Implementation and Analysis of Improved Apriori Algorithm In Data Mining

M N Mlambo

orcid.org/0000-0002-2311-9309

Dissertation submitted in fulfilment of the requirements for the degree *Master of Science in Computer Science* at the North-West University

Supervisor: Prof N Gasela

Graduation May 2018

Student number: 25215302

NWU®
NORTH-WEST UNIVERSITY
NOORDWES-UNIVERSITEIT
UNIBESITI YA BOKONE-BOPHIRIMA

DECLARATION

I, MERCY NYASHA MLAMBO, hereby declare that this project report titled *"Implementation and Analysis of Improved Apriori Algorithm in Data Mining"* is my own work carried out at North West University, Mafikeng Campus and has not been submitted in any form for the award of a degree to any other university or institution of tertiary education or published earlier. All the material used as source of information has been duly acknowledged in the text.

Signature: _____ Date: _____

Student: **Mercy Nyasha Mlambo**

APPROVAL

Signature: _____ Date: _____

Supervisor: **Prof N Gasela**

Department of Computer Science

North West University

Mafikeng Campus

South Africa

Signature: _____ Date: _____

Co-supervisor: **Prof MB Esiefarienrhe**

Department of Computer Science

North West University

Mafikeng Campus

South Africa.

DEDICATION

I dedicate my dissertation work to God Almighty for provision, guidance and strength. A special feeling of gratitude to my loving husband Simbarashe Mlambo who has supported me throughout my study. I also dedicate this dissertation to my handsome and wonderful boys, Victor Modern and Daniel Russel who were always by my side as motivation and encouragement.

ACKNOWLEDGEMENTS

I would like to give my gratitude to the most high, God who is my tower of refuge and for making it all possible. My heartfelt gratitude goes to my special husband for his patience and unwavering support. A special thanks to Prof. Naison Gasela for his many days of reading, motivating and advice during my study. I will not forget Prof. Michael Esiefarienrhe for his guidance throughout the research period.

My sincere gratitude goes to the research committee members who were more than generous with their expertise.

I also acknowledge and express my gratitude to Mathematical and Physical sciences school for the assistance and for providing academic assistance requested. Extraordinary acknowledgments to the staff development team and human resources department for their continued support.

Finally, I recognize the lecturers in computer science department who contributed to the dissertation. The department's interest and readiness to provide research opinion made the completion of the research a pleasant experience.

Lastly, would like to extend my appreciation to North West University (NWU) for allowing me to further my studies. Special thanks to everyone who contributed to this study.

ABSTRACT

The production and growing power of computer and network technology has considerably amplified the collection, storage, and manipulation ability of data. The exponential growth in data has produced a commanding need for new techniques and more efficient algorithms to extract useful patterns from such massive data. Data mining emerged in the early 1980's as the technique for mining large datasets. Data mining employs tools and techniques that can automatically unearth valuable hidden facts from large databases over different locations. The Apriori algorithm is an association rule learning based technique that identifies the most frequent itemsets from massive data and extends them to bigger itemsets as the items appear in the database. There have been several popular implementations of Apriori algorithm in sequential computing but big data analysis over distributed systems or cloud systems requires parallel and distributed algorithms. Computational and memory costs for the classic Apriori algorithm are very high for massive data. In serial processing, single processors (CPU) are used and the storage resources are limited resulting in poor performance. In this research we design and implement a fast, scalable and efficient Apriori algorithm for association rules learning in data mining. The research involves analyzing modifications and improvements on Apriori algorithm, founded by Agarwal et al. in 1994, and then develops an enhanced version. To improve the Apriori algorithm, we employ a parallel and distributed computing platform known as MapReduce. The improved Apriori extracts frequent itemsets to produce association rules. The Apache Hadoop MapReduce framework provides multi-processing for huge datasets as key-value pairs in parallel. The implementation of the enhanced Apriori algorithm proved to have better efficiency and scalability compared to past sequential improvements on the Apriori as shown by the experimental results.

Keywords: Association rule mining, Data mining, Distributed, Frequent Itemset, Hadoop MapReduce, YARN, Candidate itemsets

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

API	Application Program Interface
C_k	C for Candidate and k for size/length or the number of items the expression
CLT	Central Limit Theorem
CUDA	Compute Unified Device Architecture
GPU	Graphical processing unit
HDFS	Hadoop Distributed File System
IoT	Internet of Things
KDD	Knowledge Discovery in Databases
L_k	Frequent item set of size k
MR	MapReduce
NUMA	Non- Uniform Memory Access
PALM	Pre-processed Apriori for Logical Matching
SOT	Size Of Transaction
SPMF	Sequential Pattern Mining Framework
YARN	Yet Another Resource Negotiator

GLOSSARY OF TERMS

Association rule:	Refers to statements that find the relationship between data in any database.
Antecedent:	Refers to the term used for that item that existed before or logically precedes another found in the database.
Apriori algorithm	Refers to an algorithm for frequent item set mining and association rule learning over transactional databases.
Consequent	Refers to the term used for the item that is the second part of a conditional proposition, whose truth is stated to be implied by that of the ant the second part of a conditional proposition, whose truth is stated to be implied by that of the antecedent.
Confidence	Refers to a measure- of the reliability of the inference made by the rule.
Candidate generation	Refers to the operation that generates new candidate k-itemsets based on the frequent (k-1)-itemsets found in previous iteration.
Candidate pruning	Refers to the operation that eliminates some of the candidate k-itemsets using the support count pruning.
Data Science	Refers to the processing and analyzing data to extract meaning
Frequent itemsets	Refers to an itemset whose support is greater than or equal to a minimum support threshold.
Hadoop Cluster	Refers to a special type of computational cluster designed specifically for storing and analysing huge amounts of unstructured data in a distributed computing environment.
Hadoop software library	Refers to an open source framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.
MapReduce	Refers to a programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster
Support count	Refers to Frequency of concurrency of an itemset

CHAPTER 1: INTRODUCTION

Chapter 1 provides the introduction and the background of the research and the areas that has been investigated. Firstly, the concepts of Data extraction, Association rule mining, frequent itemsets extraction and Hadoop MapReduce (MR) platforms are explained. Secondly the background of data analysis and big data analysis are described. Furthermore, the research problem is discussed from which the research questions are derived. The justification of the study is explained with the significance of data mining and the need for new data mining algorithm development. Finally the structure to be followed for the rest of the dissertation is proposed.

1.1 Introduction

Due to explosive data growth collected through transactional systems, the need for new and efficient techniques to discover useful hidden patterns in the data has arisen. Data mining and data science have gained more importance nowadays as a result of the need for efficient and faster techniques [1]. Data mining can be defined as a process of determining unseen and interesting patterns from large amounts of data. Data science is a multidisciplinary blend of data inference, algorithm development, and technology with the aim of solving analytically complex problems. As a result of data mining, an interesting field called data science is fast gaining importance where technology and skills are used to increase awareness, clarity and direction for mined data [2].

Many data mining activities employ a technique for finding frequent itemsets to discover interesting and valuable patterns from big data. The patterns can be discovered using mining techniques which include correlations, association rules, clustering, sequences and classification [3]. Relationships between data can be unearthed by using association rules. An association rule states that the purchase of a certain item/product will result in the customer buying another related product in a retail store. An example is when a customer buys a product X and a product called Y at the same time and they appear on the same receipt. We can say that whenever item X is bought, Y is also purchased. We can express the example above as follows: Buy **{X =>Y}** where X is the *antecedent* and Y is the *consequent* [4]. The rule proposes that a relationship

exists between the sale of X and Y due to the manner in which customers are purchasing the goods. An association rule can extract data by using the frequently used data for marketing, inventory control and customer relationship management in retail stores, bioinformatics in science and financial models and customer relationship management in the banking industry, medical diagnosis, scientific data analysis and content optimisation.

We can measure the strength of association rules by using the support and confidence properties. Support is an expression of how many times a rule can be utilized in a dataset whereas confidence measures the occurrence of Y in the transactions that also have X [5]. The formulas (1) and (2) are used to calculate *support* and *confidence* of item X:

$$\text{Support S } (X=>Y) = \frac{\sigma(X \cup Y)}{N}, \quad \ldots\ldots (1)$$

$$\text{Confidence con } (X=>Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}, \quad \ldots\ldots (2)$$

where N is number of items X and σ is the support count [2].

There are a number of association rule algorithms that utilise support and confidence properties to help them discover interesting and frequent patterns. In our research we studied Apriori algorithm which is an association rule learning approach. The Apriori algorithm has two phases which are explained as follows: The first phase is the mining of frequent itemsets and the second phase is association rule discovery through machine learning. The frequent itemsets are generated by using the Apriori principle which states that: "If an item set is frequent, then all of its subsets must also be frequent", according to Han et al. [3]. Apriori utilises breadth first search to generate the $(k+1)$ candidate itemsets based on k-frequent items. In market basket analysis, the classic algorithm is used to determine association rules which reveal general trends in the database that will give insight to decision makers [6].

The Apache® Hadoop© MapReduce™ programming model is scalable and mostly used for multiprocessing data in a parallel and distributed manner. Apache Hadoop MR was adopted for this research mainly due to its ability to schedule and process datasets in parallel. More

importantly MR is a scalable framework, making it best fit for our needs. MR is a programming tool with parallel computing capabilities that can handle high volumes of data over the cloud and even on standalone computers. MR utilizes a map and reduce technique. According to Zhen Liu et al. [7] this simplifies application development in cloud environments by providing a parallel architectural design thus it can be employed to obtain a better performance for Apriori algorithm. Additionally, Apache Hadoop provides a fault-tolerant Hadoop distributed file system (HDFS) which can store both structured and unstructured files in either a centralized or distributed set up. HDFS offers extraordinary throughput access to application data making it appropriate for large datasets [8].

The greatest benefit of MR is that it has the ability to hide system level details from the system developers by generalization [9]. This means the programmers need not understand how computations are carried out but only the functionalities that they have to program. Furthermore, Hadoop uses a functional and object oriented programming languages called java for programming which allows the developers to inherit most of the inbuilt MR functions, methods and classes. Computations are distributed by the HDFS and MR thus the programmer need not be aware of how data and code are split and processed. The benefits of the MR model allow us to use it as a parallel programming model to implement a parallel Apriori algorithm to discover frequent itemsets in mining large customized retail data sets.

1.2 Problem Statement

The Apriori algorithm pioneered the use of support based pruning in an effort to overcome large and unnecessary candidate itemsets generation using association rule learning techniques [3, 10, 11]. There have been various Apriori algorithm modifications proposed and developed over the years. Some research approaches have been based on candidate itemset generation, testing and pattern growth changes have been proposed [12]. Unfortunately, in candidate itemset generation and test approach, the candidate itemsets generated are infrequent and worse still, a transaction database must be scanned repeatedly in order to eliminate the infrequent itemsets. Furthermore, these modifications have not given a scalable and efficient algorithm for mining voluminous data sets, RAM processing capacity and storage of the machine being the major setback. As a result serial and sequential algorithm improvements have proven to be expensive due to iterative scan of the database and resource requirements [10, 13, 14].

Setting of minimum support count requires someone with experience which may decrease the efficiency and availability of the rule if such a person is not available. In authors proposed parallel formulations of Apriori algorithm where candidate itemsets were partitioned among different processors. Communication overhead and redundant work were the main challenges experienced in these parallel proposed implementations of Apriori. Apriori modifications have been done with the goal of making parallel and distributed algorithm [3, 15]. The proposed parallel implementations also proposed a scalable and distributed framework that is efficient and with improved performance [16, 17]. The Apache Hadoop MR is a communication reduction and is a fault tolerant platform. We propose to improve Apriori algorithm by employing a parallel Apriori approach and then utilizing Apache Hadoop MR platform. We also propose to implement pruning locally on each split to all infrequent itemsets. We then propose to increase the mappers in order to increase the improved parallel Apriori algorithm's performance.

1.3 Research Questions (RQs)

Based on the above stated research problems, this study sought to answer the following questions:

RQ1: How can the Apriori formulation be extended so that it can work well in data mining?

RQ2: How can we best increase the efficiency and scalability of the modified Apriori?

RQ3: How can asymmetric binary Apriori algorithm be best implemented to discover relationships in data mining?

RQ4: How can the new enhanced Apriori algorithm and the traditional Apriori be compared with items of scalability and efficiency?

1.4 Research Aim and Objectives

We present the aim to be achieved in this research in the next section. We also describe the research objectives which addresses the research questions.

1.4.1Research Aim

The aim of this research is to produce a fast, scalable and efficient Apriori algorithm for data mining.

1.4.2Research Objectives

The following objectives are of interest to achieve the aim of this research:

RO1: To design an efficient parallel Apriori algorithm model for data mining.

RO2: To design and implement data mining in Hadoop framework with MR and Java Runtime Environment implementation.

RO3: To implement the improved Apriori algorithm on Hadoop framework with MR that is efficient and effective.

RO4: To extract observations from a retail datasets using improved Apriori and compare the results with other prior Apriori modification.

1.5 Justification of the study

The rapidly increasing phenomenon of unstructured data demonstrates a large increase of available data which needs processing, data mining, and machine learning technologies to manage and extract useful information from the data. Figure 1.1 shows that emerging markets have experienced explosive data growth over time and this is estimated to continue in the near future. Data mining helps businesses to analyze their past years' data and draw different conclusions, ideas, theories, and solutions that are relevant to business today. Data mining has created opportunities for retail stores to keep track of purchase patterns, loyalty programs and what is selling the most. Market basket analysis is used to search for data sales and histories to predict which products are bestselling.

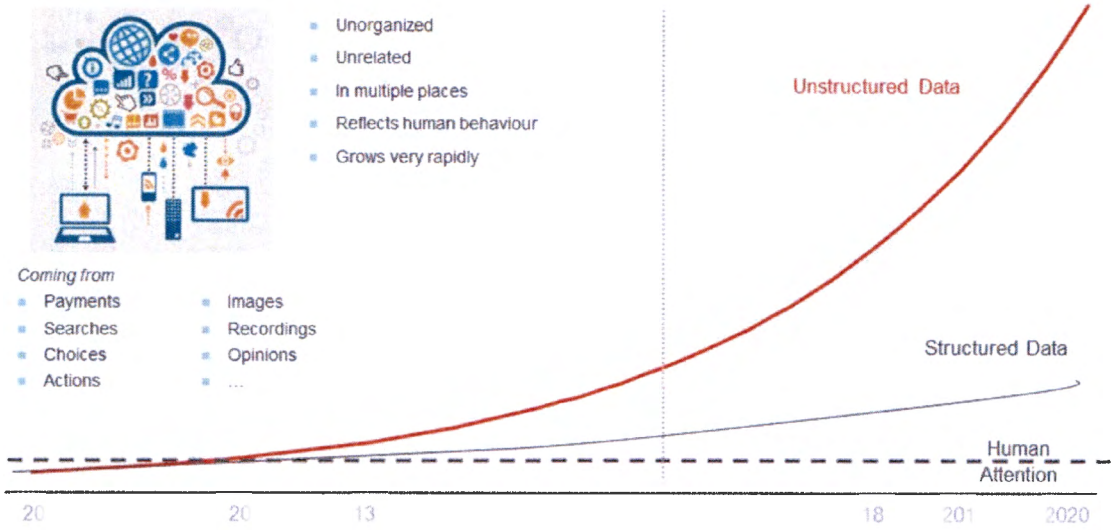

Figure 1.1:Structured and Unstructured Data Growth Projection [18]

Finding hidden relationships between data entities is critical in data mining and data science. These relationship patterns can be extracted using mining associations such as correlations, and classification. Apriori has become the most well-known algorithm for discovering most occurring itemsets from data [3]. The main hindrance is that many database scans are needed in traditional Apriori algorithm in order to generate a huge candidate itemsets. Major research work has been done on frequent itemset mining. However, very little has been done based on retail datasets. In this research we investigate the problem of extracting useful itemsets using customized retail data sets on a parallel Apriori technique based on Hadoop™ Cloudera™. We considered Apriori because it is easy to understand and we can perform a parallel implementation of the algorithm on a scalable MR based framework to address many of the challenges of the Apriori algorithm. As a result, the retail stores may improve their business and increase their sales through implementing this proposed innovation.

1.6 Scope and limitations

The following limitations and assumptions will be taken into consideration:

- We used Apache Hadoop Cloudera framework to build the clusters.

- Experiments were conducted on a multicore processor standalone computer.
- The Eclipse IDE was utilized for functional programming of java.
- We used 2, 4 and 6 processors to conduct the experiments and to measure the effects on the speedup and the execution time of the proposed modification.

1.7 Significance of the Research

The main rationale of the research was to enhance the process of using the Apriori algorithm in frequent itemsets mining in retail systems by improving its efficiency and performance using MR. The investigation aimed at reducing the time spent in determining frequent itemsets and enforcing efficient use resources that is the inputs and output devices. We present a cost effective method to discover frequent itemsets and extract association rules from retail data by identifying unknown valuable relationships that can be manipulated for decision making and prediction. We targeted to improve the Apriori algorithm to fit within MR framework and to benefit from the uncommon scalability attributes of the software to overcome the poor performance of sequential Apriori algorithm.

1.8 Methods of Investigation

Based on the goal of this research, quantitative and qualitative research methods were used as follows:

- Qualitative Research: The framework (software and hardware tools) selection to be used. Finding of data sets in the cloud or internet of things (IoT) and or populating of the data node with data from electronic store.
- Quantitative Research: Simulation of the Apriori algorithm using experiments. Comparative Data analysis of the data outputs from the two algorithms (Apriori algorithm and improved parallel Apriori algorithm) by analyzing the data produced within a time frame.

1.8.1 Literature Review

Literature review includes the knowledge, substantive findings, as well as theoretical and methodological contributions on previous work related to data mining and machine learning and

data science in general. The implementation of the algorithm on different software tools is unearthed and analyzed.

1.8.2 Model Design

Based on the literature review, the Apriori algorithm model will be designed and optimized by parallel implementation using MR for processing and HDFS for storage.

1.8.3 Model and Algorithm Implementation

Based on the designs, improved Apriori algorithm pseudo code is formulated and the parallel architectural design based on MR will be implemented on Hadoop framework with Java Runtime Environment.

1.8.4 Analysis and Evaluation

The designed improved Apriori algorithm is analyzed and evaluated for performance and comparison with other algorithms on Hadoop framework.

1.9 Dissertation Organization

Chapter 1 of this dissertation discusses the introduction of the study, the research problem as well as the research questions for the study. The chapter also demonstrates how the research questions drawn from the research problem are answered through research objectives. Chapter 2 provides detailed review of the work done currently and previously by researchers as basis for current work. Chapter 3 presents the designed model and give implementation details of the algorithm developed. The chapter also demonstrates how the model and algorithms are used to achieve the main goal of this research. Chapter 4 gives a critical discussion of the results obtained in the implementation of the model and proposed algorithms. In conclusion it will give a brief analysis of results. Finally, chapter 5 concludes dissertation and provides possible future work.

CHAPTER 2: LITERATURE REVIEW

In this chapter, we present an overview of approaches related to the main concepts of the study from the literature. The first section presents data mining paradigms as well as the Apriori algorithm for association rule mining and the works conducted in improving the performance of the Apriori as sequential and parallel algorithm.

2.1 Introduction

In data mining there are both mathematical and computing procedures which are combined to achieve the main goal of unearthing valuable relationships from large amounts of data to predict future trends. The mathematical and computing concepts that have been adopted into data mining include methods and processes from artificial intelligence, machine learning, statistics, and database systems. Intelligent mechanisms can be employed to obtain undetected patterns from data warehouses and databases.

In [19] the main goal of mining relationships from datasets in data mining is to transform it into understandable and usable format for analytics and future predictions. Data mining methods can either be based on supervised learning or unsupervised learning. In supervised learning data mining algorithms and activities are inferred from training data in which a set output value is expected whilst in unsupervised learning are unlabeled data set are used to extract hidden features from the data. In this research we will utilize unsupervised learning using association rule learning properties [20].

Data mining offers a connection between transaction and analytical systems as there is an exponential growth of data within enterprises [21]. Software applications have been developed to facilitate analyzing relationships in stored transactional data [22]. Extracting information from data has been made simpler by the use of specific data mining models that can be created based on the kind of relationships obtained from data. Past work in data mining have presented the following possible relationships to exist:

- **Classes**: When data is extracted and grouped into predetermined groups to form classes. For example Pick and Pay (PNP) can use its purchasing options of a PNP shopper card,

bank cards, account or cash to classify its customers. The retail store can then use the information to advertise to each class of customers depending on purchase trends [23].

- **Clusters**: When data items with similar attributes in a dataset are grouped in accordance with logical relationships clusters are formed. This can be illustrated by consumer affinities through the use of consumer preferences [4].
- **Associations**: Associations show the connections within data to be mined. One popular example is the beer-diaper where data collected and reviewed from main database, most fathers were seen to be buying beer and diapers on Thursdays and Fridays [4, 13, 24].
- **Sequential patterns**: As discrete data structures are mined various data behavior and patterns are extracted [14].

Elements of Data mining

According J.C Shafer et al [25], there are activities which are crucial to data mining processes [26]. These activities follow the knowledge discovery in databases (KDD) process in extracting strong rules within the retail data which we used in our research.

- Data is extracted, transformed and loaded as transaction data onto the data warehouse system. The data is then stored and managed in a multidimensional database system.
- Business analysts and information technology professionals are given access to the data.
- The data is then analyzed using appropriate software.
- Data is presented in a useful format in the form of graphs and tables.

Figure 2.1 illustrates the overall data mining process.

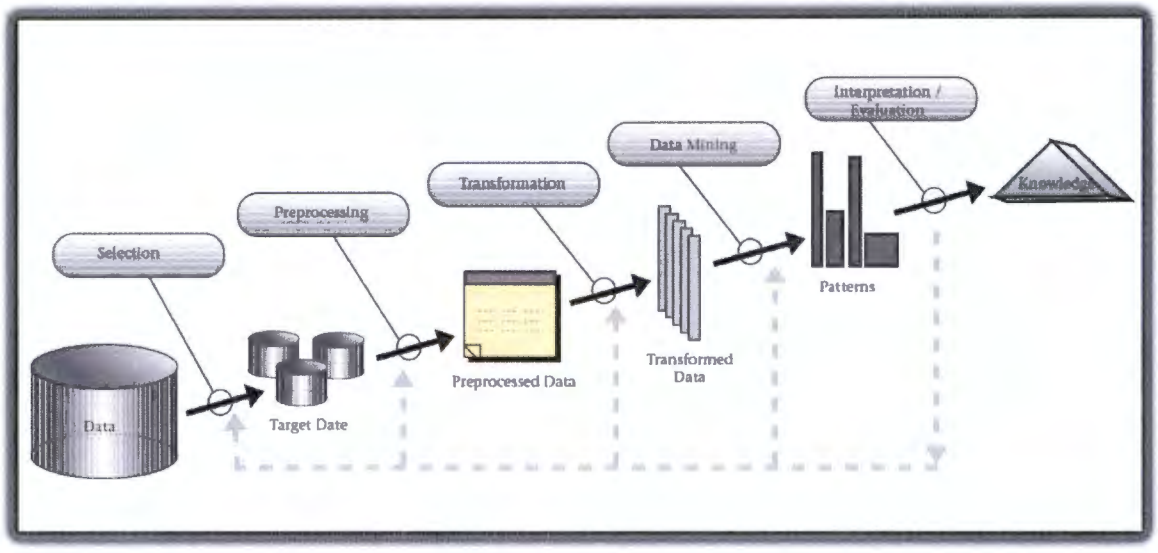

Figure 2.1:Data Mining KDD Process [19]

Selection involves the extraction of useful features from existing data and getting most meaningful dataset [24]. Data integration and cleaning are the preprocessing steps. Raw data is transformed into understandable and usable format. During transformation, data inconsistency and errors are removed and data is also consolidated [27]. Information is mined by extracting specific patterns from the data sets. The mined information is then interpreted and presented using knowledge representation and visualization techniques [26].

2.2 Association Rules Mining

Association rule mining (ARM) is a method for discovering relationships between variables in databases. Researchers discussed how strong interactions can be obtained from databases by using support and confidence measure [5, 27-31].

Researchers define association rules with *if then* statements that help to uncover relationships between unrelated data in databases. An association rule is an implication expression of the form $X \rightarrow Y$, where X and Y are disjoint itemsets, that is , $X \cap Y = \emptyset$. The strength of an association rule can be measured in terms of its support count threshold and confidence. Support determines how often a rule is applicable to a given dataset, while confidence determines how frequently items in Y appear in transactions that contain X.

Support is a measure of the occurrence of items in a transactional dataset. Support can be used to remove unexciting rules and to discover some connections between data entities/rules [3]. Confidence of an item in a given dataset measures the reliability of the implication derived from the rule. For example, the implication $\mathbf{X} \rightarrow \mathbf{Y}$, if the confidence threshold is greater than 50% that means item Y is likely to be present in transactions with X item. Conditional probability of item Y given X can also utilise confidence to determine strong rules [13]. In association rule mining the user must estimate a minimum support which is used to eliminate infrequent items and the user has to also set a minimum confidence [25]. Many algorithms exist for ARM. These may include AIS, SETM, Apriori, AprioriTID, Apriori hybrid and FP growth [4, 6]. In this research we present Apriori algorithm.

Association rules mining has two phases namely:

- Frequent Itemset Generation where all items and subsets whose support counts ≥ minimum support are generated, and
- Rule Generation is a process where frequent itemsets are generated from a user specified confidence threshold.

2.2.1 The Apriori algorithm

The Apriori algorithm has been described as an influential and popular algorithm that helps to extract frequent itemsets by past researchers in ARM [3]. Apriori works by traversing the transactional database looking for connections and relationships between data resulting in frequent itemsets generation [25]. Unfortunately, the algorithm is computationally expensive and its running times can be several days and there is a need to scan databases many times [4, 5]. Most implementations have been based on simple and easy to understand data structures and breadth-first search to traverse through the data.

According to authors in [32] we can divide the algorithm into four sections, as illustrated in Figure 2.1. These sections are initial frequent itemset sets, candidate generation and candidate

pruning and support calculation. In summary the process involves the determination of frequent itemsets by support count calculation and feeding the results into the system. Frequent itemsets are joined to generate candidate itemsets and infrequent candidate itemsets are pruned. Candidate support count is executed every time infrequent itemsets has to be trimmed. The process is repeated several times until the final frequent itemsets are discovered.

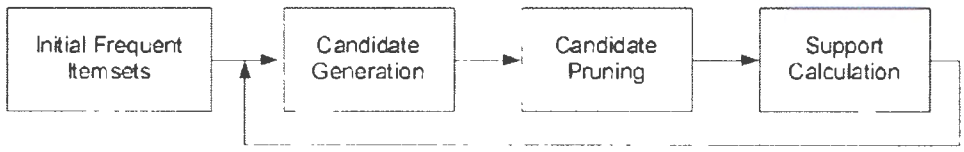

Figure 2.2: Apriori Algorithm Process Flow [33]

Description of Apriori algorithm

Apriori is designed to operate on databases containing similar transactions (for example, collections of items bought by customers, details of a website frequentation, creditors and debtors in banking systems database and DNA research analysis). Apriori uses a breadth first approach, where frequent subsets are extended one item at a time in order to generate candidates, and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found [2].

Below is the Apriori algorithm according to Agarwal and Srikanth [2,39]:

F_1= (Frequent item sets of cardinality 1);

For (k=1; Fk≠φ; k+ ·) do begin

C_{k+1}=Apriori-gen (F_k); New candidates

For all transactions t ε Database do begin

C_t'=subset (C_{k+1}, t); //Candidates contained in t

For all candidate c ε C'_t do

c.count ++;

end;

F_{k+1}={C ε C_{k+1}|c.count≥ minimum count support}

End for

End for

Until $F_k = 0$

Answers= $\cup F_k$

The function Apriori-gen in the algorithm generates C_{k+1} from F_k in two steps below:

- **Joining**: Generate R_{k+1},the initial candidates of the frequent item sets of size k+1 by taking the union of the two frequent item sets of size k,P_k, and Q_k that have the first k-1 elements in common.

  $R_{k+1}=P_k \cup Q_k = \{item_1, item_2, ... itemk_{-1}, item_k, item_k^1\}$

  $P_k = (item_1, item_2, ..., itemk\text{-}1, item_k\}$

  $Q_k = \{item_1, item_2, ..., item_{k-1}, item_k, item_k^1\}$

  Where $item_1 < item_2 < \ldots item_k < item_k^1$

- **Pruning:** Check if all the item sets of size k in R_{k+1} are frequent and generate C_{k+1} by removing those that do not pass this requirement from R_{k+1}.

2.2.1.1 Frequent Itemset Generation

Frequent itemset mining is a computational and memory intensive task and has its application in scientific and statistical areas [4]. The datasets keep on growing into big data sources. As such efficient algorithms are required to process these amounts of data. In general candidate itemset generation involves an iterative process. Frequent itemset generation is computationally expensive. The steps below are involved in frequent itemset generation.

- **Generation of first frequent itemset (1-itemsets):** Involves calculation of support count and updating the frequent itemsets as the itemset size in through the repetitive candidate generation and pruning activities. Given a dataset with an average transaction width of w that means the operation of generating frequent 1-itemsets requires O (NW) time (N is the total number of transactions). As a result the need to employ efficient methods and functions for candidate generation [2, 4].
- **Candidate generation:** Involves the joining of frequent itemsets to form candidate itemset. Assuming d items available then there will be 2^d possible candidate itemsets hence there is a greater need for researchers to come up with efficient methods to generating candidate itemsets [33]. Figure 2.2 illustrates how multiple and unnecessary candidate itemsets are produced using Apriori algorithm during candidate generation.

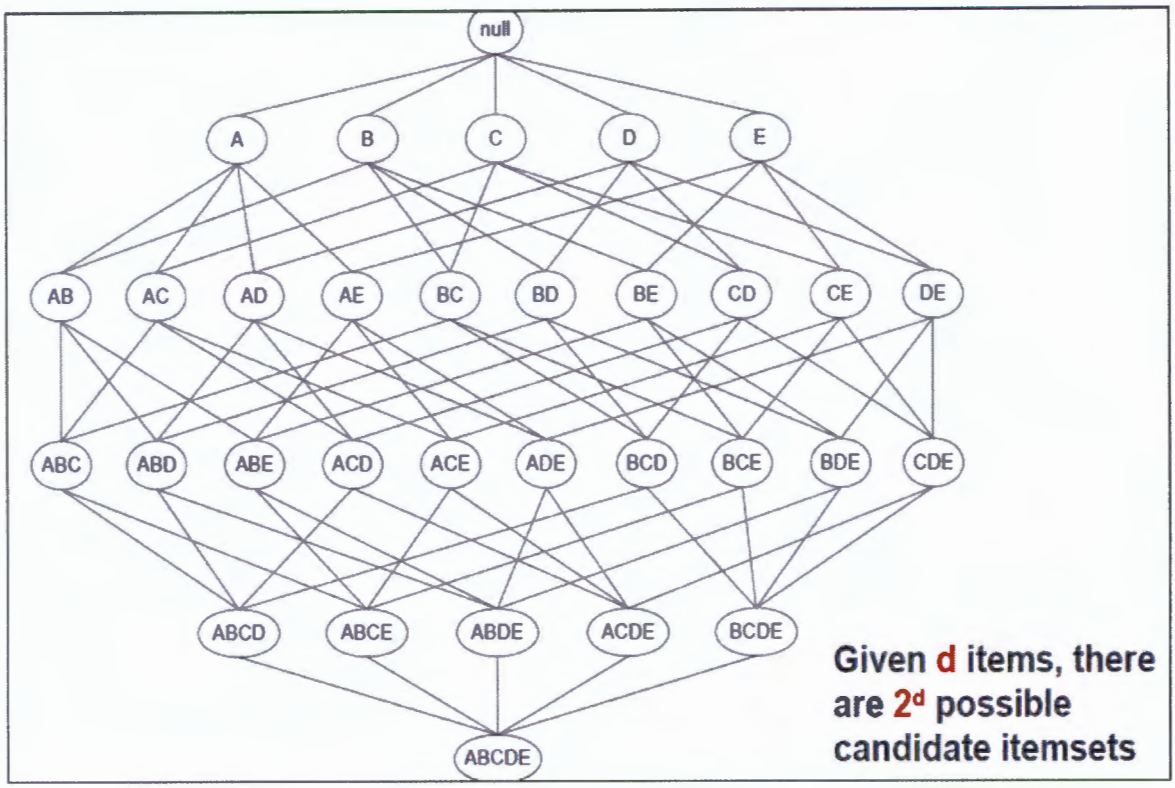

Figure 2.3: Candidate Generation [5]

In the process of generating candidate itemsets *(k-itemsets),* pairs of frequent itemsets *((k-1)-itemsets)* are joined to 1-itemset and their occurrence *I* the transactional database is also verified and counted. The itemset with the highest support is moved into the hash table and then recorded according to its occurrence and support count. Thereafter, all infrequent itemsets are pruned and frequent itemsets undergo joining or candidate generation. As the subset of itemset becomes large enough the Apriori ends by executing the most frequent itemsets. The produced frequent itemsets are useful in predicting future trends recommendation for customers and packing and arranging grocery in the retail shelf. Alternatively, in the worst-case scenario, Apriori algorithm combine every pair of the most occurring *(k-1)-itemsets* discovered in the preceding iteration. The resultant cost amount for the producing the frequent itemsets is expensive to process. Apriori algorithm has more challenges when processing big data sequentially compared to when parallel processing [34]. The formula (3) presented below can be used to estimate the average emerging cost for traditional Apriori.

$\sum_{k=2}^{w}(k-2)|C_k|<$ Cost of merging $<\sum_{k=2}^{w}(k-2)|Fk-1|^2$ ..... (3)

Candidate Pruning is a process where all infrequent itemsets are deleted from the candidate itemsets. This operation is an iterative process that eliminates some unnecessarily generated candidate k-itemsets which are appearing less frequently by using the support-based pruning strategy. Candidate pruning also helps reduce input/output cost due to memory usage during generation of frequent itemsets.

2.2.2Main drawbacks of Apriori

The traditional Apriori algorithm has a clear execution process and can be utilized to discover connections between data. Apriori algorithm is mostly employed in processing big data and when a small size of minimum support count is used the dimension becomes large. Therefore the execution efficiency of the algorithm is decreased. Many researchers have proposed modifications on Apriori algorithm but, the following are still drawbacks and challenges:

- Exponential generation of unnecessary candidate items. In previous studies it was difficult to generate few candidate itemsets or to eliminate the process of candidate generation which generate a huge and infrequent itemset mining. Once candidate itemsets are large, the multiple connections to be derived become huge and complex. Additionally, the matching process becomes time requiring process. As a result we require a mechanism that can help reduce the time spent on candidate generation and matching.
- Multiple database scans. Apriori algorithm scans the database several times to initially find and extract transactions that possess some relationships. Next, the generated candidate itemsets have to be matched every time there is a generation, thus the database or transactions table has to be rescanned. The value of k determines the number of times support counting has to be carried out for example for k= 1, 2, 3 ... , n, database scans are at least n times [35].
- matching candidate itemsets consumes a lot of time [3]

- Support counting takes a lot of time and requires more I/O resources which results in poor performance of the algorithm [35].

Previous studies have come up with two approaches in their attempt to improve and modify the Apriori algorithm namely candidate generation and frequent pattern formation. Researchers in data mining moved from sequential processes towards parallelization of Apriori algorithm activities but there was no fundamental and scalable platform to enhance it [25, 36-38]. In recent years much work has been on parallelization of Apriori based on scalable and multiprocessing frameworks in big data mining, IoT and cloud computing with software platforms like Apache Hadoop.

2.2.3 Applications of Apriori algorithm

The Apriori is a useful algorithm in association rule learning and frequent itemsets. Some of the application areas of the Apriori are as follows:

- Credit card Fraud Detection
- Market basket analysis
- Recommendation on websites
- Drug effectiveness analysis
- Medical diagnosis
- Bioinformatics
- Handwriting digit recognition
- Crime pattern detection
- Web mining
- Scientific data analysis

2.2.4 Evolution of Apriori Algorithm

The original Apriori algorithm scans the database multiple times, using a breadth first search approach, generating candidates and examining every transaction in the database. In an effort to reduce database scans, Aggarwal et al. [4] proposed AprioriTID algorithm where transaction identity is implemented and reference to support counting from the database is only made at the

first count [39]. The AprioriTID was an improvement on Apriori in that there was reduced database scans and because the generated set is used for support counting [12].

Apriori and AprioriTID algorithms exploit candidate generation technique and similar support counting method on generated itemsets. Apriori Hybrid [28], proposed in 2013, uses Apriori properties in its first phase of discovering first frequent itemsets and changes to AprioriTID when candidate item sets are in memory. This results in reduced number of database scans. Frequent pattern (FP) tree algorithm proposed by Han [4] overcomes the problem of candidate generation found in the first two Apriori versions by using pattern fragment growth method to reduce the amount of candidate generation. The FP-Tree algorithm adopts a divide and conquer tactic in obtaining the algorithm activities and routing their pathways using a specialized tree in an effort to effectively use I/O resources. The prefix tree representation helps minimize the memory used for storing the transactions. All nodes referring to the same item are linked together in a list, so that all the transactions containing the same item can easily be found and counted [40]. The main setback of FP-Tree growth is that it becomes computationally costly to traverse through the tree when there are many patterns.

As databases continued to exponentially grow over a short period of time, A Agarwal et al [25] proposed and discussed distributed and parallel algorithms. Unfortunately distributed computing over a large candidate itemset experienced high communication cost. Based on [25, 38] the count distribution and distributed mining parallel algorithms have been proposed in order to minimise storage problems and communication overheads. Additionally, the four parallel algorithms had challenges of avoiding duplication of work and employing the most optimal search method. Algorithm comparison based on distribution and parallelism of algorithms without considering the hardware and load balancing that does not improve the algorithm performance were presented in [25]. As a result parallelism was noted to contribute on the improvement to provide scalability and algorithms reduce execution time.

2.2.5 Apriori Modifications to Reduce Memory Utilization

Researchers have presented different methods to optimize and reduce memory utilization in Apriori algorithm. Singh et al. [34] proposed a novel frequent itemset mining mechanism using probabilistic frequent pattern growth. They employed vertical sorting to maintain order and

support probability distribution function (SPDF) to prune infrequent data patterns. Sangita et al. [41] proposed an improvement to Apriori algorithm by introducing a new attribute on the transaction database in an effort to reduce the scans over the main database. They also proposed parallel candidate generation using multi-core processors to reduce the processing time.

According to Raghuvanshi et al. [46], the best way to improve the Apriori algorithm in pattern mining is by using the set theory concept of intersection with the record filter approach. They used hash table to record different width within transaction thereby achieving memory pointers. Thus the Apriori algorithm achieved an optimized candidate generation phase.

Zhang [43] proposed a solution where a new transaction identifier is generated and two linear tables store items and respective TIDs composed of transaction identifiers. Pratibha et al. [44] presented an improvement on Apriori to remove the drawbacks of generating ample itemset quantities and to reduce the high frequency in querying that required a huge amount of resources in relation to time or space. The researchers reduced the number of database scans. However, there was still an overload on I/O devices.

2.2.6 Modifications of Apriori to Reduce Pruning and Support Counting

The proposed Apriori algorithm improvement by Darshan [78] decreased the deleting procedures of second candidate itemsets, C_2, thereby increasing efficiency of the algorithm slightly. The researcher used candidate pruning to eliminate infrequent candidate k-itemsets by implementing support-based pruning. The researcher utilized the Apriori principle, to remove some of the unnecessarily generated candidate itemsets without counting their support values thereby reducing the processing time [4].

A converse principle to the Apriori is when infrequent itemsets are discarded by removing all subsets of all infrequent itemsets [5]. Figure 2.3 illustrates how the Apriori principle can be employed to prune infrequent data to extract useful patterns.

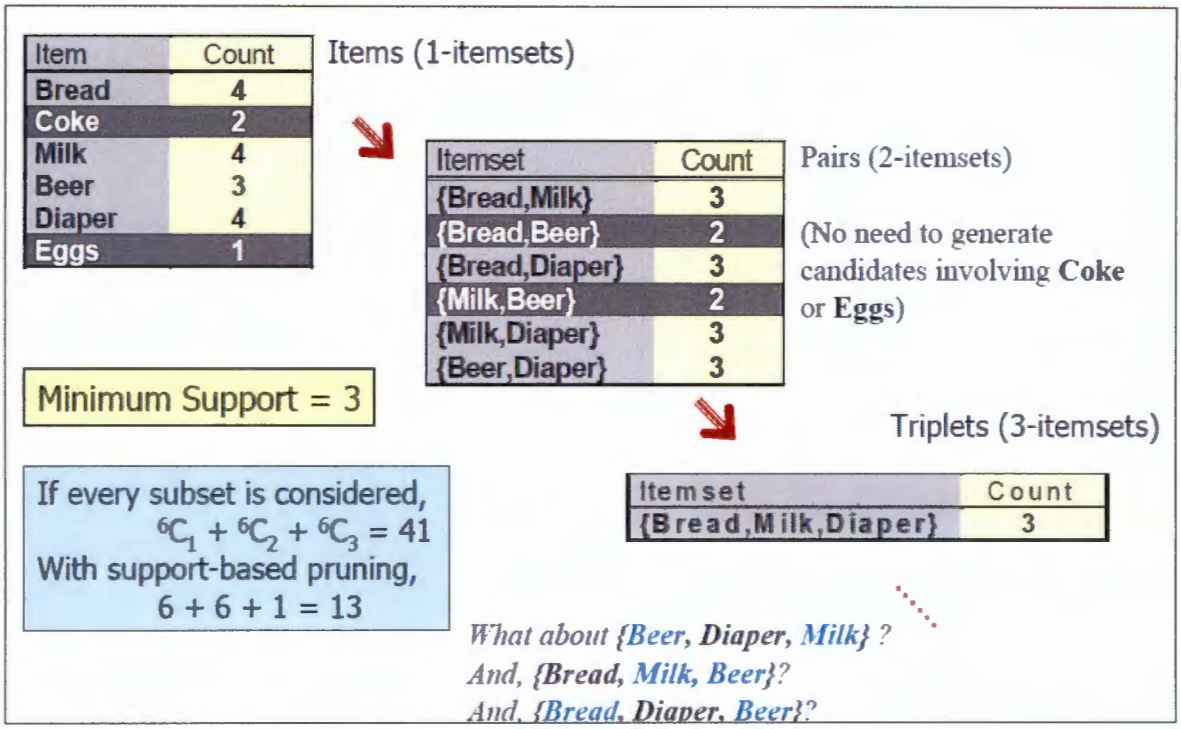

Figure 2.4: Support Pruning Using Apriori Principle [6]

The Apriori principle called the monotone property implements two approaches on the entire algorithm. This Apriori principle works due to the anti-monotone property of the support measure given in [39]. Support of an itemset never exceeds support of its subsets.

$$\forall X, Y: (X \subseteq Y) \Rightarrow s(X) \geq s(Y) \quad (4)$$

Therefore, any dataset that possesses an anti-monotone property can be incorporated directly into the mining frameworks so that the association algorithm can be used effectively to prune the large and unnecessary search space of candidate itemsets. The Apriori is the primary association rule mining algorithm that pioneered the use of support-based pruning to systematically control the large number of the unnecessary generated candidate itemsets [2, 4].

Hardware acceleration is used to propose an improvement on Apriori algorithm since there is high data complexity. Several strategies were presented on implementation of systolic array on data which is processed by an integrated device which can carry out massive computation [37]. Their implementation which used FPGA was successful on candidate generation phase by

reducing the process time. Mohammed et al. [35] proposed an approach to minimise the search time for discovering frequent itemsets in database transactions. Their research focused on the enhancement of the frequent itemset extraction algorithm to decrease time consumption of candidate itemset joining and matching by incorporating the TID in candidate generation. The proposed improvement made it easier and quicker to search for the specific transactions in a database. However, the proposed approach did not overally optimise the algorithm since there was still a need to match the candidate itemsets. In 2012, researchers in [46] improved the Apriori algorithm by making use of a matrix library and implemented it on MATLAB framework. The use of matrices caused a reduction in the computing costs and also improved operational efficiency significantly. On the other hand, Abaya [3] modified the Apriori algorithm by introducing set size and set size frequency which allowed the early deletion on infrequent candidate keys generated thus saving on memory utilisation.

Based on the studies presented above, the drawbacks of sequential approach have been outlined. These setbacks include a lack of scalability and speed of processing and hence the need to overcome the setbacks in our implementation.

2.2.7 Apriori Algorithm Based On Parallel Computing

New developments in parallel computing have also made data mining easier. Sophisticated multi-core processors and embedded systems have been designed for data mining which decrease processing time and increase performance [37]. The execution of a new multi-core architectural design allowed helper threads to process more blocks of data simultaneously [47]. The designed and developed parallel computers have multi-threading and multi-tasking capabilities which can occur in a single step.

Based on work presented in [48] parallel algorithms that can carry multiple tasks concurrently. The algorithms proposed for data mining and embedded systems were to carry out parallel tasks at a lower cost than previous sequential algorithms

The adopted distributed memory structure involves database splitting for each processor that has to be carried out in a logical way so that data can be fairly allocated for processing. Data partitioning must also accompany resource management to enable quick processing of the data. Nilesh et al. [53] presented parallel implementation of the Apriori algorithm on quad core

processors. Abaya [3] proposed an implementation of Apriori on CUDA invented by NVIDIA where a collection of multiple running threads employ the power of graphics processing unit to escalate computer performance. Baddal et al [50], concluded that the implementation of Apriori algorithm on MATLAB improved its performance.

2.3 MapReduce (MR) Based Parallel Algorithms for Large Data Processing

MapReduce (MR) software framework introduced by Google in 2004 is a special tool for processing massive data in a parallel manner. As presented in [48] MR is a software framework that is utilized for processing and generating data sets that are extremely large in parallel. Data mining algorithms from association, clustering and classification paradigms have all been explored on MR for data processing.

All map and reduce operations are fully parallelizable. The map functions are executed simultaneously and independent from other activities. Similarly, reduce operations can take several datasets and allocate them to each reduce function so that they can be executed in parallel [51].

Gowraj et al. [52] described the concept [48] of classification, association and clustering in their proposed algorithm, Preprocessed Apriori for Logical Matching (PALM). The researchers outlined the problem that users are faced with when trying to utilize information found on the internet. They used PALM to do web mining based on MapReduce.

Li et al. [12] presented a solution for parallelizing Apriori algorithm and implementing it on MR by using concurrent activities in its execution. In the paper presented with the aim of improving the ARM algorithm Zheng et al. [53] utilized the MR model to provide a parallel design methodology. They also outlined a simplified platform for application developments. MR possesses the mechanism to minimize the communication costs that are due to the synchronization of tasks between nodes or mappers [54].

MR uses two functions which are a mapper and a reducer. Key-value structured data formats can be processed using the map and reduce methods. MR was used to parallelize Apriori and have the data stored in the HDFS to ensure that all frequent itemsets were mined [53]. White et al. [55] presented Hadoop as a multi structured framework with different functional units for job

execution and file staging. There are also several utilities in Hadoop that allow different formats of files to be stored and used. The datasets are automatically divided into segments by the HDFS and a Map function can be executed on each data segment. MR is a parallel processing tool that accepts inputs in the format of key-value pairs and processes them to produce intermediate components which also are stored as key-value pairs.

The final output that is produced from MR are key and value data structures and these are stored in HDFS.

Input/output	Map function	Reduce function
Input: *key/value pairs*	*Key:Line No.; Value: one row of data*	*Key: candidate item sets;* *Value:1*
Output : *key/value pairs*	*Key: candidate itemsets;* *Value:1*	*Key: frequent subitems;* *Value: support*

Figure 2.5: Map/Reduce Key-Value Pair Structure [53]

The dataset is first loaded into HDFS and the format changed to suit the required key/value structure. As illustrated in Figure 2.4 key/value pairs act as both inputs and output to the simple map and reduce functions. It is therefore crucial to note that the output of map functions are inputs in reduce function. The candidate itemsets are generated and then collected by the mappers whilst the reducer functions sum their support and prune the unnecessary candidate itemsets which possess infrequent items [9]. Consecutive execution of map and reduce functions result in determination of frequent itemsets.

Figure 2.5 shows how data flows in the MR framework from the time it is ingested into HDFS until it is fully processed. There have been several computation iterations for MR operations which also yield the frequent itemset [18].

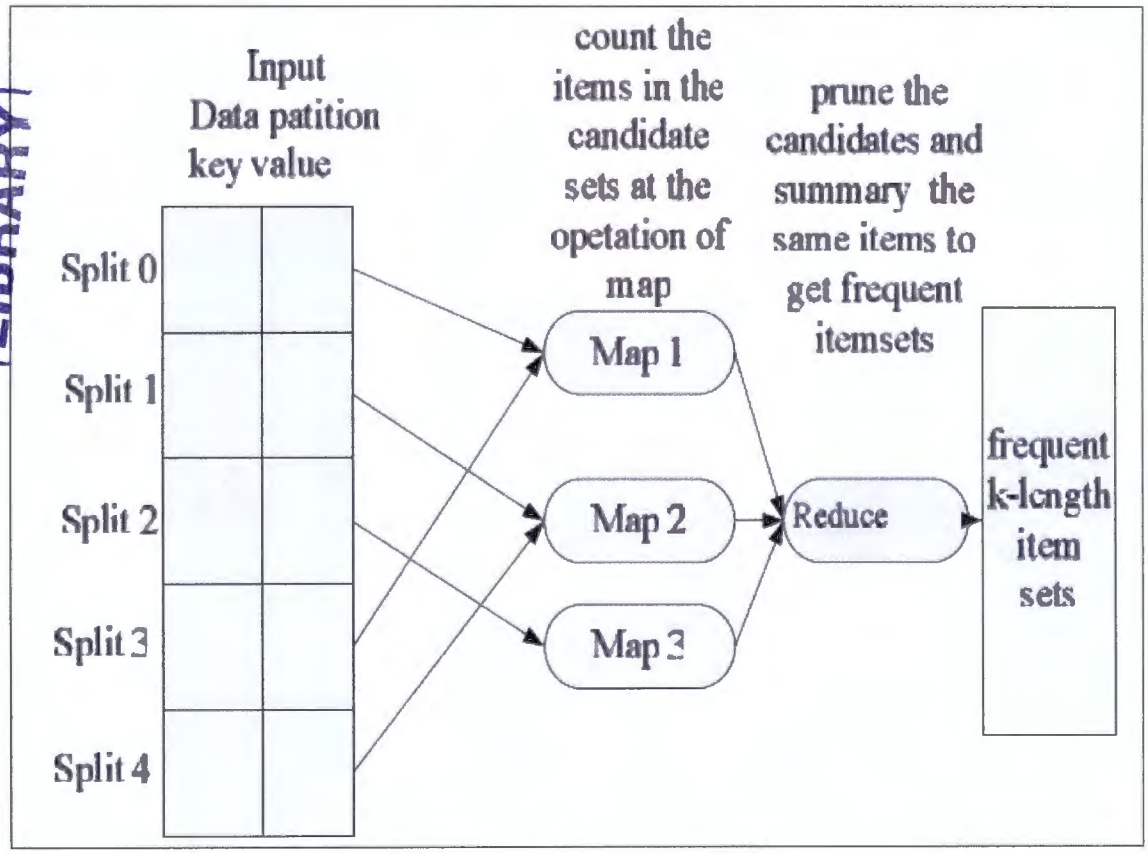

Figure 2.6: Data flow of Apriori in MR Programming Model [53]

Shulei et al. [57] implemented a distributed Apriori algorithm on a MR distributed framework to investigate its effectiveness over centralized Apriori in MR. In their proposed distributed structure there is no communication between slaves to reduce communication problems and communication costs.

Figure 2.6 illustrates the components and data flow of the distributed Apriori in MR using the map and reduce functions. The reduce phase takes input from the map phase and route them to master node to collect count per item.

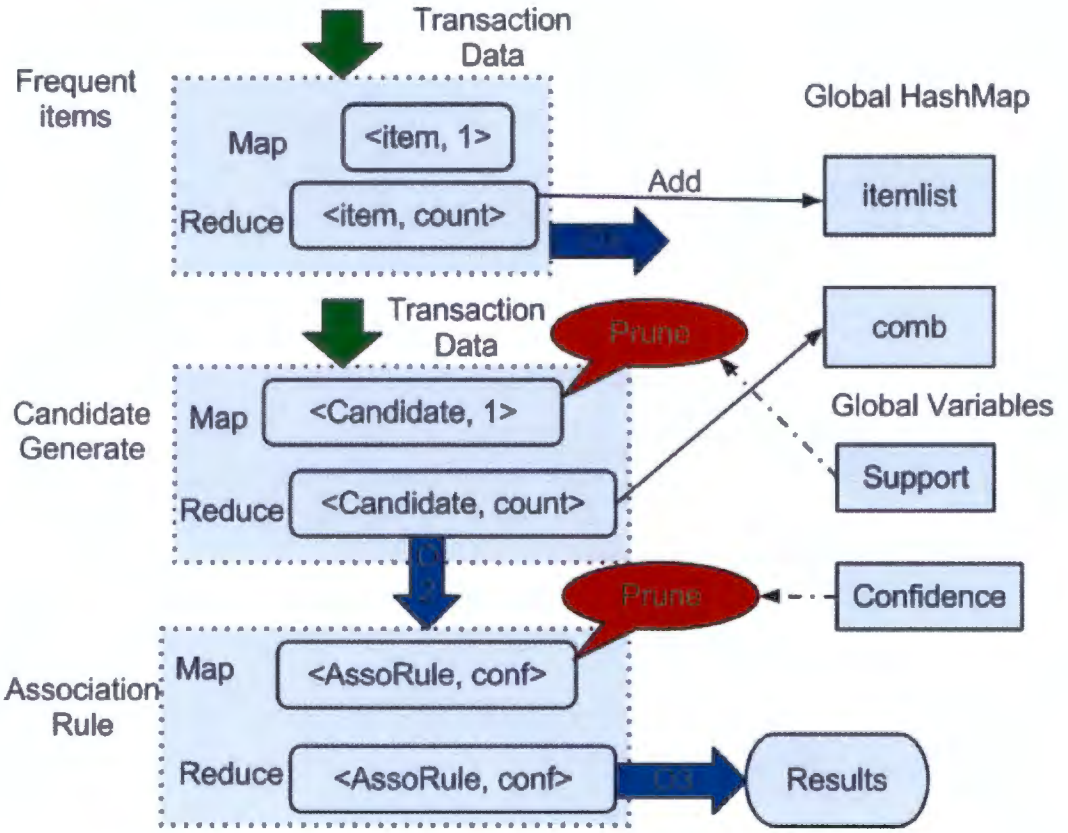

Figure 2.7: Distributed Apriori in MapReduce [57]

Hadoop is an open source platform that can reduce cost by allowing old computers to be upgraded and utilized. Hadoop software framework can process large data sets on clusters of computer nodes in a parallel manner [16]. Anjan et al. [17] presented Apriori algorithm implementation on Hadoop framework by employing distributed HDFS to generate a duplication of several data blocks and allocates them to different mappers. HDFS automatically schedules mapper and reducer activities and stores the data. Hadoop also has components that handle the

problems related to network communication, concurrency control and fault tolerance [7, 58]. HDFS is also an instrument to decrease overhead involved in the scheduling network communication.

Shaosang et al. [59] proposed improvements on the Apriori algorithm to discover frequent itemsets by splitting data into data blocks and allocating them to nodes for process. The Spark Implementation of Apriori in Parallel (SIAP) implementation process is shown in Figure 2.7

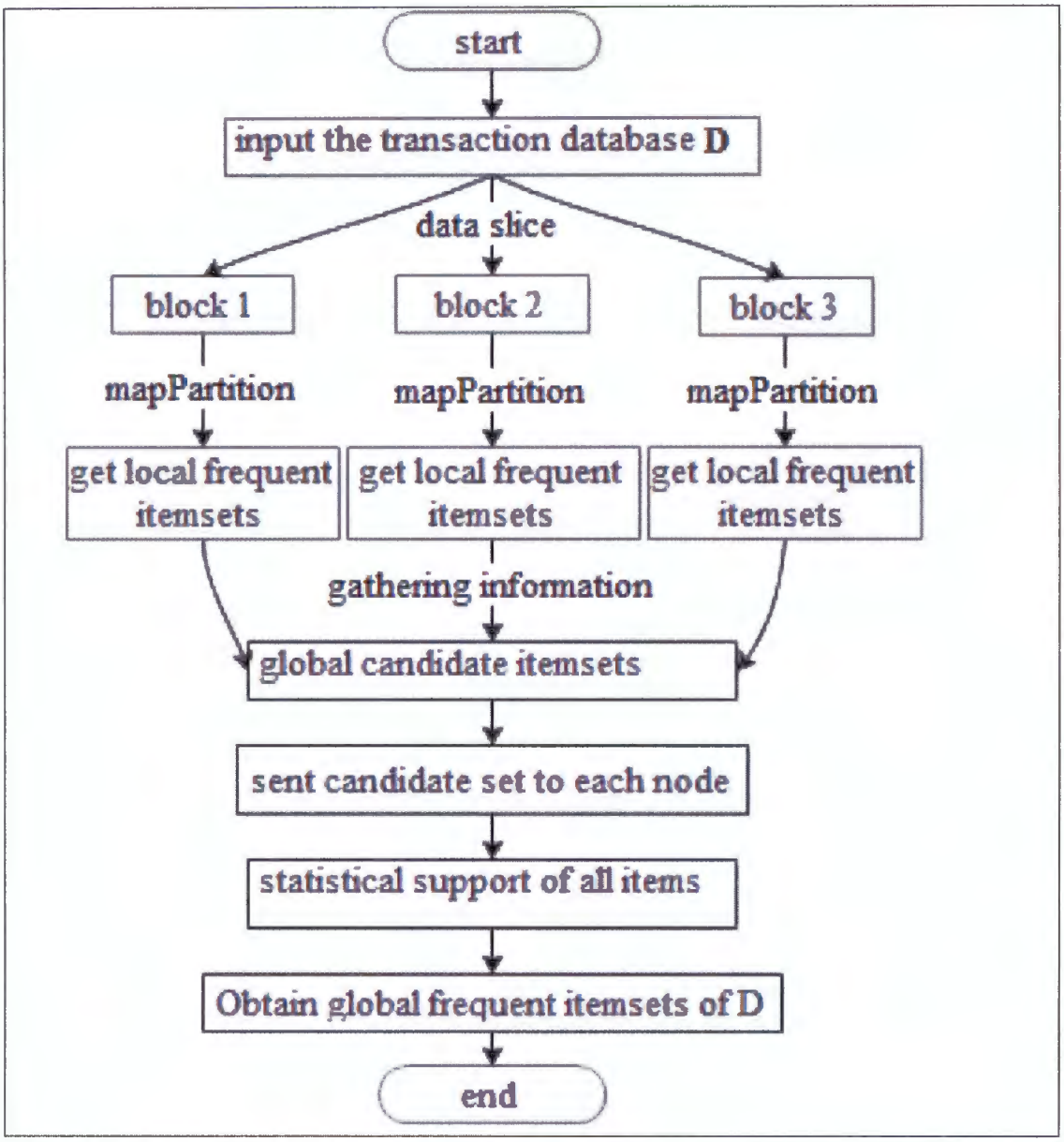

Figure 2.8:The Implementation of IAP Algorithm in Spark Environment [59]

2.4 Improvements of Apriori Based On Customer Habits

In an effort to make the Apriori algorithm more beneficial to users and to improve its efficiency, Researchers in made a few changes to the algorithm by adding new parameters which make it profitable and efficient in selecting a combination of items that will give greater profits to the business [43].

Based on Shuo et al. [60] work in 2012, Apriori was used to generate association rules to recommend useful products. The proposed modification recommends the products to customers as proposed by the researcher. They applied Apriori in E-commerce by developing an online store and used the algorithm to suggest to customers what products they could buy on their next purchase. The proposed algorithm offered decreased access to database thereby increasing its performance.

Dai Li [40], presented an implementation of Apriori algorithm to improve SQL statements that queried relational tables for candidate set support, frequent item sets and their support if they were stored in those relational tables. The use of customer habits does not contribute to the improvement of Apriori but rather helps on algorithm efficiency. The research resolved the problem of ARM only by determining the presence of items in database and treating them equally.

In [61] they proposed an enhancement of Apriori to come up with an aggregate function based on the Central Limit Theorem (CLT) that converts the database into a bitmap. They also presented new functions to calculate the smallest support for each itemset based on the probability of collision [36]. As a result they managed to resolve the problem of setting the minimum support count to obtain frequent itemsets. Setting the support threshold predicts if the number of rules is too large, and would produce only a small number of rules when support count initializes or even no rules to conclude [34]. The threshold value should be decreased to do the mining again, which may or may not give a better result, as by setting the threshold too small, too many results would be produced and would require a long time to compute and screen these rules.

In an effort to find interaction between items purchased by customers in market basket analysis, researchers proposed an approach for frequent itemsets using cluster based master slave structural design [62,72]. They presented a technique that integrates both the bottom up and top-down exploration of Apriori algorithm and adopted candidate partition in their efforts to make Apriori more efficient.

2.5 Data Mining Software Frameworks

There are many software frameworks that have been developed to automatically and quickly analyse data mining algorithms. We present some of data mining platforms that are readily available in this section. Firstly we have the Weka tool funded by the New Zealand Government's Foundation for Research, Science and Technology which applies algorithms directly to a dataset or called from your own Java code [45, 63]. Tools contained in WEKA framework are used for data pre-processing, classification, regression, clustering, association rules, and visualization. It is a platform-independent mining software that possess facilities for scripting based programs but lacks the capabilities for parallel processing [47]. Secondly we have Anaconda tool which is open source platform powered by Python. Anaconda divides workloads into parallel threads to maximise investments in multicore CPUs and GPUs and employs Python compiler and harness modern computer infrastructure where it also configures Hadoop apache within its anaconda module to scale up data extraction and transformation [64]. Thirdly there is MATLAB which is a multi-digital framework which uses fourth-generation programming language. Although MATLAB is purposefully meant for numerical computing it can be used for strings. Fourthly, is SPFM a software platform implemented in Java and it specializes in identifying patterns [65]. Finally, there is Hadoop which is an open source product from Apache that provides many ways for storage and processing of large volumes of data [66]. Hadoop comprises of MR module for parallel data processing and is mostly used to process lots of data in a parallel and distributed manner on a cluster of connected nodes [7, 8, 17, 22, 58, 74].

2.5.1Benefits of Using Hadoop MapReduce

Apache Hadoop was adopted for this research mainly for its ability to schedule and process data in parallel. It is scalable and conducive to managing large volumes of data by utilizing parallel and distributed computing on the cloud. The parallel and distributed architecture is composed of a master node and multiple slave nodes. A mapping function is harnessed on a list of values by the mapper and the output is directed to a reducer which can combine functions of values to output results.

MR can be employed in data extraction to efficiently utilize the current hardware and software without buying new expensive computing systems [67]. As network and computer technology is developing cloud computing is becoming an area of interest for most big data researchers

[73, 75]. Notably single processor's memory and CPU resources cannot handle big data to be employed for data mining solutions in huge data [7, 76]. Thus, there is no need to buy new expensive computers for Hadoop experimental solutions. An MR platform simplifies the programming needs by incorporating them on map and reduces tasks. Therefore MR when deployed speeds up the processes of support count and obtaining new candidate sets for data mining. According to Liu et al. [7] the MR model offers simplified application development in distributed settings to the Apriori algorithm to improve parallel processing. MR intermediate results are transferred to the reducer through a repetitive process allowing it to handle values too big to be accommodated by the memory. The presentations from previous studies show that parallelization is the best solution to efficient data mining. The MR framework is designed to process voluminous data using hundreds or thousands of machines hence the library must be able to endure machine failures. MR is used on jobs with high rates of failure and guides them to successful completion on a large cluster [18].

2.6 Theories Based on Hadoop MapReduce

We describe theories based on Hadoop™ developed by Apache® and MR™ developed by Google®. Both MR and Hadoop are open source software which are both suitable for processing big data.

2.6.1Apache Hadoop

The Apache™ Hadoop [68] framework permits distributed handling of huge volumes of datasets across clusters of computers over the internet. The framework is built in such a way that instead of relying on hardware to offer reliable compatibility, it relies on the software library. The software library has been crafted to detect and deal with failures at each application layer. Hadoop design allows for scalability of many machines over the internet and each machine offers its own storage and computations [51, 56]. The Hadoop distributed system overs the following favorable traits [54, 57]:

- Accessibility: Hadoop is available on clusters of hardware over the internet and most of its implementations are based on cloud computing services such as Cloudera™.

- Robustness: Hadoop is intended to run on commodity hardware, thus it has strong management tools that equip it to handle hardware failures.
- Scalability: Hadoop offers a linear scale in order to ideally handle big data by increasing the number of nodes to clusters available.
- Simplicity: Hadoop possess the ability to handle and provide the environment for functional program handling.

Hadoop is an open source software which is capable of handling distributed programs for developing and running large databases. Hadoop is simple and easy to setup and configure such that even college students can quickly create their own cheap Hadoop cluster. The jobs are controlled by the master node, which is known as the *NameNode* and it's responsible for chunking the data. Hadoop activities like data replication, data splitting and monitoring the status of the cluster are carried out synchronously [54, 71].

2.6.1.1 Components of a Hadoop Cluster

The Hadoop framework uses a cluster with master-slave architecture. Figure 2.8 shows the core components of a Hadoop cluster which includes *NameNode, Secondary NameNode* which are components of the HDFS and the JobTracker carries out map and reduce functions on a single machine. In HDFS datasets can be split into chunks and then replicated across *DataNodes* in a Hadoop cluster. In the scenario for a single node, all entities are executed by the single node with a *NodeManager*.

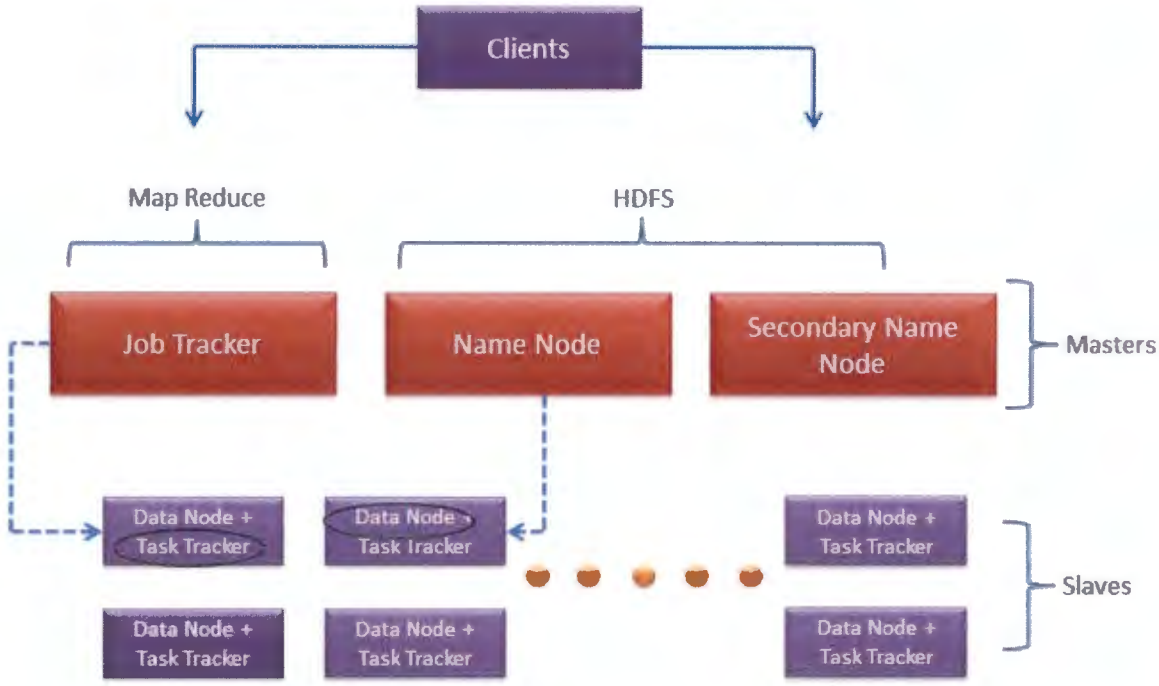

Figure 2. 9: Hadoop Core Components [18]

NameNode

The *NameNode* is a specialized node where system metadata reside for HDFS. It also maintains file location data as datasets are placed in clusters. There are two in-memory tables that store information about data mappings onto *DataNode*. The tables can also be used when there is a disk corruption on any section of the data split and when there is a node or network failure [69].

Secondary NameNode

The *Secondary NameNode* performs all time to time management and housekeeping functions for the *NameNode* [18]. It also periodically creates and monitors checkpoints of the file system existing in the *NameNode*.

DataNode

The *DataNode* engages itself in activities of storing data into HDFS and manages files with one node. It communicates with the *NameNode* concerning its information, storage and operations [70].

JobTracker and TaskTracker

JobTracker is a component that residences on Hadoop and its functions include receiving requests and assigning *TaskTrackers*. *JobTracker* holds and manages the jobs whilst *TaskTracker* manages the small activities called task kids. Tasks are assigned to *TaskTracker* based on the availability of data to be processed on the *DataNode* [18]. Tasks can be allocated to a mapper or machine for processing using their location within the racks. We can also deduce from the functionalities of *JobTracker* as presented to manage the resources and tracking their availability to be used. On the other hand, the *TaskTrackers* take the orders from the *JobTracker* and update their status periodically.

The Hadoop framework has a pre-configured *TaskTracker* where the number of slots indicates the number of activities it can accept. Therefore whenever a *JobTracker* schedules a task it has to consider the empty slots on *TaskTracker* running on the identical server which hosts the *DataNode* where the data for that task resides. Otherwise, it looks for the machine in the same rack. There is no consideration of system load during this allocation. Different *JVM* processes spawn to ensure that process failures do not bring down the *TaskTracker*. The *TaskTracker* send heartbeat messages to the JobTracker to show its activities and to keep it updated with the empty slots available for executing more tasks. Hadoop uses a web interface to keep the status information and updates of *JobTracker* and *TaskTracker*.

Hadoop does speculative execution on a slow machine in the cluster and the map/reduce tasks running on this machine and other machines runs redundant jobs on other machines to process the same task which helps if a machine fails [71].

2.6.1.2 Hadoop Distributed File System (HDFS)

The HDFS [66] is designed to store both structured and unstructured lots of data sets reliably with high bandwidth applications. The resource can grow with demand while remaining

economical at every size because of its distributed storage and computing across several servers. Files in HDFS are divided into splits, typically 64MB, and each block is stored in a local file system [66]. HDFS implemented by the *NameNode* is responsible for maintaining the HDFS directory tree, and is a centralized service in the cluster operating on a single node [72]. *NameNode* performs common file system operations that may include open, close, rename, and delete. The *NameNode* does not store HDFS data itself, but rather maintains a mapping between HDFS file name, a list of blocks in the file, and the *DataNodes* on which those blocks are stored [16]. We describe the architecture of HDFS in Figure 2.9 with two services which are *NameNode* and *DataNode*.

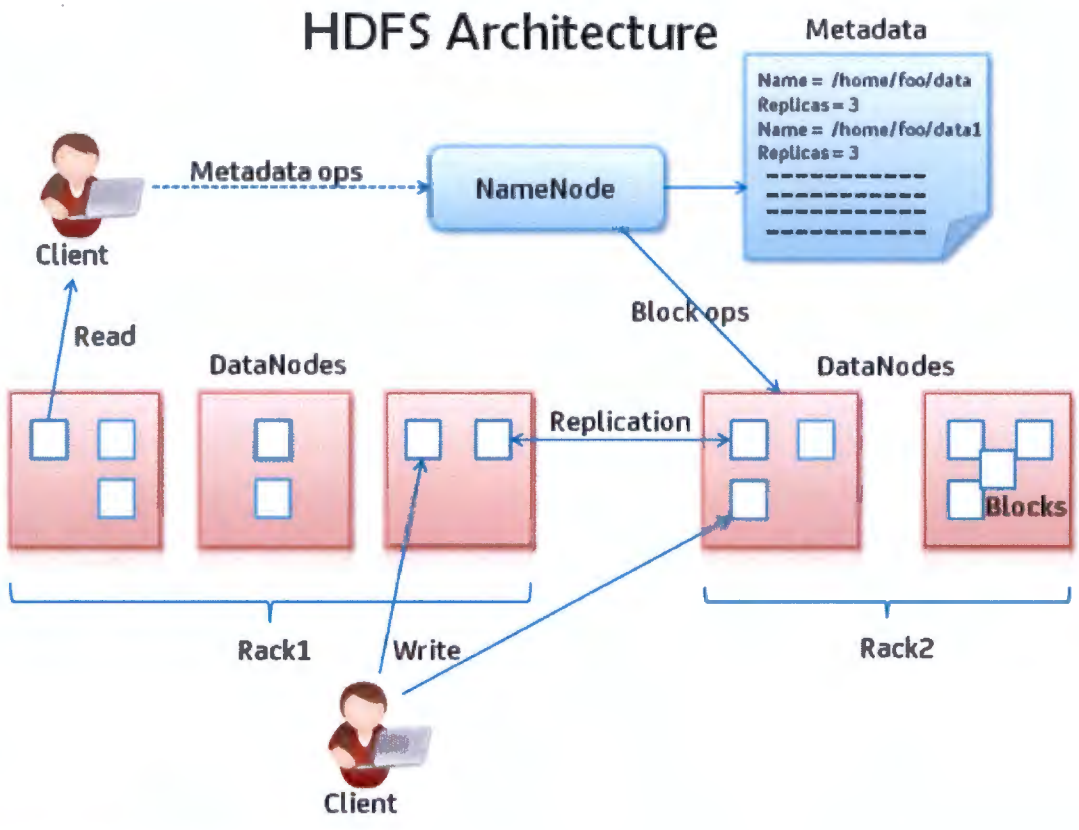

Figure 2.10: HDFS Architecture [71]

Furthermore, in scenarios where a centralized *NameNode* is used, all remaining cluster nodes provide the *DataNode* service where each *DataNode* stores HDFS blocks [73]. *DataNode* abstracts details of the local storage arrangement and saves each data block as a separate file

where the first data file to be accessed is huge. Second files are manipulated via streaming access patterns typical of batch-processing workloads [54].

Many files are written in a sequential manner. Hadoop uses a coherence model to allow data to be modified once written to exploit the streaming workloads [74]. HDFS is divided into large blocks for storage and access, typically 64MB in size. Portions of the file can be stored on different cluster nodes thus balancing storage resources and demand [75]. In [60] manipulating data at this granularity is efficient because streaming-style applications are likely to read or write the entire block before moving on to the next [41]. In addition, this design choice improves performance by decreasing the amount of metadata that must be tracked in the file system, and allows access latency to be amortized over a large volume of data [68].

2.6.2MR Programing Model

MR is a processing technique and a program model for distributed computing mainly based on object oriented language called java. The MR tool contains two important tasks, namely Map and Reduce used for parallel data processing [71]. The main advantage of MR is that it can scale data processing over multiple computing nodes and does not have data model or schema dependency. Under the MR model, mappers and reducers are agents of implementation of a MR in MR Model [8, 9]. A MR job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to reduce tasks. To ensure a job is progressing in the presence of variable system load different scheduling techniques are utilized. Finally, to avoid failures in data centres a number of failure avoidance and recovery features are enforced to ensure job completion in the environment [57].

2.6.2.1 MR Job Flow to JobTracker

When the MR job is submitted to *JobTracker* it is then the *JobTracker's* responsibility to send the job to the *TaskTracker* and schedule tasks and monitor them. *TaskTracker* gives the *JobTracker* feedback on the progress of the job. Figure 2.10 illustrates how a MR job is

processed from *JobTracker* and to other different loads in a particular manner. Listed below are the steps which are followed when any MR job is submitted by the user until it gets submitted to *JobTracker*[54]:

- The user copies input file to distributed file system (dfs)
- User submits job
- Job client gets input files information
- HDFS creates splits
- User uploads job information which include Job.jar and Job.xml
- Validation of Job Output directory occurs through HDFS API call and then client submits job to *JobTracker* using RPC call [66]

MR is ideal for operating on very large datasets and performs parallel operations on them. The MR Job goes through a map stage and a reduce stage where arithmetic operations on transaction items take place.

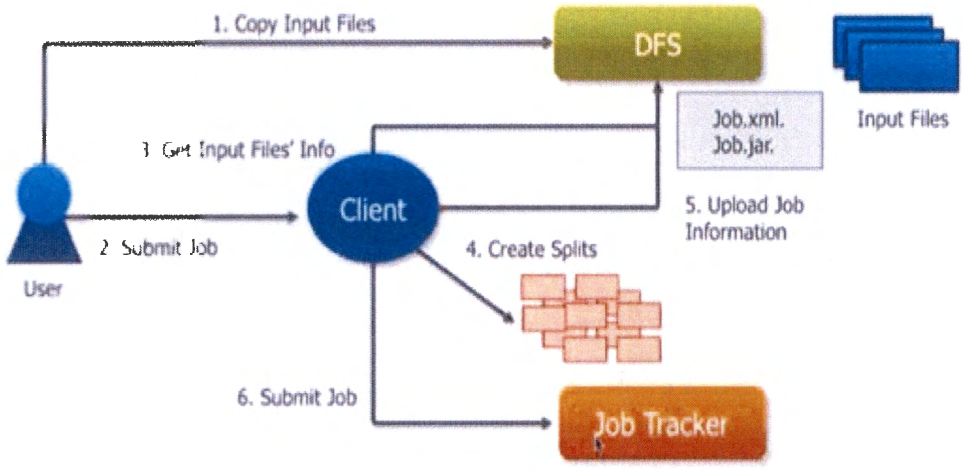

Figure 2.11: MR Job Flow to JobTracker [54]

2.6.2.2 MR Flow chart

MR works on bringing the computer to the data instead of traditional parallelism systems that would bring data to the computer. The execution of MR job includes the following listed phases as shown on the MR flow chart in Figure 2.11 demonstrating all the major processes in MR.

- Input Files

  MR task data is stored in input files and they reside in HDFS. The format of these files is arbitrary, while structured and unstructured format also will be used.

- Input Format

  Input Format defines how input files are split and read. Input Format selects the files or other objects that are used for input. It can create input splits logically and represent the data which will be processed by an individual mapper.

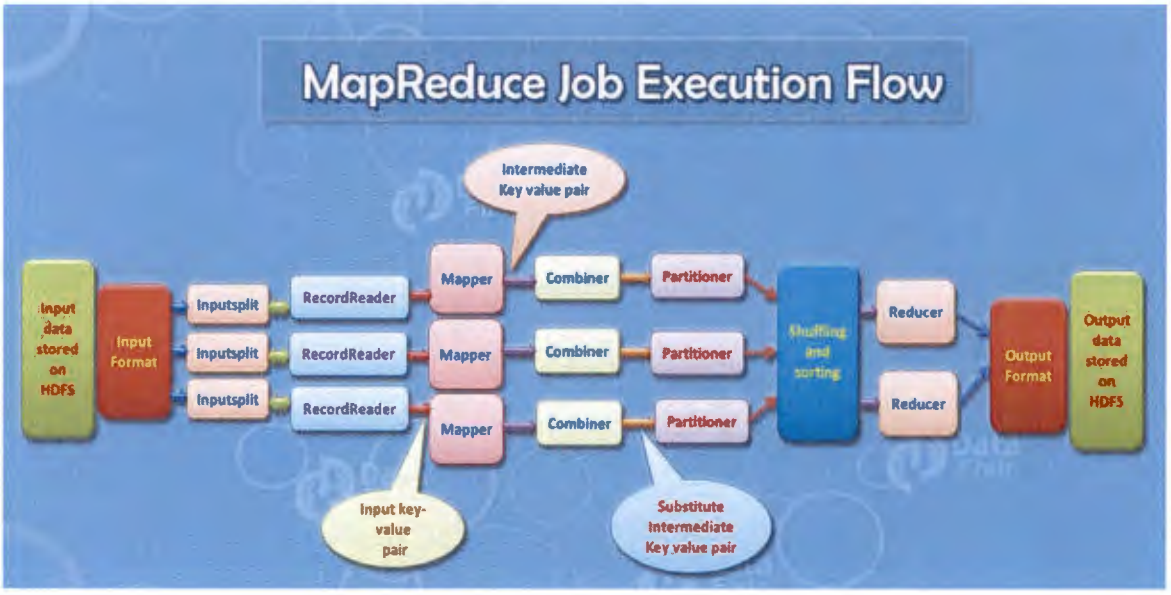

Figure 2.12:MRJob Execution Flow [18]

- Input Splits

The decision to split input data can be done as data is captured into HDFS and this is determined by the number of *DataNode*s and mappers. The replication factor determines the amount of replication to be carried out. One map task is created for each split and the number of map tasks will be equal to the number of Input Splits. The split is divided into records and each record will be processed by the mapper.

- Record Reader

  In Hadoop MR the Record Reader communicates with the Input Split and converts the data into suitable format for interpretation and analysis by the mapper. Text Input Format is used to convert data into a paired format. The Record Reader communicates with the Input Split throughout the whole process of file reading. The Record Reader assigns byte offsets (unique number) to each line present in the file. Lastly, key and value pairs are taken to the mapper for further processing [71].

- Mapper

  The mapper processes input record from Record Reader and generates new key-value pair, and this key-value pair generated by Mapper is completely different from the input pair. The output of Mapper is also known as intermediate output which is written to the local disk. The output of the Mapper is temporary data and writing on HDFS will create unnecessary multiple copies. HDFS is a high latency system so the output of mappers is passed to combiner for further processing.

- Combiner

  The combiner is also referred to as a Mini-reducer and performs local aggregation on the mappers' output to minimize the data transfer between mapper and reducer. Once the combiner functionality is executed, the output is passed to the partitioner.

- Partitioner

  Hadoop MR Partitioner is utilized when more than one reducer is implemented. The Partitioner takes the output from combiners and performs partitioning. Partitioning of output takes place on the basis of the key and then sorted using hash function and key used to derive the partition [69]. Partitioning allows even distribution of the map output over the reducer.

- Shuffling and Sorting

Shuffling is the physical movement of the data over the network. It involves transferring data from the temporal data from mappers stored in HDFS to reducers for processing. Shuffling can start even before the map phase has finished to save time[53, 58]. Sorting is a process that allows data output from mappers to be orders in items of the same identity and similar keys to help reduce processing time for the reducer. A reduce task can distinguish when a new reduce task should start since Hadoop can only start a new reduce task when the next key in the sorted input data is different from the previous key [73]. During shuffling, input data is pre-sorted (locally) in the map phase [71]and merge-sorted in the reduce phase (since the reducers get data from many mappers). Once all the mappers are finished and their output is shuffled on the reducer nodes, then this intermediate output is merged and sorted, it is then provided as input to reducer phase.

- Reducer

  Reducer takes intermediate pairs produced by the mappers as the entry data input. Reducer function is executed on the input to generate the output. Then the output of the reducer is the final output which is stored in HDFS.

- Record Writer

  Consequently, Record Writer writes these output key-value pairs from the Reducer phase to the output files.

- Output Format

  The Output Format determines the way the output key-value pairs are written in output files by Record Writer. Output Format instances provided by the Hadoop are used to write files in HDFS or on the local disk. As a result the final output of reducer is written on HDFS by Output Format instances.

Figure 2.12 illustrates an MR job from the point it is split until it is completely processed. Once the MR job is initiated, the HDFS splits the data into five data blocks which are then processed by the available three workers for the map phase. The *TaskTracker* in Hadoop is the one that is responsible for allocating tasks as it checks the progress of jobs through the *JobTracker*. Workers in the map phase read the data and pass its output to intermediate files located on HDFS. The web interface on Hadoop *Appeaser* possesses a summary of all the activities taking place and the resources allocated [47]. The two workers in the reduce phase accept the

intermediate data and process it to information then write into HDFS [71]. The reducer function performs computations on the intermediate information so that the final output can be produced. Finally, the outputs of the Hadoop reducer are outputs as key and value pairs. In the scenario given in Figure 2.12 two output files have been produced due to the presence of two reducers working parallel on data.

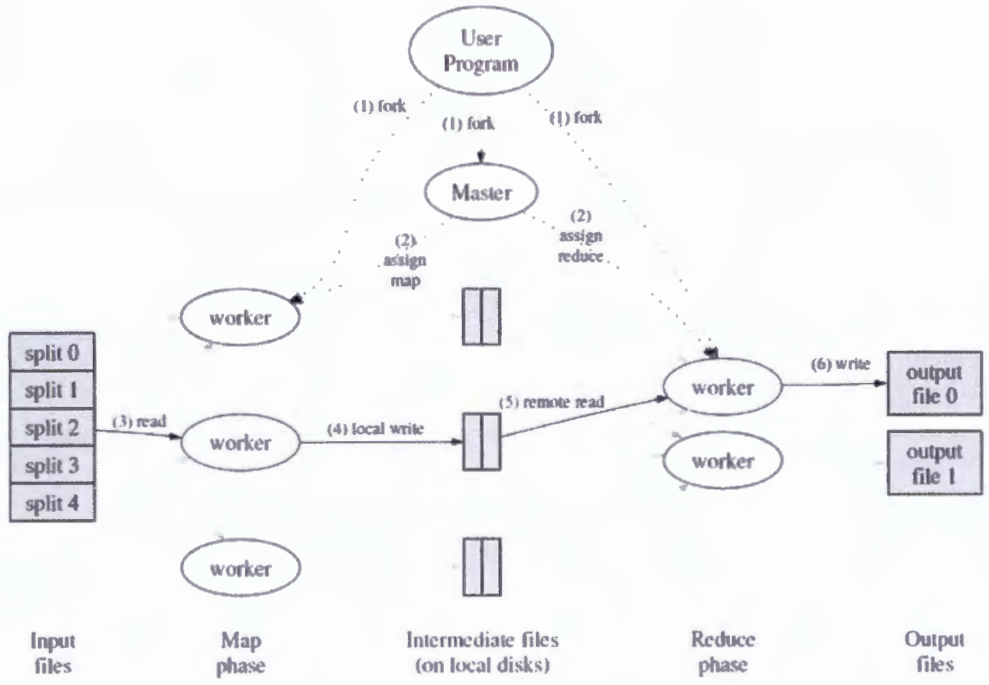

Figure 2.13: MR Execution Overview [7]

In this chapter, we presented a review of work that has been done and also identified some shortcomings of present approaches. Based on the analysis of the studies related to our study there is a need to enhance the performance of Apriori by designing the algorithm and implementing it as scalable MapReduce. The following chapter presents the proposed model of the research.

CHAPTER 3: PROPOSED IMPROVED APRIORI ALGORITHM MODEL

This chapter addresses the improved Apriori model. The first section of this chapter presents theorems and definitions of the study area. Secondly we describe how the Apriori algorithm is presented and utilized on MR followed by the flow of activities. The third section present the pseudo code of the improved algorithm and its parallel implementation. Lastly, we propose the experimental set up based on Hadoop MR and Yet another Resource Negotiator (YARN).

3.1 Definitions and Theorems

The discovery of frequent itemsets based on relationships between data in databases is a cumbersome process. Though the Apriori algorithm is easy to understand and uses a simple data structure the computation's complexity is high for the whole process. The Apriori algorithm has been revised in many ways by past researchers. We propose a parallel utilization of Apriori through the use of Hadoop MR and Java Eclipse. We also propose to utilize HDFS storage. Lastly, we propose deletion of transactions that do not satisfy minimum frequency count and the value of k size candidate itemset at each level increment.

Definition 1: C_k; where C_k the candidate itemset of size k, and L_k frequent itemset of k large [5, 20].

Definition 2: Suppose that the transaction database is split into N data splits then each item has local item frequency, β_N and the support count (global support) which is obtained by adding β_N for all data splits.

$$\sigma = \sum_0^N \beta_N \quad \text{..... (4)}$$

Where $\beta_N$support count of an itemset in the Nth is spilt and N is the quantity of processor number of the processors/split. The global support count is the total support count of an item. Apriori Principle suggest that a frequent itemset has also all its subsets frequent[4, 13]. This

principle makes the pruning process less complex and also possible to prune using either a top down or bottom up method.

We utilize Hadoop HDFS to map the database transactions and split them into N data Splits to ensure that our improvement strategy scans the database exactly once. The splitting of the data blocks is determined by the processors or mappers available. We programmed the improved Apriori algorithm using Java Eclipse and exported into a jar file. We then employed the jar file in Hadoop MR to extract frequent patterns. MR processes the data by using Map/Reduce function to obtain local item support, β_N and its global support count, σ. We utilized β_N to prune infrequent itemsets on each data split and σ to delete infrequent itemsets in the whole data set. The support counting results are stored in HDFS. Support counting for candidate itemsets generated is simultaneously calculated using the Map function of the MR concept of Apache Hadoop architecture. Pruning of infrequent itemsets is carried out by using a support based pruning strategy that exploits the Apriori principle and anti-monotone property of support measure described in the above definitions. All generated candidate itemsets whose support count does not meet the minimum support count threshold are deleted also.

We perform Candidate generation of the *k-itemsets* based on the frequent $\text{itemset}_{(k-1)}$ found in the previous iteration. The candidate itemsets for every frequent itemset_k are composed of a frequent $\text{itemset}_{(k-1)}$ and a frequent itemset_1, L_1. Henceforth, all frequent itemsets_k are part of the candidate itemset_k that will be generated.

We propose to implement a pruning strategy and to make the Apriori algorithm parallel using software frameworks that can accommodate parallel tasks such as MapReduce [71]. There are many different implementations of the MR interface based on environments [7, 76]. We propose to utilize a scalable framework based on MapReduce. The MR model requires key-values pairs for input and output. Hadoop stores both structured data and unstructured data formats making the Hadoop MR most appropriate for retail data mining [16, 53].

Figure 3.1 provides a general illustration of the improved Apriori algorithm on MR paradigm outlining the activities that are involved at each MR phase. MR provides sufficient high-level parallelization [77]. Transactions from the transactional database are split into N splits and data

converted into key-value pairs for processing within the HDFS [4]. Hadoop allocates each data split to a processor so that it can be allocated a mapper.

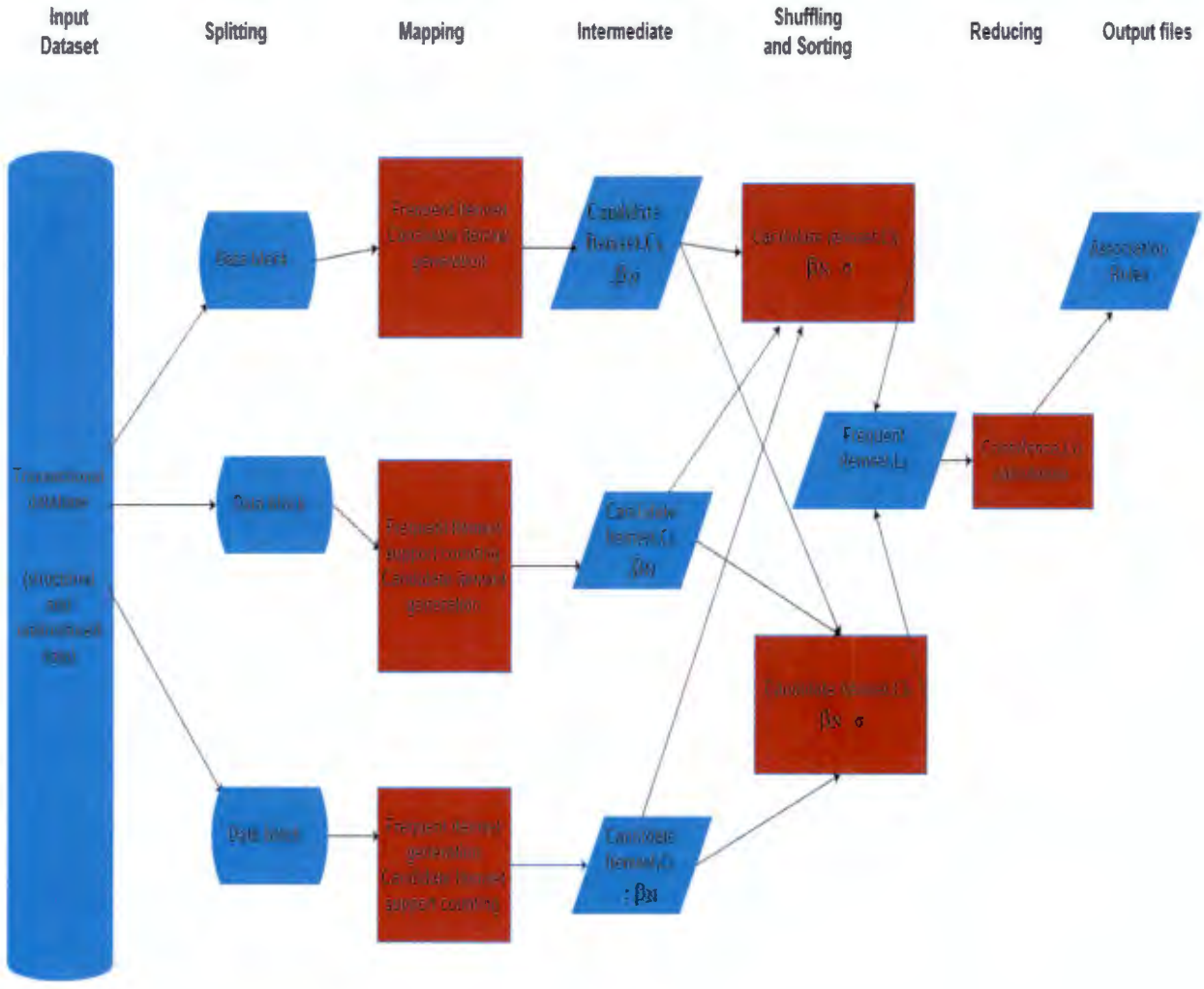

Figure 3.14: Improved Apriori Algorithm Implementation Process

Each Map function obtains local support count of each item, β_N. Mapping involves first copying of relevant data from local drives to HDFS and data conversion to key - value pair's format. In the improved algorithm we allow all the algorithm steps to be parallel thus increasing its performance. In this case the Mapper will input (itemset, 1) and output (itemset, β_N (frequency of item)). The reducer take the pairs (itemset, β_N) and perform a reducing function then outputs

(itemset, σ). The first pruning takes place in the mappers after aggregating the intermediate output of each mapper the item β_N is compared to minimum support count. Any itemset with an occurrence less than the minimum support threshold is deleted (β_N>= min support). Second elimination takes place after integrating the results of all mappers at the masters' node. Thus there is local deletion at the mappers (splits) and global deletion for the whole data still based on minimum support conditions. Reduce function adds the respective support count of the item from each data split and outputs the items with σ >= minimum support. The frequent itemsets, are produced at both the reduce phase and mapper phase. All available mappers iterate on the candidate generation process utilizing a complete effective candidate generation method. All candidate itemsets with β_N >= minimum support count are considered for strong rules. The frequent itemset L_k is obtained by merging the output of all reducers.

3.2 Proposed Flow of Activities

In Figure 3.2 we illustrate and discuss a sequential flow of activities to be followed in implementing the improved Apriori algorithm.

- Collect and generate retail data
- Customize and fine tune retail data
- Data splitting and uploading on hadoop
- Determine frequent itemset and candidate generation
- Generate frequent itemset for split
- Determine the frequent itemset for whole data set
- Determine the strong association rules

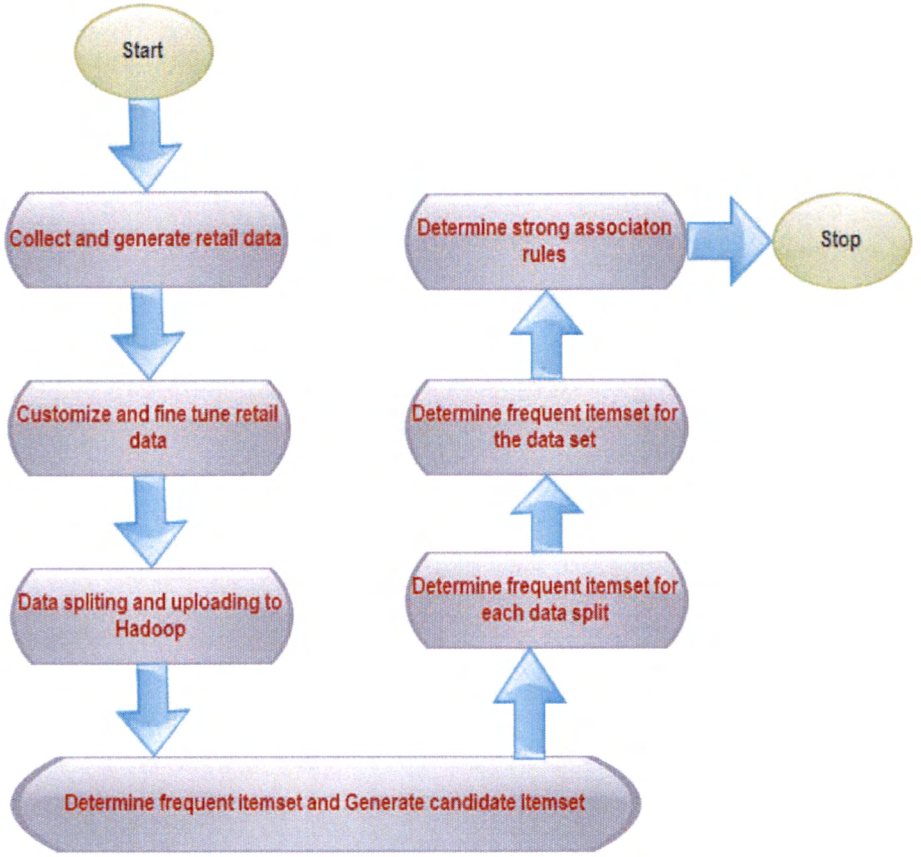

Figure 3.15: Proposed Flow of Activities

3.3 Improved Apriori Pseudo code

The customized retail data is loaded and partitioned into splits by Hadoop. The map function starts processing the split as initiated by the Map Task. The map function sequentially reads and converts data to (key, value) pairs. Intermediate results are written to the HDFS where they temporarily reside whilst they await the combiner class to process them. In Algorithm 3.1 we present the pseudo code of the *FrequentItemMapper* class that implements the map function. In Algorithm 3.2 we present the *FrequentItemReducer* class for arithmetic sum. The reducer adds up the value associated with the same key and writes results in the output file in ascending order of the keys. The output file consists of a list of frequent itemset items. The output of the *FrequentItemsetReducer* is stored temporarily on HDFS.

FrequentItemMapper

Input: Data split D_{Si}, min support

Output: *(key, value),*

Key: frequent itemset$_k$ item

Value: $split\ support\ count, \beta_N$),

Map (item, D_{Si}) *// Map function*

L_1=Get frequent itemset$_1$ (D_{Si})

For (k=2; Lk-1≠ ∅; k ++)

Candidate itemset generation. C_k;

For each candidate itemset c ∈C_k

c.count =c.count +1;

End for

L= {c∈C_k | c.count ≥ $min\ support$}

If itemset I ∈ L

Output (I, itemcount);

End map

End

Algorithm 3.1: Frequent Itemset Mapper

FrequentItemReducer

Input: *(key2, value2), where key2 is frequent itemsetk item and value2: split support count* β_N).

Output: *(key3, 1), where key3 is frequent candidate itemsetK, item*

Reduce (key2, value2) *// Reduce function*

Output (key3, 1); *// collected in Li*

End reduce

End

Algorithm 3.2: Frequent Itemset Reducer

Algorithm 3.3 describes the computation mapper of improved parallel Apriori algorithm discovering of frequent itemsets from all splits is an interactive process that involves an iteration of map and reduce function. The Computation mapper class takes input from Frequent Itemset Reducer class. The input (key, value) is the list of frequent itemsets. Then the frequency of each item in the frequent $itemset_k$ is counted using the map function. The support count of each itemset in a split ($suppitem_N$) is computed.

ComputationMapper

Input: *Data split, Dsi. Li and Li is stored as a temp intermediate file.*

Output: *(key. value). where key is item of the list, Li and value is split support count* β_N)

For each itemset I in Li

Map (item, Dsi) *// Map function*

count = count I frequency in Dsi

Output (I, count)

End Map

End for

Algorithm 3.3: Computation Mapper

The above algorithm describes the computation mapper of improved parallel Apriori algorithm. Discovery of frequent itemsets from all splits is an interactive process that involves an iteration of mapping and reduction functions. Algorithm 3.3 Computation Mapper class takes input from Frequent Itemset Reducer class. The input (key, value) is the list of frequent itemsets. Then the frequency of each item in the frequent $itemset_k$ is counted using the map function. The support count of each itemset in a split ($suppitem_N$) is computed. The reducer function in Algorithm 3.4 obtains the input from the Computation Mapper class and sums the frequency of the associated key for all the splits of the data sets.

Both Algorithm 3.5 and Algorithm 3.6 are map and reduce functions to determine the association learning rules. Firstly, Rule Mining Mapper will extract all frequent associations that have been discovered during frequent itemset mining. The mapper does this by listing the frequent itemsets with their support count globally by taking the temporally saved output file from the computation reducer class. The inputs are shuffled and sorted then sent as output to the reducer. Finally the improved algorithm outputs frequency frequent itemsets, L_1; L_2; L_k.

ComputationReducer

Input: *(key2, value2), where key1 is candidate itemset$_k$, items and value1 is split support count.β_x).*

Output: *(key3, value3), where key is frequent itemset$_k$, item and value2 is support count σ*

```
Reduce (key2, value2)                          // Reduce function
    Sum = 0;
While (values.hasNext ()) {
   Sum += values.next ().get ();
    Output (key3, (sum));
   End if
If (sum>=min support)
Out (key3, sum);
End if
End reduce
End
```

Algorithm 3.4: Computation Reducer

RuleMiningMapper

Input: *frequent itemset list, L_{kg}, support count where L_{kg} is the global frequent itemsets*

Output: *Associated rules, support count*

Map Function (L_{kg} , support count)

For each frequent item F_i in L_{Kg}

Put on same partition entries of the same key

End if

End for

Algorithm 3.5: Rule Mining Mapper

RuleMiningReducer

Input: *frequent itemset list, L_{kg}, support count where L_{kg} is the global frequent itemsets*

for all splits/mappers

Output: *R_s, As a set of strong associated rules*

Reduce Function ()

For every candidate itemset i:

Calculate confidence con = support count of i / support count;

If con >= confidence threshold

R_s = R_s union (key --> con);

End for

End

Algorithm 3.6: Rule Mining Reducer

Finally Algorithm 3.6 of the improved Apriori algorithm pseudo code describes rule mining reducer class where the confidence of each candidate itemset is calculated by the reducer function. All frequent itemsets with confidence >= minimum threshold are produced as strong association

FrequentItemPartitioner

Input: (key, value, ReduceTask Value), where key is the itemset, value is 1 and Reduce task Value is the number of reduce tasks.

```
GetPartition (key, value, ReduceTask number) {
Read keysize from dataset
Split (",").Length
If (numReduceTasks==0)
Return 0;
If (keysize ==1)
Return 0;
Else (keysize==2)
Return 1;
Else Return 2;
}
```

Algorithm 3.7: Frequent Itemset Partitioner

Algorithm 3.7 presents the Partitioner class which is responsible for dividing and allocating intermediate data to respective reducers based reducer tasks available. The Partitioner search through the temporal data stored into HDFS after map operations and extracts the number of

reducer tasks and assigns them accordingly. A Partitioner class is designed only when there are at least two reducer functions to process data.

3.4 The parallelization of Improved Apriori based on YARN MR Model

We propose to make all core activities of improved Apriori parallel. The parallelizable activities include determining frequent itemset and pruning for every split, computing frequent itemset for all splits and implementing unsupervised learning techniques in MR to discover association rules. Figure 3.10 presents a parallel and distributed improved Apriori algorithm with all its activities. We generate and customise retail data and copy it into Hadoop using the terminal.

The HDFS first divides the data sets into N splits that correspond to N mappers. We have divided the improved algorithms into three phases. During the first phase, data partitions are allocated to mappers. The mappers read and process the assigned splits. The map function executes and provides a value pair from each. Map task determines the support count of the candidate itemset and outputs the candidate set and their support counts. The output is stored temporarily in HDFS. Candidate itemsets are generated from the allocated mappers. The partitioner allocates the map output to the reducers available. MR then gathers all the value pairs and pass them to reducers. The reducer receives a key and a list of values and produce key and sum. The reduce function takes assigned itemsets and produces frequent itemsets. All infrequent itemsets are deleted using Apriori support measure where all itemsets with support count that does not meet set conditions for minimum support frequency set are eliminated. The obtained most occurring itemsets are saved and stored in HDFS.

The second phase involves processes that will determine the frequent itemsets for the whole data sets. The map function takes a combination of the output from the HDFS and part of the input data sets. The Map task sums the local support count of the candidate itemsets and outputs the candidate set and their support counts. All frequent itemsets from each split are combined into associated values. The global support (σ) is an output of support count of the itemsets and is calculated based on the reduce function. The reduce task handles all reduce tasks assigned from map tasks and outputs the item and their support count. As a result itemsets with support count less than minimum support are pruned and the rest of the data is transmitted to HDFS as output. The frequent itemsets are then generated from the results-.In the final phase, the mappers collects all frequent itemsets and their support counts that have been saved in HDFS. The combiner

collects and sorts data from mappers and directs it to reducer. The reduce function sorts the results and prunes infrequent rules using minimum confidence thresholds and minimum support counts to obtain strong association rules. Therefore strong rules are generated and stored in HDFS.

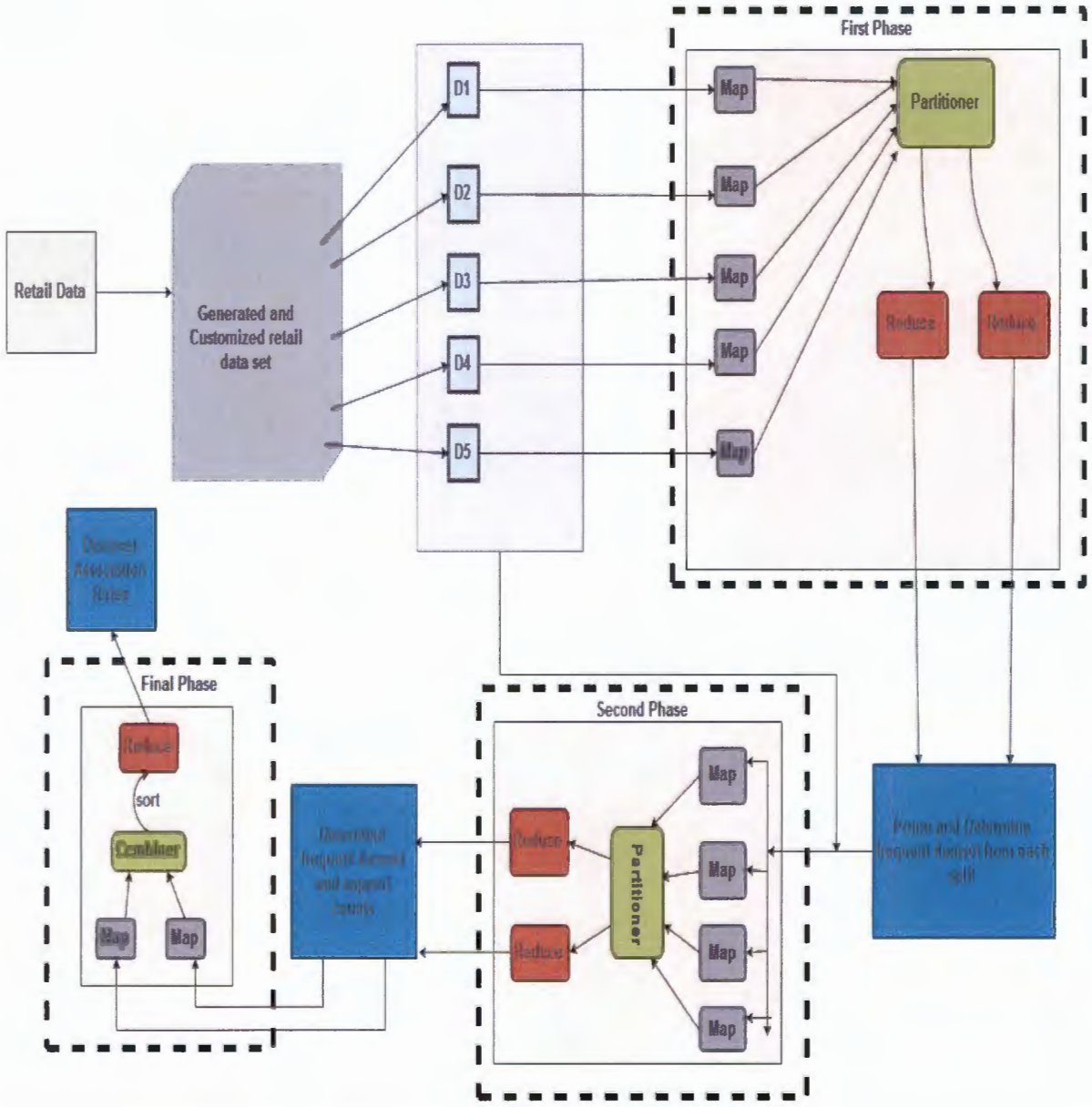

Figure 3.16: Parallel Improved Apriori Implementation

3.5 Hadoop (MR2x) YARN Implementation

In this section we present the Hadoop MR YARN (Yet another Resource Negotiator) framework which we employed to run our implementation. We also discussed the components of the YARN based framework and the execution of MR job on YARN which comes with Cloudera CHD5 set up version that we utilized. Firstly we load the customised data set into the hadoop using the provided terminal to submit MR jobs. Once the MR task is submitted *ApplicationMaster* negotiates for the resources from the *ResourceManager*. YARN incorporated open source systems are best in controlling distributed applications over internet [32]. The YARN container controls the access to resources on the host to applications. It also optimizes cluster utilization by keeping all resources in use all the time. *ApplicationMaster* partners with *NodeManager* to run and manage the container's operations. The responsibility of *NodeManager* includes monitors launching containers and monitoring them. Lastly the History server maintains information about the submitted MR jobs after the ApplicationMaster terminates. In Figure 3.11 the activities that take place during the execution of MR on YARN are summarized in sequential order of execution.

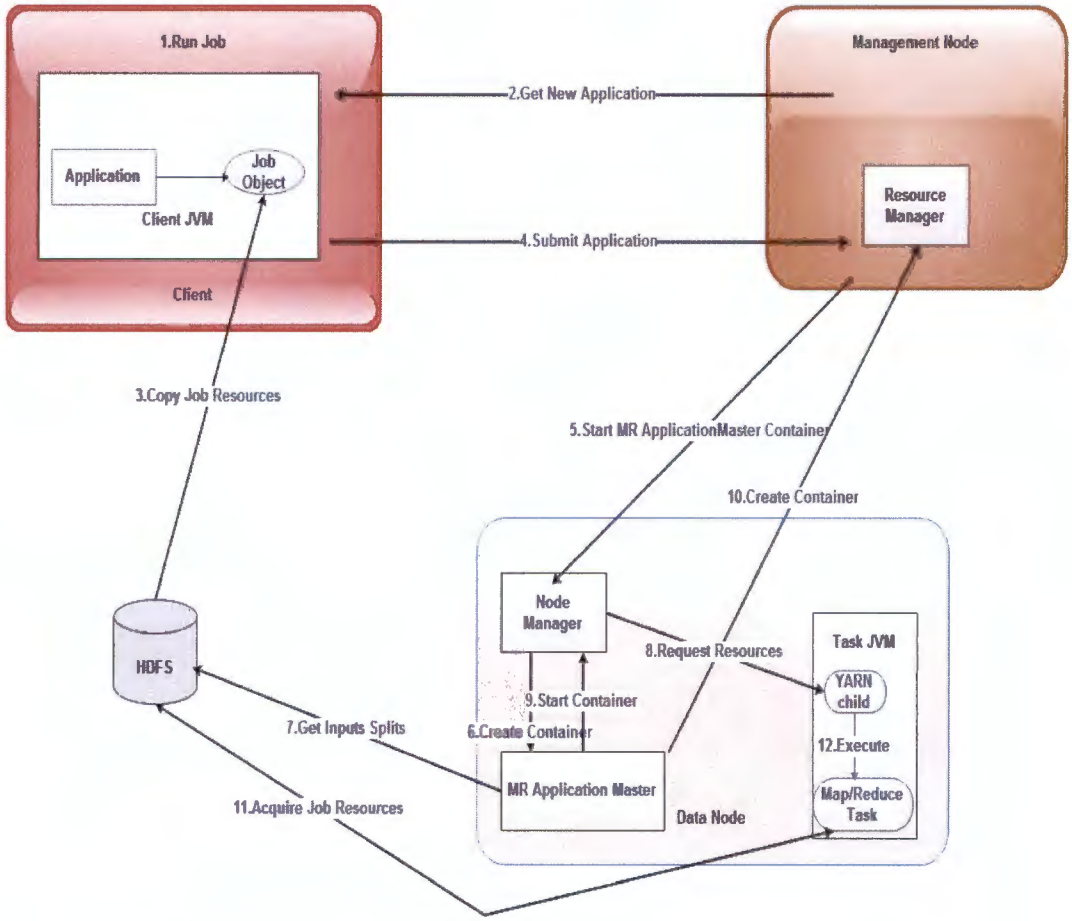

Figure 3.17: YARN MR Application Execution Flow

1. A user program like Cloudera Hadoop submits the request to launch the application.
2. The ResourceManager allocates a container to start Application master.
3. The ApplicationMaster registers with the ResourceManager enabling it to have a connection with ApplicationMaster[16].
4. ApplicationMaster asks containers from ResourceManager through resource request protocol.
5. ApplicationMaster notifies NodeManager to launche the container
6. The application code is executed within the container.

7. Clients contacts ResourceManager /Application master to monitor application's status.
8. ApplicationMaster deregisters with the ResourceManager when job is completed.
9. The container is started
10. Create the container
11. The job resources has to be acquired.
12. The whole programme is executed.

In this chapter we presented the proposed improved algorithm and its implementation framework. The improved Apriori algorithm is based on MR on YARN. We also described the experimental set up based on Hadoop framework. The next chapter presents the experimental results and performance evaluation of the improved algorithm.

CHAPTER 4: EXPERIMENTAL RESULTS AND ANALYSIS

In this chapter we present and analyze the experimental results to determine that the improved Apriori algorithm determines association rules in a faster way. The chapter presents the parallel improved algorithm based on two measures of strength that are support count and the confidence. Additionally, the experimental set up is presented and discussed. Lastly we present and discuss the experimental results.

4.1 Experimental Data

We considered retail data sets based on structured and unstructured presentations as HDFS can accommodate both formats. We downloaded grocery and supermarket data sales transactions from the internet in its raw format as csv as shown in Figure 4.1. The researchers generated and customized the data and copied it into HDFS where it was replicated by replication factor 1. In Figure 4.2 some of the dataset samples are presented. The transactions had an estimated average transaction magnitude of 11 products and a maximum frequent transaction of 5 products. Sixteen mappers were implemented and three reducers. In cloudera Hadoop cluster, these datasets are first stored in HDFS before all data input files are processed.

1	citrus fru	semi-fini	margarin	ready soups		rice	sugar	pple	fruit cak	buns	coke	sweets	rolls/bun	soap	oranges						
2	tropical f	yogurt	coffee	pots	pants	soverthead		coke	banana	beef	butter	powdered	milk	powdered	soap	beer	bread				
3	whole milk		milt	dear		rice	UHT-milk	flour	soda	coke	beer	jam	soup	spinach	beans	bread	pampers				
4	pip fruit	yogurt	cream ch	meat spreads		eab	fruit cak	fanta	pampers	soup	veges		tea	cooking	mealie meal						
5	other ve	whole mi	condens	long life	bakery	product	jam	dairy	yoghurt	pampers	eab	bread	tennis	chocolat	mealie m	coke	pork				
6	whole mi	butter	yogurt	rice	abrasive	cleaner	green be	bean	peanut b	soda	rolls	buns	spinach	covo							
7	rolls/buns		cholate	swekk	tropical	eak	butter mi	beef	chicken	sausage	curd	yoghut	bananas	fanta	coke	eab	oranges	scones	jelly		
8	other ve	UHT-mill	rolls/bun	bottled b	liquor (appetizer)		coke	pork	chips	pap	powdered	milk	flour	chocolat	curd	oranges					
9	pot plant	yoghut	pampers	eab	milk	soda	curd	chocolat	oranges	pap	potatoes	coke	jelly								
10	whole mi	cereals	coffee	chocolat	apples	oranges	bananas	pens	books	chicken	coke	potatoes	scones	bread	sugar	brown br	jam	chicken			
11	tropical f	other ve	white br	bottled v	chocolat	fruit	jelly	scones	coke	bond paper		butter	beer	tomatoes	spinach						
12	citrus fru	tropical f	whole mi	butter	curd	yogurt	flour	bottled v	dishes	lunch bo	towel	spinach	coke								
13	beef	spinach	covo	soda	newspap	butter	milk	pork	scones	fanta	coke	potatoes	oranges								

Figure 4.18: Sample Transactional Data File

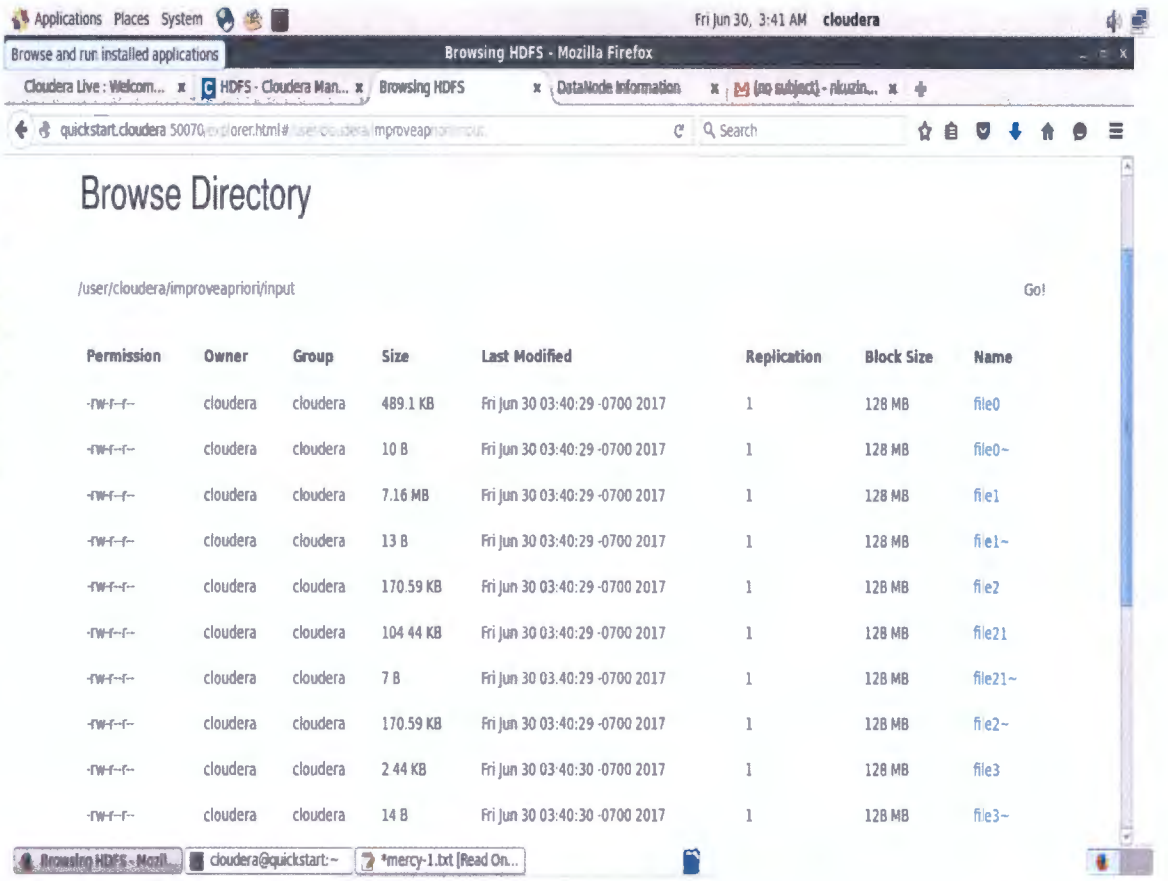

Figure 4.19: Sample Replicated Data in HDFS

4.2 Experimental Setup

We ran our experiment on Intel® Core™ i7-3770 CPU with a 3.40GHz Intel Core processor. Eclipse™ IDE (Eclipse Luna service release 2) was utilized to implement the improved Apriori algorithm using Java programming language. We developed the Improved Apriori jar.file that was then executed in Hadoop® MapReduce™. The implementation that was tested on Cloudera™ Hadoop MR environment has 8 processors running on Windows® 7 Enterprise as the host operating system as well as Linux Redhat® CentOS 6.7 version running on Oracle™ VM Virtual box. The investigation set up had 12 GB of globally accessible memory and 10 GB of accessible memory for the virtual machine. The system was a multiprocessing platform where HDFS share files and resources with 4 CPUs out of 8 CPUs were allocated to the virtual box. The Hadoop MR cluster was configured with 16 mappers and 16 reducers. HDFS split data from

a range 16 MB to 64 MB chunks presented as a map task with a factor 1. HDFS stores made duplicate files and stored the data block to ensure that data reliability and availability which in turn increased the performance of the algorithm.

4.3 Progress of MR job Hadoop

We implemented the improved Apriori algorithm on the Hadoop MR data processing framework. The algorithm first extracts frequent itemsets for each split which are then integrated. Then the algorithm determines hidden relationships and discovers frequent itemsets for all splits and eventually generates association rules. Then Apriori is executed and confidence is used to determine strong association rulesThe map function had to be completed first then the reduce function would start. Figure 4.4 represents 16 input paths being created meaning that 16 mappers had to be executed before the reduce function could start. We present Figure 4.3 which shows the progress of the map and reduce function. When Reduce function reaches 100% that means the job has been completed.

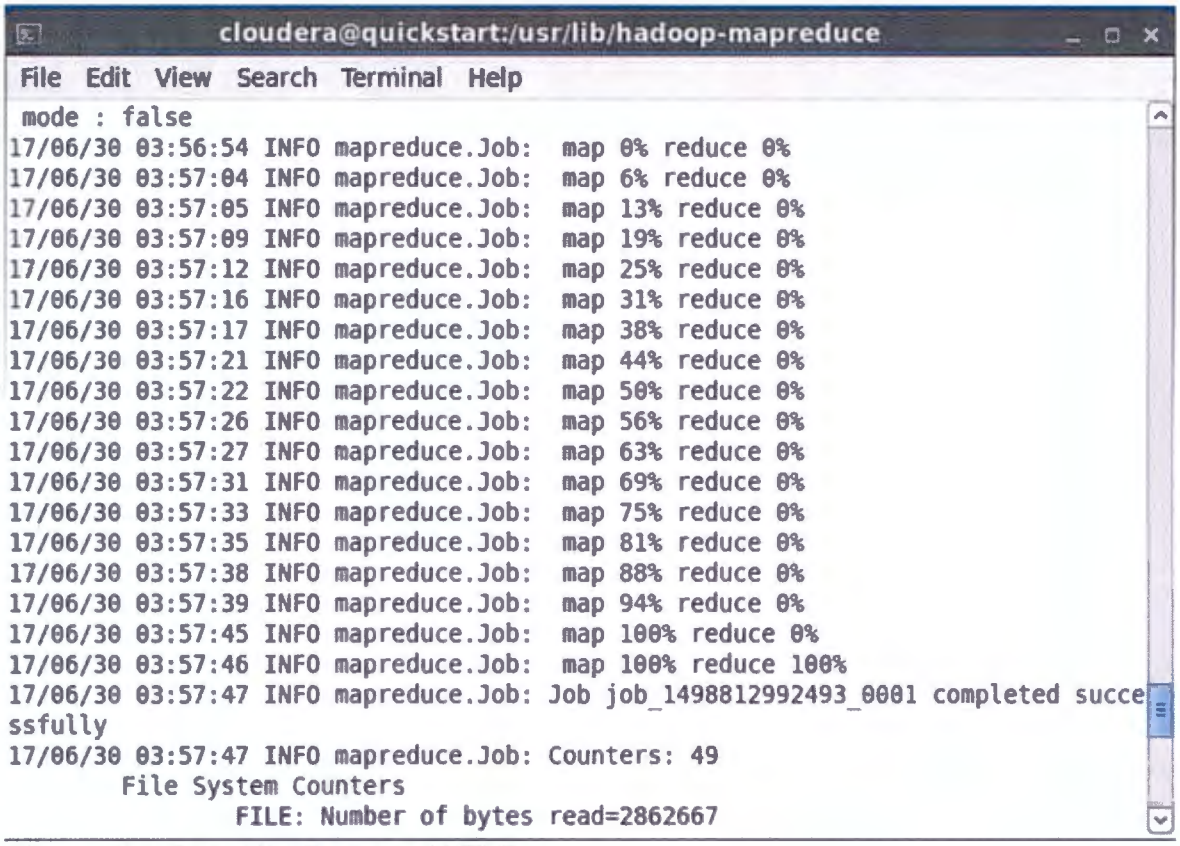

Figure 4.20: Execution Process of MR Job

Figure 4.4 presents the results of running parallel Apriori algorithm on MR platform.

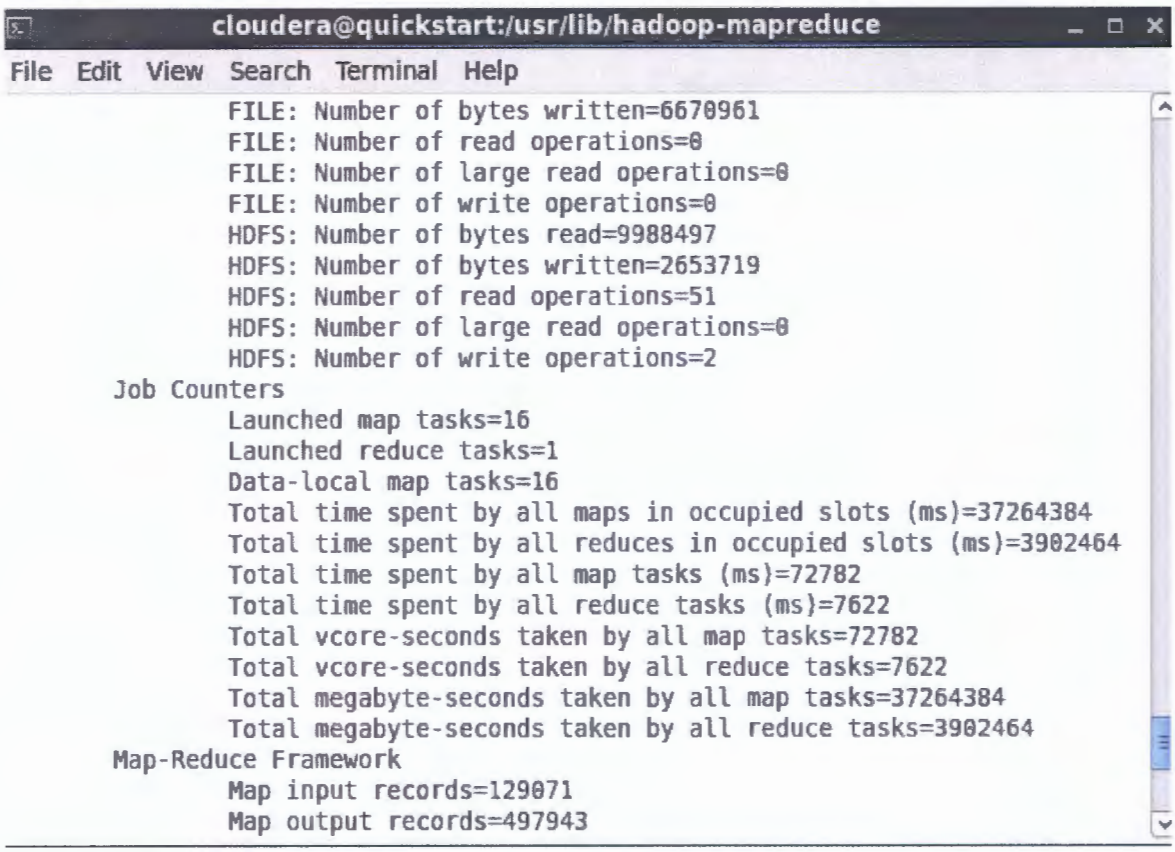

Figure 4.21: Results of Improved Apriori Algorithm

4.4 Experimental Results and Evaluation

Figure 4.4 shows sample extraction from the results of the investigation. In the next section we present and explain the metrics that we used to evaluated the results but firstly results are presented. A summary and discussion of the results of the experiments conducted follows. The metrics we used to measure performance include speedup, scalability and sizeup.

SpeedUp determines how much a parallel algorithm is faster than a corresponding sequential algorithm. It is determined by the following formula:

$$\textbf{SpeedUp} = \frac{T_1}{T_p} \quad \text{.....(7)}$$

Where p is the number of processors, T_p is the execution time for parallel algorithm, T_1 is the execution time for algorithm with processor 1. The execution times obtained from running improved Apriori on different datasets using different amount of processors is recorded in Table 4.1. SpeedUp and SizeUp is calculated from these results to measure and evaluate the algorithm performance. The results in Figure 4.5 show that the improved Apriori has a high speed up .There is a steady increase in the gradient of speed up which is due to the fact that when processors increases in number, speed up increases also but not linearly as it will be ideally in a serial system. We then conclude that the improved Apriori algorithm can efficiently mine association patterns in large retail data sets as can be seen by steady increase of the speedup gradient. Linear speedup is difficult to achieve due to increase on communication cost. Generally speedup improves for all the number of processors.

Table 4.1: Execution Time for the Improved Apriori Algorithm

Execution Time (seconds)						
Number of processors / Number of transactions	1	2	3	4	5	6
300	5.52	3.29	3.07	2.39	2.17	1.59
400	6.07	3.34	2.54	2.38	2.08	1.34
450	6.57	4.25	3.47	3.34	3.14	1.58
700	8.42	6.43	5.14	4.25	3.38	2.21
1000	9.05	8.03	6.57	5.58	4.07	2.38
2000	11.07	10.40	8.43	7.07	5.10	2.47
3000	13.52	11.11	10.02	8.30	6.04	4.52

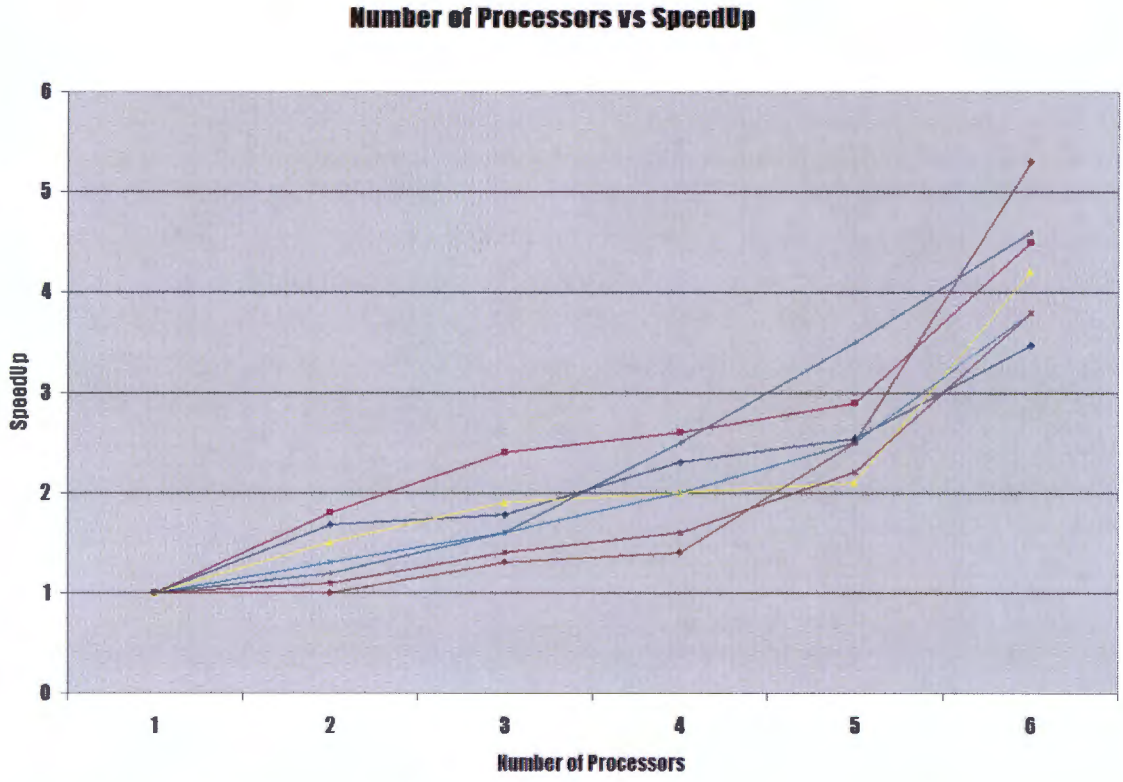

Figure 4.22: Number of Processors vs Speedup

In Figure 4.6 speedup is analyzed using a three dimensional graph. Speed up increases slightly by adding a processor this is attributed to the fact that as processors /nodes increase the need for communication increases as well.

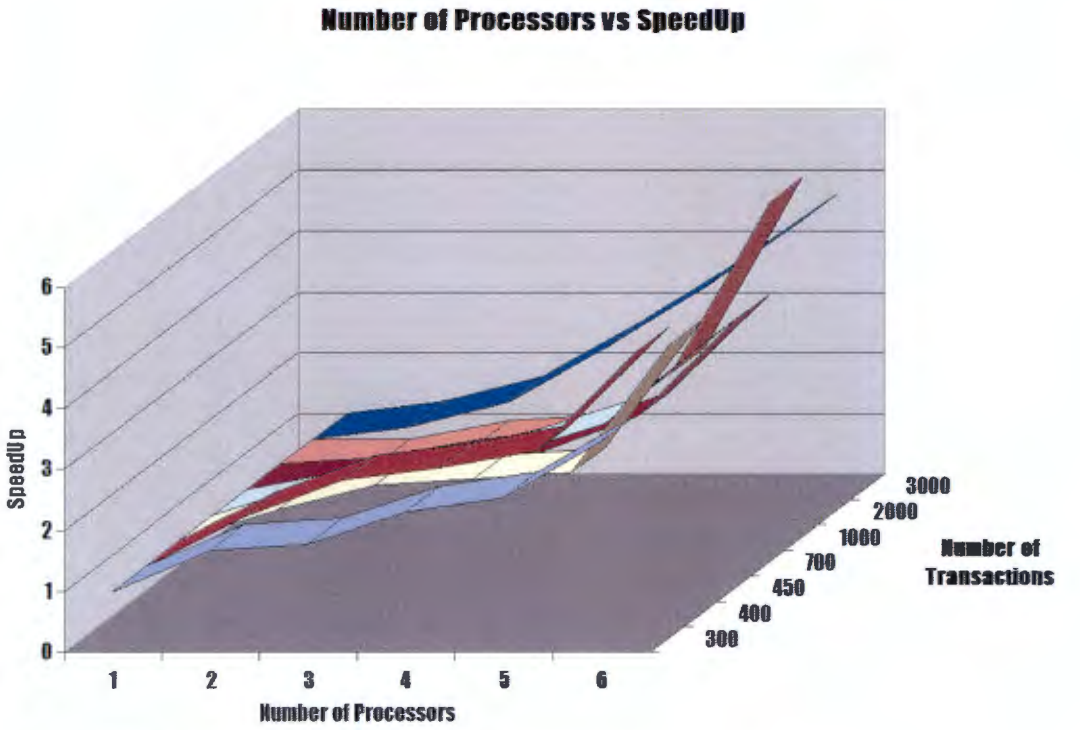

Figure 4.23: SpeedUp Analysis

SizeUp refers to analysis of the number of cores in a constant system and that it tends to grow the datasets by a factor of ***n*** [73]. In general sizeup measures how long it takes for a given system when datasets grow n times. Sizeup can be calculated using the following formula:

$$\textbf{SizeUp} = \frac{T_n}{T_1} \quad \text{.....(8)}$$

Where T_n is the execution time for processing n* data, where n is a multiple data transaction, and T_1 is the execution time for processing data with the least number of transaction or the smallest size.. For example If Tn is the time it takes to execute 500 transactions then T_1 is the time it will takes to process 250 transactions on similar processor quantities.

Figure 4.7 presents sizeup results of different processors working on different sizes of datasets. When a single processor is used in experiment the average Sizeup is less than 2.5.When the number of processors is increase to 2 and then to 3 there is also an increase in sizeup which does

not exceed 3.5. We obtain the highest value of sizeup when the highest number of processors are utilized in this experiment which applies the fact that minimum time required to execute the job due to many tasks happening at the same time. Therefore execution time is less when more processor are used compare to when the number of processor is smaller.

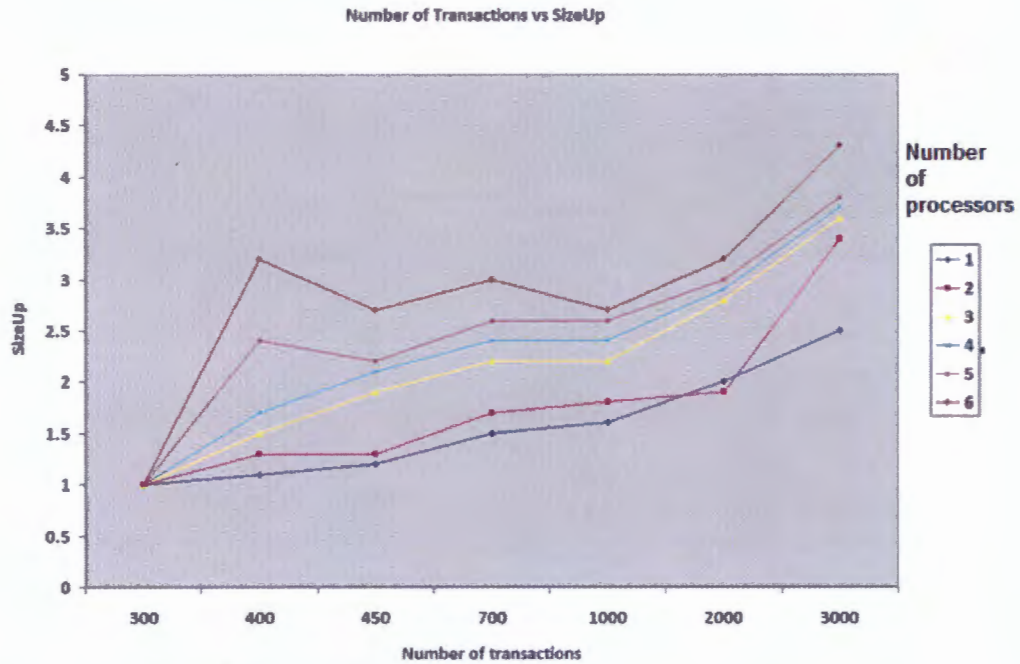

Figure 4.24: Number of Transactions vs SizeUp

4.5 Analysis and evaluation of Improved Apriori and traditional Apriori

We utilized 5 virtual processors to process different data quantities. Table 4.2 shows the results obtained from the experiment.

Table 4.2: Execution Times for Traditional Apriori

Execution Times of Traditional Apriori Algorithm	
Transaction Size	Time (seconds)
400	5.43
450	15.53
750	26 14
1000	30.41
2000	56,11
3000	72 32
3000	93 12

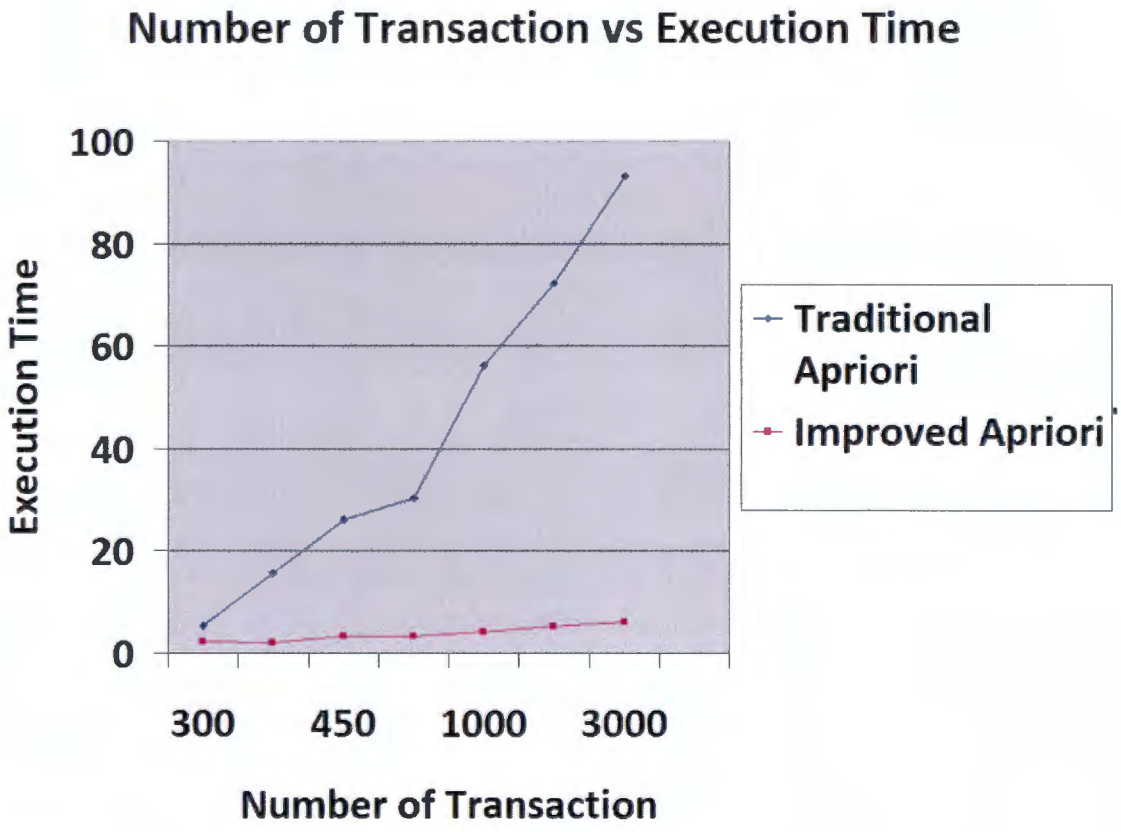

Figure 4.8: Number of Transaction vs Execution Time

Figure 4.8 shows a comparison of execution times between the traditional Apriori and the proposed Apriori. Based on the experiment results, when using improved Apriori algorithm to

process a given transaction size the execution time is less than the execution time required to process the same size of transaction with the traditional Apriori algorithm. For example, to process about 3000 transactions it takes traditional Apriori 93.13 seconds to process the data, whereas improved Apriori takes only 6.04 seconds to process the same data. Similarly, according to the Figure 4.8 execution time of 450 transactions with traditional algorithm take approximately 20seconds compared to almost 6seconds required to process 450 transactions with the modified algorithm. The improved Apriori algorithm has a steady increase on time as the number of transactions increases whilst traditional algorithm experiences a sharp increase in execution time as the number of transactions increases. From the results shown in Figure 4.8 we can conclude that when the number of transactions is small processing time for both algorithms is almost similar whereas when the transaction size increases the processing times change significantly. We therefore conclude that localized pruning, support counting and shared execution of activities proposed on improved Apriori algorithm helped enhance the performance of the algorithm.

Disk latency is the time delay between a request for data and the return of the data which is critical to the performance of a system. Figure 4.9 shows a sample presentation of a real time online experimental results carried out at North West University in South Africa using Hadoop Cloudera cloud computing resources that are freely available. The time shown on the results is based on EDT time format were America which hosts Cloudera Hadoop server. This is why the experiment appears to have been done in the early morning hours. The latency in the graph in Figure 4.9 shows that there was delay from about 03:50 AM when data was ingested into Hadoop until to 03:56AM.There is a sharp increase of disk latency when the experiment was running from 03:33 to AM to 04:00 AM. After 04:00 AM there was also a sharp drop in disk latency as the results were produced. The sharp decrease in disk latency is a good for the investigation because there is reduced delay from the reading process of data from memory to RAM. The sharp increase in latency at the beginning of the experiment was may be attributed by little RAM

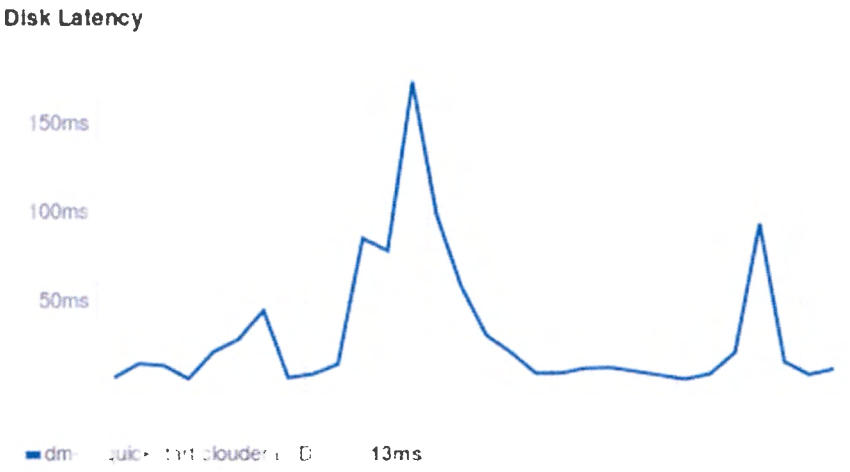

Figure 4.9: Disk Latency

Figure 4.10 shows an extraction of graphs taken from Hadoop Cluster after running some of the experiments. The first presentation of transaction shows that there was an increase in operations per second on transactions and this is so because the transactions were loaded into Hadoop around 03:50 AM and the MR processing started thereafter. Thus there is an increase of transactions being processed between 03:50 AM and 04:05 AM. As the algorithm continues to process the transactions there is a steady decrease time for processing transaction. The other three presentations show that there have been transactions loaded into MR for processing, the edit log and lastly the presentation of average edit logs based on the transactions loaded in the session of this experiment.

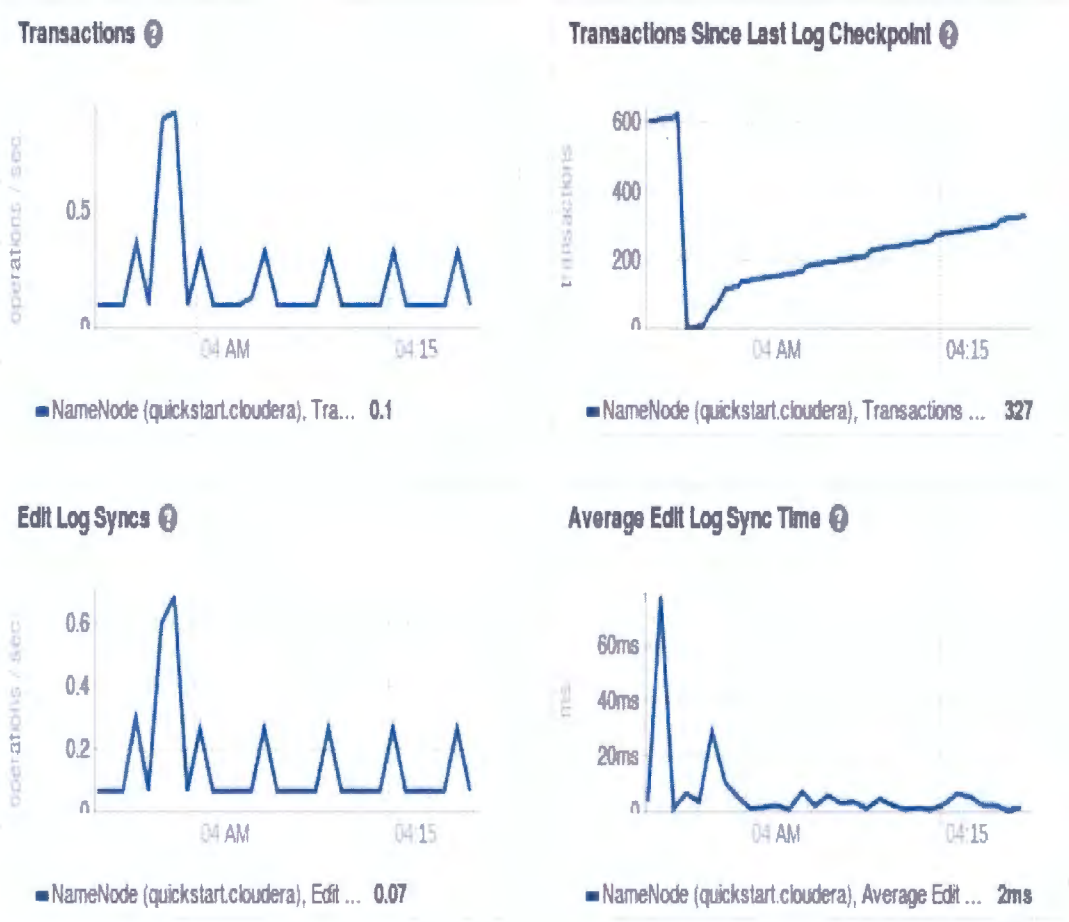

Figure 4.10: Transactions Validation

The improved Apriori algorithm is able to process large retail data transactions on MR with 16 mappers as presented on the experiments. The speed up and execution time were improved. We used speedup to evaluate the performance of the improved Apriori and it can be seen that the algorithm has been enhanced as compared to most sequential modifications of the Apriori algorithm proposed in the past. We have managed to reduce the computational and memory cost and thereby achieving our goal of mining association rules more efficiently.

This chapter presented and analyzed the experimental results. The experiment environment was explained and the implementation of the improved parallel Apriori algorithm using MR model

was discussed. We presented experimental results of the improved Apriori as well as the evaluation of its performance. The results show that there was slight improvement in the execution time, speedup and sizeup of the improved Apriori algorithm.

CHAPTER 5: CONCLUSION AND FUTURE WORK

This chapter gives conclusion of the dissertation work and presents possible guidelines on future research work.

5.1 Conclusion

Association rule mining is applicable in many areas and can also be implemented easily. There have been significant improvements and modifications on sequential Apriori algorithms [22] . There have been many proposals on parallel Apriori implementation based on serial frameworks which has proved to be currently expensive. A few studies have been conducted based on parallelizing the Apriori algorithm using MapReduce. Some researchers worked on Apriori but focused their work on its implementation on word count based on English and Arabic corpora. Though there have been significant researches on Apriori algorithm based on MapReduce, very little of that has been based on market basket analysis. The results of this research show that parallel algorithms on parallel systems provide better performance and scalability.

In our research we have improved Apriori algorithm by parallelizing all the core activities and allowing distributed pruning of infrequent itemsets at both their mappers and reducers. We also employed a hybrid implementation between in-memory and hadoop by utilizing YARN to get better execution times [21]. Based on the experimental results, we demonstrated that the improved Apriori has been enhanced.. Implementation of the improved Apriori on MR makes the algorithm scalable.

5.2 Problems encountered during implementation

Data mining is an area whose research interest is growing rapidly. The growth in research proposals in data mining or data science is mainly the need for better and efficient algorithms and mechanisms to enhance the field of research interest. It has been observed and proven experimentally over the years that advancement in microprocessor technology and embedded systems has also contributed to past achievements in data mining. Recently research in data

mining has proved that software frameworks in that area aided by complementing hardware platforms have produced better performances. Due to circumstances beyond our control we encountered some challenges during the implementation stage of the research which affected the results of the experiment. We present some of the drawbacks to our implementation as follows:

- Hadoop Cloudera minimum requirements for RAM 16GIG using Cloudera express application on trial, we managed to acquire 12GIG RAM which we used to conduct the investigation. Though we managed to install and use standard tools for data mining (Hadoop cloudera express) and complete the implementation, we had many interruptions when running java applications and hence we could not load more data since the system was complaining in terms of RAM, processors and data size ratio.
- As a result we could not fully exploit Hadoop Cloudera, a parallel platform, for processing data as some functions could not work as RAM and multiprocessor requirement were not met.
- There was an increase in disk latency during the peak processing time due to inadequacy of RAM.
- There was a limited number of processors on the computer that could grant us an efficient parallel implementation.
- Java programming was constrained by the RAM size and we could not program within the planned time because eclipse IDE would promptly shut down as RAM processing was very low thereby affecting our productivity

5.3 Future work

We used a small scale cluster with one master node based on Cloudera Express to evaluate the performance of the improved Apriori. We would want to have our master installed as a Hadoop server that can accommodate many clients/nodes so that we are able to implement a master node-slave nodes architecture. Furthermore, there is a need to fully optimize the Apriori algorithm for market basket analysis where there is no need to input a user specific minimum support and minimum confidence threshold. Further work can also be based on improving parallel and distributed Apriori algorithms based on faster pruning strategies and an efficient candidate

generation method targeting node or mapper level and at the same time devising methods to reduce communication costs.

REFERENCES

[1] J. Woo, "Market basket analysis algorithm on Map/Reduce in AWS EC2," *International Journal of Advanced Science and Technology*, vol. 46, pp. 25-38, 2012.

[2] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*: Elsevier, 2011.

[3] S. A. Abaya, "Association rule mining based on Apriori algorithm in minimizing candidate generation," *International Journal of Scientific & Engineering Research*, vol. 3, pp. 1-4, 2012.

[4] J. Han, M. Kamber, and J. Pei, "Mining Frequent Patterns, Associations, and Correlations," *Data Mining: Concepts and Techniques (2nd ed., pp. 227-283). San Francisco, USA: Morgan Kaufmann Publishers*, 2006.

[5] M. J. Zaki, W. Meira Jr, and W. Meira, *Data mining and analysis: fundamental concepts and algorithms*: Cambridge University Press, 2014.

[6] M.-S. Chen, J. Han, and P. S. Yu, "Data mining: an overview from a database perspective," *IEEE Transactions on Knowledge and data Engineering*, vol. 8, pp. 866-883, 1996.

[7] X. Y. Yang, Z. Liu, and Y. Fu, "MapReduce as a programming model for association rules algorithm on Hadoop," in *Information Sciences and Interaction Sciences (ICIS), 2010 3rd International Conference on*, 2010, pp. 99-102.

[8] J. H. Yeung, C. Tsang, K. H. Tsoi, B. S. Kwan, C. C. Cheung, A. P. Chan, *et al.*, "Map-reduce as a programming model for custom computing machines," in *Field-Programmable Custom Computing Machines, 2008. FCCM'08. 16th International Symposium on*, 2008, pp. 149-159.

[9] S. Dhanya, M. Vysaakan, and A. Mahesh, "An enhancement of the MapReduce Apriori algorithm using vertical data layout and set theory concept of intersection," *Intelligent Systems Technologies and Applications*, pp. 225-233, 2016.

[10] X. Liu and H. Liu, "An improved apriori algorithm for association rule," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 11, pp. 6521-6526, 2013.

[11] X. W. Vipin Kumar, J.Ross Quinlan,Joydeep Ghosh,Qiang Yang,Hiroshi Motoda,Geoffrey J. Mclachlan,Angus Ng,Bing Liu,Philip S.Yu,Zhi-Hua Zhou,Michael Steinbach,David J.Hand and Dan Steinberg, "Top 10 algorithms in data mining," presented at the IEEE International Conference on Data Mining (ICDM), Hong Kong, 2007.

[12] L.-J. Huang, "FP-growth Apriori algorithm's Application in the Design for Individualized Virtual Shop on the Internet," in *Machine Learning and Cybernetics, 2007 International Conference on*, 2007, pp. 3800-3804.

[13] C. Zhang and S. Zhang, *Association rule mining: models and algorithms*: Springer-Verlag, 2002.

[14] F. Masseglia, M. Teisseire, and P. Poncelet, "Sequential Pattern Mining," in *Encyclopedia of Data Warehousing and Mining*, ed: IGI Global, 2005, pp. 1028-1032.

[15] C. C. Aggarwal and J. Han, *Frequent pattern mining*: Springer, 2014.

[16] F. Kovacs and J. Illés, "Frequent itemset mining on hadoop," in *Computational Cybernetics (ICCC), 2013 IEEE 9th International Conference on*, 2013, pp. 241-245.

[17] A. Ezhilvathani and K. Raja, "Implementation of parallel apriori algorithm on hadoop cluster," *International Journal of Computer Science and Mobile Computing (Apr 2013)*, vol. 2, 2013.
[18] D. Flair. (2017). *Hadoop MapReduce Tutorial* [Online]. Available: http://data-flair.training
[19] Wikipedia. (2017). *Data mining* [Online]. Available: https://en.wikipedia.org
[20] O. R. Zaïane, "Introduction to data mining," 1999.
[21] M. Riondato, J. A. DeBrabant, R. Fonseca, and E. Upfal, "PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 85-94.
[22] Y. Chen, F. Li, and J. Fan, "Mining association rules in big data with NGEP," *Cluster Computing*, vol. 18, pp. 577-585, 2015.
[23] A. Rathod, M. A. Dhabariya, and C. Thacker, "A Survey on Association Rule Mining for Market Basket Analysis and Apriori Algorithm," *International Journal of Research in Advent Technology*, vol. 2, 2014.
[24] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI magazine*, vol. 17, p. 37, 1996.
[25] R. Agrawal and J. C. Shafer, "Parallel mining of association rules," *IEEE Transactions on knowledge and Data Engineering*, vol. 8, pp. 962-969, 1996.
[26] M. Kantardzic, *Data mining: concepts, models, methods, and algorithms*: John Wiley & Sons, 2011.
[27] C. C. Aggarwal and C. Zhai, *Mining text data*: Springer Science & Business Media, 2012.
[28] C. C. Aggarwal, *Data mining: the textbook*: Springer, 2015.
[29] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou, "Re-optimizing data-parallel computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012, pp. 21-21.
[30] C. Anshu and C. Raghuvanshi, "An algorithm for frequent pattern mining based on apriori," *IJCSE) International Journal on Computer Science and Engineering*, vol. 2, pp. 942-947, 2010.
[31] M. H. Awadalla and S. G. El-Far, "Aggregate function based enhanced apriori algorithm for mining association rules," *International Journal of Computer Science Issues*, vol. 9, pp. 277-287, 2012.
[32] C.-F. Tsai and M.-Y. Chen, "Variable selection by association rules for customer churn prediction of multimedia on demand," *Expert Systems with Applications*, vol. 37, pp. 2006-2015, 2010.
[33] H. M. Najadat, M. Al-Maolegi, and B. Arkok, "An improved Apriori algorithm for association rules," *International Research Journal of Computer Science and Application*, vol. 1, pp. 01-08, 2013.
[34] S. S. Lodhi, S. Rawat, and P. Arya, "Apriori Based Novel Frequent Itemset Mining Mechanism," *International Journal*, vol. 1, 2012.
[35] M. Al-Maolegi and B. Arkok, "An improved apriori algorithm for association rules," *arXiv preprint arXiv:1403.3948*, 2014.
[36] G. P. Quinn and M. J. Keough, *Experimental design and data analysis for biologists*: Cambridge University Press, 2002.

[37] Z. K. Baker and V. K. Prasanna, "Efficient parallel data mining with the Apriori algorithm on FPGAs," in *Submitted to the IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, 2005.
[38] S. S. Bagul, "Survey on Parallel Algorithms," 2013.
[39] R. Srikant, Q. Vu, and R. Agrawal, "Mining association rules with item constraints," in *KDD*, 1997, pp. 67-73.
[40] S. Rao and P. Gupta, "Implementing Improved Algorithm Over APRIORI Data Mining Association Rule Algorithm 1," 2012.
[41] S. Chaudhari, M. Borkhatariya, A. Churi, and M. Bhonsle, "Implementation and Analysis of Improved Apriori Algorithm," *databases,* vol. 5, 2016.
[42] J. Singh, H. Ram, and D. J. Sodhi, "Improving efficiency of apriori algorithm using transaction reduction," *International Journal of Scientific and Research Publications,* vol. 3, pp. 1-4, 2013.
[43] K. Zhang, J. Liu, Y. Chai, J. Zhou, and Y. Li, "A Method to Optimize Apriori Algorithm for Frequent Items Mining," in *Computational Intelligence and Design (ISCID), 2014 Seventh International Symposium on*, 2014, pp. 71-75.
[44] P. Mandave and M. Mane, "Data mining using Association rule based on APRIORI algorithm and improved approach with illustration," *International Journal of Latest Trends in Engineering and Technology (IJLTET), ISSN,* 2012.
[45] E. Frank, M. Hall, G. Holmes, R. Kirkby, B. Pfahringer, I. H. Witten, *et al.*, "Weka-a machine learning workbench for data mining," in *Data mining and knowledge discovery handbook*, ed: Springer, 2009, pp. 1269-1277.
[46] D. Goswami, A. Chaturvedi, and C. Raghuvanshi, "Frequent pattern mining using record filter approach," *IJCSI International Journal of Computer Science Issues,* vol. 7, 2010.
[47] Y. Ye and C.-C. Chiang, "A parallel apriori algorithm for frequent itemsets mining," in *Software Engineering Research, Management and Applications, 2006. Fourth International Conference on*, 2006, pp. 87-94.
[48] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*: Morgan Kaufmann, 2016.
[49] F. Jiang and C. K. Leung, "A data analytic algorithm for managing, querying, and processing uncertain big data in cloud environments," *Algorithms,* vol. 8, pp. 1175-1194, 2015.
[50] N. Baddal and S. Bagga, "Implementation of Apriori Algorithm in MATLAB using Attribute Affinity Matrix," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 4, pp. 10-15, 2013.
[51] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM,* vol. 51, pp. 107-113, 2008.
[52] N. Gowraj, S. Avireddy, and S. Prabhu, "Palm: Preprocessed apriori for logical matching using map reduce algorithm," *International Journal on Computer Science and Engineering,* vol. 4, p. 1289, 2012.
[53] N. Li, L. Zeng, Q. He, and Z. Shi, "Parallel implementation of apriori algorithm based on mapreduce," in *Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on*, 2012, pp. 236-241.
[54] H. Apache. (2017). *Welcome to Apache™ Hadoop®!* [Online]. Available: http://hadoop.apache.org

[55] T. White, *Hadoop: The definitive guide*: " O'Reilly Media, Inc.", 2012.
[56] G. Klockwood. (2017). *Conceptual Overview of Map-Reduce and Hadoop* [Online]. Available: http://www.glennklockwood.com
[57] S. Zhao and R. Du, "Distributed Apriori in Hadoop MapReduce Framework," ed, 2013.
[58] M.-Y. Lin, P.-Y. Lee, and S.-C. Hsueh, "Apriori-based frequent itemset mining algorithms on MapReduce," in *Proceedings of the 6th international conference on ubiquitous information management and communication*, 2012, p. 76.
[59] S. Yang, G. Xu, Z. Wang, and F. Zhou, "The Parallel Improved Apriori Algorithm Research Based on Spark," in *Frontier of Computer Science and Technology (FCST), 2015 Ninth International Conference on*, 2015, pp. 354-359.
[60] S. Yang, "Research and Application of Improved Apriori Algorithm to Electronic Commerce," pp. 227-231.
[61] Y. Li and R. P. Gopalan, "Effective Sampling for Mining Association Rules," in *Australian Conference on Artificial Intelligence*, 2004, pp. 391-401.
[62] M. N. Saxena, M. N. Bhargava, and M. U. Mahor, "A Competent Technique on Cluster Based Master Slave Structural Design," *International Journal*, vol. 1, 2012.
[63] E. Frank, M. Hall, G. Holmes, R. Kirkby, B. Pfahringer, I. H. Witten, *et al.*, "Weka," *Data Mining and Knowledge Discovery Handbook*, pp. 1305-1314, 2005.
[64] R. Phelps, M. Krasnicki, R. A. Rutenbar, L. R. Carley, and J. R. Hellums, "Anaconda: simulation-based synthesis of analog circuits via stochastic pattern search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 703-717, 2000.
[65] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, and V. S. Tseng, "SPMF: a Java open-source pattern mining library," *Journal of Machine Learning Research*, vol. 15, pp. 3389-3393, 2014.
[66] B. Hedlund. (2017). *Understanding Hadoop Clusters and the Network* [Online]. Available: http://bradhedlund.com
[67] H. Liu and H. Motoda, *Feature selection for knowledge discovery and data mining* vol. 454: Springer Science & Business Media, 2012.
[68] MapReduce. (2017). *MapReduce Products* [Online]. Available: https://mapr.com
[69] M. R. Raval, I. J. Rajput, and V. Gupta, "Survey on several improved Apriori algorithms," *IOSR Journal of Computer Engineering*, pp. 57-61, 2013.
[70] J. R. Nayak and D. J. Cook, "Approximate Association Rule Mining," in *FLAIRS Conference*, 2001, pp. 259-263.
[71] Available. (2017). *Hadoop MapReduce* [Online]. Available: https://www.tutorialspoint.com
[72] F. Zhang, M. Liu, F. Gui, W. Shen, A. Shami, and Y. Ma, "A distributed frequent itemset mining algorithm using Spark for Big Data analytics," *Cluster Computing*, vol. 18, pp. 1493-1501, 2015.
[73] D. Nandakumar and N. Yambem, "A Survey on Data Mining Algorithms on Apache Hadoop Platform," *International Journal of Emerging Technology and Advanced Engineering*, vol. 4, pp. 563-565, 2014.
[74] J. S. Yoo, D. Boulware, and D. Kimmey, "A parallel spatial co-location mining algorithm based on MapReduce," in *Big Data (BigData Congress), 2014 IEEE International Congress on*, 2014, pp. 25-31.

[75] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "MapReduce in the Clouds for Science," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 565-572.

[76] J. Dean and S. Ghemawat, "MapReduce: a flexible data processing tool," *Communications of the ACM,* vol. 53, pp. 72-77, 2010.

[77] C. Fan, F. Xiao, and C. Yan, "A framework for knowledge discovery in massive building automation data and it's application in building diagnostics," *Automation in Construction,* vol. 50, pp. 81-90, 2015.

[78] Darshan M. Tank, "Improved Apriori Algorithm for Mining Association Rules", International Journal of Information Technology and Computer Science, Vol. 6, pp.15-23, No. 7,June 2014, doi:10.5815/ijitcs.2014.07.03.