A DISSERTATION
ON

FAST ASSOCIATION RULE MINING USING DATA STRUCTURE

SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

MASTER OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Himani Digga

University Roll No.:- 2K14/CSE/08

Under the supervision of

Mr. Manoj Sethi

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

DELHI TECHNOLOGICAL UNIVERSITY

DELHI – 110042, INDIA

June 2016

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I, Himani Digga, Roll No. 2K14/CSE/08 of M.Tech (CSE), hereby declare that the dissertation titled "Fast association rule mining using data structure" under the supervision of Mr. Manoj Sethi of Computer Science and Engineering Department, Delhi Technological University in partial fulfillment of the requirement for the award of the degree of Master of Technology has not been submitted elsewhere for the award of any Degree.

Place: Delhi

Date: ______________

Mr. Manoj Sethi

SUPERVISOR

Assistant Professor
Department of Computer Science and Engineering
Delhi Technological University

ACKNOWLEDGEMENT

First of all, I would like to express my deep sense of respect and gratitude to my project supervisor Mr. Manoj Sethi for providing the opportunity of carrying out this project and being the guiding force behind this work. I am deeply indebted to him for the support, advice and encouragement he provided me without which the project could not have been a success.

I am also grateful to Dr. O.P Verma, HOD, Computer Science and Engineering Department, DTU for his immense support. I would also like to acknowledge Delhi Technological University library and staff for providing the right academic resources and environment for this work to be carried out.

Last but not the least I would like to express sincere gratitude to my parents, friends and seniors for constantly encouraging me during the completion of work.

Himani Digga
University Roll no: 2K14/CSE/08
M.Tech (Computer Science and Engineering)
Department of Computer Science and Engineering
Delhi Technological University
Delhi – 110042

ABSTRACT

In the field of data mining, frequent pattern mining has become an important task and has many utilities in various areas. One of the methods to find these frequent patterns for large datasets is using distributed data mining where the database is stored in a distributed way at multiple sites. FDM (Fast Distributed Mining) is one of the many algorithms available for distributed data mining which uses the Apriori algorithm to find the local candidate itemsets at each of the site for a number of passes.

In recent years, many new structures like, Node-list, N-list and Nodesets have been introduced to find the frequent itemset in a more efficient way which uses both the tree structure of FP-Tree and the intersection property of vertical datasets. So, a new algorithm named as FDM-FIN (FDM using FIN) is proposed by using the Nodeset structure in place of array and hash tree structures in the FDM algorithm to generate and store the candidate itemsets by applying the FIN algorithm to generate the candidate itemsets locally at each site instead of using the apriori algorithm. The performance of this algorithm is then compared to the FIN algorithm and FDM using FP Growth where FP Growth is used to generate the candidate sets.

CONTENTS

LIST OF FIGURES AND TABLES

CHAPTER 1

INTRODUCTION

1.1 Association Rule Mining and Approaches

Association rule mining means finding interesting association, correlations and frequent patterns among a large items or objects that are contained in a transaction database, relational database or some other kind of data repository. It helps in decision making process for business purpose like in supermarkets for catalog design, basket data analysis etc. A famous example of this is market basket analysis. Association rule mining involves two main steps:

1. Finding frequent itemsets: It means to find the itemsets which appear as frequently in the dataset as a pre-defined minimum support count.
2. Generating strong association rules using the frequent itemsets: Those rules satisfying conditions of minimum support and minimum confidence are generated.

Among the steps listed above, finding frequent itemsets is an important step. Various algorithms have been proposed to find the frequent itemsets. Association rule mining was first introduced by R.Agrawal in 1993 [4]. After that a lot of work has been done in this area and many new approaches and algorithms have been proposed which can be mainly categorized in three types: Apriori based, frequent pattern growth based and Vertical database format based.

1.2 Distributed Association Rule Mining and Approaches

The voluminous amount of structured and unstructured data created by a company is termed as **Big Data**. It costs too much and takes too much time to load into a relational database for analysis. It is used when speaking about petabytes or exabytes of data and does not refer to any particular quantity. The three characteristics of big data are volume, velocity and variety. Volume refers to the quantity of data generated, velocity refers to the speed with which the data is generated and variety refers to the type and nature of data. Since the size of data is increasing at a much faster rate as compared to increase in the capacity to process the data, there is a need to find a solution for it. The solution for this is either parallelization (Multiple processors in a single machine) or Distributed computing (Multiple computers connected via a network). Among various applications of big data, mining frequent patterns is also one since many interesting patterns can be mined from this huge amount of data.

With the rising size of databases and the need of mining patterns from it, there is need of distributing the mining process among a number of nodes. In distributed mining, the data is stored at different locations and various processors work parallel to provide a fast and efficient result. Distributed data mining finds local frequent patterns, communicates with other nodes and finally finds the global frequent items. Some algorithms have been proposed for distributed frequent pattern mining but lesser as compared to centralized mining of frequent itemsets. Some of the famous distributed data association rule mining algorithms are AprTidRec, Fast Distributed Mining of association rules (FDM), Optimized distributed association rule mining (ODAM) etc.

1.3 Data Structures used for Mining

Data Structure is a way of organizing data in a database. They have an important role in reducing the computational complexity of an algorithm and making it better. In association rule mining also, different types of data structures are used and based on this different frequent itemset mining algorithms have been proposed. Various data structures used in data mining process are:

- **Trie data structure:** It is also called as digital tree, prefix tree or radix tree. It is a prearranged data structure where strings are used to store the keys.
- **FP-tree:** FP tree is used by the FP-Growth algorithm to mine frequent itemsets in a database. It is used so that the database scans can be reduced as compared to the apriori algorithm.
- **N-List:** It is obtained from a prefix tree, PPC-tree, which is a structure like the FP-tree. The N-List stores information obtained from PPC-tree about the itemsets. It stores the preorder, postorder and count in form of PP-code.

- **Nodesets:** It is a more efficient data structure as compared to the N-list since it stores only the preorder or postorder of nodes. It stores count and preorder or postorder in the form of N-info.

1.4 Motivation

In the last few decades the amount of data created in the digital world has rose to a large amount. So there is a dire need to deal with this huge data and find the hidden and interesting patterns from this. We are getting data from sensors, devices in different and independent formats through various applications. Although this data has exceeded our ability to handle, manage, process, store and analyze. For example, in 1998 Google indexed one millions of pages and in 2000 which quickly arrived one billion and in 2008 Google is indexing 1 trillion of pages. This quick explosion of data is also supported by the social networking sites like Facebook, Twitter and other sites like Amazon, flipkart etc. For this much data available, it is not fair to just handle it and store the information. This data can be used to find the knowledge which can be used to uplift our business by getting support in decisions. We can mine the data and find many kinds of interesting things and patterns. One of the interesting thing is association rule, for example by considering the transaction data of marts of goods selling sites we can get the rules like in 90 percent case customer who buys bread also buys milk now this information can be used by placing the bread and milk together.

Mining the association rules is a two steps process:
1. Firstly, all frequent itemset of every size is generated using any of the various algorithms available.
2. Association rules are generated from these frequent itemsets found.

Frequent itemset mining is the first step of Association rule mining. After employing a Frequent itemset mining algorithm like Apriori or FP-Growth on the dataset, frequent itemsets having minimum support count are generated. And then, after getting these frequent itemsets, we can generate association rules.

Here in this work we are focusing on distributed computing where both the data and the processing are distributed among a number of systems. One of the available algorithms for distributed association rule mining is the FDM (Fast Distributed Algorithm for mining Distributed Association Rules) where Apriori algorithm is used to generate the local frequent itemsets at each of the site. Also, FIN (Fast mining frequent itemsets using Nodesets) is a new and improved approach based on the PrePost algorithm which combines the advantages of both the apriori like algorithms and the FP-Growth like algorithms. So, in an attempt to improve the performance of FDM, instead of using apriori algorithm in FDM to generate the local frequent itemsets, we are using the FIN algorithm.

1.5 Research Objective

As explained in the motivation above, the research objective of this work is to use the FDM algorithm to find the frequent itemsets in a distributed environment where both the data and the processing are distributed among multiple nodes.

The problem statement is given below as taken from [1] is defined below:

Let$I = \{i_1, i_2, \dots, i_m\}$be a set of items. Let DB be a database of transactions, where each transaction T consists of a set of items such that$T \subseteq I$. Given an itemset$X \subseteq I$, a transaction T contains X if and only if $X \subseteq T$. An association rule is an implication of the form $\Rightarrow Y$, where,$X \subseteq I, Y \subseteq I$and$X \cap Y = \phi$.The association rule $X \Rightarrow Y$ holds in DB with confidence c if the probability of a transaction in DB which contains X also contains Y isc. The association rule$X \Rightarrow Y$has support s in DB if the probability of a transaction in DB contains both X and Y is s. The task of mining association rules is to find all the association rules whose support is larger than a minimum support threshold and whose confidence is larger than a minimum confidence threshold.

For an itemset X, its support is the percentage of transactions in DB which contains X, and its support count, denoted by X.sup, is the number of transactions in DB containing X. An itemset X is large (or more precisely, frequently occurring) if its support is no less than the minimum support threshold. An itemset of size k is called a k-itemset. It has been shown that the problem of mining association rules can be reduced to two sub problems [4]: (1) find all large itemsets for Q given minimum support threshold, and (2) generate the association rules from the large itemsets found. Since (1) dominates the overall cost of mining association rules, the research has been focused on how to develop efficient methods to solve the first sub problem [4].

Distributed algorithm for mining association rules as stated in [1]:

We examine the mining of association rules in a distributed environment. Let DB be a database with D transactions. Assume that there are n-sites $S_1, S_2, \dots, S_n$in a distributed system and the database DB is partitioned over the n sites into $\{DB_1, DB_2, \dots, DB_n\}$respectively.

Let the size of the partitions DB_ibe D_i, for i = 1 , 2, . . . , n. Let X.supand $X.Sup_i$be the support counts of an itemsetX in DB and DB_i, respectively. X.supis called the global support count, and $X.Sup_i$ the local support count of X at siteS_i. For a given minimum support threshold s, X is globally large if $X.sup \geq s \times D$; correspondingly, X is locally large at site S_i, if $X.sup_i \geq s \times D_i$. In the following, L denotes the globally large itemsets in DB, and $L_{(k)}$the globally large k-itemsets in L. The essential task of a distributed association rule mining algorithm is to find the globally large itemsets L.

1.6 Report Organization

We have started with an introduction of association rule mining, distributed association rule mining and the data structures used for this. Rest of the report is organized into 5 more chapters. Chapter 2 is a literature survey of many different algorithms and techniques used in this field of association rule mining and the distributed association rule mining. Next chapter 3 is related algorithms which gives a detailed description of the FDM algorithm and FIN algorithm with which our algorithm is closely related. Chapter 4 is the problem definition where a detailed description of our problem is given. In chapter 5, the proposed work is described where the proposed algorithm is explained. Next chapter 6 is of results and Evaluation where the performance of the proposed algorithm is compared with some existing algorithm. Then the conclusion and future work is described.

CHAPTER 2

LITERATURE REVIEW

2.1 Data Mining

Data mining [5] is the analyzing step of the KDD process or knowledge Discovery in Database process. It is a somewhat intersection of machine learning, database system, artificial intelligence and statistics and thus we can say it is the subfield of the computer science which, discover patters in the database. The main purpose of data mining is to take out the useful information from the largest database and translate it into the understandable form. Apart from these, data mining also involves data preprocessing, modeling, interestingness metrics, complexity consideration, visualization and post processing.

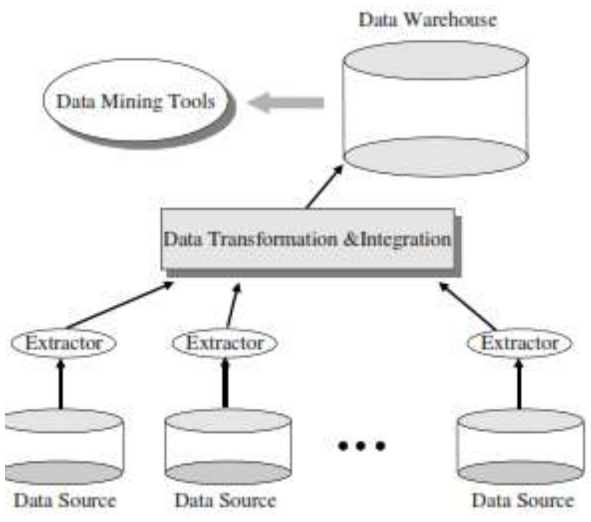

Figure 2.1: Steps in Data Mining

The Process of data mining consists of following steps:

(1) Selection

(2) Pre-processing

(3) Transformation

(4) Data Mining

(5) Interpretation/Evaluation.

Also, there are still many variables in the process, CRISP DM specifies six steps:

(1) Business Understanding

(2) Data Understanding

(3) Data Preparation

(4) Modeling

(5) Evaluation

(6) Deployment

Or, process such as (1) pre-processing, (2) data mining, and (3) results validation.

2.1.1 Pre-processing

Data in a data warehouse or database is very large, it needed to be clean and transformed into a more concise and precise way, so that we can easily uncover the hidden patterns present in the database. To analyze the multivariate data sets, we need preprocessing to remove unwanted information or noise or missing data.

2.1.2 Data mining

Data mining consists mainly 6 classifications of tasks:

- Anomaly detection –Information that need a further probe.
- Association rule learning – Exploration for relationships between variables like finding customer habit of purchasing.
- Clustering – Gathering the point into groups on some basis like similarity.
- Classification –Generalization of tasks
- Regression – Finding method having least error.
- Summarization – Allowing for a more compact and efficient representation of the data.

2.1.3 Results validation

Data mining produces so many unnecessary results which at first seem to be useful but are actually not useful for other sample of data or for future predictions. So there is a great need of executing proper statistical testing of all the hypotheses.

2.2 Frequent Itemset Mining and algorithms

Frequent itemset mining is the first and the main step in association rule mining where the frequent occurring itemsets having count greater than the support value are found. After finding the frequent itemsets, association rules are made from these frequent itemsets. Many surveys have been done on this [14,20].

Association rule mining gained popularity with an article published in 1993 by Agrawal [4] which has been sited more than 17000 times according to google scholar. Many algorithms for frequent itemset mining have been proposed in the last 2-3 decades. These can be divided into three types:

- Apriori based
- Frequent pattern growth based
- Vertical database format based

The apriori like algorithms are based on anti-monotone property [6], called Apriori, which states that any k-size itemset will be frequent if and only if all its (k-1) sized itemsets are frequent. These algorithms apply a candidate set generation and test strategy to find the frequent itemsets which means first it generates all the candidate itemsets and then check for their support count. If the count is greater than or equal to the support threshold, only then that candidate itemset is frequent and then it is used to generate the larger candidate sets.

Along with the Apriori, Agrawal also proposed its two variations namely AprioriTID [6] and AprioriHybrid [6]. Apriori is the famous algorithm in which we passes the database through the number of scans and count the frequency of the candidate itemsets previously calculated. In the first scan we get the frequency of the entire item in the database and minimum support items are size one frequent itemsets. Those who pass the minimum support criteria are called frequent itemset, and these are used to calculate the candidate itemsets for the next iteration. AprioriTID [6] has an impressive feature, after the first pass, it does not use the database for finding the frequency of the items it uses another data structure to prune the transaction in the database. AprioriHybrid [6] is another variation that uses Apriori in the first pass and shift to AprioriTID [6] when it expects that the candidate itemset will fit in the memory at the end of the pass. Apripori like algorithms perform good by reducing the size of candidate sets but they are highly expensive also since database is to be traversed many times.

In 2000, another algorithm came named as Eclat [7], discovered by M.J Zaki for faster discovery of association rules in a database having vertical format of dataset using the structural properties of itemsets that are frequent. It organized the lattice search space into small chunks or sublattice. Using an effective traversal method of the approach, which can identify long frequent itemsets quickly, many other algorithms based on eclat have also been proposed.

Apart from these approaches, also came another algorithm FP-Growth[3] in 2000 where no candidate sets are generated to mine frequent itemsets, Here database is scanned once to find size-1 frequent itemsets and then a tree like structure called FP-Tree is made by scanning the database for second time. Then the frequent itemsets are mined from this tree structure. Its advantage is that it reduces the search space and finds frequent itemsets without candidate generation but the process of constructing and mining the FP-Tree is complex.

For sparse dataset, another algorithm [15] was introduced where a FP-array technique is used. This approach might use a lot of memory but is efficient for sparse datasets. A space preserving algorithm was introduced in 2007 [16]. A data structure H-struct was used in it. This algorithm has a predictable main memory cost and runs very quickly in memory-based settings.

In 2012, came a new algorithm for fast mining frequent itemsets using a new structure called, N-List which is a novel vertical data representation originating from a FP-Tree like structure called as PPC-Tree [9]. Based on this an algorithm called as PrePost is developed to mine all the frequent itemsets. Though it is very fast algorithm but consume more memory for sparse datasets. A hybrid approach was introduced which presented an improved version of PrePost that uses a hash table to enhance the process of creating the N-lists associated with 1-itemsets and an improved N-list intersection algorithm.

Based on the PrePost algorithm, came another algorithm FIN, Fast mining frequent itemset using Nodesets in 2014 [2]. It is also based on PPC-Tree, but makes a different structure Nodeset to store information of each node using only preorder or postorder of the node. It is better than the previous algorithms both in running time and memory usage. DFIN [21] is another algorithm based on nodeset structure and using a structure named as diffnodesets.

Another algorithm based on this PPC-Tree structure came in 2015 [10], named as PrePost$^+$. It also uses the N-List to represent the itemsets and set enumeration search tree to discover the frequent itemsets. The difference is that it employs an efficient pruning strategy named as Children-Parent equivalence pruning to reduce the search space. To minimize the pointers between the nodes a structure called linear prefix tree [26] was introduced which is composed of array forms.

2.3 Distributed data association rule mining and algorithms

Distributed database environment is database is partitioned and stored at different locations and processing is done in a parallel way. Here local frequent itemsets are found by each site, then communication takes place among the sites and finally global frequent itemsets are found. It requires proper communication, storage space and processing capabilities.

Count Distribution (CD) [11] is a simple algorithm where data is parallelized and apriori algorithm is run parallely. In it, local support count for each itemset is found and then communicated to each site and hence the global frequent itemsets are found by all the sites. The number of messages exchanged for support count exchange is just O(n^2).

AprTidRec, an algorithm based on apriori algorithm was proposed in 2011 [12].It is different from apriori because it has only the joint step and no pruning step. It generates a record structure called tidRec and their intersection for each candidate frequent itemset. This has lesser running time than apriori algorithm.

FDM (Fast distributed mining of association rules), proposed in 1996 [1] is based on apriori and Count Distribution algorithms. It first finds the locally large itemsets, which are then communicated to the respective polling sites and then the global count is found to find the global frequent itemsets. The number of messages exchanged for support count exchange is just O(n).

DMA [17] is another algorithm for distributed association rule mining. It generates a small number of candidate sets and requires only *O(n)* messages for support count exchange for each candidate set, where *n* is the number of sites in a distributed database.

ODAM (Optimized Distributed Association Rule Mining) [13] is another algorithm for distributed data association rule mining. After discovering the global frequent-1 itemsets, it removes the infrequent ones and inserts the transactions and their count in a temporary file which is then used to find the frequent itemsets of larger lengths. It focuses on the communication and synchronization issues.

Using the map-reduce approach also, many algorithms can be processed in distributed environment, like the MRPrepost [27] algorithm gives the processing of prepost algorithm on the Hadoop platform, cloud implementation of Prepost [29] and apriori map reduce [28] is also there.

2.4 Problem Definition

The fast distributed mining algorithm uses the apriori algorithm to generate the local candidate itemsets at each of the site. It uses an array structure and a hash tree to store the candidate sets. To create the candidate itemsets for each pass, there are multiple passes in the fast distributed mining algorithm also. It ends when either no candidates are generated or no global large frequent itemsets are found in a pass. So, it scans database at each site multiple times which is time consuming for large databases.

Now nodeset is a new structure used in FIN algorithm which makes a preorder tree or a postorder tree and then nodesets for all frequent size-1 and size-2 itemsets are made and also it does not require scan the database multiple times. Now since the nodeset structure used in the FIN alggborithm uses the approach of FP tree and the intersection property of vertical database to generate the frequent itemsets and is much more efficient than the apriori algorithm, it can be used to generate the local candidate itemsets at each of the site in a better way.

Also since the data is distributed and the processing is also distributed, the distributed version of FIN algorithm should give better performance as compared to the sequential FIN.

CHAPTER 3

RELATED ALGORITHMS

3.1 FDM [1]

FDM is an interesting distributed association rule mining algorithm which finds an important relationship between locally large and globally large itemsets. It was proposed back in 1996 by D.W.Cheung, J.Han, V.T.Ng, A.W.Fu and Yongjian Fu.

Symbol	Description
D	Number of transactions in DB
s	Support threshold minsup
L_k	Globally large k-itemsets
CA_k	Candidate sets generated from L_k
X.sup	Global support count of X
D_i	Number of transactions in DB_i
$GL_{i(k)}$	gl-large k-itemsets at S_i
$CG_{i(k)}$	Candidate sets generated from $GL_{i(k-1)}$
$LL_{i(k)}$	Locally large k-itemsets in $CG_{i(k)}$
$X.sup_i$	Local support count of X at S_i

Table 3.1: Notation Table of FDM

Pseudocode for FDM-LP: FDM with Local Pruning algorithm

$Input: DB_i(i = 1,2, \dots n): the\ database\ partition\ at\ each\ site\ S_i.$

$Output: L: the\ set\ of\ all\ globally\ large\ itemsets.$

$Method: Iteratively\ execute\ the\ followin\ program\ fragment\ (for\ the\ k - th\ iteration)\ distributively\ at\ each\ site\ S_i. The\ algorithm\ terminates\ when\ either\ L_{(k)} = \emptyset, or\ the\ sets\ of\ candidate\ sets\ CG_{(k)} = \emptyset.$

```
1)  $if\ k = 1,\ then$
2)          $T_{i(1)} = get\_local\_count(DB_i, \emptyset, 1)$
3)  $else$
4)          $CG_{(k)} = \cup_{i=1}^{n} CG_{i(k)}$
                    $= \cup_{i=1}^{n} Apriori\_gen(GL_{i(k-1)});$
5)          $T_{i(k)} = get\_local\_count(DB_i, CG_{(k)}, i);\}$
6)  $for\_all\ X \in T_{i(k)}\ ,\ do$
7)          $if\ X.sup_i \geq s \times D_i\ then$
8)                  $for\ j = 1\ to\ n\ do$
9)                          $if\ polling\_site(X) = S_i then$
                                    $insert(X, X.sup_i) into\ LL_{i,j(k)};$
10) $for\ j = 1\ to\ n,\ do\ send\ LL_{i,j(k)}\ to\ site\ S_j;$
11) $for\ j = 1\ to\ n,\ do\ \{$
12)         $receive\ LL_{i,j(k)};$
13)         $for\ all\ X \in LL_{j,i(k)}\ do\ \{$
14)                 $if\ X \notin LP_{i(k)}\ then$
                            $insert\ X\ into\ LP_{i(k)};$
15)                 $update\ X.large\_sites;\}\}$
16) $for\_all\ X \in LP_{i(k)}\ do$
17)         $send\_polling\_request(X);$
18) $reply\_polling\_request(T_{i(k)});$
19) $for\_all\ X \in LP_{i(k)}\ do\ \{$
20)         $receive\ X.sup_j\ from\ all\ the\ sites\ S_j$
            $where\ S_j \notin X.large\_sites;$
```

```
21)         $X.sup = \sum_{i=1}^{n} X.sup_i;$
22)         $if\ X.sup \geq s \times D\ then$
23)                 $insert\ X\ into\ G_{i(k)}$ ; }
24) $broadcast\ G_{i(k)}$ ;
25) $receive\ G_{j(k)}\ from\ all\ the\ other\ sites\ S_j\ (j \neq i);$
26) $L_{(k)} = \bigcup_{i=1}^{n} G_{i(k)}.$
27) $divide\ L_{(k)}\ into\ GL_{i(k)}, (i = 1, \dots, n);$
28) $return\ L_{(k)}.$
```

The **get_local_count( ) function** used in line 2 and line 5 is used to get the count of the candidate itemsets in the database.

Apriori_gen( ) function in step 4 takes $L_{(k-1)}$as input, which is the set of large k-1 itemsets and outputs the set of k-itemsets. This function contains two steps, first join step second prune step.

Join step join $L_{(k-1)}$ with $L_{(k-1)}$

Insert into $C_{(k)}$

Select p. item_1 , p. item_2 , p. item_3 ,…., p. $\text{item}_{(k-1)}$, q. $\text{item}_{(k-1)}$

From $L_{(k-1)}$ as p, $L_{(k-1)}$as q

Where p. item_1 =q.item_1,…….,p.$\text{item}_{(k-1)}$ = q.$\text{item}_{(k-1)}$, p.$\text{item}_{(k-1)}$<q.$\text{item}_{(k-1)}$

Prune step is the one in which we delete all the subsets of the join step whose subset of size k-1 is not present in the $L_{(k-1)}$.

```
For all itemsets x ϵ $C_{(k)}$do
        For all (k-1) subsets $s$ of x do
                if ($s$ is not in $L_{(k-1)}$)
                        delete x from $C_{lil}$
```

Explanation of the algorithm:

Every site takes up the roles of home site, polling site and then the remote site. The activities involved in these are grouped below:

1. Home Site: generate the candidate itemsets and send them to the polling sites (line 1-10). Here the database is scanned for the support count and the locally large ones are sent to their respective polling sites along with their support count.
2. Polling Site: receive candidate sets and send polling requests (line 11-17). The candidate sets from other sites are received and request is sent to the sites which are not in the large sites to collect the remaining support count.
3. Remote Site: return support count to polling site (line 18). The site receiving polling request will find the count of that candidate set from its hash tree and return the result.
4. Polling Site: receive support count and find large itemsets (line 19-23). After receiving count from all sites, global support is found and then the globally large itemsets are broadcasted.
5. Home Site: receive large itemsets (line 24-27). The globally large k-itemsets are received and hence the set of size-k large itemsets is found which is used in next iteration to find the candidate sets.

3.2 FIN [2]

This algorithm was proposed by Zhi-Hong Deng and Sheng-Long Lv in [2] in 2014. It mines the frequent itemsets using the structure called nodeset. The algorithm for this is explained below:

Pseudo code for FIN:

$Input: A\ transaction\ Database\ DB\ and\ a\ minimum\ support\ \xi.$

$Output: F, the\ set\ of\ all\ freuent\ itemsets.$

```
1)  $F \leftarrow \Phi$
2)  $Construct\ the\ POC\ tree\ and\ find\ F_1, the\ set\ of\ all\ frequent - 1\ itemsets.$
3)  $F_2 \leftarrow \Phi$
4)  $Scan\ the\ POC - tree\ by\ the\ pre - order\ traversal\ do$
5)  N$\leftarrow currently\ visiting\ node;$
6)  $i_y \leftarrow the\ item\ registered\ in\ N;$
7)  $For\ each\ ancestor\ of\ N, N_a\ , do$
8)  $i_x \leftarrow the\ item\ registered\ in\ N_a;$
9)  $If\ i_x i_y \in F_2, then$
10)         $i_x i_y.support \leftarrow i_x i_y.support + N.account;$
11)         $E$lse
12)         $i_x i_y.support \leftarrow N.account;$
13)         $F_2 \leftarrow F_2 \cup \{i_x i_y\};$
14)         $Endif$
15)         $Endfor$
16)         $For\ each\ itemset, P, inF_2\ do$
17)         $If\ P.support < \xi\ \times |DB|, then$
18)         $F_2 \leftarrow F_2 - \{P\};$
19)         $Else$
20)         $P.Nodeset \leftarrow\ \Phi;$
21)         $Endif$
22)         $Endfor$
```

```
23)    $Scan\ the\ POC-Tree\ by\ the\ pre-order\ traversal\ do$
24)    $Nd \leftarrow currently\ visiting\ node;$
25)    $i_y \leftarrow the\ item\ registered\ in\ Nd;$
26)    $For\ each\ ancestor\ of\ Nd, Nd_a, do$
27)    $i_x \leftarrow the\ item\ registered\ in\ Nd_a;$
28)    $If\ i_x i_y \in F_2, then$
29)    $i_x i_y.Nodeset \leftarrow i_x i_y.Nodeset \cup Nd.N_{info};$
30)    $Endif$
31)    $Endfor$
32)    $F \leftarrow F \cup F_1;$
33)    $For\ each\ frequent\ itemset, i_s i_t, in\ F_2 do$
34)    $Create\ the\ root\ of\ a\ tree, R_{st}, and\ label\ it\ by\ i_s i_t;$
35)    $Constructing_{Pattern\ \ Tree(R_{st},\{i|i\in F_1, i>i_s\},\emptyset)};$
36)    $Endfor$
37)    $Return\ F;$
```

In the above algorithm, frequent 1 itemsets are found. Using the POC-Tree, all candidate frequent-2 itemsets are found. Then delete the infrequent itemsets and initialize the nodesets of all frequent-2 itemsets by null. Using the preorder traversal, generate the nodesets of all frequent-2 itemsets. Then call the procedure Constructing_Pattern_Tree() to generate the frequent-k itemsets.

Procedure Constructing_Pattern_Tree(Nd, Cad_set, FIS_parent)

```
1) $Nd.equivalent\_items \leftarrow \emptyset;$
2) $Nd.childnodes \leftarrow \emptyset;$
3) $Next\_Cad\_set \leftarrow \emptyset;$
4) $For\ each\ i \in Cad_{set} do$
5) $X \leftarrow Nd.itemset;$
6) $Y \leftarrow \{i\} \cup (X - X|1|);$
7) $P \leftarrow \{i\} \cup X;$
8) $P.Nodeset \leftarrow X.Nodeset \cap Y.Nodeset;$
```

```
9)  $P.support = X.support\ then$
10) $Nd.equivalent\_items \leftarrow Nd.equivalent\_items \cup \{i\}$;
11) $Else\ if\ P.support \geq |DB| \times \xi, then$
12) $Create\ node\ Nd$;
13) $Nd_i.label \leftarrow i$;
14) $Nd_i.itemset \leftarrow P$;
15) $Nd_i.childnodes \leftarrow Nd_i.childnodes \cup \{Nd_i\}$;
16) $Next_{cad_{set}} \leftarrow Next_{cad_{set}} \cup \{i\}$;
17) $Endif$
18) $Endif$
19) $Endfor$
20) $If\ Nd.equivalent\_items \neq \emptyset\ then$
21) $SS \leftarrow the\ set\ of\ all\ subsets\ of\ Nd.equivalent\_items$;
22) $PSet \leftarrow \{A | A = Nd.label \cup A', A' \in SS\}$;
23) $If\ FIS\_parent = \emptyset, then$
24) $FIT\_Nd \leftarrow PSet$;
25) $Else$
26) $FIT\_Nd \leftarrow \{P' | P' = P_1 \cup P_2, (P_1 \neq \emptyset \wedge P_1 \in PSet)\ and\ (P_2 \neq \emptyset \wedge P_2 \in FIS\_parent)\}$;
27) $Endif$
28) $F \leftarrow F \cup FIT\_Nd$;
29) $Endif$
30) $If\ Nd.childnodes \neq \emptyset\ then$
31) $For\ each\ Nd_i \in Nd.childnodes\ do$
32) $Constructing\_Pattern\_Tree(Nd_i, \{j | j \in Next_{Cad_{set}}, j \succ i\}, FIT\_Nd)$;
33) $Endfor$
34) $Else\ return$;
35) $Endif$
```

Here in the above algorithm, three input parameters are there: Nd, Cad_set, ex_frequent_itemsets. Nd is the current node in set enumeration tree. Cad_set are available items used to extend Node Nd. FIS_parent are the frequent itemsets generated on the parent of Nd. A pruning technique, promotion is used here.

CHAPTER 4

PROPOSED WORK

This chapter deals with the proposed work for the problem defined in the above chapter. In FDM, the apriori algorithm has multiple passes and uses a less efficient data structure. So to make the FDM algorithm perform in a better way, a new data structure, nodeset proposed in [4] is used to generate the candidate itemsets to find the local frequent itemsets at each of the site.

The main steps in the algorithm are explained below:

1. The size-1 local frequent items are found at each site and sent to polling sites.
2. The polling site receives these local large items and send request to sites where the item was not frequent.
3. Global count is found and hence the global frequent size-1 items are broadcasted to all sites.
4. Based on these globally large size-1 items, the infrequent items are removed.
5. With the application of FIN algorithm, all local large itemsets having size greater than 1 are found and are sent to respective polling sites.
6. Again the polling site receives these local large itemsets and send request to sites where the item was not frequent.
7. Global count is found and hence the global frequent itemsets are broadcasted to all sites.

Symbol	Description
D	Number of transactions in DB
s	Support threshold minsup
L_k	Globally large k-itemsets
CA_k	Candidate sets generated from L_k
X.sup	Global support count of X
D_i	Number of transactions in DB_i
$GL_{i(k)}$	gl-large k-itemsets at S_i
$CG_{i(k)}$	Candidate sets generated by FIN algorithm
$LL_{i(k)}$	Locally large k-itemsets in $CG_{i(k)}$
$X.sup_i$	Local support count of X at S_i

Table 4.1: Notation Table for proposed algorithm

The algorithm is named as FDM-FIN (FDM using FIN). It is described below:

$\boldsymbol{Input}$: $DB_i(i = 1,2,\dots n)$: $the\ database\ partition\ at\ each\ site\ S_i, minimum\ support,$

$total\ transactions\ count, number\ of\ nodes$

$\boldsymbol{Output}$: L: $the\ set\ of\ all\ globally\ large\ itemsets.$

$\boldsymbol{Method}$: $Execute\ the\ following\ program\ twice\ ie. k = 2, first\ for\ size$

$-1\ items\ and\ then\ for\ itemsets\ with\ size\ greater\ than\ 1.$

```
1) $if\ k = 1, then$
2)         $T_{i(1)} = get\_local\_count(DB_i)$
3) $else$
4)         $CG\ \ = Algo\_FIN(DB_i, minimum\ support, transaction\ count);$
```

```
5)          $T_{i(k)} = get\_local\_count(DB_i, CG_{(k)}, i); \}$
6)  $for\_all\ X \in T_{i(k)}$ , $do$
7)           $if\ X.sup_i \geq s \times D_i\ then$
8)                  $for\ j = 1\ to\ n\ do$
9)                        $if\ polling\_site(X) = S_i then$
                                  $insert(X, X.sup_i) into\ LL_{i,j(k)};$
10) $for\ j = 1\ to\ n,\ do\ send\ LL_{i,j(k)}\ to\ site\ S_j;$
11) $for\ j = 1\ to\ n,\ do\ \{$
12)          $receive\ LL_{i,j(k)};$
13)          $for\ all\ X \in LL_{j,i(k)}\ do\ \{$
14)                 $if\ X \notin LP_{i(k)}\ then$
                          $insert\ X\ into\ LP_{i(k)};$
15)                 $update\ X.large\_sites; \}\ \}$
16) $for\_all\ X \in LP_{i(k)}\ do$
17)          $send\_polling\_request(X);$
18) $reply\_polling\_request(T_{i(k)});$
19) $for\_all\ X \in LP_{i(k)}\ do\ \{$
20)          $receive\ X.sup_j\ \ from\ all\ the\ sites\ S_j$
             $where\ S_j \notin X.large\_sites;$
21)          $X.sup = \sum_{i=1}^{n} X.sup_i;$
22)          $if\ X.sup \geq s \times D\ then$
                  $insert\ X\ into\ G_{i(k)}\ ; \}$
23) $broadcast\ G_{i(k)}$ ;
24) $receive\ G_{j(k)}\ from\ all\ the\ other\ sites\ S_j\ \ (j \neq i);$
25) $L_{(k)} = \bigcup_{i=1}^{n} G_{i(k)}.$
26) $if(k = 1)$
27) $remove\_infrequent(DB_i);$
28) $return\ L_{(k)}$
```

Explanation of the algorithm:

1. Home Site: generate candidate sets and submit them to the respective polling site (line 1-10)

   When the value of k is equal to 1, the site calls get_local_count to scan the DB_i once and store the local count of size-1 items and support in a map structure $T_{i(1)}$. And for the next pass when all local frequent itemsets of size greater than 1 are to be found, the candidate sets are generated using the FIN algorithm and are then stored in CG. Then the database is scanned and the local count of each itemset in CG is stored in map structure T_i. The polling site of the locally large itemsets is found and the itemsets are sent to respective sites.
2. Polling Site: receive candidate sets and send polling request (line 11-17)

   As the polling site, site S_i receives the candidate itemsets from other sites. It stores the itemsets in $LP_{i(k)}$ and the sites from which the itemsets are received are stored in X.large_sites. Then a polling request is sent to sites not in X.large_sites to collect the remaining support of that itemset.
3. Remote site: return support count to polling site (line 18)

   When a site receives polling request from another site, it checks the support count of that particular itemset in its map structure T_i and send it to the polling site.
4. Polling site: receive support counts and find the large itemsets (line 19-23)

   As a polling site, S_i receives the support count from the other sites for a candidate itemset. Then it computes the global count of the candidate itemset and comparing it with the minimum support condition, the global large itemsets are found and are stored in $G_{i(k)}$. This is finally broadcasted to all the sites.
5. Home site: receive globally large itemsets (line 24-28)

   Finally as a home site, all the frequent itemsets are received. And if it is the first pass then the dataset is updated and all the infrequent size-1 itemsets are removed from the database. And the final set of large itemsets is returned.

CHAPTER 5

RESULTS AND EVALUATION

This chapter deals with the environment used to run the algorithm and it is then evaluated depending upon various parameters.

5.1 Environment used

The configurations of the sites are listed below in table 5.1:

Operating System	Windows 7
RAM	2GB
Hard Disk Drive	500 GB
CPU	2.10 GHz
System Type	32 bit
JAVA	JDK 1.7
Eclipse	Eclipse Indigo version

Table 5.1: Configuration of the Node

The algorithms are run on a dataset named **Mushroom.** The Mushroom dataset is a real dataset and is downloaded from FIMI repository [30]. This dataset contains characteristics of various species of mushrooms. Various specifications are given below:

- Average length = 23
- Number of Items = 119
- Number of Transactions = 8124

Three algorithms are taken to run for comparison on the basis of execution time. These three algorithms are FIN, FDM using FIN and FDM using FP Growth. The FIN algorithm is taken as the base algorithm for comparison and other two are the new ones created by us. FIN is a sequential algorithm and other two are having both distributed data and distributed processing.

5.2 Results

The algorithms are run with five minimum supports of 0.5, 0.6, 0.7, 0.8, 0.9 and three setups where 2 nodes, 3 nodes and 4 nodes are included and compared on the basis of execution time. The results are shown and compared below using line graphs.

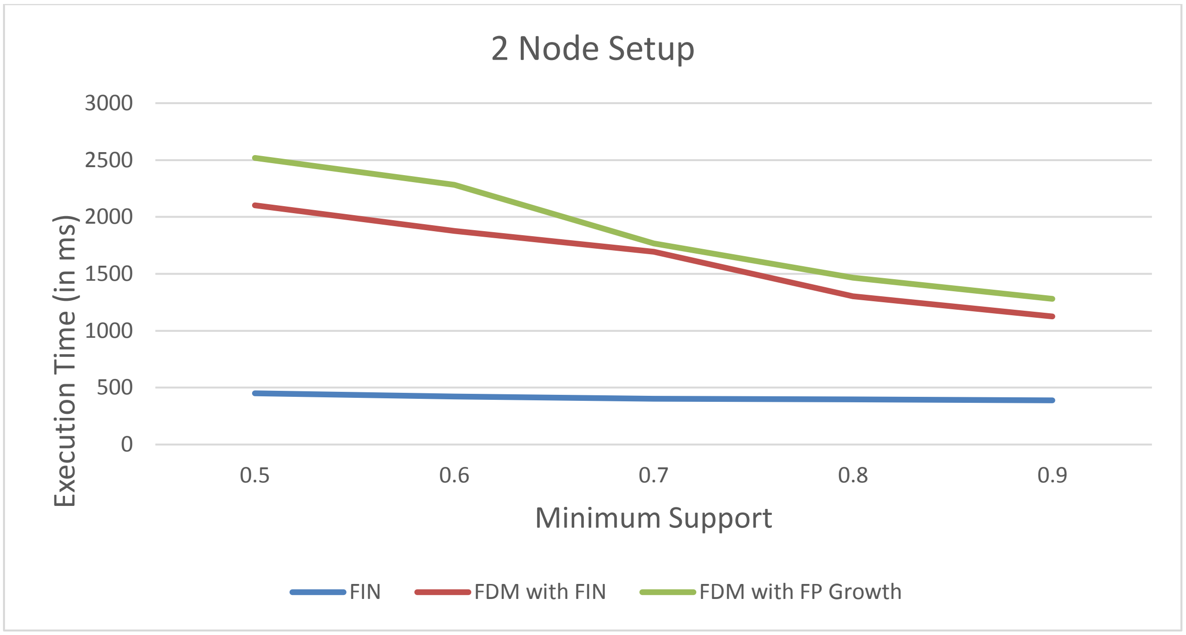

Figure 5.1: Graph for 2-Node setup

From the above graph we can easily that our proposed algorithms are not better than the sequential algorithm in terms of the execution time.

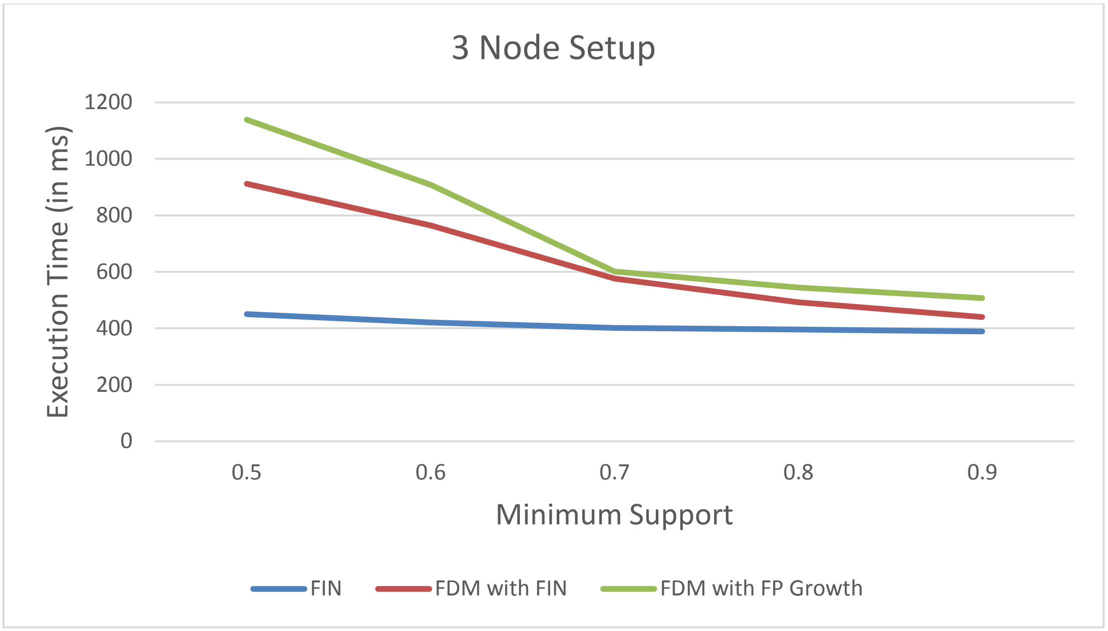

Figure 5.2: Graph for 3-Node setup

Now from the above figure, we can see that still the performance is not good though it is better than the one where two nodes were used to run the algorithm.

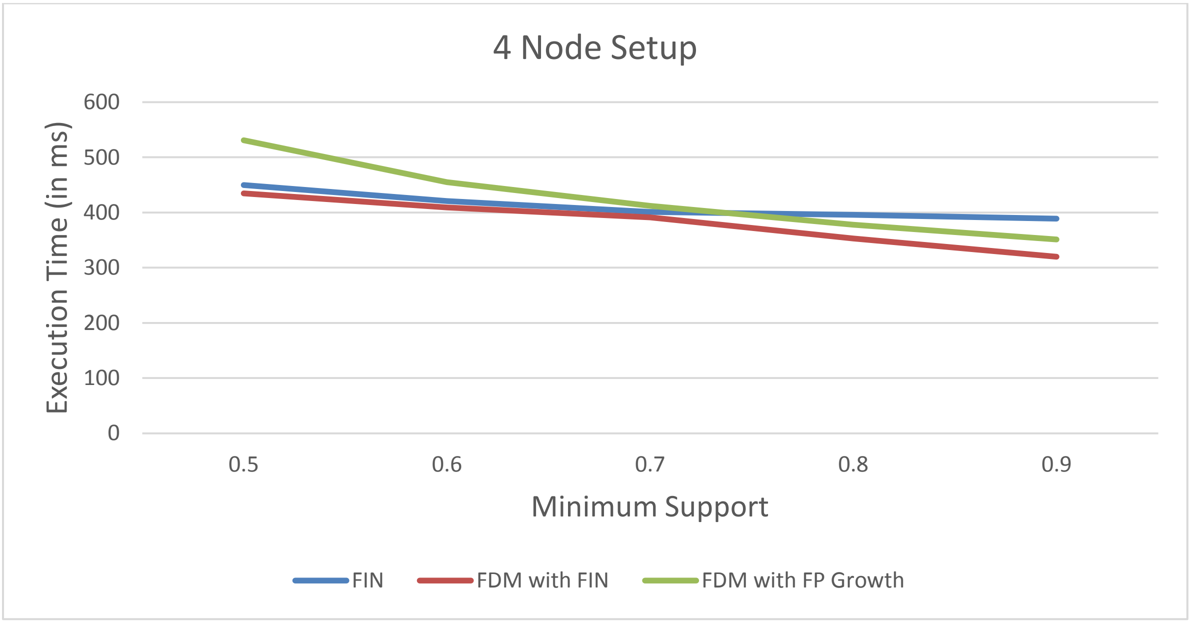

Figure 5.3: Graph for 4-Node setup

The above graph for four node setup shows that now our algorithm is performing better than the sequential FIN algorithm in terms of execution time.

To have a look of how the addition of nodes in the setup is helping to improve the performance of the system, a graph is shown below in figure 5.4. Here the execution time for different number of nodes for different support counts are compared.

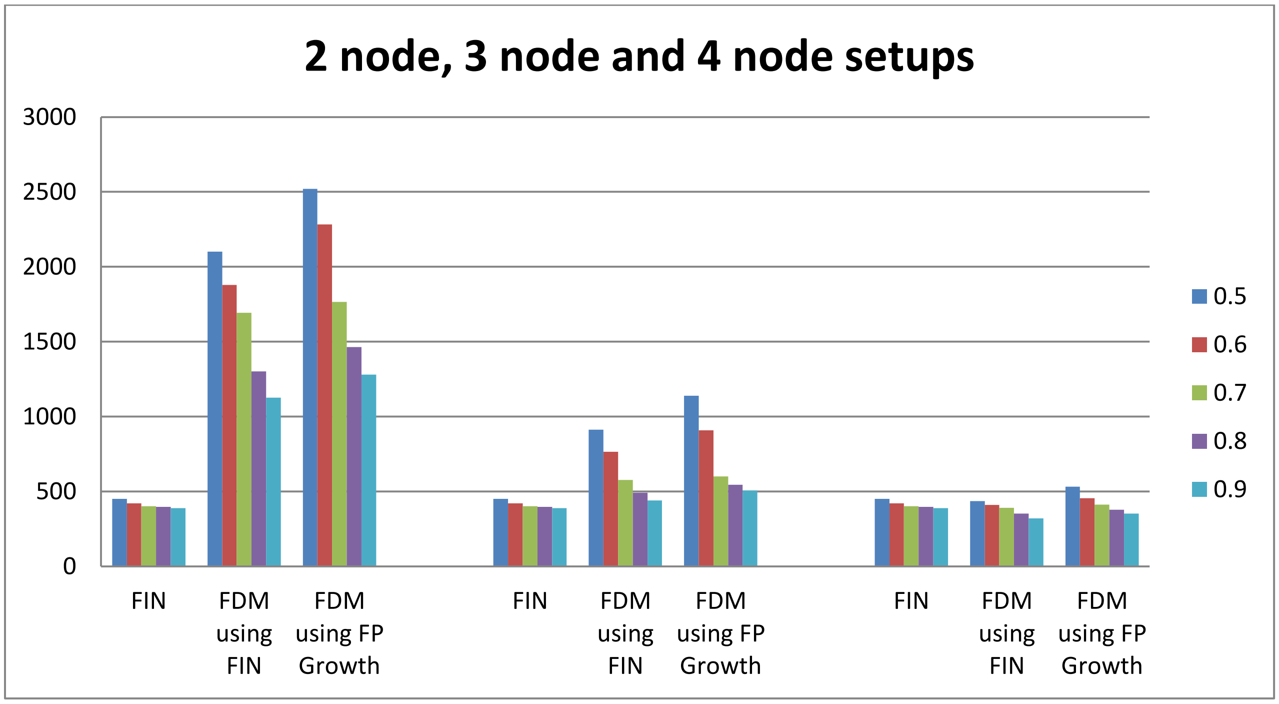

Figure 5.4: Graph for comparing performance with different number of nodes

From the above graph, it can be clearly seen that the performance becomes better on addition of nodes in the setup. Since more is the number of nodes, the less processing is to be done by each node and the faster will be the work done.

Also there is a tradeoff between the actual processing time and the communication time where the packets have to be sent among the nodes. So it can be seen that when only 2 nodes were there, apart from the processing time lots of time was taken for the communication, due to which it could not perform good. As we keep on increasing the nodes in the system, though the communication may increase but processing time per node is decreased and hence the overall performance becomes good.

CONCLUSION AND FUTURE WORK

In this project, parallel processing on distributed data is implemented using a new data structure, nodeset where an algorithm named FIN, using the nodeset structure, is used to generate the candidate itemsets at each site locally. It can be seen that the performance is not good for less number of nodes, but as the number of nodes in the system are increased, there is an improvement in the overall performance in terms of the execution time as compared to its sequential counterpart.

So, this kind of approach can be used in an environment where more number of nodes are available for parallel processing and each node has a part of dataset stored in it. But for small datasets, it does not perform well as the communication overhead will be more than processing time and hence the sequential algorithm performs better in that situation.

As a future work, it can be further improved by using some advanced systems with more memory or in a setup where more nodes are there to decrease the overall execution time. The algorithm can also be compared with other algorithms using different data structure to generate candidate itemsets. Also there can be improvement by using some newer data structures like diffnodesets proposed in [21].

REFERENCES

1. Cheung, David W., et al. "A fast distributed algorithm for mining association rules." Parallel and Distributed Information Systems, 1996., Fourth International Conference on. IEEE, 1996.
2. Deng, Zhi-Hong, and Sheng-Long Lv. "Fast mining frequent itemsets using Nodesets." Expert Systems with Applications 41.10 (2014): 4505-4512.
3. Han, Jiawei, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation." ACM Sigmod Record. Vol. 29. No. 2. ACM, 2000.
4. Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases." ACM SIGMOD Record 22.2 (1993): 207-216.
5. http://en.wikipedia.org/wiki/Data_mining
6. Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules." Proc. 20th int. conf. very large data bases, VLDB. Vol. 1215. 1994.
7. Zaki, Mohammed J. "Scalable algorithms for association mining." Knowledge and Data Engineering, IEEE Transactions on 12.3 (2000): 372-390.
8. Zaki, Mohammed J., and Karam Gouda. "Fast vertical mining using diffsets." Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2003.
9. Deng, ZhiHong, ZhongHui Wang, and JiaJian Jiang. "A new algorithm for fast mining frequent itemsets using N-lists." Science China Information Sciences55.9 (2012): 2008-2030.
10. Deng, Zhi-Hong, and Sheng-Long Lv. "PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via Children–Parent Equivalence pruning." Expert Systems with Applications 42.13 (2015): 5424-5432.
11. Agrawal, Rakesh, and John C. Shafer. "Parallel mining of association rules."IEEE Transactions on Knowledge & Data Engineering 6 (1996): 962-969.
12. Ailing, Wang. "An Improved Distributed Mining Algorithm of Association Rules." Journal of Convergence Information Technology 6.4 (2011): 118-122.
13. Ashrafi, Mafruz Zaman, David Taniar, and Kate Smith. "ODAM: An optimized distributed association rule mining algorithm." IEEE distributed systems online 3 (2004): 1.

14. Vinaya Sawant, Ketan Shah. “A Survey of Distributed Association Rule Mining Algorithms.” Journal of Emerging Trends in Computing and Information Sciences(2014) :391-398.
15. Grahne, Gosta, and Jianfei Zhu. "Fast algorithms for frequent itemset mining using fp-trees." Knowledge and Data Engineering, IEEE Transactions on17.10 (2005): 1347-1362.
16. Pei, Jian, et al. "H-Mine: Fast and space-preserving frequent pattern mining in large databases." IIE transactions 39.6 (2007): 593-605.
17. Cheung, David W., et al. "Efficient mining of association rules in distributed databases." Knowledge and Data Engineering, IEEE Transactions on 8.6 (1996): 911-922.
18. Han, Jiawei, Micheline Kamber, and Jian Pei. Data mining: concepts and techniques. Elsevier, 2011.
19. Zaki, Mohammed J. "Parallel and distributed association mining: A survey."IEEE concurrency 4 (1999): 14-25.
20. Han, Jiawei, et al. "Frequent pattern mining: current status and future directions." Data Mining and Knowledge Discovery 15.1 (2007): 55-86.
21. Deng, Zhi-Hong. "DiffNodesets: An Efficient Structure for Fast Mining Frequent Itemsets." Applied Soft Computing (2016)…..
22. Savasere, Ashok, Edward Robert Omiecinski, and Shamkant B. Navathe. "An efficient algorithm for mining association rules in large databases." (1995)…….
23. Zaki, Mohammed J. "Scalable algorithms for association mining." Knowledge and Data Engineering, IEEE Transactions on 12.3 (2000): 372-390……
24. Bayardo Jr, Roberto J. "Efficiently mining long patterns from databases."ACM Sigmod Record. Vol. 27. No. 2. ACM, 1998…..
25. Vo, Bay, et al. "A hybrid approach for mining frequent itemsets." Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on. IEEE, 2013.
26. Pyun, Gwangbum, Unil Yun, and Keun Ho Ryu. "Efficient frequent pattern mining based on linear prefix tree." Knowledge-Based Systems 55 (2014): 125-139.
27. Liao, Jinggui, Yuelong Zhao, and Saiqin Long. "MRPrePost—A parallel algorithm adapted for mining big data." Electronics, Computer and Applications, 2014 IEEE Workshop on. IEEE, 2014.
28. Woo, Jongwook. "Apriori-Map/Reduce Algorithm." Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.

29. Thakare, Sanket, Sheetal Rathi, and R. R. Sedamkar. "An Improved PrePost Algorithm for Frequent Pattern Mining with Hadoop on Cloud." Procedia Computer Science 79 (2016): 207-214.

30. https://www.fimi.ua.ac.be/data