A DISSERTATION
ON

EFFECT ON COMMUNICATION USING N-LIST STRUCTURE FOR DATA MINING IN DISTRIBUTED DATABASE

SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF
MASTER OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by:
Pallavi Batra
University Roll Number :- 2K14/SWE/12

Under the supervision of
Mr. Manoj Sethi

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

DELHI TECHNOLOGICAL UNIVERSITY

DELHI – 110042, INDIA

June 2016

DECLARATION

I hereby declare that the dissertation titled "**Effect on communication using N-List structure for data mining in distributed database**" which is being submitted to Delhi Technological University, in partial fulfillment of requirements for the award of degree of Master of Technology (Software Engineering) is a bonafide report carried out by me. The material contained in the report has not been submitted to any university or institution for the award of any degree.

Pallavi Batra
University Roll no: 2K14/SWE/12
M.Tech (Computer Science and Engineering)
Department of Computer Science and Engineering
Delhi Technological University
Delhi – 110042

CERTIFICATE

This is to certify that the dissertation titled "**Effect on communication using N-List structure for data mining in distributed database**" submitted by **Pallavi (University Roll No 2K14/SWE/12)** for partial fulfillment of the requirement for the award of degree Master Of Technology (Software Engineering) is a record of the candidate work carried out by her under my supervision

Mr. Manoj Sethi
SUPERVISOR
Assistant Professor
Department of Computer Science and Engineering
Delhi Technological University

ACKNOWLEDGEMENT

First of all I would like to thank the Almighty, who has always guided me to work on the right path of the life. My greatest thanks are to my parents who bestowed ability and strength in me to complete this work.

I owe a profound gratitude to my project guide Manoj Sethi who has been a constant source of inspiration to me throughout the period of this project. It was his competent guidance, constant encouragement and critical evaluation that helped me to develop a new insight into my project. His calm, collected and professionally impeccable style of handling situations not only steered me through every problem, but also helped me to grow as a matured person.

Secondly, I am grateful to Dr. O.P.Verma, HOD, Computer Engineering Department, and Delhi Technological University 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 and friends for constantly encouraging me during the completion of work.

Pallavi
University Roll no: 2K14/SWE/12
M.Tech (Computer Science and Engineering)
Department of Computer Science and Engineering
Delhi Technological University
Delhi – 110042

Date: 29th June, 2016

ABSTRACT

Finding association rules through data mining among different items in a large database distributed over a large number of nodes is one of the challenges in the field of discovery of knowledge. Extraction of frequent patterns in transaction-oriented database is crucial to several data mining tasks such as association rule generation, time series analysis, classification, etc. Most of these mining tasks require multiple passes over the database and if the database size is large, which is usually the case, scalable high performance solutions involving multiple processors are required. When the database is distributed among several different systems with share-nothing memory architecture, the problem of mining data for finding frequent patters can be done using distributed data mining algorithms. One such proposed algorithm is FDM (Fast Distributed Mining) and CD (Count Distribution) which are Apriori based algorithms that generates candidate set on each iteration.

The generation of candidate sets is same as that of Apriori algorithm. Once the candidate sets have been generated, two pruning techniques, local pruning and global pruning, are developed to prune away some infrequent candidate sets at each individual sites. All sites share a common globally frequent itemset with identical support counts, so rules that are generated at different participating sites have identical confidence. This approach focuses on a rule's exactness and correctness.

The main problem with these algorithm is the number of iterations it goes through before generating the final frequent itemsets. Every time it finds the candidate itemset, it communicates them as per the polling site resulting in high communication cost and network bandwidth. We propose a new algorithm which uses the advantage of N-List structure to find out all the candidate itemsets in a one single scan resulting in less communication. We have also proposed a solution to further study the effect on communication by communicating both frequent and infrequent itemsets in a single pass rather than sending request and reply messages for every infrequent itemset.

TABLE OF CONTENT

Declaration i
Certificate ii
Acknowledgement iii
Abstract iv

Contents

1. Introduction 7
1.1. Association Rule Mining 7
1.2. Distributed Association Rule Mining and Approaches 8
1.3. Data Structures used for Data Mining 9
1.4. Motivation 10
1.5. Research Objective 11
1.6. Report Organization 13
2. Literature Review 14
2.1. Data Mining 14
2.2. Classification of Data Mining tasks 16
2.3. Association Rule Mining Algorithms 17
3. Problem Definition 20
4. Related Algorithms 22
4.1. PrePost[3] 22
4.2 FDM with FP-growth 27
5. Proposed Work 29
6. Results 33
6.1 Environment Used 33
1.2 Comparison of Running Time 34
7. Conclusion and Future Work 38
8. References 39

LIST OF FIGURES AND TABLES

Table	Description
Table 1. (a)	A Transactional database
Table 1. (b)	Some Association Rules found
Table 4. (a)	Notation Table for FDM-FP
Table 6. (a)	Configuration of the Node
Table 6. (b)	Summary of database

Figure	Description
Figure 6. (a)	Graph for 2-Node setup with PUMSB database
Figure 6. (b)	Graph for 2-Node setup with PUMSB database
Figure 6. (c)	Graph for different number of nodes
Figure 6. (d)	Graph for comparing performance with different number of node

1. Introduction

1.1.Association Rule Mining

The method of extracting useful and previously unknown or we can say implicit information from data is called Data mining. From over the past two decades there has been a tremendous acceleration or we can say increase in the amount of data being stored in database system as well as the quantity of database applications in business and the scientific domain. The use of relational model for storing data and development accelerated the explosion in the amount of electronically stored data and the development and maturing of data retrieval and manipulation techniques. Very little emphasis was being given to develop software for analyzing the data as more importance was being given to the technology for storing the data fast to keep up with the market demands. But recently the hidden information within these masses is explored by the companies. This tremendous pool of information which was earlier being ignored is of great value to the company. This vast quantity of stored data contains information about a number of characteristics of their business waiting to be explored and used for more effective business decision support. Database Management Systems used to manage these data bundles as of now only allow the user to access information implicitly present in the database system i.e. the data. The data which is there in the database is only a small part of the huge information' available from it. Resided implicitly within this data is information about quantity of aspects of their business waiting to be harnessed and used for more effective business decision support. One of the most important domain of data mining is extracting association rules. We define Association rules as the dependency rules which predict the occurrence of individual item who are dependent on occurrences of other items. It is Simple as well as effective and can help the commercial decision making like the storage layout, appending sale etc. Association rule mining [1] consists of discovering associations between sets of items in transactions. It is one of the most important data mining tasks. It has been integrated in many commercial data mining software and has numerous applications [2].

Association rule mining is primarily defined as finding the association, correlations among the data to be observed [1] and it is also used for pattern searching or finding the patterns which are the most frequent in the whole database being under observation stored in a data warehouse or some other repository of data. It guides in various business deals for making some decisions regarding sales of the product or making decisions for business deals like in big retail stores for their product placement technique, their store design or for the sales of their product or to compete in the market. Association rule mining consist of two main steps:

1. Frequent itemset: The count of items that has a frequency greater than the minimum support count as set by the user are find in the database.
2. Association Rules: The itemsets which have a support count greater than the threshold value are used to generate association rules and a confidence with a minimum value is generated.

Of the two steps that are written above, the dominant one is the first step of finding frequent itemsets in the database. There have been a number of algorithms that are being proposed to calculate the frequent itemsets in an efficient manner. The term Association Rule Mining was first proposed by R.S. Agarwal in 1993 [1]. Massive amount of work has been done in data mining area and the algorithms that have been proposed so far that are formally divided into these three main types: Apriori based, frequent pattern growth based and vertical database format based.

1.2.Distributed Association Rule Mining and Approaches

Structured and unstructured data generated in bulk by a company is termed as **Big Data**. Distributed data mining algorithms are needed to mine frequent itemsets or association rules when data is very large and saved in a distributed environment. Relational database is unable to handle such a voluminous amount of data and it takes too much time to load the data into the relational database for analysis. Volume, Velocity and variety are the three characteristics for a data to be called big data. All are three characteristics are defined as follows:

Volume refer as the amount or the quantity of the data generated for the analysis purpose.

Velocity is defined as the pace with which the data is being generated by the Internet or business world.

Variety refers to the nature and the characteristic type of data.

The ability to extract the useful information from data or to process the data is not as fast as the quantity of data that is being generated being created at a tremendous rate. There is a demand to solve the problem of analyzing such a huge amount of data to meet the current trends. Parallel computing (Single machine with multiple processor or also known as shared memory processors) or Distributed processing (Network consisting of many local computers with shared nothing memory) can be the solutions to tackle such problem with huge data. Mining frequent patterns from such a huge amount of data is one the various applications of big data.

The need of mining frequent patterns and the increase in the size of database demands distributed mining being processed among number of nodes. The data to be mined is stored at various locations and many different processors work in parallel to refine the data and provide a fast and efficient result. Local frequent patters are mined and are communicated to the all the nodes in the system to find the global frequent items. Many centralized as well as distributed algorithms have been proposed to do the first step of association rule mining but these days a lot of emphasis is on distributed mining. Some of the well-known algorithms are AprTidRec, Fast DM, and Optimized DAM etc.

1.3.Data Structures used for Data Mining

We organize the data through data structure in a database. Data structures are useful in a way that help in reduce the complexity of the code and help in minimizing the computational complexity of the implemented algorithms and making it better. A various number of frequent item set mining algorithms have been proposed using different data structures for mining association rules. The different data structures used in data mining are:

N-List: N list is a vertical data structure originated from FP-tree-like prefix tree. Also known as PPC tree. Single path property of N-List is exploited to find frequent item sets without actually finding candidate items in that database.

Trie data structure: in order to store dynamic set where the items are generally in the form of strings we use tree data structure.

FP-tree: FP tree is a tree like data structure used to find frequent itemsets. The benefit of using FP-Tree like data structure is that it does not generate candidate itemsets every time. It is more efficient than Apriori like algorithms.

1.4.Motivation

Last few decades have witnessed a tremendous amount of flow of information in the form of data in this digital world to explore the hidden information and potentially useful patterns from this large quantity of data in petabyte or Exabyte. We are getting data from social media data collected from data sensors, customer retail data, transactional data and many others form of financial data to be harnessed. The type of data is different from different source and the current technologies are not able to handle, store such a huge pool of data at once and get relevant information out of the system. Trillions of data is coming to these large multinational company like Google, Facebook. Today e-commerce have gained a huge popularity resulting in gathering more information about the customers through clicks, cookies, feedback forms, and from their account information. Cleaning, segregating, handling these chunks of data to get insights about the information is done by data developers for the benefits of their companies. It helps to better understand the business and gather knowledge about the market trends and also against how the business is performing against the competitors. Many decisions of the future works are calculated based on this information available from this data. It is not fair to just handle and store this large information without harnessing it. This data is of great value to the company to succeed in the market. Mining is one of the crucial part while extracting information after cleaning the data. It is used to find association or the relation between different set of data which can be relevant from business point of view. Association rule mining or we can say finding association rules among different patterns can be explained as follows. Consider a transactional data of a store retail chain selling daily commodities. Using association rule mining we can find from this data what is the most common product that is bought by the most of the population in that respective area or we can also find which products are bought together most frequently. This result is helpful in the sense that it can give us the combination of products that we should put together to increase our profits and data show that people have more inclination towards buying these products together.

The two main steps involved in mining are as follows:

1. In the first step all the item sets which are frequent of every size is generated using the various association rule mining algorithms available.
2. From these frequent item sets generated in the first step, strong association rules are generated.

Frequent itemset mining is the very first step while finding association rules. After employing a frequent item set algorithm for mining like Apriori or FP-Growth on the database stored in data warehouse, those itemsets which are frequent item sets i.e. having support more than minimum support count are generated. Once these itemsets which are frequent are obtained, we can generate association rules.

The power of distributed systems for very large computation can be used which working with this problem. Some of the already known algorithms in the field of distributed association rule mining are Fast Distributed Mining (FDM) algorithms in which Apriori algorithm has been used to generate local frequent item sets at each computational site at every round which then further computes the global frequent item sets by communicating the local information i.e. the local frequent item sets. A new algorithm has been proposed called PrePost algorithm which is based on a novel data structure called N-List which use both the advantages of Apriori and FP growth and can efficiently mine the data with better computational speed. Using N-List we compact the space as well as the number of iterations the program goes through before outputting the final result. So, in order to improve the performance of FDM, the apriori algorithm has been modified And PrePost has been used instead. We have also worked on improving the communication cost of the overall algorithm by sending the relevant information beforehand to save the bandwidth and overall communication.

1.5. Research Objective

We generally use Distributed system for mining association rules when there is a mass data that is available to us in data warehouse. As the technologies in web and distributed systems are advancing, we are trying towards keeping the database in distributed environment. Mining of association rule by exploring distributed system is gaining a tremendous hype as it can marginally reduce the computation amount. They are highly available, less vulnerable to get fail and are easy to be maintained. Though mining of frequent item sets in distributed environment may require iterative scanning of all the database being fed into each system which sometimes become a costly process. Therefore there is a need of an efficient mining algorithm for transaction or relational database which can generate frequent item sets without much of iterations and this has become a major part of database studies

Problem statement for finding association rules is as follows:

Let I be a set of items called an itemset. Suppose there is a database called DB which contains a set of transactions, where each transaction can be represented as having a particular set of items in some specific order. Given a transaction containing an item set, we can say that the implication of the form $X \Rightarrow Y$, where, $X\ and\ Y$are itemsets in a transaction and is also called association rule.

With a given a minimum threshold minimum confidence ‘c’, the association rule among itemsets holds in DB if the likelihood of a operation $A \implies B$ holds true. Let T be the number of transaction in DB which comprises A also contains B. The association rule $A \Rightarrow B$ has user defined threshold least support "s" in DB if the likelihood of a transaction in DB incorporates both X and Y is "s". The problem of mining association rules deals with finding all the rules where their individual support is larger than the minimum support threshold and confidence "c" is larger than least confidence threshold.The confidence of a rule is defined as conf () = sup () / sup (). For example, Figure 1 shows a transaction database (left) and the association rules found for minsup = 0.5 and minconf = 0.5 (right).

ID	**Transaction**
t_1	{a, b, c, e, f, g}
t_2	{a, b, c, d, e, f}
t_3	{a, b, e, f}
t_4	{b, f, g}

Figure 1. (a) A Transactional Database

ID	**Rules**	**Support**	**Confidence**
r_1	{a}→ {b}	0.75	1
r_2	{a}→ {c, e, f}	0.5	0.6
r_3	{a, b}→ {e, f}	0.75	1
….	….	….	….

Figure 1. (b) Some association rules found

Mining associations is done in two steps [1]. Step 1 is to find all frequent item sets in the database satisfying this given mathematical formula (minsup × |T| transactions) [1, 9]. Step 2 is to use all the frequent item sets produced in the step 1 to conclude all the association rules. For each frequent itemset , pairs of frequent item sets and = – are selected to form rules of the form →. For each such rule →, if sup (→) ≥ and conf (→) ≥ , the rule is output. For an itemset, its support is defined as the number of count in which it appears in a transaction. An itemset is called a frequent occurring item if its support is not less than minimum support value. -item set is an item set containing items. Step 1 mostly determines the total computation speed of the overall association rule mining algorithms and there has been a lot of focus on developing fast and efficient solutions to tackle this sub problem to overall minimize the time it takes to find association rules.

1.6. Report Organization

The thesis is organized in following manner:

Chapter 2 gives an impression of previous work in the field of data mining.

Chapter 3 discusses the related algorithms that have been proposed so far.

Chapter 4 contains the new proposed algorithm on PrePost Distributed Mining

Chapter 5 analyze the result and compare the proposed with other algorithms.

Chapter 7 concludes the thesis work along with future work that can be done.

2. Literature Review

2.1. Data Mining

A lot of studies have been done to examine efficient mining of association rules from many different perspectives. Apriori was developed for rule mining in databases with large number of transactions, which is one of the most influential algorithms. A DHP algorithm is kind of an extension of Apriori which use a hashing technique. The scope of this study has also been extended to efficient mining of sequential pattern, generalized association rules, quantitative association rules, multiple-level association rules, etc. another area of study is maintenance of discovered association rules by incremental updating. Although most of studies are on sequential data mining techniques, parallel or distributed mining algorithms of association rules have also been proposed recently. It is felt that the development of distributed algorithms for efficient mining of association rules has its own unique role, based on the following reasoning. (1) Large amount of data is stored in data warehouses and databases. Substantial processing power is required for mining association rules in such databases, and a possible solution for this is distributed system. Many large databases are distributed in nature. For example, transaction records of thousands of retail department stores will be most probably stored at different sites. This observation inspires us to study better and more efficient distributed algorithms for mining association rules in databases. New light may also be shown upon parallel data mining. Furthermore, a distributed mining algorithm should be used to mine association rules in a single large database by dividing the database among a set of sites and processing all the tasks in a distributed manner. Distributed system offers the scalability, high flexibility, low cost performance ratio, and easy connectivity which makes DS an ideal platform for mining association rules with ease. A lot of Literature on Data Mining consists of ARM algorithms which are either parallel or distributed in nature. However, these algorithms were designed with collective memory parallel execution environments. We can bifurcate the above mentioned algorithms into two groups: parallel ARM and Distributed ARM (DARM) given their structure and implementation.

Parallel ARM algorithms can be characterized as data-parallelism or task-parallelism algorithms. In data-parallelism algorithms the data sets are divided among different nodes while in task-parallelism algorithms each site must access the entire data set but performs the task independently. One of the simplest data-parallelism algorithm is the the Count Distribution (CD) algorithm [2]. CD algorithm uses Apriori algorithm in a parallel environment and it is assumed that data sets are partitioned horizontally among different sites. The main advantage of CD algorithm is that it doesn't exchange data tuples between processors but only exchanges counts. Depending on the items present in its local partition, local candidate itemset is generated by each processor in the initial scan. The global counts are obtained by exchange of local counts with all the other processors. The $O(|C| \cdot n)$ calculates the communication overhead of algorithm at each phase, where $|C|$ is the size of candidate itemsets and n is number of datasets. Data Distribution is one of the task-parallelism-based algorithm that divides the candidate itemsets among all the processors.

(2) Each processor computes the counts of the locally stored subset of the itemsets of candidate for each transactions occurring in the database. Each processor must scan the portions of transactions assigned to its locally stored portion as well as portions of different processors. Therefore, it results in high communication overhead and performs inferior than the CD. Candidate Distribution (CD) divides the candidates during recurring iterations resulting in generation of all the disjoint candidates independently [2]. It selectively replicates the database at the same time so that the processor can create global counts in comparison to other systems.CD performs better than Candidate Distribution. Common Candidate Partitioned Database in a shared-memory architecture follows a data parallel approach. The algorithm then creates database partitions logically into chunks of same size. A disjoint candidate subset is generated by each processor which results in much better computational division. The PEAR algorithm [3] is based mainly on the sequential SEAR algorithm. Apriori and SEAR algorithms are very similar but it uses a prefix tree rather than a hash tree used by SEAR algorithm, hence improving its working performance. Masaru Kitsuregawa, along with his colleagues have proposed four algorithms known as Non Partitioned Apriori, Hash Partitioned Apriori, Simply Partitioned Apriori, and Hash Partitioned Apriori with Extremely Large Itemset Duplication [4]. The candidate itemsets are copied into all processors in NPA, so each processor can work autonomously. The final data are gathered from the coordinator processor and support count is observed. The candidate itemsets created by SPA are distributed among different processors which shares specifications to its local transaction to all processors. HPA differs from SPA by using the hash functions like hash join so that it reduces the broadcasting cost.

The candidate sets generated in HPA-ELD is more compact and the memory is more efficiently used as compared to other algorithms that generate the same candidate sets and it efficiently use the system and cache memory. Demographically partitioned database is used by DARM to determine association rules which satisfy the minimum support criteria. The major issue with distributed mining algorithms is that there is a lot of communication cost involved while transferring of data and the network connection is not that fast in parallel environment. To handle the problem of mining data from distributed database, Fast Distributed Mining (FDM) algorithm was proposed by researchers which helped to tackle the problem of non-availability of data ta one place. In each of these sites, all the itemsets which satisfy the local support count are searched by FDM and all the infrequent itemsets are pruned away. Once the local pruning I done, the frequent itemsets are broadcasted to all the sites and support count of remaining itemset is also requested.

2.2. Classification of Data Mining tasks

So many redundant outcomes are produced during Data Mining which firstly appear to be useful but are essentially not for other example of data or for future forecasts. So there is a great need of implementing proper statistical testing of all the assumptions.

Association rule mining comprises two major steps:

1. Finding frequent itemsets: Itemsets which appear as frequently in the dataset as a pre-defined minimum support count are found by the miner.

2. Using the frequent itemsets to generate strong association rules: The rules satisfying conditions of minimum support and minimum confidence are generated. Finding frequent itemsets is a vital step. Various algorithms had been created to find the recurring items. R. Agrawal introduced the Association mining rule in 1993[1]. A lot of research has been done in this field since then and a lot of new techniques and set of rules have been planned which are primarily categorized in three types: Apriori based, frequent pattern growth based and Vertical database format based.

Data mining is the analyzing step of the KDD process or knowledge Discovery in Database process. It is a somewhat combination of machine learning, database system, artificial intelligence and statistics and therefore we can say that it is the subfield of the computer science which discover patterns and object relations in the database. The core objective of data mining

is to take out the valuable information from the largest database and translate it into an easily understandable form. Data mining also includes data preprocessing, modeling, interestingness metrics, complexity consideration, visualization etc.
Database that is under observation in data warehouse is very vast and it is required to be segregated, cleaned and altered into a more concise database so that we can find the hidden patterns present in the database. To analyze the multivariate data sets, we need preprocessing to remove unwanted and redundant information along with noise and missing data.

2.3. Association Rule Mining Algorithms

The foremost and the most important step in association mining rule is to look for recurrent itemsets along with their count which satisfy the minimum support criteria. Association Rules are then calculated from these frequent itemsets generated in the first step. The significant algorithms, which have led to the emergence of distributed data mining from centralized mining, are as follows:

Major push in popularity of Association rule mining was done by an article published by by Agrawal in 1993 [1] which according to google scholar has been cited over 17000 times. Last 2-3 decades saw evolution of many algorithms for frequent itemset. These can be further partitioned into three types:

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

The anti-monotone property forms the base of the apriori based algorithms [6], called Apriori, which says that any k-size itemset will be frequent if and only if all its (k-1) sized itemsets are frequent. These algorithms generates a candidate set and apply test plan to find the frequent itemsets, which implies that all the candidate itemsets are created first followed by check of their support counts. If the count is greater than or equal to the support threshold the candidate itemset is termed frequent and then it is used to generate the larger candidate sets.

Agrawal also proposed two variations of Apriori, viz. AprioriHybrid [6] andAprioriTID [6]. Apriori is a renowned algorithm in which the database passes through lot of scans and counts the frequency of the candidate itemsets formerly calculated. In the initial scan we achieve the frequency of the entire item in the database and least support items are size one frequent itemsets. Those who allow the least support criteria are labeled frequent itemset, and these are used to compute the candidate itemsets for the subsequent iteration. Apriori TID [6] has a very beneficial feature i.e. after the initial pass, the database is not used for computing the frequency of the items rather it uses alternative data structure to trim the transaction in the database. Apriori Hybrid [6] is an alternative variation that uses Apriori in the initial pass and moves to Apriori TID [6] at the end of the pass when it assumes that the candidate itemset will fit in the memory. Apriori like algorithms performs fine by decreasing the size of candidate sets, however, they are very costly since database has to be traversed several times over and over.

In 2000, another algorithm appeared termed as Eclat [7], discovered by M.J Zaki for quick discovery of association rules in a large database having vertical layout of dataset using the structural properties of itemsets that are frequent. It restructures the lattice search space into small portions or sub-lattice. A lot of algorithms based on Elcat have also been developed using an effective technique of the approach that could detect long frequent itemsets quickly.

Beside these methodologies, another algorithm FP-Growth [3] also found attention in 2000. Instead of generating candidate itemsets the whole database is scanned once to find all the possible frequent itemsets by using a tree like structure which stores all the items along with their count. Later, this tree structure is utilized to mine the frequent itemsets. Its advantage is that it diminishes the search space and discovers frequent itemsets without generating candidate rather it lacks is that the process of constructing and mining process of the FP-Tree is too complex than the alternative approaches.

AprTidRec [17] is based on basic apriori mining algorithm and it uses a record like structure TidRec for each candidate frequent itemset that is generated. There is only one joint step but no pruning step so there is only one scan of database and the time spent on I/O is saved marginally resulting in less communication cost and in this algorithm if support is decreased the communication cost increase.The disadvatages of using AprTidRec is that it requires large memory space when database is large and there is no balance between time and space complexity.

ODAM [18] is also based on Apriori that firstly remove infrequent items from all the transactions and insert each transaction into the main memory if it is not there into the memory. It sends support count of each frequent item to a single site and are stored into a temporary file. This file is further used to generate the global itemsets of various lengths. Advantages of using ODAM involves using single site for communication from all the nodes in the network which result in less communication cost and fewer exchange of messages. But one of the disadvantages of using ODAM is that it is less secure and there are privacy errors also.

Distributed Mining of Association Rules [19] uses an optimization technique to eliminate the duplicate itemsets from the candidate sets and there is no scanning of the partition for calculating the support count. As the number of nodes increases the performance of DMA increases. It requires more storage for keeping the messages exchanged and only those nodes having identical schemas are used.

Distributed Decision Miner[20] also known as DDM is a well-known distributed mining algorithm that works on unskewed data and it verifies if the itemset is large before aggregating its support count from all the sites in the system. It is easily scalable and less communication is required as itemsets are pruned beforehand but it requires more space as compared to other algorithms.

3. Problem Definition

Most of the challenges faced by data miners stem from the fact that data stored in real-world databases was not collected with discovery as the main objective. Storage, retrieval and manipulation of the data were the main objectives of the data being stored in databases. Thus most companies interested in data mining poses data with the following typical characteristics:

- The stored data is large and noisy
- Conventional methods of data analysis are not useful due to the complexity of the data structures and the size of the data
- The data is distributed and heterogeneous due to most of the data being collected over time in legacy systems

Distributed data mining started gaining popularity from centralized mining because of the following factors
Non-trivial and expensive integration of subset of distributed data. Scalability and performance issues associated with data mining.
The advantage of using distributed data mining is that it provides a framework for scalability that helps in splitting this large amount of data into smaller chunks that require less computational power individually.

Now we inspect the excavating of association rules in a distributed environment. Let DB be a database with D transactions. Suppose in a distributed system there are n-sites and the database DB is partitioned over the n sites into are the partitions created by dividing the database DB.
Let be the size of each partition of database for. Let the support counts be   and for the database in and, respectively. The global support count is called, and the local support count of X at site is called. X is globally large for a given minsup (minimum support) if it satisfies; correspondingly, for a site,X is called the locally large if it satisfies . Gobally large itemsets in DB are denoted by L for the above following algorithm and be the k size itemsets in L which are globally large. The important job of a distributed association rule mining algorithm is discovery of the globally large itemsets L.
The distributed association rule mining algorithm that have been proposed generally uses the Apriori algorithm at each iteration to produce the local candidate itemsets at each of the site.It

generally do not use any specific data structure but we can use hash tree to store the itemsets generated after each iteration. There are multiple passes in which candidate sets are generated and at teach passes these candidate sets are passed to all the other systems present in the network. The termination of the algorithm happens when there are no candidates generated or no global large frequent itemsets are found in the current pass. So, it traverse database at each system many times which is a very time consuming for large databases.

Now in PrePost algorithm we have a new data structure called N-List which makes a tree containing both the preorder and postorder traversal values along with each node and then N-Lists for all frequent size-1 and size-2 itemsets are made and also it does not require scan the database multiple times. The N-List structure present in PrePost Algorithm take advantages present in both the horizontal and vertical data mining algorithm. It is more efficient from both Apriori and FP-growth algorithm because instead of generating candidate sets at each iteration it generates a data structure that does not requires iteration for candidate set generation for size-1 itemset and the rest itemsets can be generated from that only.

4. Related Algorithms

4.1.PrePost[3]

DENG ZhiHong⁎, WANG ZhongHui & JIANG JiaJian first introduced the PrePost algorithm in 2012[6]. N-List data structure is the data structure that is used to effectively find the frequent itemsets in data mining. It ease up the process of finding items by using benefits of both vertical and horizontal mining techniques. PrePost is efficient from the rest of the algorithms in three ways. The transaction that have common prefix or the starting items share the same node in N-List which helps in making it compact. The N-Lists generated by all the itemsets are transformed by intersecting them and this process is achieved in an efficient manner of order O (m+n) where cardinalities of the two lists are defined by m and n respectively. The plus point of using PrePost algorithm is that without generating candidate we can directly mine frequent itemsets without generating candidate itemsets like in Apriori by exploiting the single path property of N-List.

Algorithm 1 Construction of PPC-tree

Input $A\ transactional\ databaseDB\ and\ a\ minimum\ support$ £

Output Frequent-1 item sets and a PPC Tree

Method Construct PPC-Tree (DB, £)

```
    1) $[Frequent\ 1 - itemsets\ generation]$
    2) $According\ to\ £, scan\ the\ DB\ to\ find\ F_1, the\ set\ of\ frequent\ 1 -$
       $itemsets\ and\ their\ support.$
    3) $Sort\ the\ F_1\ in\ support\ descending\ order\ as\ L_{1,} which\ is\ the\ list$
       $of\ ordered\ frequent\ items.$
    4) $PPC - tree\ Construction$
    5) $Create\ root\ of\ a\ PPC - tree, T_r, and\ label\ it\ as$ "null".
    6) $for\ each$
           $transaction\ Trans\ in\ DB\ do$
           $Select\ the\ frequent\ items\ in\ Trans\ and\ sort\ out\ them\ according\ to\ the$
           $order\ of\ F_1. Let\ the\ list\ be\ [P|p], where\ p\ is\ the\ first\ element\ and\ P$
                          $is\ the\ remaining\ list.$
           $Call\ insert\_tree([p|P], T_r)$
```

```
7) $end\ for$

8) $[Pre\ Post\ Code\ generation]$
9) $Scan\ PPC - tree\ to\ generate\ the\ pre - order\ and\ the\ post -$
   $order\ of\ each\ node.$
10) $[Function\ insert\ tree([p|P], T_r)$
11) $if\ T_r\ has\ a\ child\ N\ such\ that\ N.items - name = p.item - name\ then$
        1) $increase\ N'scount\ by\ 1$
12) $else$
        1) $T_r\ create\ a\ new\ node\ N, with\ its\ count\ initialiazed\ to\ 1, and\ add\ it\ to\ T_r$
13) $end\ if$
14) $end\ if$
```

Algorithm: N-lists construction

Input: PPC-tree and L_1, the set of frequent 1-itemsets.

Output: NL_1, the set of the N-lists of frequent 1-itemsets.

Procedure N-lists construction (PPC-tree)

```
1. $Create\ NL_1, let\ NL_1[k] be\ the\ N - List\ of\ L_1[k].$
2. $for\ each\ node\ N\ of\ PPC - tree\ accessed\ by\ pre - order\ traversal\ do$
3.  $if(N.item - name = L_1[k].item - name)\textbf{then}$
4.       Insert $(N.pre - order, N.post - order) : N.count\ into\ NL_1[k]$
5.    $end\ if$
6. $end\ for$
```

Algorithm 3: Mining frequent 2-itemsets

Input: PPC-tree and L_1, the set of all frequent 1-itemsets

Output: L_1, the set of all frequent 2-itemsets.

Procedure L_1 Construction (PPC-tree)

```
1.  $Let\ L_1[k] be\ the\ kth\ element\ in\ L_1,\ set\ L_1[k].order\ =\ k.$
2.  $Create\ Temp_2 =\ int[L_1.size()][L_1.size()].$
3.  $for\ each\ node\ N\ of\ PPC\ tree\ accessed\ by\ pre-order\ traversal\ do$
4.    $for\ each\ ancestor\ node\ of\ N, Let\ it\ be\ N_a\ do$
5.        $Temp_2[N.item-name.order][N_a.item-name.order] +=\ N.count.$
6.    $end\ for$
7.  $end\ for$
8.  $\boldsymbol{for}\ each\ element\ Temp_2[i,j]\ in\ Temp_2\ do$
9.    $if\ Temp_2[i,j] =\ \xi\ \times |DB| then$
10.     Insert L1[i] ∪ L1[j] into $L_2$
11.   **end if**
12. **end for**
```

Algorithm 4: NL intersection

Input: $NL_1 = \{(x_{11}, y_{11}): z_{11}, (x_{12}, y_{12}): z_{12}, \ldots, (x_{1m}, y_{1m}): z_{1m}\}\ and\ NL_2 = \{(x_{21}, y_{21}): z_{21}, (x_{22}, y_{22}): z_{22}, \ldots, (x_{2n}, y_{2n}): z_{2n}\}, which\ are\ the\ N-list\ of\ P_1 = i_u i_1 i_2 \cdots i_{k-2}\ and\ P_2 = i_v i_1 i_2 \cdots i_{k-2} (i_u \succ i_v)\ respectively.$

Output: NL3, the N-list $P3 = i_u i_v i_1 i_2 \cdots i_{k-2}$.

Procedure: $NL\ intersection\ (\ NL_1, NL_2\ )(\ PPC - tree$

```
1. $i \leftarrow 1;$
2. $j \leftarrow 1;$
3. $while\ i \le m\ \&\&\ j \le n\ do$
4.     $if\ (x_{1i} < x_{2j})\ then$
5. $if\ (y_{1i} < y_{2j}) then$
6.   $insert\ \{(x_{1i}, y_{1i}): z_{2j}\}\ into\ NL_3;$
7.   $j++;$
```

```
8.  $else$
9.      $i++$;
10. $end\ if$
11.     $else$
12.         $j++$;
13.     $end\ if$
14. $end\ while$
15. $ptr_1 \leftarrow NL_3.first\ element$; $////the$ first $element\ of\ NL_3$
16. $ptr_2 \leftarrow ptr_1.next\ element$; $////the\ next\ element\ of\ ptr_1$
17. $while\ ptr_1\ is\ not\ the\ last\ element\ of\ NL_3\ do$
18.         $if\ ptr_1.pre-code = ptr_2.pre-code\ and\ ptr_1.post-code = ptr_2.post-code\ then$
19.         $ptr_1.count \leftarrow ptr_1.count + ptr_2.count$;
20.         $delete\ ptr_2\ from\ NL_3$;
21.         $ptr_2 \leftarrow ptr_1.next\ element$;
22. $else$
23.         $ptr_1 \leftarrow ptr_2$;
24.         $ptr_2 \leftarrow ptr_1.next\ element$;
25.     end if
26. end while
```

Algorithm 5: Mining frequent k-itemsets

Input: the minimum support ξ, the frequent 1-itemsets L_1 and their N-lists NL_1. Note that frequent 1-itemsets in L_1 are sorted in support descending order**.**

Output: The frequent itemset set *F*.

Method: Call mining *(L_1, NL_1)*

Procedure mining L (L_k, NL_k)

```
1: for $i \leftarrow L_k.size() - 1\ to\ 1$ do
2: $L_{k^i+1} \leftarrow \emptyset$ ;
3: $NL_{k^i+1} \leftarrow \emptyset$
4: for $j \leftarrow i - 1$ to 0 do
5:     Assume $L_k\,[i] = x_1x_2 \cdots x_k\ and\ L_k\,[j] = yx_2 \cdots x_k (y \succ x1 \succ x2 \succ \cdots \succ x_k); y \in L_1, x_s (1 \le s \le k) \in L_1$
6:     $l \leftarrow yx_1x_2 \cdots x_k; L_k[i] \cup L_k[j]$
7:     $l.N-list \leftarrow NL\_intersection(NL_k\,[i], NL_k[j])$;
8:     if $l.count \ge |DB| \times \xi$ then
```

```
9:          $L_{k^i+1} \leftarrow L_{k^i+1} \cup \{l\}$;
10:         $F \leftarrow F \cup \{l\}$;
11:         $NL_{k^i+1} \leftarrow NL_{k^i+1} \cup \{l.N - list\}$;
12:     **end if**
13: **end for**
14:   **if** $L_{k^i+1} = \emptyset$ **then**
15:     **if** $NL_k[i].length() = 1$ **then**
16:       Assume $L_{k^i+1} = \{P_1, \ldots, P_n\}$ $where\ P_i = y_i x_1 x_2 \cdots x_k$
17:       **for** any $p = y_{v1} y_{v2} \cdots y_{vu} x_1 x_2 \cdots x_k (1 \leq v_1 < v_2 < \cdots < v_u \leq n)$ **do**
18:         $p.count \leftarrow NL_k[i].count$;
19:         $F \leftarrow F \cup \{p\}$;
20:       **end for**
21:     **else**
22:       $Call\ mining\ L\ (L_{k+1^i}, NL_{k+1^i})$;
23:     **end if**
24:   **end if**
25: **end for**
```

4.2 FDM with FP-growth

Symbol	Description
D	Total number of transactions
s	Minimum support
L_k	Global itemsets of size k
CA_k	Candidate sets generated from L_k
$X.sup$	Support count of global itemset
D_i	Each partition in DB_i
$GL_{i(k)}$	Global large k-itemset S_i
$CG_{i(k)}$	K size itemset generated from $GL_{i(k-1)}$
$LL_{i(k)}$	k-itemsets in $CG_{i(k)}$ which are locally large
$X.sup_i$	Support count of locally large X at S_i

Table 4. (a) Notation Table for FDM-FP

Pseudocode for FDM-FP: FDM with FP-growth 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)          $CG_{(k)} = \cup_{i=1}^{n} CG_{i(k)}$
                    $= \cup_{i=1}^{n} fp\_growth(GL_{i(k-1)});$
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) $divide\ L_{(k)}\ into\ GL_{i(k)}, (i = 1, \ldots, n)$;
28) $return\ L_{(k)}$.
```

5. Proposed Work

This chapter briefly explains the basic version of the proposed algorithm implemented to support the work. The work is divided into two parts. Section 1 explain the new modified distributed mining algorithm called Pre Post Distributed Mining. Section 2 explains a technique on how to effectively reduce the communication while sending the data in one iteration. Most of the distributed mining algorithms use Apriori algorithm that generate the candidate itemsets after passing the candidate itemsets at each iteration. In order to improve the efficiency of FDM, we added the N-List data structure of PrePost algorithm in FDM instead of using Apriori. There are many advantages of using N-List instead of Apriori as we need only one scan of the database to generate the N-List structure and from that we can generate all the candidate sets.

The main steps in the algorithm are explained below:

1. Firstly, the whole database is scanned to generate the support count of each item and if satisfy the minimum support criteria, they are considered as the size-1 local frequent items and polling site of each item is found and are send to their respective system which are assigned their responsibility using the network.
2. After receiving the local size-1 frequent itemsets from each locally large site, the polling site send request to all the nodes which have not send these items as the count was less than the minimum support for these items at these sites.
3. After receiving the count from each node including the frequent and non-frequent sites, the total count is calculated which determines whether the item is global or not. For an itemset to be considered Global, its global count should be greater than the minimum support count. These itemsets generated are called Global size-1 itemsets and are broadcasted everywhere.
4. Once every site receive the broadcast itemset, it removes the local infrequent itemsets that are not there in the broadcast list.
5. For the second pass, PrePostDM generates the N-list using the global frequent itemset list. From this N-List all the possible candidate sets that can be generated are created.
6. Once the candidate sets are generated, those itemsets that satisfy minimum support are again send to their respective polling site like we did in step 2-3.
7. Once the polling site find the count from non-frequent itemsets, it finally calculates the global itemsets and broadcast the final result

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. (b) Notation table for PrePostDM

We named the algorithm as PrePost Distributed Mining (PrePostDM).It is described below:

$\boldsymbol{Input}$: $DB_i(i = 1,2, \ldots n)$: $the\ database\ partition\ at\ each\ site\ S_i, minimum\ support,$ $total\ transactions\ count, number\ of\ nodes$

$\boldsymbol{Output}$: 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)          $CG_k = Algo\_FIN(DB_i, minimum\ support, transaction\ count)$;
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)                          $if\ polling\_site(X) = S_i then$
                              $insert(X, X.sup_i) into\ LL_{i,j(k)}$;
                          $else$
                              $insert(X, X.sup_i) into\ iLL_{i,j(k)}$;
10) $for\ j = 1\ to\ n,\ do\ send\ LL_{i,j(k)} and\ iLL_{i,j(k)}\ to\ site\ S_j$;
```

```
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) $for\ j = 1\ to\ n,\ do$ {
17)         $receive\ iLL_{i,j(k)}$;
18)         $for\ all\ X \in iLL_{j,i(k)}\ do$ {
19)                 $if\ X \in LP_{i(k)}\ then$
20)                 $X.sup = X.sup_i\left(LL_{j,i(k)}\right) + X.sup_i\left(iLL_{j,i(k)}\right)$ ;
21)         $if\ X.sup \geq s \times D$
            $then$
            $insert\ X\ into\ G_{i(k)}$ ;}
22) $broadcast\ G_{i(k)}$ ;
23) $receive\ G_{j(k)}\ from\ all\ the\ other\ sites\ S_j\ \left(j \neq i\right)$;
24) $L_{(k)} = \bigcup_{i=1}^{n} G_{i(k)}$.
25) $if(k = 1)$
26) $if\ k = 1, remove\_infrequent(DB_i)$;
27) $return\ L_{(k)}$
```

Explanation of the algorithm:

1) Home Site: sets of candidates were created and they were submitted to the respective sites of polling (line 1 to line 10)

When k equals to 1, the site calls get_local_count to input the DB_i one time and save the local tally of size-1 objects and support in a map assembly$T_{i(1)}$. And for the next pass when all local recurrent itemsets of size more than 1 are found, the candidate sets are created using the PrePost [3] algorithm and are then saved in CG. Then the database is scanned and the local value of sum of each itemset in CG is saved in map assembly T_i. The polling site of the locally huge itemsets is found and the itemsets are directed to respective sites.

2) Polling Site: candidate sets were received and polling request was sent (line 11-17)

As the polling site, site S_i gets the candidate itemsets from other sites. It saves the itemsets in $LP_{i(k)}$ and the sites from which the itemsets are received are saved in X.large_sites. Then a polling request is sent to sites not in X.large_sites to collect the residual support of that itemset.

3) Remote site: support count was returned to polling site (line 18)

When a site receives polling request from some other site, it checks the support count of that specific itemset in its map structure T_i and transmit it to the polling site.

4) Polling site: support counts are received and the large itemsets are found (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.

6. Results

An in depth evaluation of the proposed work has been done to compare the PDM (Prepost distributed mining) algorithm with FDM (fast distributed mining) on the various parameters related to the field of mining and distributed mining.

6.1 Environment Used

The above algorithms are implemented on a distributed system. A series of two to five stations , running the windows system, are connected by 5Mb LAN to perform the experiment. The database used in this experiment is taken from http://fimi.cs.helsinki.fi/testdata.html). Three real data and two synthetic data has been used in the experiment. These data has also been used in previous study of frequent itemset mining. PUMSB, Accidents and Retail are the real database. Retail database contains the real market basket data from some retail store. Census data is contained in PUMS data.

In the experiment result, the number of candidate sets found in PrePostDM at each site is between 10 - 25% of that in FDM. The overall message size in PrePostDM is between 10 - 15% of that in FDM. It can be clearly seen from the graphs that the performance gain of PrePost DM over FDM as well as normal sequential algorithm is higher in distributed systems in communication bandwidth is an important performance factor. For example, if the mining is being done on a distributed database over wide area or long haul network.The performance of PrePost DM against sequential PrePost in a large database is also compared.

The configurations of the sites are listed below:

Operating System	Windows 8.1
Random Access Memory	4GB
Hard Disk Drive	500 GB
CPU usage	2.10 GHz
System Information	64 bit
JAVA used	JDK 1.8
Eclipse used	Eclipse MARS

Table 6.1: Configuration of the Node

PUMSB is the dataset on which these algorithms are being run**.** The pumsb dataset is a real dataset and is available in FIMI repository. Pumsb is quite dense, so a large number of frequent itemsets will be mined even for very high values of minimum support various specifications are given below:

Database	Average Length	#Items	#Transactions
PUMSB	74	2113	49046
Accidents	33.8	468	340183
Retail	10.3	16470	88162

Figure 6.2 Summary of database

The results with those in some published papers may differ because of the different experiment platforms, such as software and hardware, may differ marginally in the runtime for the same algorithms. So, it is very fair that we compare these algorithms in the same running environment. The data distributed among different database is different so we are using different range of minimum support to determine the performance in a reasonable time. The total execution time is the difference between the start time and the end time of the whole program.

1.2 Comparison of Running Time

On the basis of execution time, three algorithms are considered to run. These algorithms are PrePost, PrePost DM and FDM using FP Growth. The PrePost algorithm is taken as the reference algorithm for comparison and rest of the two are the new ones created by us. PrePost is a sequential algorithm and rest of the two algorithms are having both distributed data and distributed processing.

The algorithms are run with five minimum supports of 0.98, 0.9, 0.85, 0.8, 0.7, 0.6 and three setups where 2 nodes, 3 nodes and 4 nodes are included and are compared on the basis of execution time. The results are shown and compared below using line graphs.

The X and Y axes in the three figures show the running time and minimum support, respectively. Figure 7(a) represents the running time of the compared algorithms on Pumsb. PrePostDM runs fast in comparison to FDM with FP growth when the support is high but less than sequential prepost when the support is low

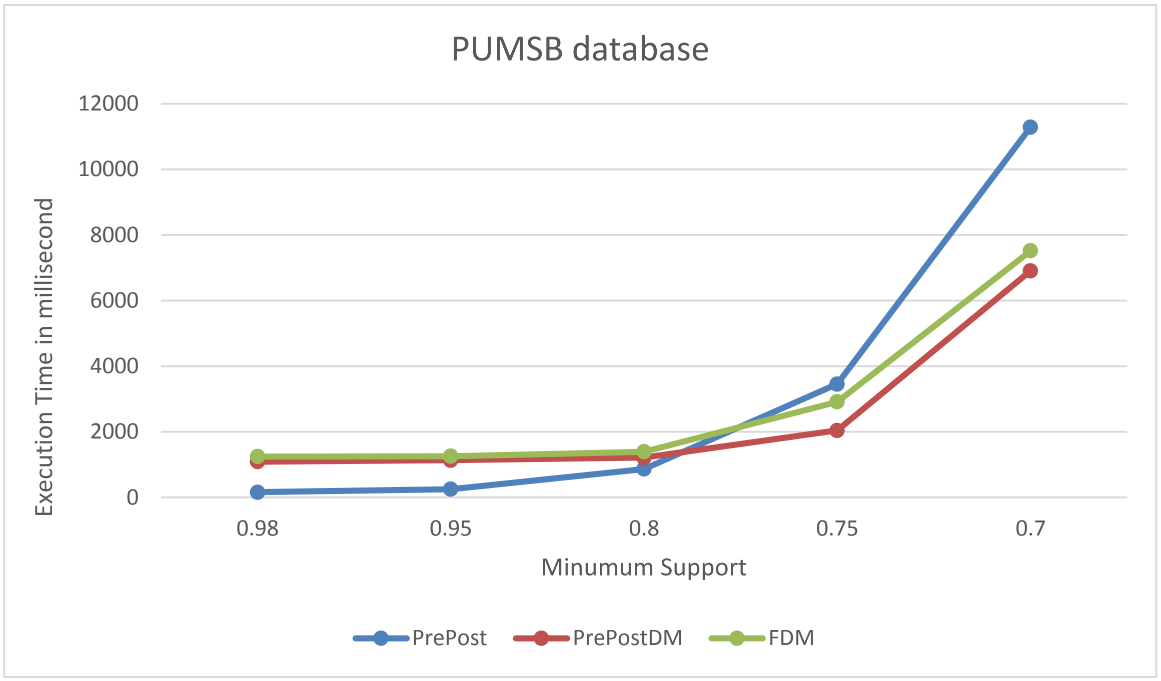

Figure 6.4: Graph for 2-Node setup with PUMSB database

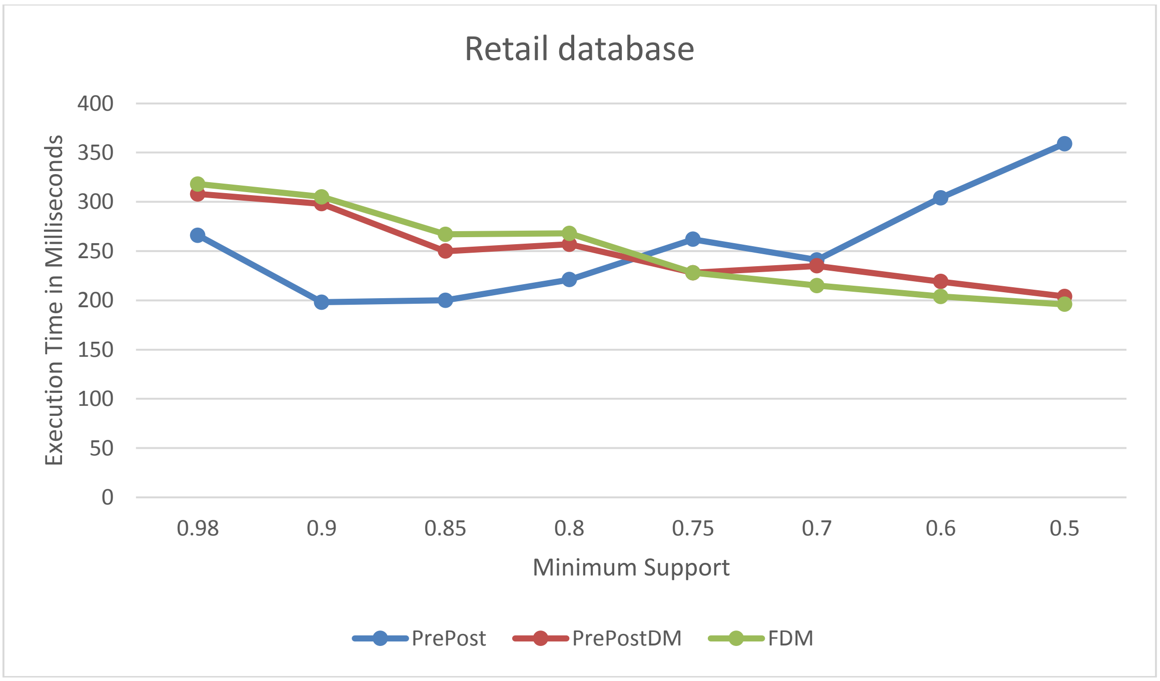

Figure 6.5: Graph for 2-Node setup with PUMSB database

From these above two graphs we can infer that when the minimum support is high, sequential data mining performs better than distributed mining algorithms as the communication time is more than the processing time but when we decrease the value of minimum support i.e. when there are more frequent itemsets generated distributed mining performs better as the task of finding the items is divided between the system and execution time becomes more than the communication time. And if we consider PrePostDM with FDM, PrePostDM execution time is almost equal or better than FDM because PrePostDM implements N-List data structure which requires less number of passes in comparison to FDM which is based on Apriori.

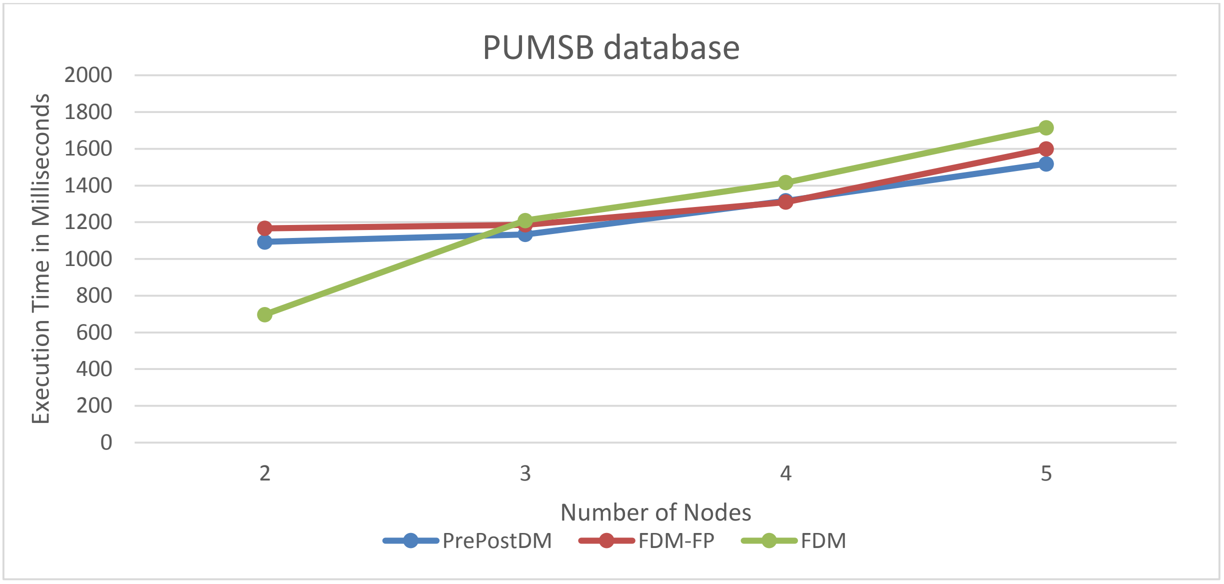

Figure 6.5: Graph for different number of nodes

The above graph shows how all the algorithms PrePostDM, FDM-FP, FDM execute as per the number of nodes in the system. Initially FDM performs better than the rest because there is no data structures being created. PrePostDM is better than FDM-FP because of using N-List structure which can be more efficiently traversed. Both PrePostDM and FDM-FP outperforms FDM because it takes only two scans to complete the whole process while FDM iterates every time it generates the candidate sets.

The above graph for four node setup shows that now our algorithm is performing better than the sequential PrePost 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 6.4. Here the execution time for different number of nodes for different support counts are compared.

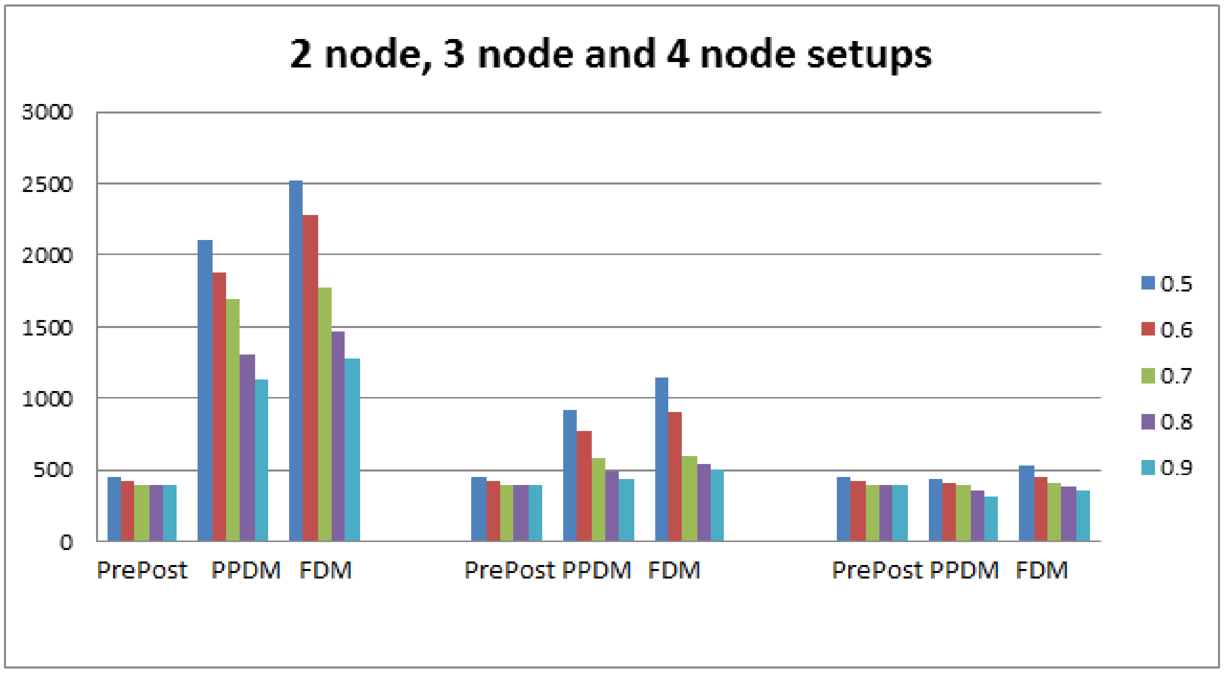

Figure 6.4: Graph for comparing performance with different number of nodes and different minimum support.

It can be clearly seen from the above graph that the performance improves upon addition of nodes in the setup. The higher 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 communication time and the actual processing time where the packets have to be sent among the nodes. So it can be seen that when only 2 nodes were there, lots of time was taken for the communication apart from the processing time, due to which it could not perform good. As we keep on increasing the nodes in the system, there might be an increase in the communication but the processing time per node is decreased and hence the overall performance becomes better.

7. Conclusion and Future Work

This research work focused mainly on the study of various approaches being used to find the Frequent itemset in the field of data mining and the various data structures to make the process more efficient. We propose a new algorithm which is based on PrePost algorithm using N-List data structure to determine frequent itemsets in a more effective manner in terms of communication and execution time. The evaluation results are better than the traditional distributed mining algorithms. We saw that the performance is improving as compared to sequential algorithms as we are using distributed system to find the desired result. There are two major changes that effected the running time of the proposed algorithm

1. Using new data structure called N-List instead of using either Apriori or FP-tree
2. Reduction in communication cost by sending the infrequent itemsets along with the frequent itemsets to their respective polling site.

There is a marginal improvement in the overall performance in terms of the overall execution time as compared to its sequential counterpart while having less communication cost. Several issues related to the extension of the proposed algorithm can be discussed. The technique of candidate set reduction and global pruning of infrequent itemsets can be integrated with PrePostDM to perform mining in a parallel environment which will be better than other distributed mining algorithms when considering both message passing and synchronization of all the nodes in the system. It can be further improved by using some advanced systems with better configurations to decrease the overall execution time. Also there can be improvement by using some newer data structures like N-List proposed. The data distribution technique that we have used for deciding polling site can also be further improved. Study of performance of PrePostDM using the skewness of data distribution and the relaxation of support thresholds is also discussed. Future study include comparing of this algorithm with other distributed mining algorithm containing different data structure. Recently, there have been interesting studies on the mining of generalized association rules, multiple level association rules, quantitative association rules etc. Extension of our method to the mining of these kinds of rules in a distributed or parallel system are interesting issues for future research. Also, parallel and distributed data mining of other kinds of rules, such as characteristic rules, classification rules, clustering etc. is an important direction for future studies.

8. References

[1] Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases." ACM SIGMOD Record 22.2 (1993).

[2] 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).

[3] Deng, ZhiHong, ZhongHui Wang, and JiaJian Jiang. "A new algorithm for fast mining frequent itemsets using N-lists." Science China Information Sciences55.9 (2012).

[4] Manoj Sethi, Rajni Jindal "Distributed Data Association Rule Mining: Tools and Techniques". Proceedings of the 10th INDIACom; INDIACom-2016 3rd 2016 International Conference on "Computing for Sustainable Global Development", 16th – 18th March (2016).

[5] 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).

[6] Han, Jiawei, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation." ACM Sigmod Record. Vol. 29. No. 2. ACM, (2000).

[7] Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases." ACM SIGMOD Record 22.2 (1993).

[8] Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules." Proc. 20th int. conf. very large data bases, VLDB. Vol. 1215. (1994).

[9] Zaki, Mohammed J. "Scalable algorithms for association mining." Knowledge and Data Engineering, IEEE Transactions on 12.3 (2000).

[10] 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).

[11] Deng, ZhiHong, ZhongHui Wang, and JiaJian Jiang. "A new algorithm for fast mining frequent itemsets using N-lists." Science China Information Sciences55.9 (2012).

[12] 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).

[13] Agrawal, Rakesh, and John C. Shafer. "Parallel mining of association rules."IEEE Transactions on Knowledge & Data Engineering 6 (1996).

[14] Ailing, Wang. "An Improved Distributed Mining Algorithm of Association Rules." Journal of Convergence Information Technology 6.4 (2011).

[15] Ashrafi, Mafruz Zaman, David Taniar, and Kate Smith. "ODAM: An optimized distributed association rule mining algorithm." IEEE distributed systems online 3 (2004).

[16] Vinaya Sawant, Ketan Shah. “A Survey of Distributed Association Rule Mining Algorithms.” Journal of Emerging Trends in Computing and Information Sciences (2014).

[17] Wang Ailing (2011), "An improved Distributed Mining Algorithm of Association Rules", Journal of Convergence Information Technology, Vol. 6, No. 4, 2011.

[18] Mafruz Z. Ashrafi, Taniar, D. and Smith, K (2004), “ODAM: An Optimized Distributed Association Rule Mining Algorithm” Distributed Systems Online, IEEE, Vol.5, 2004.

[19] David W. Cheung, Vincent T. Ng, Ada W. Fu and Yongjian Fu(1996a), “Efficient Mining Of Association Rules In Distributed Databases”, IEEE Transactions On Knowledge And Data Engineering, Vol. 8, No, 6, Digital Object Identifier: 10.1109/69.553158, December 1996.

[20] Assaf Schuster and Ran Wolff (2001), "Communication-Efficient Distributed Mining of Association Rules", Proceedings of the 2001 ACM SIGMOD international conference on Management of data (SIGMOD '01) 2001.

[21] VO, Bay, et al. "A hybrid approach for mining frequent itemsets." Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on. IEEE, 2013.

[22] Pyun, Gwangbum, Unil Yun, and Keun Ho Ryu. "Efficient frequent pattern mining based on linear prefix tree." Knowledge-Based Systems 55 (2014).