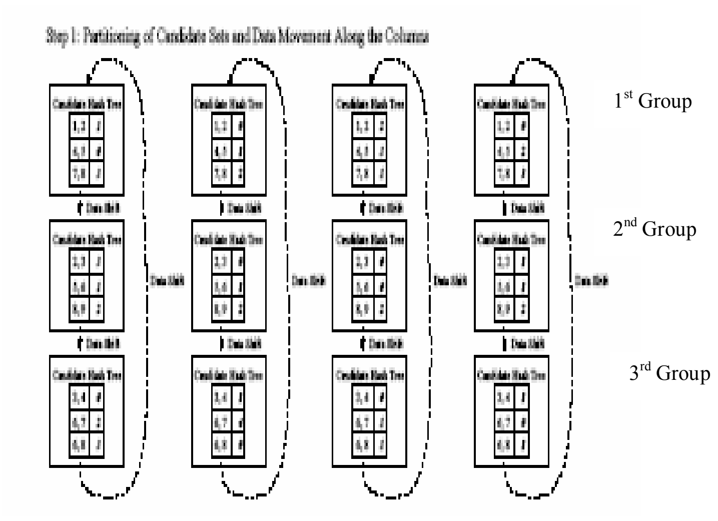
The Intelligent data Distribution algorithm [10] locally stores a portion of the database sent to all processors by using a ring based all-to-all broadcast. Each processor communicates by its left and right neighbor. Each processor has two buffers one for sending and one for receiving. The send buffer initially fills with one block of local data. Each processor initiates an asynchronous send operation to the next right neighbor and receives an asynchronous receive operation from left neighbor and computes the counts of candidates assigned to each processor. Then the role of send and receive buffer is switched and it continues for P-1 times. The example of IDD algorithm is below. The main point of this algorithm is partitioning the candidate itemset.
By partitioning the candidate itemset the redundant work will reduce. As you can see in figure 4 the candidate set is partitioned based on bit map value.
Proc. 0 hashes all the candidates starting with item 1 and 7.
Proc. 1 hashes all the candidates starting with item 2 and 5.
Processors 2 hash all the candidates starting with item 4.
Proc. 3 hashes all the candidates starting with item 3 and 6.
The candidate starting with bit map value will process otherwise it will skip then the IDD algorithm will reduce the redundant work.



5) Hybrid Algorithms:

The Hybrid algorithm [4], computation is structured into multiple stages and it is sometimes necessary to apply different types of decomposition in different stages. It is a combination of the Count Distribution and the Intelligent Data Distribution algorithm. Let assume we have P processors that we have divided them into G groups of equal size. Therefore each group has P/G processors. If we have twelve processors and divide them into three groups then four processors have the same candidate itemsets. Therefore, each column has combination of all candidate itemsets. In this example, CD algorithm will apply to each column and each column will be looked at as one processor. Then each processor (in this case, column) computes the local count of all candidate itemsets.
At the second step, we apply IDD algorithm on each row to find the global count of candidate itemsets. Therefore reduction operation happened in each row and the count in each row is the same for processors in each row. At the third step, all-to-all broadcasting for each column will show the same frequent itemset for all processors. Finally, the system is ready to advance to next pass.
The number of groups calculated is based on the memory size of each processor that can hold a hash tree of m candidate set. And finally, the data movement will decrease to (1/G). In figure 5 you can see the three steps of the Hybrid algorithm. In step one you can see the partitioning of candidate set and data movement along the columns. In step two the reduction operation is shown along the rows and in the last step all-to-all broadcast operations happens along the columns.

Figure 5: [4] (P/G = 4 → P = 12)


Step 1: Partitioning of Candidate Sets and Data Movement Along the Columns
Candidate Hash Tree
Data Shift
1st Group
2nd Group
3rd Group

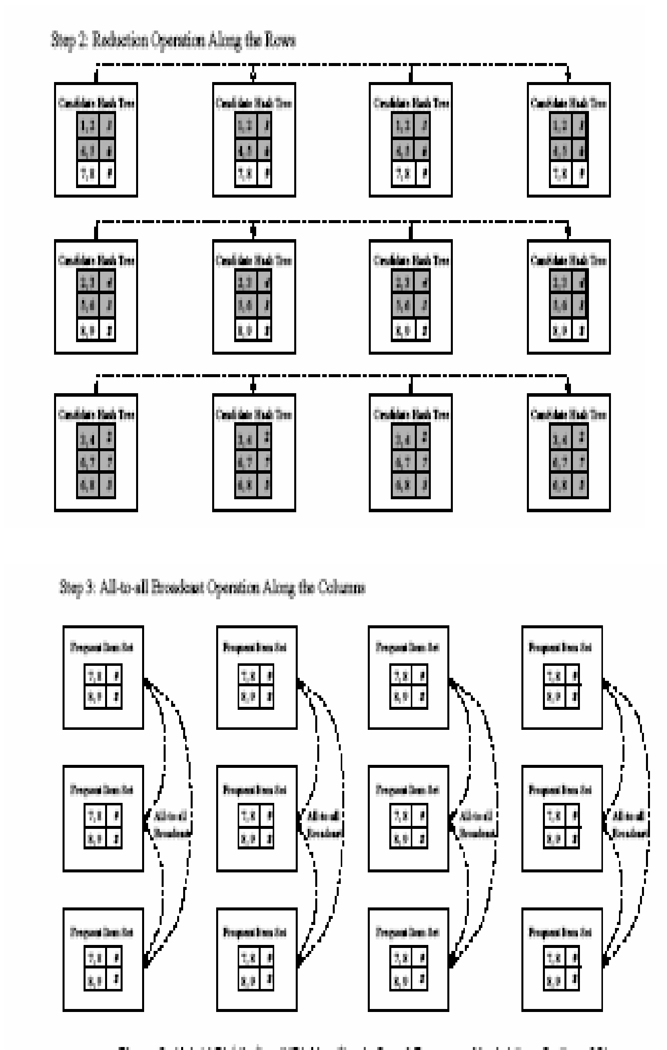

Step 2: Reduction Operation Along the Rows
Candidate Hash Tree
Step 3: All-to-all Broadcast Operation Along the Columns
Frequent Item Set
All-to-all Broadcast

Figure 9: Hybrid Distribution (HD) Algorithm in 3 × 4 Processor Mesh ($G = 3$, $P = 12$)

6) Inverted Matrix Algorithms:

There are some problems with other algorithms like scan database repeatedly, large amounts of computations and communications, and memory size for finding the frequent itemset. Authors believe the Inverted Matrix [8] is covered these problems. This algorithm has two main parts. In first part database is fully scan and generates a data structure is called inverted matrix. In second part inverted matrix is repeated between different processors. In the Inverted Matrix algorithm if support threshold changes it will be necessary to rescan the database.

There are three different approaches for transaction layout: horizontal layout, vertical layout, and inverted matrix layout that is used in Inverted Matrix algorithm. The most common approach is horizontal layout that works on transaction id, and the vertical layout works on the transactions that have same item. Then the key in horizontal is transaction id and in vertical layout is item. The inverted matrix layout is combination of two layouts. Then the items and transactions check together, first it finds the occurrence of items in each transaction, then points each transaction to all items included in the transaction. Inverted Matrix uses pointer instead of transaction id and points to the location of next item in the transaction that ordered in ascending order.

For building the Inverted Matrix first scan the database to find frequent itemset in ascending order, then scan again for sorting the transaction in ascending order based on frequency result. As you can see in the below L in transaction one has location 3 in Inverted Matrix, E has location 4 and so on.

<u>Ascending Order</u>

T_1: A L D E		T_1: L E D A
T_2: P D E A	$\Rightarrow$	T_2: P E D A
T_3: A P L R		T_3: R P L A

Item Frequency: A: 3, L: 2, D: 2, E: 2, P: 2, R: 1

Loc	Index	Transactional array
1	R,1	2,2
2	P,2	4,2 3,2
3	L,2	4,1 6,3
4	E,2	5,1 5,2
5	D,2	6,1 6,2
6	A,3	∅,∅∅,∅∅,∅

As you can see L is linked to E and E is first empty element, then link for L becomes 4,1. E is linked to D and D is first empty element, then link for E becomes 5, 1. D is linked to A and A is first empty element, then link for D becomes 6, 1. A is last item then link become∅, ∅. Now in next transaction, P is linked to E and E is second empty element, then link for P becomes 4,2 and so on.

For implementation the Inverted Matrix in parallel, Inverted Matrix must be repeatedly sent to all processors. Because the whole database is in each processor the communication time for generating global frequent itemset will decrease. In this approach the frequent itemset evenly distributed between processors. For example in example above if support count (δ) >1 and we have two processors the location 2, 4, and 6 process in processor 1. Each processor read sub transactions from Inverted Matrix and build Co-Occurrence Frequent Item tree (COFI-tree) and after mining, the trees are discarded.

<u>Advantages and Disadvantages</u>

In Data Mining Server (DMS) the communication between server and manager and client is one problem.

We have memory problems in the Count Distribution (CD) algorithm. Memory problem of the CD algorithm is covered with partitioning of the candidate itemset between processors in Data Distribution (DD) algorithm. If the numbers of processors in the DD algorithm increase, we will face problems because of communication patterns and redundant work. We have P-1 send to other processors and P-1 receives operation from other processors. In DD we have a finite number of communication buffers in each processor then the all-to-all communication sometimes makes the process become idle. For example when one processor finishes an operation on local data and sends the buffer to all processors if the communication buffer of receiving is full then the sends must wait till the buffer becomes empty. In DD all N transactions have to go through the hash tree of the M/P candidate but in CD only N/P times go through the hash tree of M candidate.

In Intelligent Data Distribution (IDD) we have point-to-point communication between neighbors, then the chance of idling will decrease and we may have idling only for a short time.

IDD requires the algorithm to have good load balancing, since good partitioning will create an equal number of candidates in all the processors. We can achieve a load balancing distribution by using a bin-packing partition algorithm. [4] In this algorithm it stores a number of candidate itemset with each item. Then by partitioning items in P bucket, the number of itemset in each bucket will be equal. In IDD each processor has M/P number of candidate. If number of processors increase we have fewer candidates per processor and it will be difficult to balance the work and we have smaller hash tree and it reduces the efficiency of algorithm, because the communication time will be greater than computation time.
In Hybrid algorithm (HD) by combining CD and IDD the disadvantages of these two will cover each other.
In Inverted Matrix after creating the Inverted Matrix it will replicated between processors then communication will decrease and we will have only two full database scan for creating the Inverted Matrix.

Comparison of different Algorithms

In this comparison some notation are used like: Computational time per transaction (T_{trans}), Number of candidate set (C), Traverse of hash tree based on potential candidates ($T_{travers}$), Number of leaf visited per transaction ($V_{C,L}$), and L is average number of leaf node in hash tree.
Then: $T_{trans} = C * T_{travers} + V_{C,L}$

Serial Algorithm

In Serial algorithm it must process N number of transactions

$$T_{comp} = (N)(T_{trans})$$

DMS

The candidate itemset is partitioned between different servers. Each server separately partitions the transaction between processors. We will be faced with more communication. Each server must communicate with the Manager then Manager with Client

$$T_{comp} = (N/ S) (T_{trans} + T_{Comm(S,M)} + T_{Comm(M,C)})$$

CD

Each processors handles N/P number of transactions.

$$T_{comp} = (N/ P)(T_{trans})$$

DD

Number of candidate per process decrease to (C=M/P, M :Total # of candidate set). Number of leaf node per transaction ($V_{C,L/P}$), L/P is average number of leaf node. Each processor computes all transactions.

$$T_{trans} = C * T_{travers} + V_{C,L/P}$$
$$T_{comp} = (N)(T_{trans})$$

IDD

Number of potential candidate set per process decrease to (C/P). Then number of leaf node per transaction ($V_{C/P,L/P}$). Each processor computes all transactions.

$$T_{trans} = (C/P) * T_{travers} + V_{C/p,L/p}$$
$$T_{comp} = (N)(T_{trans})$$

Hybrid

Number of potential candidate set per process decrease to (C/G), then number of leaf node per transaction ($V_{C/G,L/G}$) L/G is average number of leaf node. Number of candidate per process decrease to (M/G).Number of transactions per processor is N/(P/G) = NG/P

$$T_{trans} = (C/G) * T_{travers} + V_{C/G,L/G}$$
$$T_{comp} = (G x N / P)(T_{trans})$$

Inverted Matrix

Number of transactions that support threshold will be divided between processors (m). Number of potential candidate set per process decrease to (m/p).

$$T_{trans} = (C/P) * T_{travers} + V_{C/P,L/P}$$
$$T_{comp} = (m / P)(T_{trans}) + 2 T_{scan}$$

Conclusion

Data Mining is useful in analyzing large amounts of data in different area.
Different types of data mining are useful in different types of applications based on behavior of the data. Association rule mining is a type of data mining that handles the transactional application.
There are different algorithms for parallel implementation of association rule:
DMS, CD, DD, IDD, Hybrid, and Inverted Matrix

References:

[1] J.Han, M.Kamber, Data Mining Concept and Techniques, Morgan-Kauffman, 2001
[2] P-N Tan, M.Steinbach, V.Kumar, Introduction to Data Mining, Addison-Wesley, 2005
[3] Felicity George, Amo Knobbe, A Parallel Data Mining Architecture for Massive Data Sets
[4] Eui-Hong Han, George Karypis, Vipi Kumar, Scalable Parallel Data Mining for Association Rules
[5] Zoltan Jarai, Aashu Virmani, Liviu Iftode, Towards a Cost-Effective Parallel Data Mining Approach
[6] R.Agrawal, R.Srikant, Fast algorithms for Mining Association Rules
[7] R.Agrawal, J.C.Shafer, Parallel Mining of Association Rules
[8] El-Hajj, R.Zaiane, Parallel Association Rule Mining with Minimum Inter-Processor Communication
[9] R.Agrawal and R.Srikant. Fast algorithms for mining association rules. In Proc. Of the 20th VLDB Conference, pages 487-499, Santiago, Chile, 1994
[10] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. Introduction computing: Algrithm Design and Analysis. Benjamin Cummings/ Addison Wesley, Redwod City, 1994.