نموذج رقم (1)

إقـــــرار

أنا الموقع أدناه مقدم الرسالة التي تحمل العنوان:

عنوان رسالة Apriori للبيانات العربية باستخدام نموذج MapReduce

Apriori Algorithm for Arabic Data Using MapReduce

أقر بأن ما اشتملت عليه هذه الرسالة إنما هو نتاج جهدي الخاص، باستثناء ما تمت الإشارة إليه حيثما ورد، وإن هذه الرسالة ككل أو أي جزء منها لم يقدم من قبل لنيل درجة أو لقب علمي أو بحثي لدى أي مؤسسة تعليمية أو بحثية أخرى.

DECLARATION

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification

Student's name: اسم الطالب: خلف عبد الناصر الحضرمي

Signature: التوقيع:

Date: التاريخ: 10/10/2015

The Islamic University – Gaza

Denary of Higher Studies

Faculty of Information Technology

الجامعة الاسلامية – غزة

عمـــادة الدراســـات العليـــــــا

كليـــة تكنولوجيـــا المعلومـــات

Apriori Algorithm for Arabic Data Using MapReduce

Submitted By:

Ola Abed El-nasser El-khoudary

Supervised By:

Dr. Rebhi Baraka

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master in Information Technology

July, 2015 – Shawal , 1436H

بسم الله الرحمن الرحيم

الجامعة الإسلامية – غزة
The Islamic University - Gaza

مكتب نائب الرئيس للبحث العلمي والدراسات العليا هاتف داخلي: 1150

الرقم ج.س.غ/35/..... Ref

التاريخ 2015/08/19م..... Date

نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة شئون البحث العلمي والدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحثة/ علا عبدالناصر حسن الخضري لنيل درجة الماجستير في كلية *تكنولوجيا المعلومات* برنامج تكنولوجيا المعلومات وموضوعها:

خوارزمية Apriori للبيانات العربية باستخدام نموذج MapReduce

Apriori Algorithm for Arabic Data Using MapReduce

وبعد المناقشة التي تمت اليوم الأربعاء 04 ذو القعدة 1436هـ، الموافق2015/08/19م الساعة الثانية مساءً، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

د. ربحي سليمان بركة	مشرفاً و رئيساً	.....
أ.د. علاء مصطفى الهليس	مناقشاً داخلياً	.....
د. سناء وفا الصايغ	مناقشاً خارجياً	.....

وبعد المداولة أوصت اللجنة بمنح الباحثة درجة الماجستير في كلية *تكنولوجيا المعلومات*/ برنامج تكنولوجيا المعلومات.

واللجنة إذ تمنحها هذه الدرجة فإنها توصيها بتقوى الله ولزوم طاعته وأن تسخر علمها في خدمة دينها ووطنها.

والله ولي التوفيق،،،

نائب الرئيس لشئون البحث العلمي والدراسات العليا

أ.د. عبدالرؤوف على المناعمة

P.O. Box 108, Rimal, Gaza, Palestine fax: +970 (8) 286 0800 فاكس Tel: +970 (8) 286 0700 ص.ب. 108 الرمال، غزة، فلسطين هاتف
public@iugaza.edu.ps www.iugaza.edu.ps

Abstract

Aprioi is the most popular algorithm that is used to extract frequent itemsets from large data sets where these frequent itemsets can be used to generate association rules. Such rules are used as a basis for discovering knowledge such as detecting unknown relationships and producing results which can be used for decision making and prediction.

When the data size is very large, both memory use and computational cost are very expensive. And in this case single processor's memory and CPU resources are very limited which make the algorithm performance inefficient. Parallel and distributed computing is effective for improving algorithm performance.

In our research we propose a parallel Apriori approach for large volume of Arabic text document using MapReduce with enhanced speedup and performance, Apriori algorithm that has been popular to collect the itemsets frequently occurred in order to compose Association Rule, MapReduce is a scalable data processing tool that enables to process a massive volume of data in parallel.

The experiments show that the parallel Apriori approach can process large volume of Arabic text efficiently on a MapReduce with 16 computers, which can significantly improve the execution time and speedup and also generate strong association rules.

Keywords: *Apriori, frequent itemset, Association Rule, MapReduce and Hadoop.*

الملخص

عنوان البحث: خوارزمية Apriori واستخدامها في النص العربي باستخدام نموذج MapReduce

تعتبر خوارزمية Apriori من اشهر الخوارزميات التي تستخدم لاستخراج مجموعة من البنود المتكررة (frequent itsemsets)هذه البنود تستخدم للحصول على قواعد (Association rules) لاكتشاف المعرفة. هذه القواعد تساعدنا في التنبؤ بالعلاقات غير المعروفة والحصول على النتائج التي تساعد في التنبؤ واتخاذ القرار.

عندما يكون حجم البيانات كبير فيعتبر استخدام كل من الذاكرة والحسابات للمعالجات التقليدية مكلفة جدا بالإضافة الى ذلك فان ذاكرة المعالج الواحد ومصادره محدودة جدا وهذا يؤدي ضعف في اداء الخوارزمية. لذلك تعتبر الحوسبة المتوازية والموزعة فعالة جدا في تحسين اداء الخوارزمية.

في هذا البحث قمنا باقتراح خوارزمية Apriori باستخدام نموذج الحوسبة المتوازية MapReduce مع تعزيز التسريع (speedup) والأداء (performance), خوارزمية Apriori الاكثر شهرة تستخدم في جمع البنود (itemsets) التي تظهر تكرارا وتستخدم هذه البنود من اجل تكوين قواعد (Association rules), يعتبر نموذج الحوسبة المتوازية (MapReduce) اداة لمعالجة البيانات حيث تستطيع معالجة كم ضخم من البيانات بالطريقة المتوازية.

اظهرت النتائج ان تطبيق خوارزمية Apriori المتوازية المقترحة تعالج بكفاءة النصوص العربية ذات الحجم الكبير حيث اجريت التجارب على نموذج MapReduce مكون من 16 جهاز حاسوب واظهرت النتائج تحسنا كبيرا على التسريع والاداء وايضا الحصول على قواعد قوية.

Dedication

To my beloved mother and father...

To my husband and children Lilia and Naya...

To my sisters and brothers...

To my best friends...

Acknowledgments

Thanks to Allah for giving me the power and ability to complete this thesis.

Many thanks and sincere gratefulness goes to my supervisor Eng. Dr. Rebhi S. Baraka without his help, guidanc and continuous follow-up, this research would never have been.

Also I would like to extend my thanks to the academic staff of the Faculty of Information Technology at the Islamic University-Gaza who helped me during my Master's study and taught me different courses.

Last but not least, I am greatly indebted to my family for their love and support.

Table Of Contents

List of Figures

List of Algorithms

List of Tables

List of Abbreviations

AFS	Apriori for Frequent Sub path
ARM	Association Rule Mining
CA	Classical Arabic
DA	Dialectal Arabic
FP	Frequent Pattern
HDFS	Hadoop Distributed File System
KNN	K-Nearest Neighbors
LHS	Left Hand Side
MPI	Message Passing Interface
OSAC	Open Source Arabic Corpus
RHS	Right Hand Side

Chapter 1 Introduction

Text mining has introduced tools and techniques to extract interesting patterns from large data. Apriori algorithm is the most classical and important algorithm for mining frequent itemsets. Frequent patterns, are patterns that frequently appear in a data collection. Itemsets, subsequences, or substructures are different terms used to refer to patterns of different kinds.

Text mining concerns looking for patterns in unstructured text. The purpose of Text mining is to process unstructured (textual) information, extract meaningful numeric indices from the text and make the information contained in the text accessible to the various data mining algorithms. Information can be extracted to derive summaries for the words contained in the documents or to compute summaries for the documents based on the words contained in them [1].

Finding frequent itemsets is one of the most important fields of data mining. Apriori algorithm is the most established algorithm for finding frequent itemsets from dataset; however, it needs to scan the dataset many times and to generate many candidate itemsets [2]. Association Mining is one of the most important data mining's functionalities and is the most popular technique that has been studied by researchers [3]. Mining of frequent itemsets is an important phase in association mining which discovers frequent itemsets in data sets.

MapReduce is a scalable programming model. The programmer writes two functions a map function and a reduce function each of the functions

define a mapping from one set of key-value pairs to another [4]. The map function takes an input as key/value and produces a set of intermediate key/values. It groups all the intermediate values associated with same key and passes them to the reduce function. The reduce function takes an intermediate key and a set of values for the key and merges all values together to form smaller set of values [5].

One widely used implementation of MapReduce is Apache Hadoop [6] which is a collection of related services that compose an infrastructure for distributed computing. Hadoop is known for MapReduce and its Hadoop Distributed File System HDFS [7], it provides complementary services, such as Core, MapReduce, HDFS, and HBase. Hadoop MapReduce is distributed data processing model and execution environment that runs on large clusters of commodity machines.

One of the most important advantages of MapReduce is that it provides an abstraction that hides many system-level details from the programmer. Therefore, the developer can focus on what computations need to be performed as opposed to how those computations are actually carried out or how to get the data to the processes that depend on them [8]. MapReduce provides a means to distribute computation without burdening the programmer with the details of distributed computing [9]. Because of the benefits of MapReduce model we use it as parallel programming model.

The majority of the research works conducted on generating frequent itemsets is mainly related to English corpuses and little works have focused on Arabic data. Thus, this work investigate the problem of generating frequent itemsets from Arabic data using Apriori algorithm. We prefer Apriori

algorithm over other algorithms as it's the simplest and we have recently a good work on parallel Apriori algorithm so we can overcome the drawbacks.

We build a MapReduce-based parallel Apriori approach for large scale of Arabic text to generate frequent itemsets and used this itemsetse to generate association rules and also increase the algorithm performance by implementing the algorithm in parallel.

To build our approach, we collect a large volume of Arabic corpus and perform several preprocessing steps to prepare the corpus for applying Apriori. After that we design the parallel Apriori. We conduct several experiments to apply the parallel Aprior approach.

1.1 Statement of the Problem

Apriori algorithm for the generation of frequent itemsets in text mining, although simple, suffers from performance bottleneck when used with large data sets. There have been several attempts to improve its performance through parallelization, but none of these attempts have considered the attractive MapReduce programming model and Arabic data sets.

Therefore there is a need to develop a high performance Apriori approach over MapReduce programming model to process large Arabic data and generate frequent itemsets which can be used for generating association rules.

1.2 Objectives

1.2.1 Main objectives

The main objective of this research is to develop an efficient parallel Apriori approach over MapReduce to generate frequent itemsets and use these frequent itemsets for generating association rules from Arabic data.

1.2.2 Specific objectives

- Designing the suitable MapReduce computing model for parallel Apriori.
- Determining and collecting an Arabic corpus which is suitable for Apriori algorithm.
- Implementing the algorithm based on the designed model.
- Executing and testing the algorithm using the collected Arabic corpus.
- Evaluating the performance of the algorithm based on time, speedup.

1.3 Scope and limitations

The work is conducted with the following limitations and assumptions:

1- We use Apache Hadoop framework to build the clusters.
2- Experiments are conducted on a set of processors and their own exclusive memory (multicomputer cluster).
3- Some text preprocessing is performed using RapidMiner.
4- The MapReduce text preprocessing execution time is not considered.
5- We use 2, 4, 8, 12 and 16 processors to conduct the experiments and to measure the effects on the speedup and the execution time of proposed approach.
6- The strength of association rules is based on the minimum support and confidence threshold which is specify during the experiments.

1.4 Importance of the Thesis

1- Improve the efficiency of Apriori algorithm using MapReduce in terms of time, resources, and cost.
2- Generate association rules from Arabic data that can be used for detecting unknown relationships which can be used for decision making and prediction.
3- Improve the Apriori algorithm to naturally fit with the MapReduce programming model to benefit from the high scalability of MapReduce applications for Arabic data which is not exist.
4- Overcome the issue low performance for the sequential Apriori algorithm due to the large amount of computational power.

1.5 Methodology

The methodology to be followed in order to complete this research and achieve its objectives is illustrated in Figure 1.1 and consists of the following phases:

- **Research and Survey**: this includes reviewing the recent literature closely related to the thesis problem statement. After analyzing the existing methods, identifying the drawbacks or the lack of existing approaches, we formulate the solution to overcome the drawbacks.

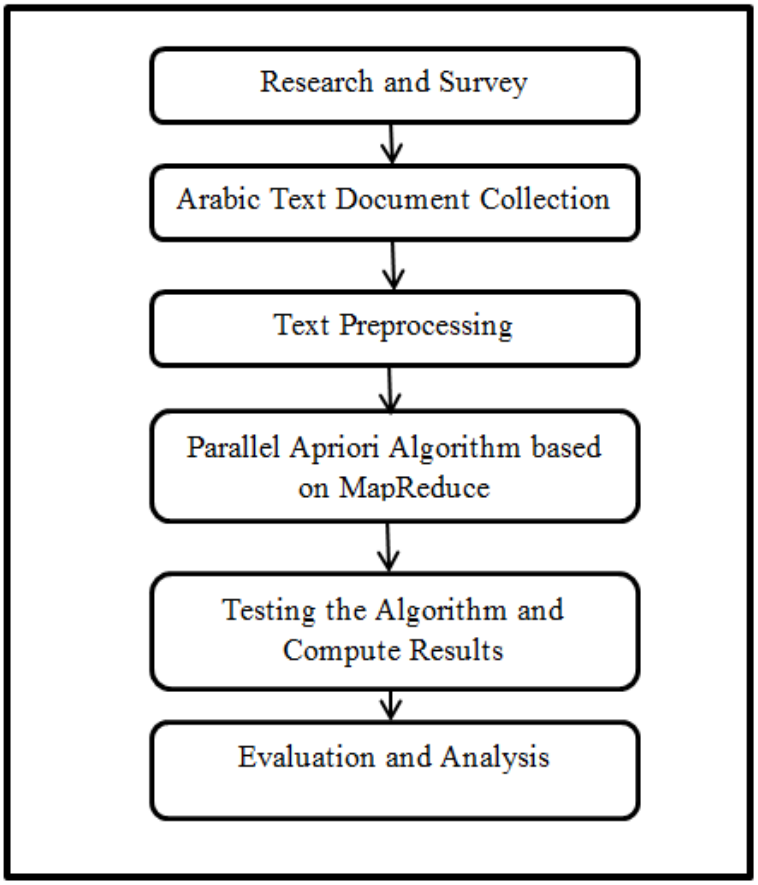


Figure 1.1 : Stages of the Research Methodology

- **Data Collection:** we collect largest freely public Arabic corpus of text documents with multiple domains.
- **Text Preprocessing:** some preprocessing in Arabic corpus is performed. It includes applying stop words removal, tokenizing string to words and applying suitable term stemming.
- **Design the Parallel Apriori Approach:** we build the parallel Apriori algorithm for large volume Arabic text based on MapReduce model.
- **Implementation of the algorithm:** we implement the proposed approach using Java programming language, and Hadoop platform with

a multicomputer cluster on the largest freely public Arabic corpus of text documents.

- **Evaluation:** the proposed approach is evaluated for speedup and execution time using different performance metrics.
- **Results and Discussion:** In this stage we analyze the obtained results and justify the effectiveness of the proposed approach.

1.6 Research Format

The thesis is organized as follows: Chapter 2 presents and discusses the state of the art and related works. In Chapter 3 includes the theoretical foundation of the research. Chapter 4 presents the proposed parallel Apriori approach. Chapter 5 presents the experimental results and evaluation. Finally, Chapter 6 presents the conclusions and future work.

Chapter 2 Related work

In this chapter, we give an overview to approaches related to the main topic of this thesis. The first section presents the Apriori algorithm for association mining and the works conducted in improving the performance of the Apriori in sequential and in parallel using MPI, the second section includes the MapReduce as a parallel programming model which is used for data processing across massive data sets. It also discuss the hadoop as a programming framework that support processing of large data.

2.1 Apriori Algorithm for Association Mining

Association mining is one of the most important data mining's functionalities and it is the most popular technique. Aprioi algorithm which used for finding frequent item set and use this item to generate association rules. The benefits of these rules are detecting unknown relationships, producing results which can be used for decision making and prediction. There are numerous of researches and projects that exploit Apriori algorithm and how to improve its efficiency.

Zoghby [10]This work introduces a new system developed to discover soft-matching association rules using a similarity measurements based on the derivation feature of the Arabic language. In addition, it presents the features of using Frequent Closed Item-sets (FCI) concept in mining the association rules rather than Frequent Itemsets (FI).

Najadat and Maolegi et al. [11] indicate the limitations of the original Apriori algorithm of wasting time for scanning the whole database searching

on the frequent itemsets, and presents an improvement on Apriori by wasted time depending on scanning only some transactions. They showed by experimental results that applied on the original Apriori and the improved Apriori that the new Apriori reduces the time consumed by 67.38% in comparison with the original Apriori, and makes the Apriori algorithm more efficient and less time consuming.

The results show that the improved Apriori algorithm that scan only some transactions instead of the whole database reduce the consumed time.

Rao and Gupta. [12] present a new scheme for finding the rules out of transactional datasets which improve the original Apriori in terms number of database scans, memory consumption, and the interestingness of the rules. It also avoids scanning the database again and again. So, they use Frequent Pattern (FP) Growth ARM (Association rule mining) algorithm that is more efficient to mine patterns when database grows.

Ali [8] propose a method for finding frequent patterns using AFS (Apriori for Frequent Sub path) and use this method to extract frequent patterns from Quran. using a graph, and apply a frequent sub-path mining algorithm on it to generate frequent patterns. The basic idea is to represent Quranic text using a complete graph, and apply an efficient frequent pattern mining algorithm which can take advantage of the graph representation.

The previous work using Arabic data (Quran) to generate frequent pattern, the algorithm showed that frequent patterns generated by the algorithm cluster similar or identical verses.

Urmila. [13] presents features of parallel algorithm for mining association rules using MPI for message passing based on the Master-Slave based structural model to achieve high-performance parallel computing, the algorithm is implemented using Master-Slave structure and communicate by MPI between the hosts, this make full use of the resources, a unified scheduling, coordination of treatment, under the cluster environment.

The previous work showed that is parallelization is the most suitable solution to efficient data mining.

Lal and Mahanti.[14] propose an algorithm to mine data from cloud using sector/sphere framework, Sector/Sphere is an open source software suite for high-performance distributed data storage and processing, Sector is a distributed file system targeting data storage over a large number of commodity computers. Sphere is the programming framework that supports massive in-storage parallel data processing for data stored in Sector. The algorithm can avoid redundant rules, the performance of the algorithm improved when compared with the existing algorithms.

Liu and Liu. [2] build a knowledge system established based on the analysis of classical Apriori. The improved algorithm adopted the matrix to express the database it scans the database once and eliminate a huge number of linking operations in finding frequent item sets. The experimental results showed that the new algorithm is improved both the efficiency and the performance.

Bayardo and Roberto [15] propose the Max-Miner algorithm for extracting the maximal of frequent itemset .the previous algorithms based on Apriori scale exponentially with longest pattern length. The idea is to expand sets over

an ordered and finite item domain where four items are denoted by their position. The particular ordering imposed on the item domain affects the parent child relationships in the set-enumeration tree but not its completeness experiments on real data showed that when the patterns are long, the new algorithm is more efficient.

Zoghby et al. [10] presents a system that uses new similarity measures and a modified association rule generation to discover soft-matching rules from Arabic textual databases automatically constructed from document corpora. The system induces accurate predictive rules. The excellence of the soft- matching over the hard exact matching is clear in the system results. They use the algorithm called CHARM. It only prunes closed itemset lattice rather than all itemsets lattice. CHARM can reduce both the number of database passes and CPU overhead incurred by the frequent itemset search as well as the efficiency and effectiveness of the rules.

The works presented above shows the importance of Apriori algorithm and the same time shows its main limitation, namely, the slowness of the sequential version of the algorithm which we try to overcome.

Urmila [13] implement Apriori using MPI they showed that parallelization is the most suitable solution to increase the performance of Apriori algorithm. In my work I used MapReduce over MPI.

Ali using AFS (Apriori for Frequent Sub path) to extract frequent patterns from Quran. using graph but the implementation is in sequential not parallel. In my thesis I used Arabic text but in parallel. Also (Zoghby) present the CHARM algorithm to produce soft-matching rules from Arabic textual databases also the implementation in serial nor parallel.

2.2 MapReduce Based Parallel Algorithms for Large Data Processing

MapReduce is a programming model and an associated implementation for processing large data sets. User specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

Woo. [16] presents Apriori algorithm that runs on paralle MapReduce framework namely Apache Hadoop over a cluster of computers. The author also explains the time complexity which theoretically shows that the algorithm have higher performance than the sequential algorithm. The item sets produced by the algorithm can be used to compute and produce association rules for market analysis. But the algorithm needs to be built and generate experimental result to prove that the proposed algorithm works.

Mahendra and Deepika [17] discusses an algorithm to mine the data from the cloud using sector/sphere framework with association rules they also discussed the integration of Sector/Sphere framework and Association rule. This enables the application of association rule algorithm to the wide range of cloud services available on the web. Sphere allow developers to write certain distributed data parallel applications with several simple APIs. A Sphere database consists of one or more physical files. Computation in sphere is done by user-define function. the result can be written to either the local disk or common destination files on other nodes.

Hegazy. [18]implements an efficient MapReduce Apriori algorithm based on Hadoop-MapReduce model which needs only two phase to find all frequent itemsets, they also compared the new algorithm with two existed algorithms which need either one or k phases to find the same frequent itemsets. Experimental results showed that the proposed Apriori algorithm is efficient and exceed the other two algorithms.

Qureshi and Bansal [19] used association rule mining as a data mining technique they have improved the Apriori algorithm to suit it for parallel computation platform. Using Amazon's web services namely EC2, S3 and EMR for cloud computing, The proposed algorithm will reduce the execution time for lower values of support count, the authors didn't mention or explain the algorithm also the result wasn't clear .

Dean and Ghemwat. [5] implement a MapReduce system that runs on a large cluster of commodity machines and is highly scalable. The author added some optimizations in the implemented system that aims to reduce the amount of data sent across the network. The system gives us a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

HaLee and Choi. [20] aim to assist the database and open source communities in understanding various technical aspects of the MapReduce framework. They discuss MapReduce framework pros and cons. Also they introduce its optimization strategies and discuss challenges raised on parallel data analysis with MapReduce. They found that MapReduce is simple but provides good scalability and fault-tolerance for massive data processing.

Yang and Dasdan [21] aim to improve the MapReduce framework by adding a Merge phase, so that it is more efficient and easier to process data relationships among heterogeneous datasets. Also they extend the MapReduce framework to the Map-Reduce-Merge framework. It also adds a new Merge phase that can join reduced outputs. They found that Map-Reduce-Merge model added to MapReduce's many features. I also contains several configurable components that enable many data-processing patterns

Agrawal and Srikant. [22] design and implement a MapReduce simulator to simulate the behavior of algorithms based on discrete event simulation which accurately simulate the hadoop environment. The simulation allows us to measure scalability of MapReduce application. The evaluation results show high level of accuracy from different aspect.

AbuTair and Baraka. [1]propose a parallel learning algorithm based on the KNN algorithm. They evaluated the parallel implementation on a multicomputer cluster that contains of 14 computers, using C++ programming language and the MPI library. They used OSAC Arabic corpus collected from multiple websites, the corpus includes 22,428 text documents with different categories (History, Sports, Health, Low, Stories, Economics, Education, Cooking Recipes). The corpus contains 18,183,511 words.

AbuShab and Baraka. [23] propose a MapReduce-based classification approach for large scale Arabic text based on Naïve Bayes Algorithm that reduces time and achieves enhanced accuracy. Also the algorithm can process large amount of Arabic text efficiently and significantly improve the speedup, and the classification results show that the proposed parallel classifier has achieved accuracy, precision, and F-measure with around 97%.

From the previous work on MapReduce the results show that using MapReduce as a parallel programming model improve the performance of the algorithm.

2.3 Summary

In this chapter, we presented a review of existing works closely related to our research and identified some drawbacks of existing approaches. From the previous works there is a needed work to enhance the performance of Apriori by implementing it in parallel using MapReduce, but there is limited work in Arabic data.

In the next chapter, we present the theoretical foundation underlying our research.

Chapter 3 Theoretical Foundation

In this chapter, the fundamental concepts which represent the basis for understanding and conducting our research are presented. First, association rules mining are presented followed with Apriori algorithm and introducing related performance metrics. Then text preprocessing is explained since it is a prerequisite for any text mining task. Since we are using MapReduce model as the underlying computing model, it will be presented with its programming model and its realization Apache Hadoop and its related Hadoop Distributed File System (HDFS).

3.1 Association Rules Mining

Association rule mining, one of the most important and well researched techniques. It aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items indata repositories. Association rules are widely used in various areas such as telecommunication networks, market and risk management, inventory control [24].

In a database of transactions D with a set of n binary attributes (items) I, a rule is defined as an implication of the form

$$X ==> Y \text{ where } X, Y \subseteq I \text{ and } X \cap Y = \emptyset.$$

There are two important basic measures for association rules, support (s) and confidence (c). Since the database is large and users concern about only those frequently items, usually thresholds of support and confidence are pre-defined by users to drop rules that are not interesting or useful. The two thresholds are called minimum support and minimum confidence [24].

Agrawal. [22] defined association rules as the implication rules that inform the user about items most likely to occur in some transactions of a database. They are advantageous to use because they are simple, intuitive and do not make assumptions of any models. Their mining requires satisfying a user-specified minimum support and a user-specified minimum confidence from a given database at the same time.

Support (s) of an association rule is defined as the percentage of records that contain **X** ∪**Y** to the total number of records in the database. The count for each item is increased by one every time the item is encountered in different transaction T in database D during the scanning process. Support(s) is calculated by the following [24].

$$\text{Support (XY)} = \frac{Support\ count\ of\ XY}{Total\ number\ of\ transaction\ in\ D}$$

Confidence (c) of an association rule is defined as the percentage of the number of transactions that contain **X** ∪**Y** to the total number of records that contain X, where if the percentage exceeds the threshold of confidence an interesting association rule **X ==>Y** can be generated [24], Confidence is a measure of strength of the association rules.

$$\text{Confidence (X|Y)} = \frac{Support\ (XY)}{Support(X)}$$

3.2 Apriori Algorithm

Apriori is an algorithm that has been proposed in [25]. The discovery of frequent itemsets is accomplished in several iterations. In each scan, a full scan of training data is required to count new candidate itemsets from frequent itemsets already found in the previous step. Apriori uses the Apriori property to improve the efficiency of the search process by reducing the size of the candidate itemsets list for each iteration. The Apriori property saysthat every sub (k-1)-itemsets of the frequent k-itemsets must be frequent.

Input: *database D, Mini Support ϵ , Mini Confidence ε*

Output: *Rt All association rules*

Method:

1- L1 = large 1-itemsets;

2- for(k=2; $L_{k-1} \neq \emptyset$; k++) do begin

3- Ck =apriori-gen(L_{k-1}); //generate new candidates from L_{k-1}

4- for all transactions $T \in D$ do begin

5- Ct=subset(C_k,T); //candidates contained in T.

6- for all candidates $C \in C_t$ do

7- Count(C)=Count(C)+1; // increase support count of C by 1

8- end

9- $L_k=\{C \in C_t \mid Count(C) \geq \epsilon \times |D|\}$

10- end

11- $L_f = \bigcup_k L_k$

12 Rt=GenerateRules(L_f , ε)

Figure 3.1 Pseudo Code for Apriori Algorithm [24]

The Apriori algorithm for finding frequent itemsets is shown in Figure 3.1, is used to produce C_k, and discarding all itemsets in C_n that do not pass the support threshold. Once these candidate itemsets are identified from C_k, then their supports are incremented. The rule generated that satisfy minimum confidence.

3.2.1 Performance Metrics for the parallel Apriori

In order to demonstrate the effectiveness of parallel processing for a problem on some platform, several concepts have been defined. These concepts will be used in later chapters to evaluate the effectiveness of parallel programs. These include speedup, scalability, and efficiency. They will be used in later to evaluate the effectiveness of our proposed parallel Apriori. Also the association rules generated must satisfy minimum support and confidence.

- **Speed up**

A Standard metric to measure the efficiency of a parallel algorithm is the speed up factor [26]. It's defined as the ratio of the time required to solve a problem on a single processor to the time required to solve the same problem on a parallel computers [27], It's defined as:

$$Sn= ts / tp$$

where ts is the execution time using only one processor and tp is the execution time using n processor.

3.3 Arabic Text Preprocessing

Arabic Language is a widely used language in the world Arabic Language it's the 5th, It is spoken by more than 422 million people as a first language . Arabic language has three forms; Classical Arabic (CA), Modern Standard Arabic (MSA), and Dialectal Arabic (DA). Arabic alphabet consists of the following 28 letters (ا، ب، ت، ث، ج، ح، خ، د، ذ، ر، ز، س، ش، ص، ض، ط، ظ، ع، غ، ف، ق، ك، ل، م، ن، ه، و، ي، أ) . The orientation of writing in Arabic is from right to left [28].

Some preprocessing in the corpus is performed. It includes tokenizing string to words, normalizing the tokenized words, applying stop word removal, applying the suitable term stemming.

- **String Tokenization**

String tokenization is the process of breaking a stream of text into words, phrases, symbols, or other meaningful elements called tokens. this tokens becomes input for further processing such text mining [29].

Text documents contains white spaces, punctuation marks, and a number of mark-ups that indicate font changes, and special characters. The aim of tokenization phase is to detect and isolate the individual words by eliminating these additional components.

- **Stopwords Removal**

Stopwords are terms or words that are too frequent in the text. These terms are insignificant. So, we can remove them to reduce the space of the items significantly.

- **Stemming**

Word stemming is an important feature supported by present day indexing and search systems. Stemming algorithm is a computational process that gathers all words that share the same stem and has some semantic relation [30]. The main idea is to improve recall by automatic handling of word endings by reducing the words to their word roots, at the time of indexing and searching. Recall in increased without compromising on the precision of the documents fetched. Stemming is usually done by removing any attached suffixes and prefixes (affixes) from index terms before the actual assignment of the term to the Index.

Stemming is needed in many application such information retrieval systems, natural language processing and compression of data [31].

Many stemmers have been developed for English and other languages. Most of the major stemming techniques in Arabic are:

1- The affix removal process is mainly achieved before the stemming process as one of the pre-processing steps. However, there are many conflicts in the literature if this process is necessary or not.
2- Surface-based stemmers that comprise from at least two morphemes as stated in [32]:
3- Root-based stemmers: the main goal of this type of stemmers is to separate the root of a specific surface word. The prefixes and suffixes are removed and they are followed by the extraction of root. The residual stem is then compared with the similar patterns and length to extirpate the root.

4- Algorithmic Light Stemmers which eliminate a few number of suffixes and prefixes without dealing with recognize patterns or infixes, and find roots that listed in [32] and [33].
5- Simple Stemmers are considered as types of Light stemmers using them the infixed vowels ا , و, ء,ي are removed from variant patterns as concluded in [33].

3.4 MapReduce Programing Model

MapReduce is one of the earliest and best known models in parallel and distributed computing area, created by Google in 2004, based on C++ language. It is a programming model and associated implementation for processing and generating large data sets in a massively parallel and distributed manner [9]. It is composed of two functions to specify, "Map" and "Reduce". They are both defined to process data structured in (key, value) pairs. The advantages of MapReduce is simple and easy to use, it's doesn't have any dependency on data model and schema, high scalability, highly fault-tolerant because each node in the cluster is expected to report back periodically with completed work and status updates [20].

- **MapReduce Architecture**

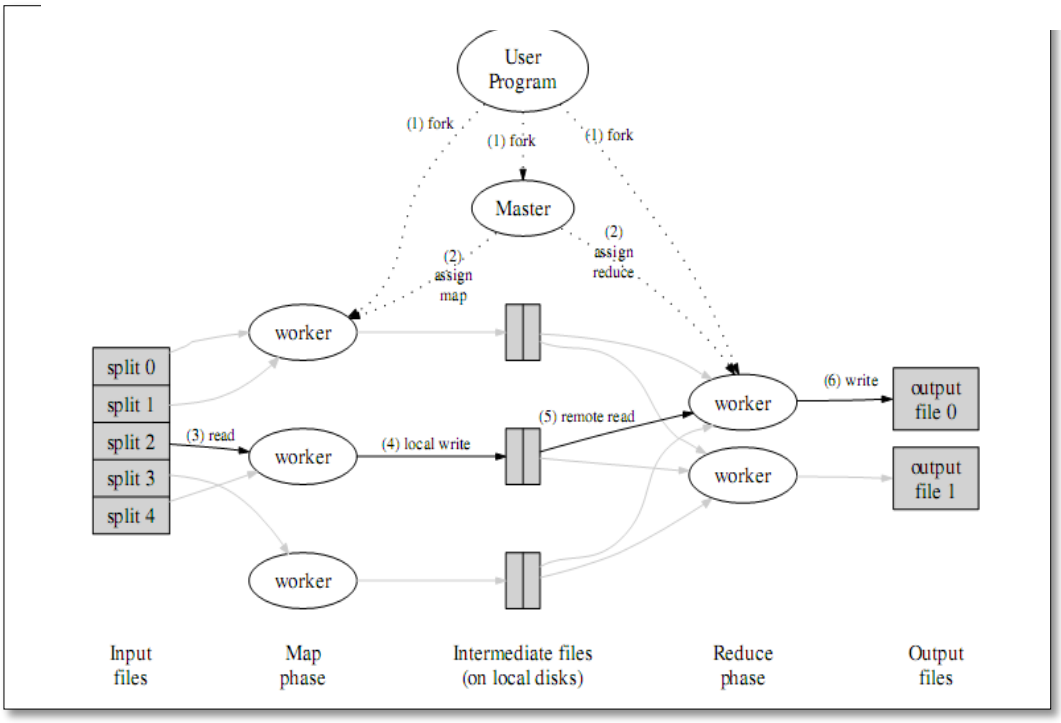
The MapReduce is consists of two primitive functions: Map and Reduce. The input for MapReduce is a list of (key1, value1) pairs and Map() is applied to each pair to compute intermediate key-value pairs, (key2, value2). The intermediate key-value pairs are then grouped together on the key equality basis [34].

The Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits. The input processed in parallel by different machines. Reduce invocations are distributed by partitioning the intermediate key into R pieces using a partitioning function. The number of partitions and the partitioning function are specified by the user. Figure 1 shows the overall flow of a MapReduce operation. When the user calls the MapReduce function the following actions occurred in sequence:

1- The MapReduce library splite the input files into N pieces with size between 16 MB to 64 MB.
2- It starts up many copies of the program on a cluster of machines. One is the master and the others is workers, master assigns map and reduce tasks to the workers.
3- A worker with a map task read the contents of it's input and pass it to the Map function as key/value pairs. The output of the Map function is buffered to a memory.
4- Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. Then the locations of these buffered pairs on the local disk are passed back to the master to forward these locations to the reduce workers.

5- When a reduce worker is notified about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers.

Figure 3.2 MapReduce Execution overview [5]



6- When a reduce worker read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together.

7- The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is appended to a final output file for this reduce partition.

8- Finally when all map tasks and reduce tasks have been finished their job, the master wakes up the user program to return the result

- **Fault Tolerance**

Since the MapReduce library is designed to help process very large amounts of data using hundreds or thousands of machines, the library must tolerate machine failures gracefully. When running jobs on a large cluster where individual nodes or network components may experience high rates of failure, MapRecuce can guide jobs toward a successful completion.

- **Worker Failure**

The master pings every worker periodically. If there is no response received from a worker in a certain amount of time, the master marks the worker as failed. Any map tasks completed by the worker are reset back to their initial idle state, and therefore become available for scheduling on other workers. Although, any map task or reduce task in progress on a failed worker is also reset to idle and becomes available for rescheduling.

Completed map tasks are re-executed on a failure because their output is stored on the local disk of the failed machine and the output is not accessible. Completed reduce tasks do not need to be re-executed because their output is stored in a global file system.

- **Master Failure**

The master write periodic checkpoints of the master data structures. If the master task dies, a new copy can be started from the last check pointed state. However, given that there is only a single master, its failure is unlikely; the program aborts the MapReduce computation if the master fails.

3.5 Hadoop as MapReduce Realization

Apache Hadoop [6] is an infrastructure for distributed computing which allow us to write and run distributed processing of large-scale data sets on high performance cluster. Distributed computing is wide and varied field, but the key distinctions of Hadoop are [35]:

- Accessible: Hadoop runs on large clusters of commodity machines or on cloud computing services such as Amazon's Elastic Compute Cloud (EC2).
- Robust: Because it is intended to run on commodity hardware, Hadoop is architected with the assumption of frequent hardware malfunctions. It can gracefully handle most such failures.
- Scalable: Hadoop scales linearly to handle larger data by adding more nodes to the cluster.
- Simple: Hadoop allows users to quickly write efficient parallel code.

Hadoop's accessibility and simplicity give it an edge over writing and running large distributed programs. Even college students can quickly and cheaply create their own Hadoop cluster [36] . On the other hand, its robustness and scalability make it suitable for even the most demanding jobs at Yahoo and Facebook.

The cluster run jobs controlled by the master node, which is known as the NameNode and it's responsible for chunking the data, cloning it, sending the data to the other slave nodes (DataNode), monitoring the status of the cluster, and collecting the results [37].

- **Component of a Hadoop Cluster**

Hadoop follows a Master-Slave architecture. As mentioned earlier. In Figure 3.3 the components of a Hadoop cluster (NameNode, Secondary NameNode and the JobTracker) are running on a single machine. Usually in production clusters having more that 20-30 nodes, the daemons run on separate nodes. a file in HDFS is split into blocks and replicated across Datanodes in a Hadoop cluster [38]. You can see that the three files A, B and C have been split across with a replication factor of 3 across the different Datanodes. The nodes in the figure are:

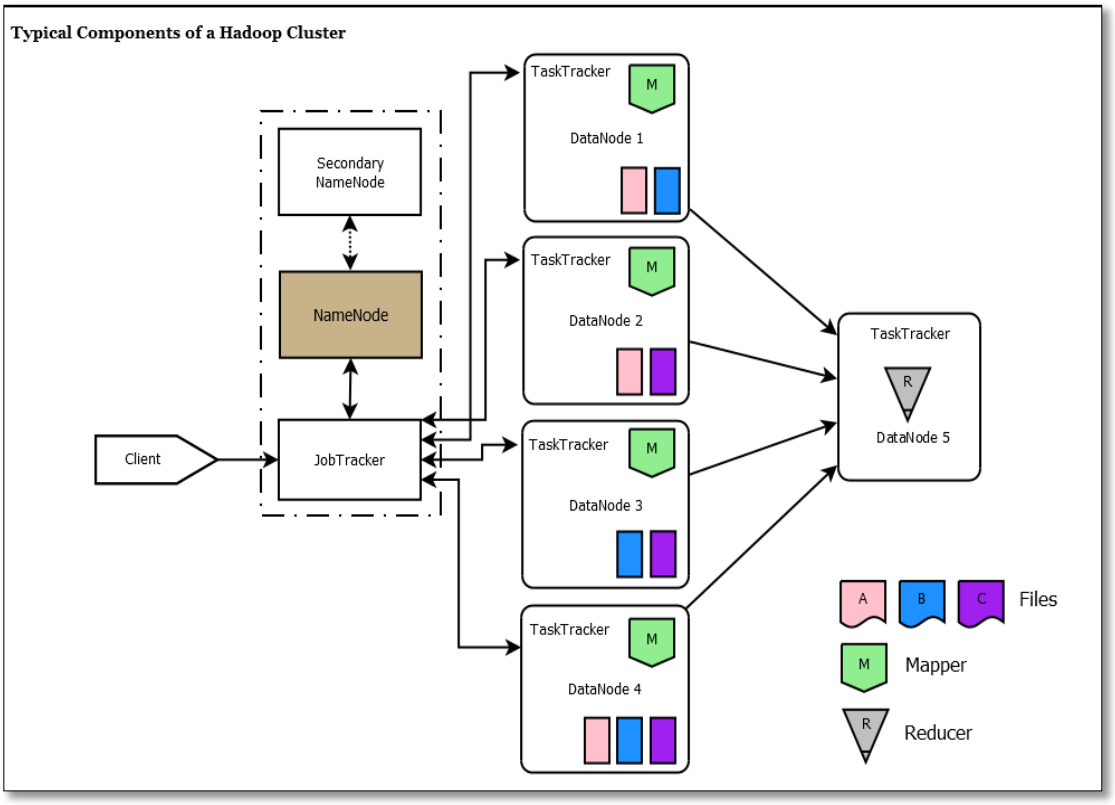


Figure 3.3: The Component of the Hadoop Cluster [35]

- **NameNode**

The NameNode in Hadoop is the node where Hadoop stores all the location information of the files in HDFS. In other words, it holds the metadata for HDFS. Whenever a file is placed in the cluster a corresponding entry of it's location is maintained by the NameNode.

- **Secondary NameNode**

The Secondary NameNode is not a failover node for the NameNode.The secondary name node is responsible for performing periodic housekeeping functions for the NameNode. It only creates checkpoints of the file system present in the NameNode.

- **DataNode**

The DataNode is responsible for storing the files in HDFS. It manages the file blocks within the node. It sends information to the NameNode about the files and blocks stored in that node and responds to the NameNode for all file system operations.

- **JobTracker**

JobTracker is responsible for taking in requests from a client and assigning TaskTrackers with tasks to be performed. The JobTracker tries to assign tasks to the TaskTracker on the DataNode where the data is locally present (Data Locality). If that is not possible it will at least try to assign tasks to TaskTrackers within the same rack. If for some reason the node fails the JobTracker assigns the task to another TaskTracker where the replica of

the data exists since the data blocks are replicated across the DataNodes. This ensures that the job does not fail even if a node fails within the cluster.

- **TaskTracker**

TaskTracker is a daemon that accepts tasks (Map,Reduce and Shuffle) from the JobTracker. The TaskTracker keeps sending a heart beat message to the JobTracker to notify that it is alive. Along with the heartbeat it also sends the free slots available within it to process tasks. TaskTracker starts and monitors the Map & Reduce Tasks and sends progress/status information back to the JobTracker.

- **Hadoop Distributed File System (HDFS)**

Hadoop Distributed File System (HDFS) [39][36,7] is a distributed file system designed for storing and supporting very large files, it provides global access to files in the cluster. For maximum portability, HDFS is implemented as a user-level files system in Java which exploits the native file system on each node. Files in HDFS are divided into large blocks, typically 64MB, and each block is stored as a separate file in the local file system. HDFS is implemented by two services: the NameNode and DataNode. The NameNode responsible for maintaining the HDFS directory tree, and is a centralized service in the cluster operating on a single node. Clients contact the NameNode in order to perform common file system operations, such as 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.

In addition to a centralized NameNode, all remaining cluster nodes provide the DataNode service. Each DataNode stores HDFS blocks on behalf of local or remote clients. Each block is saved as a separate file in the node's local file system. Because the DataNode abstracts away details of the local storage arrangement, all nodes do not have to use the same local file

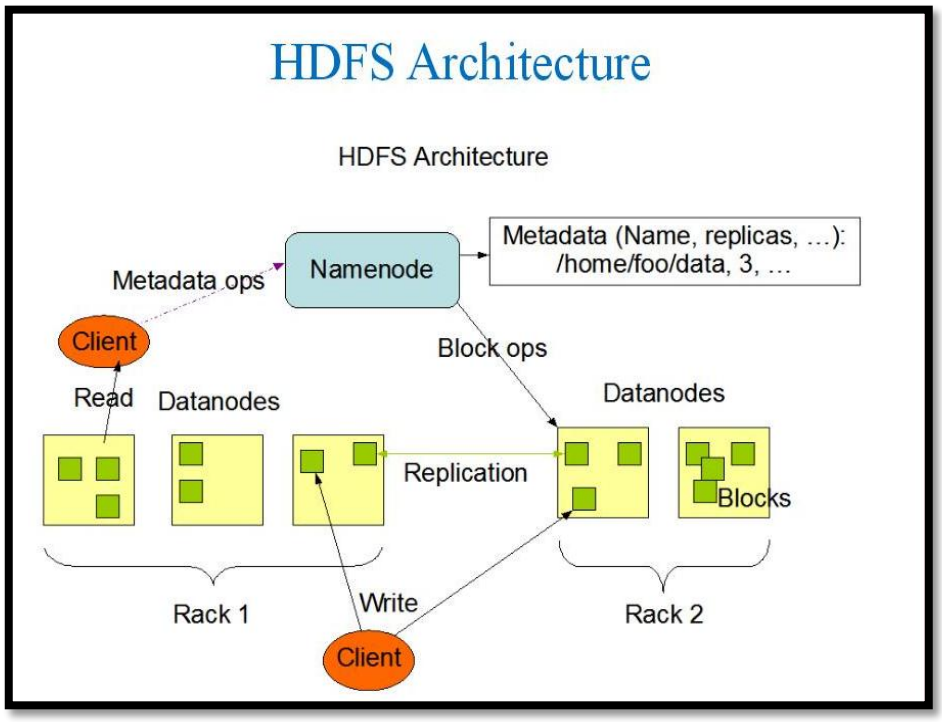


Figure 3.4: Hadoop Distributed File System Architecture [37]

system. Blocks are created or destroyed on DataNodes at the request of the NameNode, which validates and processes requests from clients. Although the NameNode manages the namespace, clients communicate directly with DataNodes in order to read or write data at the HDFS block level. Hadoop MapReduce applications use storage in a manner that is different from general-purpose computing. First, the data files accessed are large, typically tens to hundreds of gigabytes in size. Second, these files are manipulated via streaming access patterns typical of batch-processing workloads. When

reading files, large data segments (several hundred kilobytes or more) are retrieved per operation, with successive requests from the same client iterating through a file region sequentially. Similarly, files are also written in a sequential manner.

This emphasis on streaming workloads is evident in the design of HDFS. First, a simple coherence model (write-once read-many) is used that does not allow data to be modified once written. This is well suited to the streaming access pattern of target applications, and improves cluster scaling by simplifying synchronization requirements. Second, each file in 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, balancing storage resources and demand.

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. 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.

3.6 Summary

In this chapter, we presented an overview of the basic theoretical foundation that related to our research. We presented Apriori algorithm, we described performance metrics that used to evaluate the effectiveness of a parallel Apriori algorithm, MapReduce, Hadoop platform, Hadooop Distributed File System. In the next chapter, we provide a detailed description of the proposed parallel Apriori approach.

Chapter 4 The Proposed Parallel Apriori Approach

In this chapter we present the proposed parallel Apriori algorithm approach. We describe all steps of the proposed parallel Apriori using the psydocode and diagrams. We use MapReduce model to solve the problem of processing a large scale Arabic text. First, we present the steps of collecting Arabic text documents and applying text documents and applying text preprocessing. Second, we describe the steps of splitting and distributing the documents of the collected corpus as MapReduce tasks. Finally, we present the Apriori algorithm using MapReduce model.

4.1 The Overall Apriori Approach

Figure 4.1 shows the workflow of the parallel Apriori approach: the approach consists of the following phases

1- **Corpus collection and cleaning:** The corpus is collected and divided into text documents, then text preprocessing is applied to remove non-Arabic text, perform tokenization, remove Arabic stop word and perform light stemming.
2- **Text preprocessing:** some preprocessing in Arabic corpus is performed. First we applying stop words removal, tokenizing string to words and applying suitable term stemming. It is worth mentioning that the processing time is not considered as part of the performance evaluation since it is performed only once for the corpus used repetitively in the experiments.

3- **HDFS configuration, splitting and data uploading:** it's important to split the problem into sub-problem that can be executed in parallel and identify the data on which the computational performed, and then partitioning this data across various tasks.

   The task performs the computations on its own data. In our algorithm the input data (corpus) are preprocessed and transferred in to sequence of files and then we upload it to the HDFS. The HDFS splits the corpus in to 16 MB to 64 MB chunks each presented as map task and distribute them among the workers. The data replication is 3 times (by default). In addition to that there are other configuration parameter such as: document number, classes number and the documents number in each class of corpus.

   Next, steps, 4, 5 and 6 represent the core of the approach in terms of the Apriori algorithm.

4- **Generate frequent itemsets and their occurrence in each split:** in this step the frequent itemsets are generated for each split resulted from the previous step and the MapReduce model outputs the itemsets along their occurrences in the split using one map, one reduce function. This is explained in detail in Section 4.3.2.

5- **Generate frequent itemsets and their occurrence in all splits:** based on the generated itemset from the previous step, the candidate itemsets and its occurrence in the whole splits as part of Apriori algorithm are generated using one map and one reduce function. This step is detailed in Section 4.3.3.

6- **Generate strong association rules:** after generating the frequent itemsets for all the data, we generate the association rules using one

map and one reduce function with predefined minimum confidence to generate strong rules. This step is detailed in Section 4.3.4.

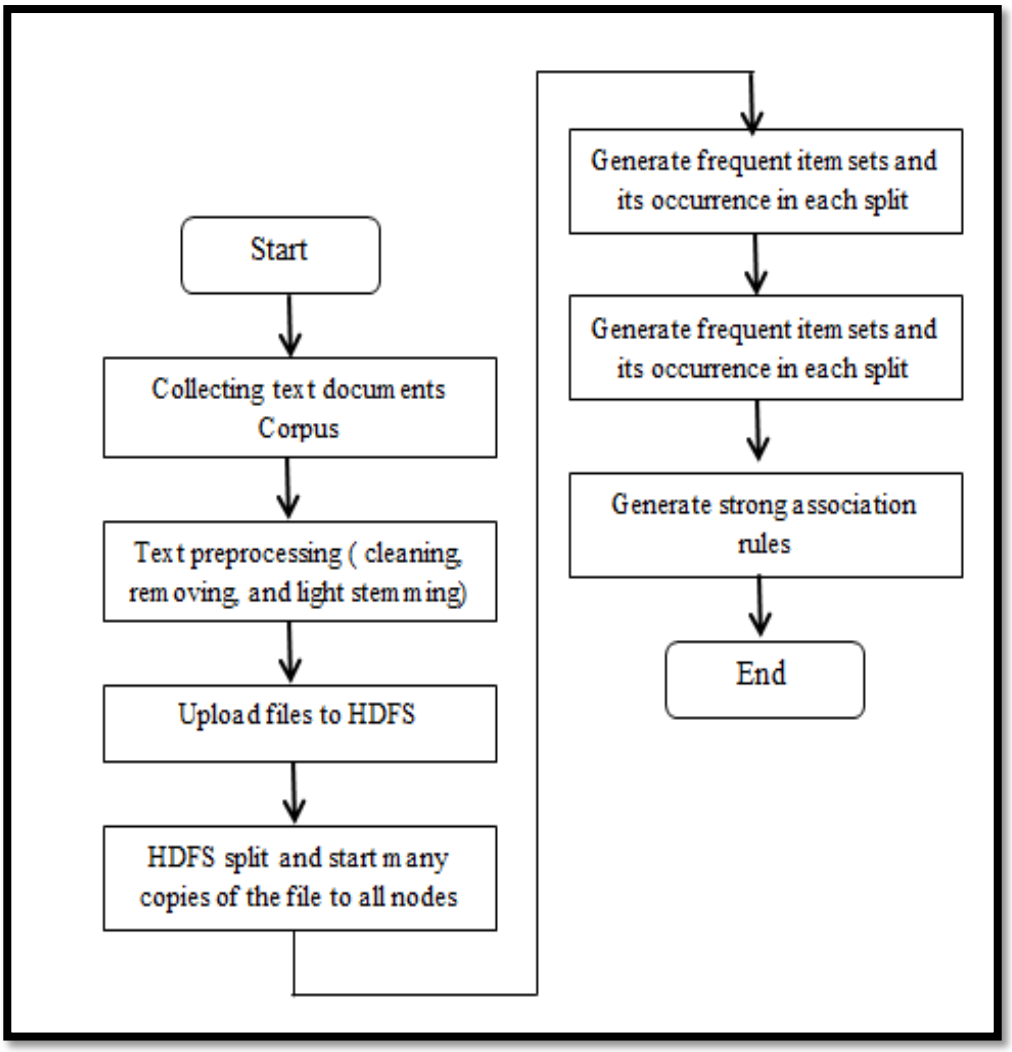


Figure 4.1 Workflow of the Proposed Approach

The above steps explains briefly the Apriori approach, next (in Section 4.2 and Section 4.3) we present each of these steps in details and how Apriori algorithm is used as a set of MapReduce functions.

4.2 Corpus Collection

Collecting the Arabic data is one of the most difficulties in this work because we want to find a large and free Arabic corpus for evaluating the parallel Apriori algorithm.

We have different datasets available for Apriori algorithm in English language while the Arabic data is fewer. The most popular Arabic text corpus used in text mining cannot satisfy our experiments data size for large-scale Arabic text corpus. So we choose to collect the real data Arabic text corpus from Shamela library [40] it contains large collection of data in different Arabic fields.

We collect the documents from Shamela library this documents needs to be processed, First compiling and labeling text documents into corpus then converting document files into text format with UTF-8 Encoding using a word to text converter (Zilla).

The Shamela corpus is categorized into eight subjects, Creed, History, Trajem, Usual, Tafsir, Sirah, Al-Hadith and Fiqh. The corpus contains 101,647 text documents with size of 5,310 MB.

Next, text preprocessing and the MapReduce parallel Apriori algorithm is described in more details.

4.3 The parallel Apriori Algorithm as a MapReduce Model

Building the parallel Apriori is the core of our approach .It includes four main phases: text preprocessing phase, generating frequent itemset for each split, generating frequent itemset for each split for all splits, and generate association rules phase. These phases are shown in Figure 4.2

As a MapReduce processing model, in the first phase two steps are conducted (i) the data divided in to (m) files (ii) the text preprocessing is performed using MapReduce computations.Each map function takes one split as input, we have also a mapper and reducer functions.

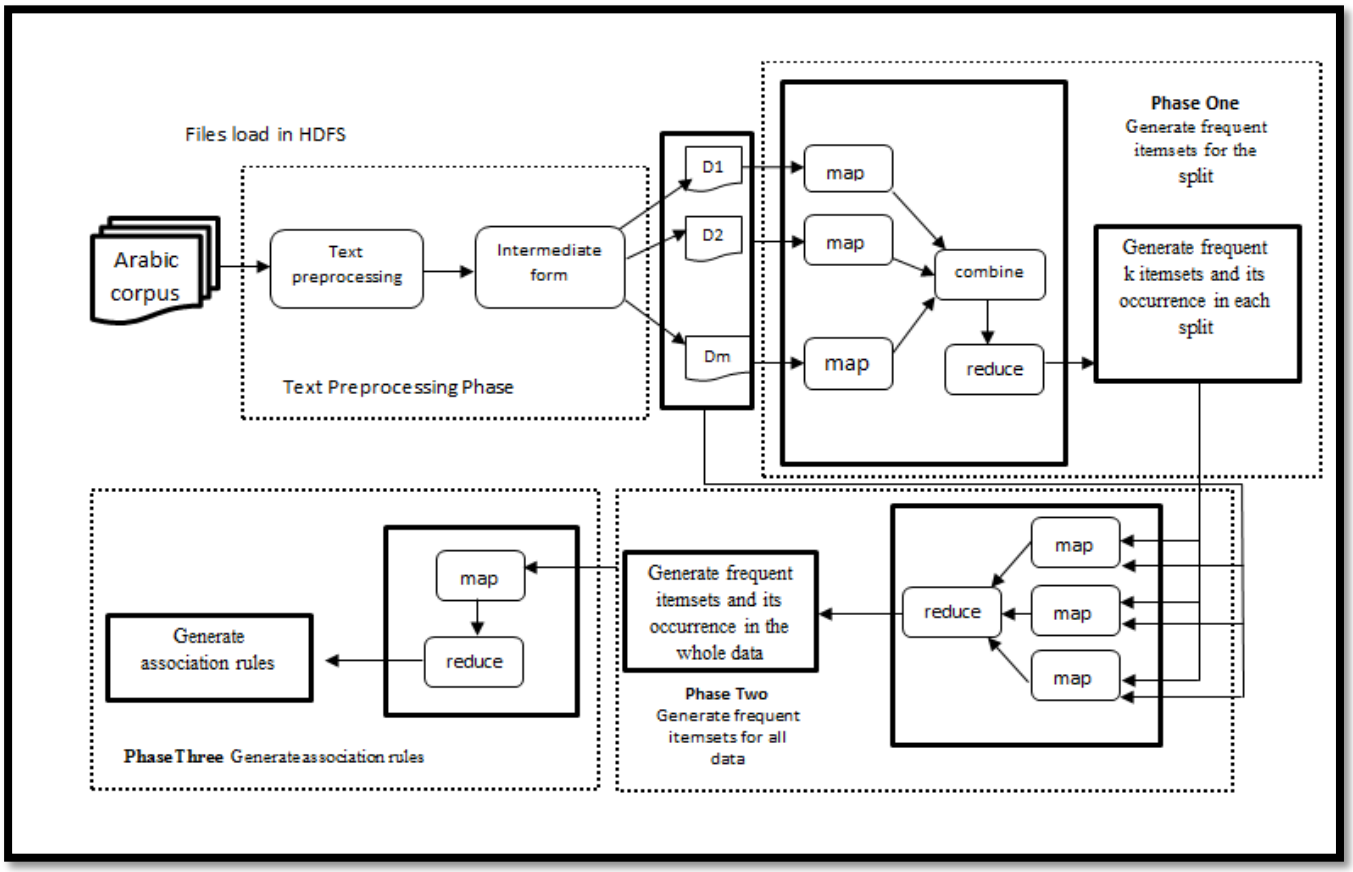


Figure 4.2 proposed parallel Apriori approach

The output of this phase is a frequent k-itemsets and their occurrence for each split as a list of intermediate key/values.

The second phase has a MapReduce computation for generating candidate frequent itemsets for all the data, the input of this process is an input split and a file contained all partial frequent k-itemsets that resulted from the first phase and the output is the frequent k-itemsets and its occurrence in the whole input data. And then used the frequent itemsets to generate the association rules.

Next we present in details these two phases and their relationships based on the proposed approach as shown in figure 5.2.

4.3.1 Text Preprocessing Phase

Applying the Apriori Approach requires usually a preprocessing stage that

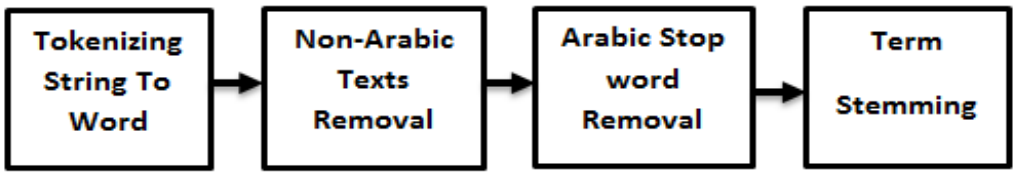


Figure 4.3: Text Preprocessing Details

would remove punctuation marks, and might returning the words to their stem or roots. Figure 4.3 shows these steps in details, they include removing non-Arabic text, tokenizing string to words, stop words removal, term stemming. These steps are details as follows:

- All the non-Arabic texts such as the digits and punctuation marks, non-Arabic letters.

- Tokenization consists of separating strings by word boundaries, we used white space tokenization because the space is the only way to separate words in Arabic language.
- Arabic stop word removal that delete tokens which is not content-bearing.
- Stemming to remove all possible affixes and reduce the word to its stem

The text preprocessing algorithm is based on [40], we covert it to parallel using MapReduce programming model as shown in (Algorithm 4.1) map phase and (Algorithm 4.2) the reduce phase.

Algorithm 4.1: The Map Phase of Text Preprocessing

Input:
Key: docnam, // one text document for each map
Value: content, //content of the document
Output:
key: docname,
Value: tokenized content, // content of the document after applying tokenization
For each line ∈ Document
Token=Tokenize (line); // tokenizing string to word
If (Token.hasMoreTokens)
docTok ← docTok ∪ Token;
end if
end for

The map phase (Algorithm 4.1) of the preprocessing algorithm takes the document as input each map function takes one document as input, this function tokenized the string to words, The aim of the tokenization phase is to detect and isolate the individual words by eliminating additional components.

For the purposes of this work, it is assumed that an Arabic word is a sequence of Arabic letters and diacritical marks without separators (space or punctuation marks).

Algorithm 4.2: The Reduce Phase of Text Preprocessing

```
Input:
Key: docname,      // one text document for each reduce
Value: content,    // content of the document
Output:
key: docname,
Value : conent,  //content of the document after preprocessing
For each line ∈  Document
  Remove non-Arabic character;
For each word  ∈  line  do
If (word  ∈  stopword.txt)
  Remove word;
end loop
  Remove prefixes;
  Remove suffixes;
If (word  ∈  similes.txt) then
   Substitute similar for word;
end loop
end loop
```

The reduce phase (Algorithm 4.2) takes the document from the map function and perform text preprocessing it includes:

- Cleaning process: to draw the list of stop words- the set of words those are deemed “irrelevant"- such as مع ,لذلك , الذي and to remove all non-Arabic elements.
- stemming is used for reducing inflected (or sometimes derived) words to their stem, base or root form.

4.3.2 Phase One: Generate Frequent Itemsets for Each Split

In this phase the data is divided into logical Input Splits, each of which is

Algorithm 4.3 The Mapper of Phase 1

Input: *split of the data Si,min-sup*
Output: *(key, value),*
key: element of frequent k-itemset,
value: the occurrence of the element,
Map(object , Si) // Map function
L1=find-frequent-1-itemsets(Si)
For(k=2; $L_{k-1} \neq \emptyset$,;k ++)
Generate new candidate C_k;
For each candidate c $\in C_k$
c.count++;
end for
L={ $c \in C_k$ | $c.count \geq min - sup$}
if itemset I $\in$ L
output (I , pcount);
end map
end

then assigned to a Map task then the map worker calls the map function to process the input split.

The Map function (Algorithm 4.3) reads one split at a time and output a list of intermediate(key , values) pairs where key is the element of the frequent itemset and the value is its occurrence. The data from the mapper is written in the temporary files in HDFS to be used by combiner.

The reducer (Algorithm 4.4) takes the inputs from the mapper and sum up the values associated with the same key, and writes the value in the output file in the increasing order of the keys. The output is a list L of (key, value) pairs where key is an element of frequent itemsets and the value equal one, this list Li is stored in a temporary file in HDFS.

Algorithm 4.4: The Reducer of Phase 1

```
Input: ( key1, value1 ),
Key1: element of frequent k-itemset,
Value1: the occurrence in each split,
Output: ( key2, 1),
Key2: element of candidate frequent k-itemsets,
Reduce( key1, value1 )    // Reduce function
Out(key2, 1);    // collected in L1
End reduce
End
```

4.3.3 Phase Two: Generate Frequent Itemsets for all Splits

In the later iteration of Map/Reduce (phase two) the Map function (Algorithm 4.5) takes in an input split and a file that contains the list of all frequent itemset (Li), This map function counts the occurrence of each element of the frequent k-itemset in the split and outputs a list of (key , value)

pairs, where key is an element of frequent k-itemset and the value is the total occurrence in the split.

Algorithm 4.5: The Mapper of Phase 2

```
Input: split of the data Si, Li (temporal file in HDFS),
Output: ( key, value ),
Key : element of the list Li,
Value : the occurrence in the split,
for each itemset I  in Li
Map(object , Si )   // Map function
count = count the occurrence of I in Si,
output (I , count);
end Map
 end for
end
```

Algorithm 4.6: The Reducer of Phase 2

```
Input: ( key1, value1 ),
Key1: element of the candidate k-itemset,
Value1: the occurrence in each split,
Output: ( key2, value2 ),
Key2: element of frequent k-itemset,
Value2: the occurrence in the whole data,
Reduce( key1, value1 )    // Reduce function
If ( value2.hasNext())
Sum+=value2.getNext();
End if
If (sum>=min-sup-count)
Out(key2, sum);    // stored in Lg (HDFS)
End if
End reduce  End
```

The reducer (Algorithm 4.6) takes the inputs from the mapper as (key1, value1) where key is an element of the candidate k-itemset and the value is its occurrence in each split. The output is a list L of (key, value) pairs where key is an element of frequent k-itemsets and the value its occurrence in the whole data.

4.3.4 Phase Three: Generate Association Rules

In this phase we generate the strong association rules using one map and one reduce function (Algorithm 4.7) , the map function takes the list of frequent itemsets and their support and takes the frequent itemset that survived the support threshold and group the entries of the same key.

Algorithm 4.7 : Map and Reduce for Generate Association Rules

***Input**: a list of all frequent itemset Lg, support suppi,*
***Output**: R set of strong associated rules*
Map Function(Lg , suppi)
for each frequent item Fi in Lg of suppi
group entries of the same key
End if
End for
Reduce Function():
for every entry i which:
calculate confidence conf = sup_i/ sup ;
if conf >= confidence threshold
R = R union (key --> conf);
End for
End

The reduce function calculate the confidence of each itemset and output the itemsets which satisfy the confidence threshold and the output will be the rule with the confidence.

4.4 Summary

In this chapter, we presented the proposed parallel Apriori approach based on MapReduce model. First we prepossess the Arabic Textual data through a MapReduce algorithm, then we used three algorithm to generate frequent itemsets for each split of the data. One for finding the frequent itemset in the input split, also we have the combiner to combine the data and the reducer to generate the frequent item set. Also we have two parallel MapReduce algorithms, one for calculate the occurrence of the frequent itemset in the split, and the other to generate the frequent itemset and their occurrence in the whole data. Finally we have a MapReduce algorithm to generate a strong association rules.

Chapter 5 Experimental Results and Evaluation

In this chapter we present and analyze the experimental results to show that our parallel Apriori algorithm can enhance execution time, speedup, and generate strong association rules in terms of support and the confidence for the association rules. The chapter includes three sections: Section 5.1 presents the corpus used in our experimentation and gives insight into the main characteristics of it. Section 5.2 describes the experimental environment and the implementation of the parallel Apriori. Finally, Section 5.3 presents and discusses the experimental results.

5.1 The Corpus

We used Shamela corpus which is the largest freely public Arabic corpus of text documents to perform our experimentations.

The Shamela Arabic corpus collected from multiple websites as presented in Table 5.1, the corpus includes 101,647 text documents. Each text document belongs to 1 of 8 categories (Creed, Usual, Fiqh, Hadith, History, Seerah, Tafsir, and trajem).

We perform all text preprocessing (Section 3.3) on the corpus. This includes non-Arabic text removal, Arabic stop word removal, stemming, and indexing.

Table 5.1 : The Shamel Corpus

Category	Number of Text Document	Size of Text Document(MB)
Hadith	23,530	1200
Trajim	14,722	784
Creed	6,776	373
Usual	2,245	128
History	9,232	488
Tafsir	18.048	973
Seerah	4,641	240
Fiqh	22,405	1180
Total	101,647	5310 MB

5.2 Experimental Setup

This section describes the experimental environment for testing our proposed approach. We implemente parallel Apriori algorithm using Java programming language. The experimental environment is built on a MapReduce cluster with 16 machines. One of these nodes is configured as Hadoop Master or as the NameNode which controls the data distribution over the Hadoop cluster and the other 15 machines acting as DataNodes. All the nodes are identical in terms of the system configuration i.e., all the nodes have identical processor - Intel Core i5 CPU with 3.20 GHz, 4.00 GB RAM, 500 GB hard disk drive and the operating system is Ubuntu 12.4 Linux with Java JDK 1.8.0, and Hadoop version 1.2.0. The computers connected through a local area network with speed of 10/100 Mbps.

HDFS splits corpus into 16 MB to 64 MB chunks each presented as a map task and then distribute them among workers with 3 replications by default, the input split includes location information for the next block and the byte offset of the data needed to complete the record. HDFS stores replicas of each data block to ensure both reliability, availability, and performance.

5.3 Implementing the Parallel Apriori over Hadoop

The proposed parallel Apriori is implemented over Hadoop distributed data processing platform as a MapReduce model that reflects the phases of the approach: text preprocessing (Section 4.3.1), generating frequent itemsets for each split (Section 4.3.2), generating frequent itemsets for all splits(Section 4.3.3), and generating association rules (Section 4.3.4). The overall results of the implementation is to increase the performance of the apriori in terms of execution time and hence speedup while generating strong association rules in terms of the confidence measure.

We follow steps in [35] for building the Hadoop cluster with Hadoop version 1.2.0. The implementation of the parallel Apriori approach using Hadoop involves the following steps:

- Step 1: All text preprocessing is performed on Shamela corpus also using MapReduce algorithm (see Section 4.3.1). It is saved as text files directories into NameNode then uploaded to HDFS which divides the input text files into data blocks of size 64 MB. The HDFS stores the metadata of each block in the NameNode and all the data blocks in the DataNode.

- Step 2: Running the Apriori algorithm over hadoop to generate the frequent itemsets. Figure 5.1 show the execution time of the parallel Apriori. The output of the Apriori is stored in file on HDFS.
- Finally, Generate the association rules using one map and reduce function. We considered the strong rules that satisfy the minimum support and confidence which we considered see (Table 5.4).

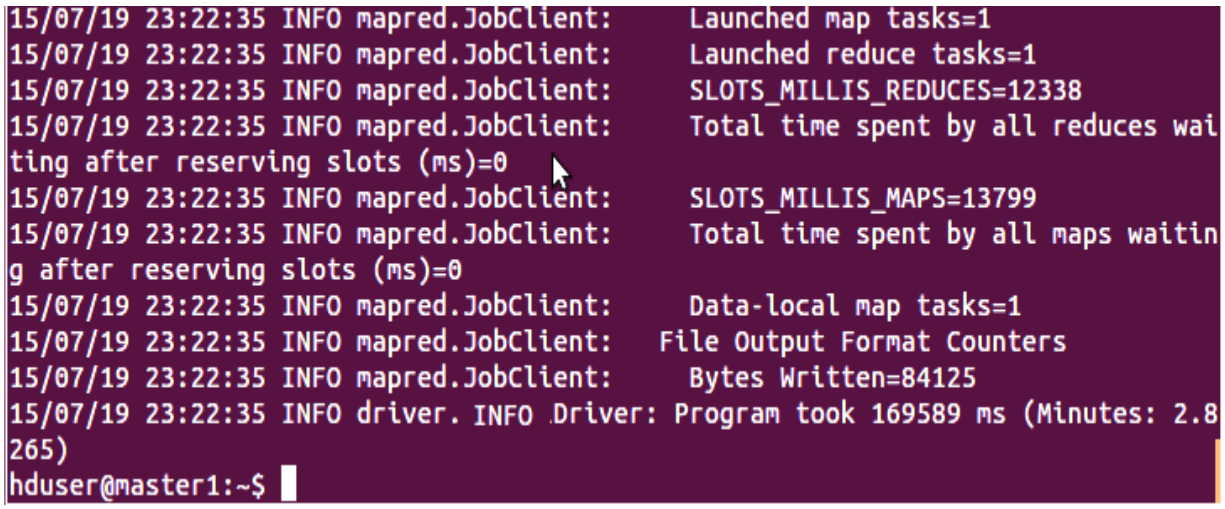


Figure 5.1: The Result of Running Parallel Apriori

5.4 Experimental Results and Evaluation

This section summarizes and discusses the results of the experiments that are conducted.

We use the collected corpus of 101,647 documents that are represented as records and 4046 words that represented attributes. We evaluate the performance of the parallel Apriori with respect to the execution time and speed up (as described in section 3.6.1).

5.4.1 Execution Time

To measure the execution time, we have executed the parallel Apriori algorithm with Support 35 on a system of clusters which varies from 2 to 16. We also have used different number of testing documents to observe the effects of different problem (documents) size on the performance. Three sets where used with the number of tested documents 5830, 10508, 20302 documents. Figure 5.1 shows a snapshot of the Hadoop run in one of the experiments, it shows total execution time of the parallel Apriori including the time of each phase except the text preprocessing. As we mentioned before text preprocessing is not part of Apriori algorithm where it is performed only once for all set of runs. The execution time here is equal to 2.8 minutes. More specific execution time values for various runs are shown in Table 5.2.

Table 5.2: The Execution Times (sec.) of One Node and Multiple Node

Problems size / No. of Nodes		5830 Documents	10508 Documents	20302 Documents
Standalone	1-Node	234.61	293.72	621.32
Parallel Apriori	2-Node	127.96	140.54	219.53
	4-Node	131.96	130.07	194.99
	8-Node	42.30	51.90	91.29
	12-Node	40.25	49.57	72.97
	16-Node	32.17	40.14	53.69

Table 5.2 shows the execution time of one node with MapReduce takes more time than the parallel version. In the parallel Apriori the execution time decreases when the number of processors increases. However, the parallel implementation achieves a good execution time compared to the standalone one. Also the execution time increases when the number of documents increases. Figure 5.1 shows the curves of the execution time based on Table 5.2.

The sequential Apriori algorithm is not appropriate for experiment, because of the large scale of documents.

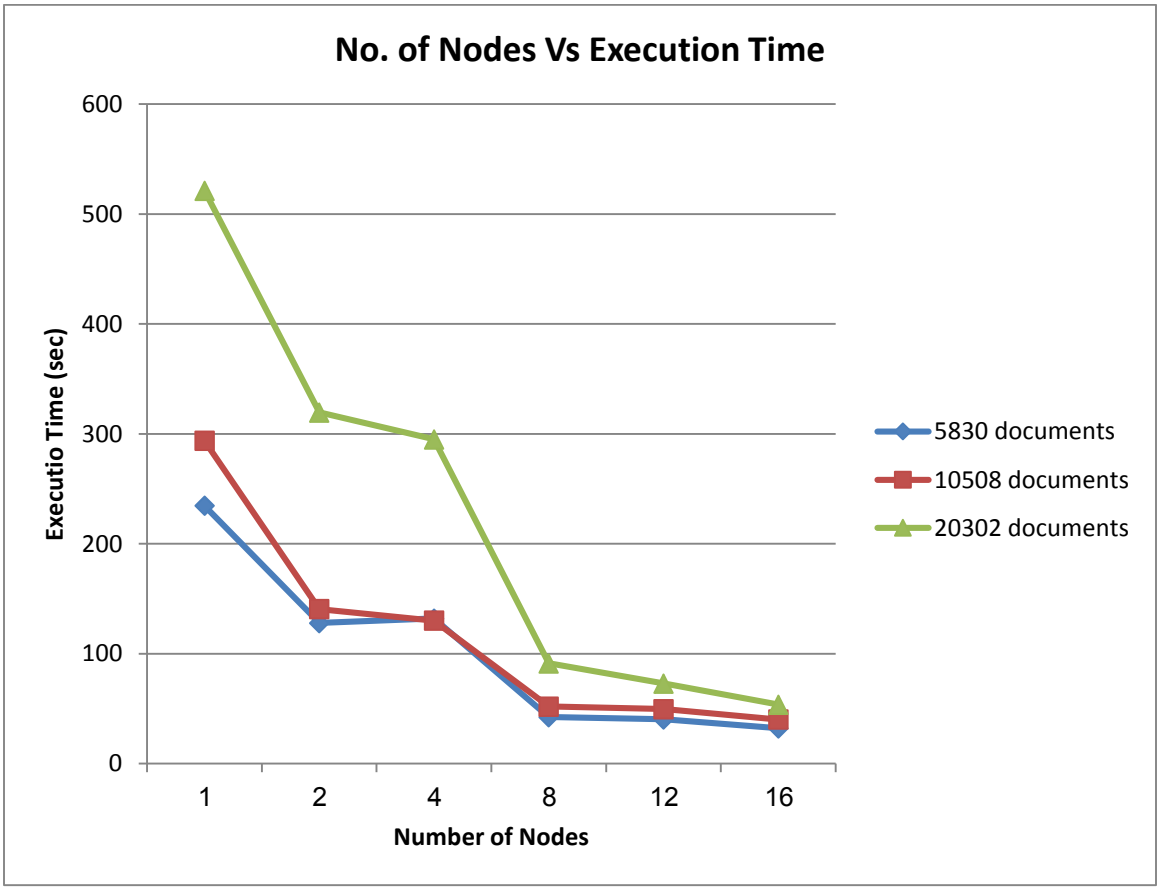


Figure 5.2 : Execution Time

5.4.2 Speedup

To compute the speedup we using the formula:

$$S_n = t_s / t_p$$

where t_s is the execution time using only one node and t_p is the execution time using n node which is gained from this parallelization as described in section . The speedup is recorded in Table 5.3 and is illustrated in Figure 5.2

The parallel algorithm demonstrates linear speed up. When running an algorithm with linear speedup, doubling the number of nodes doubles the speedup. It is difficult to achieve linear speed up due to the communication costs which increases as the number of document increases.

The time that the parallel Apriori spends does not appear to have a linear relationship with nodes, this is due to the fact that when running Hadoop jobs, starting a cluster for the first time takes some time. Also the execution time of parallel Apriori on nodes have a few changes.

Table 5.3 : The Relative Speedup of the Proposed Parallel Aprioi

Problems size / No. of Nodes	5830 Documents	10508 Documents	20302 Documents
2-Node	1.83	2.09	2.37
4-Node	1.78	2.26	2.67
8-Node	5.55	5.66	5.71
12-Node	5.83	5.93	7.14
16-Node	7.29	7.32	9.71

The results show that the Apriori algorithm have high speedup. Specifically, as the size of records increase the speedup improves. Therefore, the parallel Apriori can treat large scale Arabic text document efficiently.

The speedup improves in some cases, on the largest tested set (20302 documents), the parallel Apriori achieves relative speedups of 2.37, 2.67, 5.71, 7.14 and 9.71 on 2, 4, 8, 12 and 16 nodes respectively. When the size of the tested document is small the speedup drop from linear to sub-linear. The smallest tested documents sizes give similar results.

If we increase the number of nodes the speedup gains tend to drop. Figure 5.2 shows the speedups for the different document sets. When we used 4 nodes the speedup improve from 1.83 to 2.37, on 8 nodes it improves from 1.78 to 2.67, and on 16 nodes it improves from 7.29 to 9.71. It can be shown that our parallel Apriori algorithm gives better performance with larger volume Arabic text documents than with smaller volume Arabic text documents.

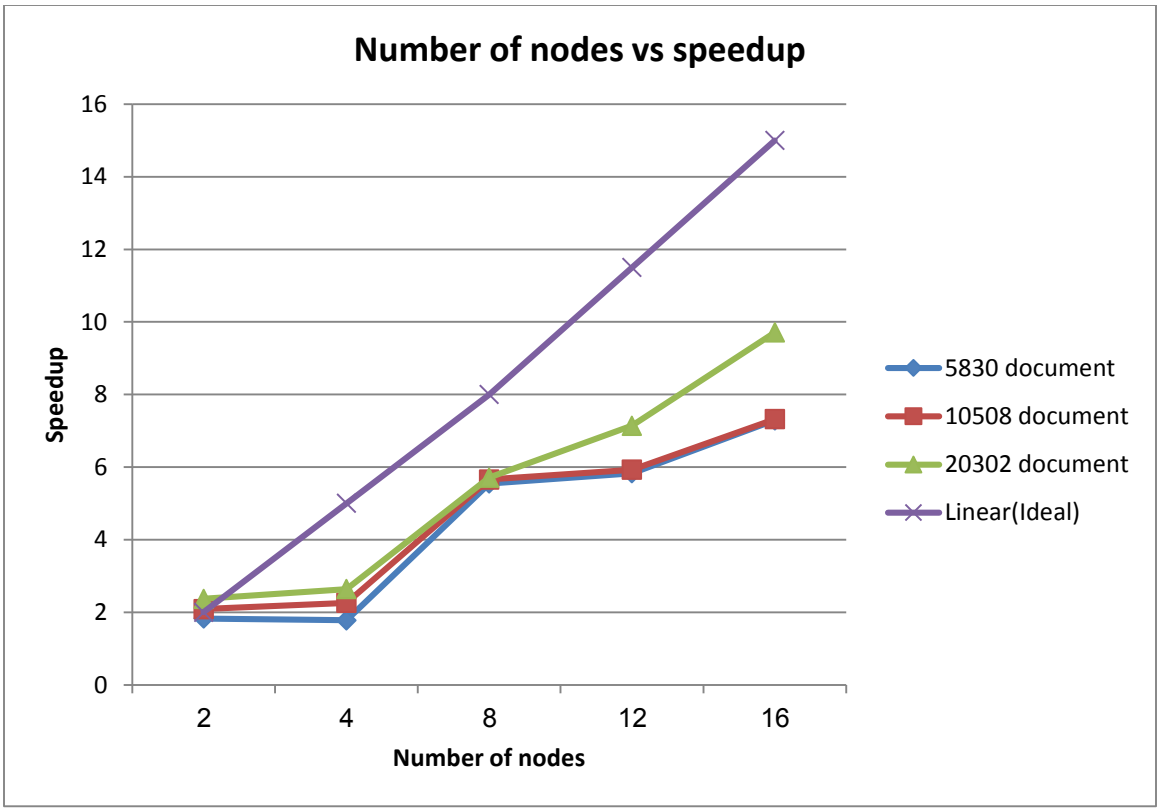


Figure 5.3 The Relative Speedup of the Proposed Parallel Apriori

5.4.3 Support and Confidence

Support is an important measure (Section 3.1) because a rule that has very low support may occur simply by chance. The smaller the minimum support threshold is, the more frequent itemsets there will be, so the execution time will increase along with the decrease of the minimum support threshold. The number of frequent items increases quickly along with decrease of minimum support threshold. Table 5.4 shows the execution time for different support values.

Table 5.4 Minimum Support Threshold with Execution Time

Support	Time (sec)
25	281.23
30	210.02
35	109.2
40	99.2
45	87.02

The confidence is used to measure the strength of the rules. Table 5.5 shows the association rules generated with different support and confidence, the number of rules decreases as the support increase. A low support rule is also likely to be uninteresting. For these reasons, support is often used to eliminate uninteresting rules. In Table 5.5, the rules with the minimum confidence specified is considered strong rules.

So we considered the value of support 35 to generate the frequent itemsets which lead to decrease the execution time if we use lower support 1 the execution time will increase and the frequent itemsets also increase, the number of rules also increase and some rules maybe occurred by a chance. We considered the minimum confidence which used to determine the strength of the rule be 60% this leads to generate a good number of rules that that are considered meaningful and strong. For example as shown in the table, when the support is 35 and confidence is 60%, the number of generated association rules that are considered meaningful and strong is 641.

Table 5.5 : The number of Rules Generated for Different Support and Confidence

Support	**25**			**35**			**45**		
Confidence	**50 %**	**60 %**	**70%**	**50 %**	**60 %**	**70 %**	**50 %**	**60 %**	**70%**
Number of rules	568	452	322	690	641	388	755	599	512

Table 5.6 show an example of some rules resulted from the experiments on hadith data. We have rules say: (افضل العمل) is (ايمان) and (الحج), this is true, we have a hadith for (افضل الاعمال).

Also we have rules say: (اشراط الساعة) is (الجهل) and (الخمر), this is true, we have a hadith for (اشراط الساعة).

Another rules say (كل مسكر) is (حرام) this is true, we have a hadith for (المسكرات).

Table 5.6: Example of the Rules

افضل عمل ← ايمان
ايمان احسان ← اسلام
افضل عمل ← الحج
اشراط الساعة ← الخمر
من ايمان ← الحياء
اشراط الساعة ← الجهل
كل مسكر ← حرام
جوف ليل ← صلاة
حسن الخلق ← البر
تقوى الله ← امر

صيام ← جنة
حج مبرور ← جنة
صيام قيام ← رمضان

5.5 Summary

This chapter presented and analyzed the experimental results. It presented the corpus characteristics, explained the experimental environment, and the implementation of the parallel Apriori algorithm using MapReduce model. Also, it presented the experimental results of the parallel Apriori and its performance. Finally The evaluation of the quality of the parallel Apriori model during sets of experiments.

Overall, results indicated that the parallel Apriori algorithm improved the performance of the Apriori and this improvement became much more obvious when the data is very large. The improvement in the execution time and speedup.

Chapter 6 Conclusion and Future Work

Mining association rules from large Arabic text documents is an important research in text mining. Apriori algorithm is of low efficiency when used with large amount of computational power for generating frequent itemsets. Such a drawback makes it unsuitable to handle a large volume of text documents with high performance and in particular in the Arabic language.

We proposed a parallel Apriori approach for large-scale Arabic text document based on MapReduce. It involves Arabic text documents collection, Arabic text processing, design the suitable MapReduce computing model for parallel Apriori over Hadoop platform, implementation of the parallel algorithm using java programming.

We tested our approach using large scale Shamela-sourced corpus which is the largest Arabic corpus of text documents. The test is performed on Hadoop cluster consisting of 16 nodes as a MapReduce model. The experimental results on the performance indicate that the parallel Apriori algorithm design has very good speedup characteristics when the problem sizes are scaled up.

The proposed approach can be used efficiently to generate frequent itemsets from large scale Arabic text with high performance in terms of execution time and speedup while generating strong association. Hence our approach overcome the problem of low efficiency of the sequential Apriori algorithm while maintaining the level of association rules generation.

There are several directions for improvement and future investigation. Our work can be extended to cover larger computer clusters and Arabic text

documents that will be more than one tera bytes. Additionally, we can apply this parallel Apriori to various application domains such as weather data, internet traffic, log files, medical information, among others to check its generalization. We will also extend our work to cover a popular distributed programming paradigms like MapReduce in a cloud environment. Further algorithms can be applied to interesting applications. Finally the work can be applied with other cloud-based technologies, where algorithms can be used with big data techniques over MapReduce model to speed up the process and give accurate results. Also for future work the Apriori suffers from the number of scanning for the data in order to generate frequent itemsets we can take this drawback for future.

References

[1] R. M. AbuTair, "Design and Evaluation of a Parallel Classirier for Large-Scale Arabic Text," *Int.J.Comput,* 2013.

[2] L. Yang, Ren-hua, "An Improved Apriori Algorithm for Association Rules," *Tlekomnika,* vol. 11, pp. 6521-6526, 2013.

[3] V. Wu, "Top 10 Algorithms in Data Mining," *Knowledge and Information System,* pp. 1-37, 2008.

[4] S. Hammoud, "MapReduce Network Enabled Algorithms for Classification based on Association Rules," *PhD Thesis,* 2011.

[5] S. G. Jeffrey Dean, "MapReduce: Simplified Data Processing on Large Clusters," *ACM,* vol. 5, pp. 107-113, 2008.

[6] "Apache Software," 2010. [Online]. Available: http://hadoop.apache.org/. [Accessed 2015].

[7] S. Hairong Kuang, "The Hadoop Distributed File System," *MSST,* pp. 1-10, 2010.

[8] I. Ali, "Application of Mining Algorithm to find Frequent Patterns in a Text Corpus," *International Journal of Software Engineering and its Applications,* vol. 6, no. 3, pp. 127-134, 2012.

[9] C. D. Jimmy Lin, "Data Intensive Text Processing with MapReduce," *Synthesis Lectures on Human Language Technologies 3.1,* 2010.

[10] A.-Z. A, "Mining Arabic Text using Soft-Matching Association Rules," *Computer Engineering & Systems, ICCES'07. International Conference,* pp. 421-426, 2007.

[11] B. A. Mohammed Al-Maolegi, "An Imporoved Apriori Algorithm for Association Rules," *International Research Journal of Computer Science and Application,* vol. 1, pp. 1-8, 2013.

[12] P. G. Sanjeev Rao, "Implementing Improved Algorithm Over Apriori Data Mining Association Rules Algorithm," *International Journal of Computer Science and Technology,* vol. 3, pp. 489-493, 2012.

[13] U. Pol, "Design and Development of Apriori Algorithm for Sequential to concurrent mining using MPI," *International Journal of Computers & Technology,* 2013.

[14] k. l. N.C.Mahanti, "A Novel Data Mining Algorithm for Semantic Web Based Data," *International Journal of Computer Science and Security* , pp. 160-175, 2010.

[15] B. J. Roberto J., "Efficiently Mining Long Patterns from Databases," *ACM,* vol. 27, pp. 85-93, 1998.

[16] J. Woo, "Apriori MapReduce Algorithm," in *International conference on Parallel and Distributed Processing Techniques and Applications,* 2012.

[17] N. T.V.Mahendra, "Data Mining for High Performance Data Cloud using Association Rule Mining," *IOSR Journal of Computer Engineering,* pp. 16-22, 2012.

[18] O. Y. Osman Hegazy, "Ann Efficient Implementation of Apriori Algorithm Based on Hadoop MapReduce Model," *International Journal of Reviews in Computing,* vol. 12, pp. 59-67, 2012.

[19] S. B. Zeba Qureshi, "Improving Apriori Algorithm to Get Better Perfrmance with Cloud Computing," *International Journal of Software and Hardware Research in Engineering,* vol. 2, pp. 33-37, 2014.

[20] H. Kyong-Ha Leem, "Parallel Data Processing with MapReduce: A Survey," *ACM,* vol. 40, pp. 11-20, 2012.

[21] A.. Hung-chih Yang, "MapReduce Merge: A Simplified Relational Data Processing on Large Clusters," *ACM*, pp. 1029-1040, 2007.

[22] R. Rakesh Agrawal, "Fast Algorithms for Mining Association Rules in Large Databases," *Computer Science and Technology*, vol. 15, pp. 487-499, 1994.

[23] R. M. Abu Shab, "Large-Scale Arabic Text Classification Using MapReduce," *MS. Thesis*, 2015.

[24] S.Q. Zaho, "Association Rule Mining: A Survey," *anyang Technological University* , 2003.

[25] T. L. Rakesh Arrawal, "Mining Association Rules Between Sets of Items in Large Databases," in *SIGMOD*, 1993.

[26] A. Grama, Introduction to Parallel Computing, Addison Wesley, 2003.

[27] D. Matei Zaharia, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," *European Conference on Computer Systems*, pp. 265-278, 2010.

[28] "Arabic language," [Online]. Available: http://ar.wikipedia.org/wiki/ اللغة العربيةٍ. [Accessed 2015].

[29] "Tokenization,"2015.[Online].Available: http://en.wikipedia.org/wiki/Tokenization.

[30] O. F. Mohammed Aljlayl, "On Arabic Search: Improving the Retrieval Effectiveness via a Light Stemming Approach," *International Conference on Information and Knowledge Managment*, pp. 340-347, 2002.

[31] M. G. Noll, "Running Hadoop on Ubuntu Linux (Single-Node Cluster)," 2011. [Online]. Available: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/#installation.. [Accessed 2015].

[32] "Improving Stemming for Arabic Information Retrieval: Light Stemming and Occurrence Analysis," *ACM Iternational Conference,* pp. 275-282, 2002.

[33] J. B. Lovins, "Development of Stemming Algorithm," *Mechanical Translation and Computational Linguistics,* pp. 23-31, 2000.

[34] C. D. Paice, "An Evaluation Method for Stemming Algorithms," *ACM SIGIR Conference,* pp. 42-50, 1994.

[35] T. White, Hadoop: The Definitive Guide, O'Reilly, 2012.

[36] C. Lam, Hadoop in Action, Manning Publications, 2010.

[37] A. Alam, "Hadoop Architecture and Its Issues," *International conference on Computational Intelligence,* vol. 2, pp. 288-291, 2014.

[38] "Apache Hadoop. Welcome to Apache Hadoop," Apache, 2010. [Online]. Available: http://hadoop.apache.org/. [Accessed 2015].

[39] H. E.-R. Mostafa Abd-El-Barr, Fundamental of Computers Organization and Architecture, Willey, 2005.

[40] "Shamela Library," [Online]. Available: http://shamela.ws. [Accessed 2015].

[41] H. K. Konstantin Shvachk, "The Hadoop Distributed File System," *MSST,* pp. 1-10, 2010.

[42] C. D. Jimmy Lin, "Data Intensive Text Processing with MapReduce," *Synth. Lect.Hum.Lang Techno,* vol. 3, pp. 1-177, 2010.

[43] U. Y. Raymond Moone, "Text Mining with Information Extraction," *Multilingualism and Electronic Language Management,* pp. 141-160, 2005.