CHAPTER 4

METHODOLOGY ADOPTED IN PARALLEL AND DISTRIBUTED DATA MINING WITH ASSOCIATION RULES

The methodology involved in this research work is as follows:

- Retrieval of CPU idle time from distributed workstations.
- Using simulated annealing algorithm to allocate the jobs in a distributed environment in the available CPU's.
- Carrying out the mining process with a parallel association mining algorithm, Hash Partitioned Apriori and Count Distribution.
- Simulation of data mining process in the distributed environment in parallel by developing a prototype.

The distributed workstations are identified and the CPU utilization is retrieved from the identified workstations. The CPU idle time retrieval is completed and the allotment of job to the distributed workstations is carried out. The following screen shows the CPU idle time retrieval process. The different workstations machine name, host id and password are given as input. After this process, the CPU idle time of the different workstations will be listed. The job allocation process is carried out based on this CPU utilization.
Based on the CPU utilization the allocation scheme is used for job allocation. The remote machines are also monitored by giving the respective machine names. After successful job allocation the mining process is carried out.

Figure 4b graphically depicts the CPU, memory and disk utilization. This shows the CPU, memory, disk and network utilization under Windows NT platforms (2000, XP). It is composed of four fully configurable small graphs and it can work in the tray area also. The windows are fully anchorable and by moving the "CPU" window the other attached windows are moved.
The following screen shot Figure 4c is the program which uses File Transfer protocol for transferring the files based on the allocation scheme. The task allocation is carried out using the simulated annealing algorithm. The mining process starts performing as soon as the file is sent to the workstations. The mined result is then obtained from the different distributed locations.

![FTP Demo Application](image)

![SEDC - UtilityPOWERnet FTP Send Form](image)

**Figure 4c: FTP Application**

The FTP server name is the name of the server where the file is stored. Authentication is provided using username, password. The source file is provided in the FTP send file field and the destination file is given in FTP.
destination file. The FTP server port is also provided. This transfers the file to the other machine. The mining process carried out after allocating to the specific processes is shown below. The XML file required to mine has to be selected and then this will be mined and the mined outcome is obtained.

![ConnectionDialogBox](image)

**Figure 4d: XML File from Market Basket Analysis**

The transaction data is selected and based on this the mining result is obtained with the different number of itemsets being generated.
Figure 4e: Mined outcome from Market Basket Analysis

After obtaining the mined result from the distributed workstations the results are integrated based on the global exceptional pattern identification. The integration is also carried using the Bayes networking model. The Figure 4e illustrates the knowledge obtained in mining a sample XML data with a minimum support of 0.2 and confidence factor of 80%.
4.1 Framework of the Research Work

The framework of the research work is illustrated in Figure 4f. The distributed workstations are identified initially to carry out the data mining process. The CPU idle time is retrieved and the job allocation is carried out accordingly. After the allocation, mining process is carried out with the necessary data mining algorithms in a parallel and distributed environment. Dynamic remote memory utilization is carried out so that the memory is utilized effectively without any overload. The knowledge which is obtained from the distributed workstations is integrated to obtain the final result. An optimized response time is obtained in this research work. The algorithms and the needed primitives used in the above framework are explained in the following sections.
4.1.1 Task allocation process

The task allocation process is carried out using the simulated annealing algorithm. Here the tasks are assigned to multiprocessors based on the idle time and the intelligent agents. The manner in which this partitioning and allocation is made by determining the efficiency of a given application when it is executed on distributed environments. If this step is not performed properly, an increase in the number of processors may actually result in a decrease in the total system throughput. This is handled properly so that there is no system degradation.

4.1.2 Association Rules

Association rule mining was first introduced by Agrawal et al. [AIS, 93], and was used for market basket analysis. The problem of mining association rules can be explained as follows: There is the itemset I = i₁, i₂, .. .., iₙ, where I is a set of n distinct items, and a set of transactions D, where each transaction T is a set of items such that TC I.

An association rule is an implication of the form X⇒ Y, where X, Y C I and X ∩ Y = Ø. The rule X⇒ Y has support s in the transaction set D if s% of transactions in D contains X∪Y. The support for a rule is defined as support (X∪Y). The rule X⇒Y holds in the transaction set D with confidence c if c% of transactions in D that contain X also contain Y. The confidence for a rule is defined as support (X∪Y)/support(X). For example, consider that people buy bread and butter, they also buy milk in 66% of the cases and 80% of the transactions with bread and butter also contains milk. Such a rule can be represented as
“bread,butter $\Rightarrow$ milk | support=0.66, confidence=0.8”

Not all the rules found are useful and the number of rules generated maybe enormous. Therefore, the task of mining association rules is to generate all association rules that have support and confidence greater than the user-defined minimum support (mins) and minimum confidence (minconf) respectively.

An itemset with minimum support is called the large (or frequent) itemset. The rule $X \Rightarrow Y$ is a strong rule iff $X \cup Y$ is in the large itemset and its confidence is greater than or equal to minconf. Normally, the task of mining association rules can be decomposed into two sub-tasks:

1. Discover all large itemsets in the set of transactions $D$.
2. Use the large itemsets to generate the strong rules. The algorithm for this task is simple. For every large itemset $I_1$, all the large itemsets $I_2$ such that $I_2 \subseteq I_1$ where $\text{support}(I_1 \cup I_2) / \text{support}(I_2) \geq \text{minconf}$. For every such large itemset $I_2$, output a rule of the form $I_2 \Rightarrow (I_1 \setminus I_2)$.

The performance of mining association rules is mainly dependent on the large itemsets discovery process (step 1), since the cost of the entire process comes from reading the database (I/O time) to generate the support of candidates (CPU time) and the generation of new candidates (CPU time). Therefore, it is important to have an efficient algorithm for large itemsets discovery.
4.1.3 Apriori Algorithm

The Apriori algorithm [AS, 94] uses a bottom-up breadthfirst approach to finding the large itemsets. It starts from large 1-itemsets and then extends one level up in every pass until all large itemsets are found. For each pass, say pass k, there are three operations. First, append the large (k-1)-itemsets to L. Next, generate the potential large k-itemsets using the (k-1)-itemsets. Such potential large itemsets are called candidate itemsets C. The candidate generation procedure consists of two steps:

1. Join step – generate k-itemsets by joining Lk-1 with itself.
2. Prune step – remove the itemset X generated from the join step, if any of the subsets of X is not large. Since any subset of a large itemset must be large.

This can be written formally as follows:

\[
\begin{align*}
L1 & := \text{frequent 1-itemsets}; \\
K & := 2; // k represents the pass number \\
\text{while} \ (L_{k-1} \neq \emptyset) \ \text{do} \\
\text{begin} \\
C_k & := \text{New candidates of size k generated from } L_{k-1}; \\
\text{forall} \ \text{transactions } t \ \text{& } D \ \text{do} \\
& \quad \text{Increment the count of all candidates in } C_k \ \text{that are contained in } t; \\
L_k & := \text{All candidates in } C_k \ \text{with minimum support}; \\
k & := k + 1; \\
\text{end}
\end{align*}
\]

Figure 4g: Apriori algorithm
In the last operation, the itemset X is selected from the candidate itemsets where \( \text{support}(X) \geq \text{minsup} \). Figure 1 gives the general Apriori algorithm and Table 4a summarizes the notation used in the algorithm.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k )-</td>
<td>itemset An itemset having ( k ) items</td>
</tr>
<tr>
<td>( C_k )</td>
<td>Set of candidate ( k )-itemsets</td>
</tr>
<tr>
<td>( L_k )</td>
<td>Set of large ( k )-itemsets</td>
</tr>
</tbody>
</table>

Table 4a: Notation

### 4.1.4 Hash Partitioned Apriori (HPA) algorithm

**Algorithm Design**

In the sequential algorithm, the count support processing requires the largest computation time, where the transaction database is scanned repeatedly and a large number of candidate itemsets are examined. A parallel algorithm for count support processing is considered. If each processor can hold all the candidate itemsets, parallelization is straightforward. Figure 4h shows the number of candidate itemsets and the large itemsets in each pass. These statistics were taken from the real point of sales data.
The candidate itemset of pass 2 is too large to fit within the local memory of a single processor. HPA partitions the candidate itemsets over the memory space of all the processors. Thus HPA can exploit the total systems memory effectively as the number of processors increases.

*Parallelization of association rule mining with HPA*

In order to improve the quality of the rule, the analysis is made on very large amounts of transaction data, which requires considerable computation time. Several parallel algorithms are analyzed for mining association rules, based on Apriori. One of these algorithms, called Hash Partitioned Apriori (HPA), is implemented and evaluated. The data allocation among the processors is also taken care of dynamically during the execution of the HPA algorithm. Depending upon the available size of the memory and contents the swap in, swap out process is carried out. HPA partitions the candidate itemsets among processors using a hash function, like the hash join in relational databases. HPA effectively utilizes the whole memory space.
of all the processors, hence it works well for large scale data mining. The steps used in the research work are as follows.

1. Generate candidate $k$-itemsets:

   All processors have all the large $(k - 1)$-itemsets in memory when pass $k$ starts. Each processor generates candidate $k$-itemsets using large $(k - 1)$-itemsets, applies a hash function, and determines a destination processor ID. If the ID is the processors own, the itemset is inserted into the hash table, otherwise it is discarded.

2. Scan the transaction database and count the support value:

   Each processor reads the transaction database from its local disk. It generates $k$-itemsets from those transactions and applies the same hash function used in phase

   The processor then determines the destination processor ID and sends the $k$-itemsets to it. When a processor receives these itemsets, it searches the hash table for a match, and increments the match count.

3. Determine large $k$-itemsets:

   Each processor checks all the itemsets it has and determines large itemsets locally, then broadcasts them to the other processors. When this phase is finished at all processors, large itemsets are determined globally. The algorithm terminates if no large itemset is obtained.
The Pseudo code for the HPA algorithm is

\[ \{C_1^p\} := \text{All items assigned to the } p\text{-th processor based on hashed value} \]

\text{forall } t \in D^p \text{ do}

\text{forall } \text{items } x \in t \text{ do}

Determine the destination processor ID by applying the same hash function which is used in item partitioning and send that item to it. If it is its own ID, increment the support_count for the item. Receive the item from the other processors and increment the support_count for the itemset. Receive k-itemset from the other processors and increment the support_count for that itemset.

\text{end}

end

\[ \{L_1^p\} := \text{All the candidates in } C_1^p \text{ with minimum support.} \]

Send \( L_1^p \) to the coordinator

Receive \( L_1 \) from the coordinator

While \( (L_{k-1} \neq \emptyset) \) do

\[ \{C_p^k\} := \text{All the candidate } k\text{-itemsets, whose hashed value corresponding to the } p\text{-th processor} \]

\text{forall } t \in D^p \text{ do}

\text{forall } \text{k-items } x \in t \text{ do}

Determine the destination processor ID by applying the same hash function which is used in item partitioning and send that k-itemset to it. If it is its own ID, increment the support_count for the itemset. Receive k-itemset from the other processors and increment the support_count for that itemset.

\text{end}

end
\{L_k^p\} := \text{All the candidates in } C_p^k \text{ with minimum support.}

Send \(L_k^p\) to the coordinator

Receive \(L_k\) from the coordinator

\(k := k + 1\)

end

\begin{center}
\begin{tabular}{|c|l|}
\hline
\textbf{L}_k & \text{Set of all large k-itemsets.} \\
\hline
\textbf{C}_k & \text{Set of all the candidate k itemsets.} \\
\hline
\textbf{D}_p & \text{Transactions stored in the local disk of the p-processor.} \\
\hline
\textbf{C}_p^k & \text{Set of k itemsets assigned with p processor.} \\
\hline
\textbf{L}_k^p & \text{Set of large k itemsets derived from } C_p^k \\
\hline
\end{tabular}
\end{center}

Table 4b: HPA algorithm abbreviations

\textit{Execution}

Each workstation of the cluster has a transaction data file on its own hard disk. Transaction data is produced using a data generation program developed by Agrawal, designating some parameters, such as the number of the transaction, the number of different items, etc [CS, 04]. The data produced is divided by the number of workstations, and copied to each workstation’s hard disk. At each workstation, two processes are created and executed. One process makes candidate itemsets from previous large itemsets, and sends them to the other process, which puts the data into a hash table. In the data counting phase, one process generates itemsets by
scanning the transaction data file, and sends them to the other processes on
the workstations selected by the hash function. The target processes check
and increment their hash table values accordingly. During the execution of
HPA, itemsets are kept in memory as linked structures that are classified by
a hash function. That is to say, all itemsets having the same hash value are
assigned to the same hash line on the same workstation, and their
structures are connected with each other to form a list.

As an example, a result of HPA is shown in Table 4c. In this
execution, the number of transactions is 10,000,000, the number of different
items is 5,000, and the minimum support is 0.5%. It is known that the
number of candidate itemsets in pass 2 is very much larger than in other
passes, as can be seen in the table. This often happens in association rule
mining.

<table>
<thead>
<tr>
<th>No of Passes</th>
<th>Candidate Itemsets</th>
<th>Large Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 1</td>
<td></td>
<td>850</td>
</tr>
<tr>
<td>Pass 2</td>
<td>312543</td>
<td>25</td>
</tr>
<tr>
<td>Pass 3</td>
<td>17</td>
<td>30</td>
</tr>
<tr>
<td>Pass 4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Pass 5</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4c: The number of candidate and large itemsets at each step

4.1.5 Count distribution algorithm

Parallel and distributed data mining is the mining of distributed data in a
parallel environment. It intends to obtain global knowledge from local data
at distributed sites. Researchers expect parallelism to relieve current
Association Rule Mining (ARM) [WST, 03] methods from the sequential
bottleneck, providing scalability to massive data sets and improving
response time. The main challenges include synchronization and communication minimization, workload balancing, finding good data layout and data decomposition, and disk I/O minimization. Many parallel data mining algorithms inherits this property from Apriori, which is why most parallel data mining algorithms are said to be a variation of Apriori.

The main challenges associated with parallel data mining include

i. Minimizing I/O

ii. Minimizing synchronization and communication

iii. Effective load balancing

A parallel data mining algorithm Count Distribution is used for the mining process. The main focus of Count Distribution algorithm is on minimizing communication. It does so even at the expense of carrying out redundant duplicate computations in parallel. This algorithm uses a simple principle of allowing redundant computations in parallel on other-wise idle processors to avoid communication. The first pass is special. For all other passes $k > 1$, the algorithm works as follows: [RJ, 96]

1. Each processor $P^i$ generates the complete $C_k$, using the complete frequent itemset $L_{k-1}$ created at the end of pass $k-1$. Observe that since each processor has the identical $L_{k-1}$, they will be generating identical $C_k$.

2. Processor $P^i$ makes a pass over its data partition $D_i$ and develops local support counts for candidates in $C_k$.

3. Processor $P^i$ exchanges local $C_k$ counts with all other processors to develop global $C_k$ counts. Processors are forced to synchronize in this step.
4. Each processor \( P^i \) now computes \( L_k \) from \( C_k \).

5. Each processor \( P^i \) independently makes the decision to terminate or continue to the next pass. The decision will be identical as the processors all have identical \( L_k \).

In the first pass, each processor \( P^i \) dynamically generates its local candidate itemset \( C_1 \) depending on the items actually present in its local data partition \( D_k \). Hence, the candidates counted by different processors may not be identical and care must be taken in exchanging the local counts to determine global \( C_1 \). Thus, in every pass, processors can scan the local data asynchronously in parallel. However, they must synchronize at the end of each pass to develop global counts.

### Count Distribution Algorithm (CDA)

| Step 1. Place data partition at each site. |
| Step 2. In Parallel at each site do |
| Step 3. \( C_1 \) = Itemsets of size one in I; |
| Step 4. Count \( C_1 \); |
| Step 5. Broadcast counts to all sites; |
| Step 6. Determine global large itemsets of size 1, \( L_1 \); |
| Step 7. \( i = 1 \); |
| Step 8. Repeat |
| Step 9. \( i = i + 1 \); |
| Step 10. \( C_i \) = Apriori-Gen(\( L_{i-1} \)); |
| Step 11. Count \( C_i \); |
| Step 12. Broadcast counts to all sites; |
| Step 13. Determine global large itemsets of size \( i \), \( L_i \); |
| Step 14. until no more large itemsets are found; |

Figure 4j: Count Distribution algorithm

107
Method

Steps 1-2 and 4-5 are similar to that of the serial algorithm. The non-obvious step is how processors exchange local counts to arrive at global Ck counts. The details of the implementations steps are dealt with efficiently, separately for passes k > 1 and pass 1.

In this parallel formulation of Apriori algorithm proposed in [KGG*, 94], each processor computes how many times all the candidates appear in the locally stored transactions. This is carried out by using the entire hash tree that corresponds to all the candidates and then performs a single pass over the locally stored transactions to collect the counts. The global counts of the candidates are computed by summing these individual counts using a global reduction operation [TSM, 03]. This algorithm is illustrated above. Note that since each processor needs to build a hash tree for all the candidates, these hash trees are identical at each processor. Thus excluding the global reduction, each processor in the CD algorithm executes the serial Apriori algorithm on locally stored transactions. This algorithm has been empirically shown to scale linearly with the number of transactions.

A detailed scalability analysis is presented by given N number of transactions and P number of processors, if M is the total number of candidates that get generated, then they show that the parallel runtime of the algorithm is $T_s/P + O(M)$, where $T_s$ is the serial runtime of the algorithm. The $O(M)$ term comes from the hash tree construction and global reduction of counts. This indicates that the algorithm is scalable in a number of
transactions; however it does not parallelize the computation of building the candidate hash tree. This step becomes a bottleneck with large number of processors. Furthermore, if the number of candidates is large, then the hash tree does not fit into the main memory [CF, 04]. In this case, this algorithm has to partition the hash tree and compute the counts by scanning the database multiple times, once for each partition of the hash tree. The cost of extra database scanning can be expensive on machines with slow I/O system. Note that the number of candidates increases if either the number of distinct items in the database increases or if the minimum support level the association rules decreases. Thus the CD algorithm is effective for small amount of distinct items and a high minimum support level.

*Performance of Count Distribution algorithm:*

The response time is calculated based on different parameters after the allocation of the task. It is the time taken to perform the mining process after the task allocation and send back the mined results to its destination.

*Scale up:*

To see how well the Count Distribution algorithm handles larger problem sets when more processors are available, Scale up experiments were performed where the size of the database is increased in direct proportion to the number of workstations in the system. Count algorithm scales very well, being able to keep the response time almost constant as the database and multiprocessor sizes increase. Slight increase in response
times are due entirely to more processors being involved in communication. Since the itemsets found by the algorithm does not change as the database size is increased, the number of candidates whose support must be summed by the communication phase remains constant.

Size up

The program is actually more efficient as the database size is increased. Since the results do not change as the database size increases neither does the amount or cost of communication. Increasing the size of the database simply makes the non-communication portion of the code take more time due to more I/O and more transaction processing. This has the result of reducing the percentage of the overall time spent in communication. Since I/O and CPU processing scale perfectly with size up, a sub linear performance is obtained.

Speed up:

In the speed up factor the number of processors has been increased and its performance has been analyzed with the same datasets. The results of running the Count algorithm on configurations of up to 16 processors are given. Larger datasets would have shown even better speedup characteristics.
4.1.6 Dynamic remote memory utilization

The process of utilizing the memory efficiently without any memory leak or overflow is taken care in this research work. In dynamic remote memory utilization the intelligent agents keeps monitoring the distributed workstations and manages the free memory effectively. Load balancing is the assignment of work to processors is critical in parallel simulations. It maximizes application performance by keeping processor idle time and interprocessor communication as low as possible. In applications with constant workloads, static load balancing can be used as a preprocessor to the computation. Other applications, such as adaptive finite element methods, have workloads that are unpredictable or change during the computation; such applications require dynamic load balancers that adjust the decomposition as the computation proceeds. This research work uses the dynamic remote memory utilization which is dealt in previous chapter.

4.1.7 Knowledge integration

There exist many distributed locations today in a business or financial organizations. For example a large company may have subsidiary companies and each subsidiary company has its own data warehouse to make decisions for the development of the company. The decision taken will be in the distributed local workstations, which has to be met globally based on the interestingness measure. The knowledge discovery process takes the raw results from data mining (the process of extracting trends or patterns from data) and carefully and accurately transforms them into useful and understandable information [SS, 08]. The mining process is carried out in
the geographically distributed locations and this outcome knowledge has to be integrated to produce a result. The integration of knowledge is done using Identify exceptional pattern and Bayes network discussed in chapter 5.

The framework of this research work is designed in a way that an optimized response time is obtained in a parallel and distributed data mining environment using certain task allocation algorithms, data mining primitives and knowledge integration models. Data mining is carried out with parallel algorithms like count distribution and hash partitioned apriori.