A Dissertation
On

"Performance Analysis of Apriori And FP Growth On Different MapReduce Frameworks"

Submitted in partial fulfillment of the requirement
for the award of degree of

MASTER OF TECHNOLOGY
Software Engineering
Delhi Technological University, Delhi

SUBMITTED BY

Ravi Ranjan
2K15/SWE/14

Under the Guidance of

Mr. Manoj Sethi
Department of Computer Science & Engineering
Delhi Technological University

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
2017

DECLARATION

I hereby declare that the work entitled "**Performance Analysis of Apriori And FP Growth on Different MapReduce Frameworks**" which is being submitted to Delhi Technological University, in partial fulfilment of requirements for the award of degree of Master of Technology (Software Engineering) is a bonafide report of thesis carried out by me. The material contained in the report has not been submitted to any university or institution for the award of any degree.

Ravi Ranjan

2K15/SWE/14

CERTIFICATE

This is to certify that the dissertation entitled **"Performance Analysis of Apriori And FP Growth On Different MapReduce Frameworks"** has been submitted by **Ravi Ranjan (Roll Number: 2K15/SWE/14),** in partial fulfillment of the requirements for the award of Master of Technology degree in Software Engineering at **DELHI TECHNOLOGICAL UNIVERSITY**. This work is carried out by him under my supervision and has not been submitted earlier for the award of any degree or diploma in any university to the best of my knowledge.

(Mr. Manoj Sethi)
Project Guide
Department of Computer Science & Engineering
Delhi Technological University

ACKNOWLEDGEMENT

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

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

I am also thankful to him for trusting my capabilities to develop this project under his guidance.

I would also like to express my gratitude to the university for providing us with the laboratories, infrastructure, testing facilities and environment which allowed me to work without any obstructions.

Ravi Ranjan

2K15/SWE/14

ABSTRACT

Association rule mining remains a very popular and effective method to extract meaningful information from large datasets. It tries to find possible associations between items in large transaction based datasets. In order to create these associations, frequent patterns have to be generated. Apriori and FP Growth are the two most popular algorithms for frequent itemset mining. To enhance the efficiency and scalability of Apriori and FP Growth, a number of algorithms have been proposed addressing the design of efficient data structures, minimizing database scan and parallel and distributed processing. MapReduce is the emerging parallel and distributed technology to process big datasets on Hadoop Cluster. To mine big datasets it is essential to re-design the data mining algorithm on this new paradigm. However, the existing parallel versions of Apriori and FP-Growth algorithm implemented with the disk-based MapReduce model are not efficient enough for iterative computation.

Hence a number of map reduce based platforms are being developed for parallel computing in recent years. Among them, two platforms, namely, Spark and Flink have attracted lot of attention because of their inbuilt support to distributed computations. But, not much work has been done to test the capabilities of these two platforms in the field of parallel and distributed mining. Therefore, this work helps us to better understand, how the two algorithms perform on three different platforms. We conducted an in-depth experiment to gain insight into the effectiveness, efficiency and scalability of the Apriori and Parallel FP Growth algorithm on Hadoop, Spark and Flink.

Contents

List of Figures & Tables

CHAPTER 1

INTRODUCTION

This chapter briefly introduces the research work proposed in the thesis. Section 1.1 gives an overview of the research undertaken. Section 1.2 briefly explains the Big data, followed by its techniques in section 1.3. The applications and challenges of big data are explored in section 1.4 & 1.5 respectively. Section 1.6 discusses motivation and scope. Section 1.7 enlightens the research objectives. Section 1.8 presents an outline of this thesis and labeling the remaining chapters. Finally, Section 1.9 gives the summary of the chapter.

1.1 Introduction

Internet has become an amalgamated, impeccable and a necessary part of our lives. It is changing swiftly so are we. As more and more people have started using it, Web is also going through a paramount expedient. In the past few years, web based documents are achieving popularity as a way that portraits individual experience and sentiments. According to www.worldwidewebsize.com, the indexed Web contains at least 4.5 billion pages (Monday, 20 March, 2017). With the massive proliferation in the velocity, volume and variety of information accessible online and the consequent need to develop viable paradigms which facilitate better techniques to access this information, there has been a strong resurgence of interest in Big data analysis research in recent years.

With the growth of Web 2.0, which emphasis user-generated content, the way people used to express their views and opinions has also changed prominently. Ideas, comments, views, suggestion, feedbacks are shared by the users. Better methods are now used to make decisions. Earlier, people use to conduct surveys but now online reviews are studied to make a conclusion from the opinions given by the user. As with the increase of amount of data on the Web, it is impossible for an individual to study, examine such a large amount of data.

Earlier, an amount of data generated was not that high and we kept archiving the data as there was just need of historical analysis of data. But today data generation is in petabytes that it is not possible to archive the data again and again and retrieve it again when needed as data

scientists need to play with data now and then for predictive analysis unlike historical as used to be done with traditional. 80% of the data getting generated today is unstructured and cannot be handled by our traditional technologies. According to the statistics the percentage of data that has been generated from last two years is 90%. This data comes from many industries like climate information collects by the sensor, different stuff from social media sites, digital images and videos, different records of the purchase transaction. This data is big data.

1.2 Big Data

As stated on www.gartner.com, Big data is huge-volume, fast-velocity, and different variety information assets that demand innovative platform for enhanced insights and decision making. In other words, big data gets generated in multi-terabyte quantities, changes fast and comes in varieties of forms that is difficult to manage and process using RDBMS or other traditional technologies. Big Data solutions provide the tools, methodologies, and technologies that are used to capture, store, search & analyses the data in seconds to find relationships and insights for innovation and competitive gain that were previously unavailable.

In simple terms, Big Data is an idea that the amount of data that we generate (and more importantly, collect) is increasing extremely quickly. More importantly, companies are recognizing that this data can be used to make more accurate predictions, and therefore, make them more money. Facebook, for example, knows how often you visit many websites (due to the pervasive Like on Facebook buttons) and wants to use that information to show you ads you are more likely to click on.

As Gary King stated, “There is a big data revolution”, as now we can use this data to get some meaningful information and utilize it in so many ways, that can change the way we conceive things now. Big data is used in multiple domains like

- Netflix Uses Big Data to Improve Customer Experience
- Promotion and campaign analysis by Sears Holding
- Sentiment analysis
- Customer Churn analysis
- Predictive analysis
- Real-time ad matching and serving

To analyze such an unstructured data to get the information we want using computers is not easy. Like in sentiment analysis field, computers still cannot get the exact emotion of a person from their tweets or status like a human can get. But, at the same time a human being even for a large group of humans it is not possible to analyze the whole data available over the web to understand the patterns, behaviors etc. The information extracted from every second increasing data helps enhance the business, a shopping website can recommend the products which a person will likely buy on the basis of his mood, his location, events happening in his life, as all such information are easily available on the social networking sites, but to analyze them is not easy. So, big data technologies are the center of the attraction of researchers these days, a lot of work has been done, but still none of the method is like a 'silver bullet' which can solve the problems faced by big data analytics. Some of the Big data technologies are discussed in next section.

1.3 Big Data Technologies

With the fast changing world, many techniques have been proposed to handle the big data. The most famous technologies used for big data analytics are Apache Hadoop, Apache Spark and Apache Flink. Big data is creating Big Impact on industries today. World's 50% of the data has already been moved to Hadoop – The Heart of Big Data. It is predicted that by 2017, more than 75% of the world's data will be moved to Hadoop and this technology will be the most demanding in the market as it is now. Further enhancement of this technology has led to an evolution of Apache Spark – lightning fast and general-purpose computation engine for large-scale processing. It can process the data up to 100 times faster than MapReduce. While Apache Flink is a streaming engine that can also do batches. So, at its core, Flink is more efficient in terms of low latency.

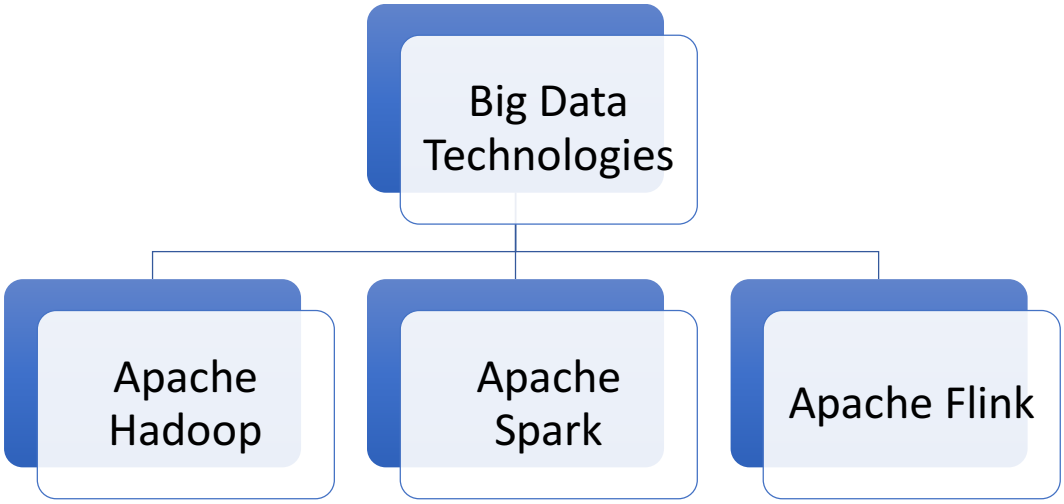

Fig 1. Big Data Technologies

Apache Hadoop: Hadoop is an open source tool from the ASF – Apache Software Foundation, which is used to store large amount of data sets. Hadoop is provided for data storage, data access, data processing and security operations. It is flexible enough to be able to work with multiple data sources, either aggregating multiple sources of data in order to do large scale processing, or even reading data from a database in order to run processor-intensive machine learning jobs. It has several different applications, but one of the top use cases is for large volumes of constantly changing data, such as location-based data from weather or traffic sensors, web-based or social media data, or machine-to-machine transactional data.

Apache Spark: It provides faster and more general-purpose data processing engine. It is basically designed for fast computation. It covers a wide range of workloads Such as batch, interactive, iterative and streaming. Easy to program and does not require any abstractions. Programmers can perform streaming, batch processing and machine learning, all in the same cluster. It has in-built interactive mode. Spark is highly fault-tolerant, no need to restart the application from scratch in case of any failure.

Apache Flink: Apache Flink is called 4G of Big Data. It is an open source framework that can handle streaming as well as batch data. Apache Flink is a streaming engine that can also do batches. Apache Spark is a batch engine that emulates streaming by micro batches. So, at its core, Flink is more efficient in terms of low latency.

1.4 Applications of Big Data

IT organizations have started considering Big data initiative for managing their data in a better manner, visualizing this data, gaining insights of this data as and when required and finding new business opportunities to accelerate their business growth. Every CEO wants to transform his company, enhance their business models and identify potential revenue sources whether he being from telecom domain, banking domain, retail or healthcare domain etc. Such business transformation requires right tools and hiring the right people to ensure right insights are extracted at right time from the available data. Some of the applications of big data in various sectors are as follows:

- Big data in manufacturing sector: Big data can be used to identify machinery and process variations that may be indicators of quality problems.
- Big data for product distribution: Based on data available, its analysis could be done to ensure proper distribution in proper market.
- Big data in Marketing field: Big data helps in knowing better marketing strategy that could increase ale.
- Price Management using Big data: To maintain position in market, price management plays a key role and Big data helps business in knowing market trend for it.
- Merchandising: Big Data plays a major role in sales for retail market also.
- Big data in Sales: It helps in increasing sale for the business. It also helps in optimizing assignment of sales resources and accounts, product mix and other operations.

Big data has enormous potential to improve the human condition, with emphasis on health and productivity. Less people will be needed to provide for a healthier, ageing population. Big data analysis can help by identify productive, or at least non-destructive occupations in the changing demographic where there won't be enough conventional "employment" to go around.

1.5 Challenges of Big Data

The single greatest challenge facing data analytics in the 21st century is the so-called “utilization gap.” Every major company has vast stores of information in increasingly complex databases. However, despite having more data than ever before, most data analytics still fail to provide actionable insights.

Efforts to bridge the utilization gap extend to BI platforms as well with business intelligence becoming more user friendly with each iteration, allowing the typical business user to query data themselves. For example, every popular BI platform provides reasonably intuitive UIs that allow normal users to find basic visualizations and charts (Tableau, Birst, etc.). However, visualizations are often not enough. Some other challenges faced by organization are:

- Tools are too expensive to acquire, deploy and maintain.
- Solutions are too complicated for normal business users to use.
- Most vendors have legacy business AND technology approaches that limit the potential for customers to succeed with Data.

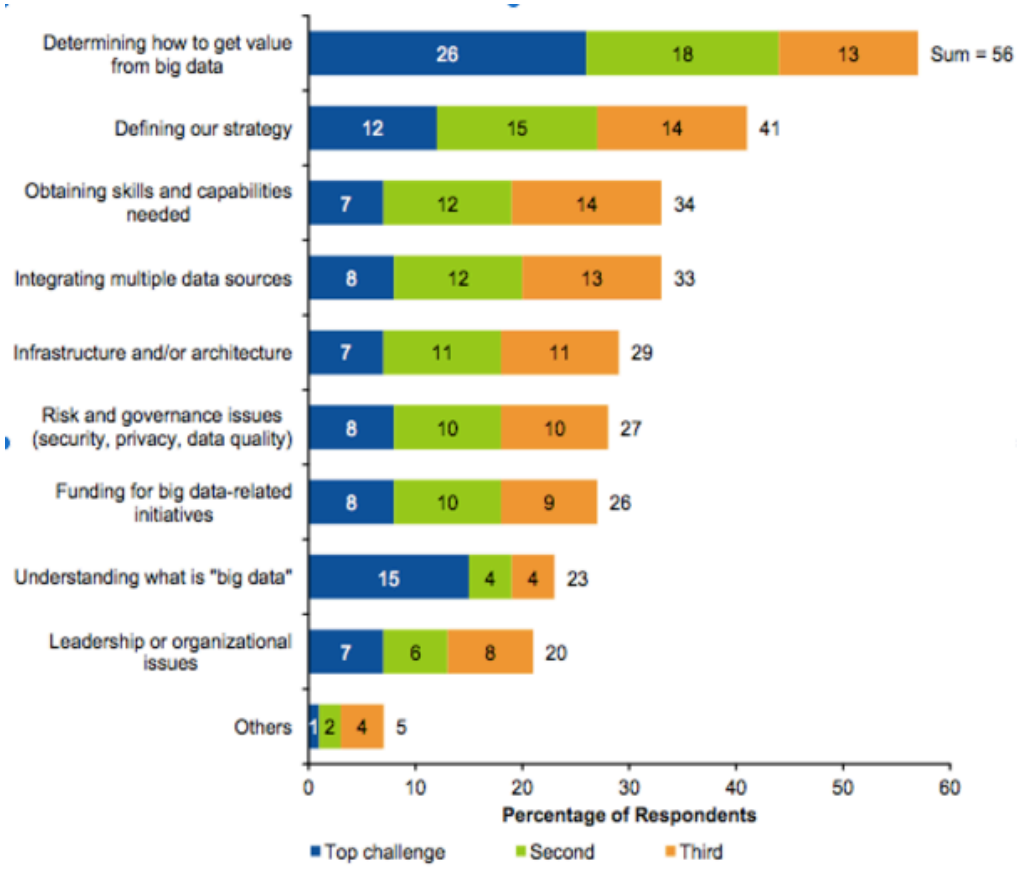

Fig 2. Top Big Data Challenges

Figure 2 shows the challenges of big data faced by multiple organization as per the data collected by www.gartner.com in 2013 by collecting the data from various industries to understand the issues which are becoming hurdles in the way of using big data analytics by most of the companies world-wide.

Big data is a fascinating area that holds a lot of promise, but investment in big data is not like investing in a financial investment where you put in some money, perhaps pay a financial specialist to manage it, and wait for it to grow. It is a lot more like investing in a gym membership where the whole organisation needs to change their lifestyle to reap the benefits.

1.6 Motivation and Scope

Increment of Web 2.0 gives the abundance services which can be helpful for user's awareness. Big Data is a way to solve all the unsolved problems related to data management and handling, an earlier industry was used to live with such problems. With Big data analytics, you can unlock hidden patterns and know the 360-degree view of customers and better understand their needs.

Web 2.0 has involved quite a large number of people to use these services. As almost every type of public is concerned, we need to have refined data which may not offend someone sentiments. So, to detect the patterns in this large amount of data multiple algorithms has been proposed. Some of the algorithms perform closed item set mining, some weighted itemset mining. Multiple big data techniques are available each having their own plus points like Spark can handle the iterative algorithms very well, while Hadoop can't. Similarly, there are multiple algorithms available for association rule mining, like Apriori, Eclat, FP-Growth, Relim etc. But each handle different types of data more precisely than other algorithms. Apriori and FP-growth are the oldest and most often used algorithms for frequent itemset mining.

There exists a study showing the comparison of Apriori and Fp-growth on Hadoop, and another study has also shown the comparison of two MapReduce frameworks Hadoop and spark for Apriori algorithm. No one has still explored the behavior of these two algorithms on all the three MapReduce frameworks with 3 datasets of different sizes.

This work helps us to better understand, how the two algorithms perform in different environments. And it also clear that which algorithm is good for large datasets and which is better for small or medium datasets. How much time does different MapReduce platforms take to run the same algorithm also determines the effectiveness of the MapReduce framework.

1.7 Research Objectives

The main research objectives of the work done in this thesis are:

Research objective 1 – To study the different techniques that has been used for Parallel Apriori and FP Growth.

Research objective 2 – To study the MapReduce frameworks that has been used for implementing Distributed Apriori and FP Growth**.**

Research objective 3 – Checking the performance of Apriori and FP growth algorithm under different environment.

The objective of this thesis is to analyze the two of the famous association rule based mining algorithms on different MapReduce frameworks.

1.8 Organization of Report

This thesis is structured into 5 Chapters followed by references and appendix.

Chapter 1 provides the overview of the research work done, about the big data, research objectives, scope and motivation of the project. Finally, analyzing the need for solution for which research is done.

Chapter 2 provides the essential background and context for this thesis and provides a complete justification for the research undertaken in this thesis.

Chapter 3 gives the details of the methodology employed and outlines the uses of algorithms and MapReduce platforms.

Chapter 4 describes the implementation. It discusses all the input sets, platform and tool used to implement and to compare the results.

Chapter 5 describes the experimental results obtained from the given datasets. It presents the analysis of tests performed.

Chapter 6 presents future scope and conclusions based on the contribution made by this thesis.

1.9 Chapter Summary

This chapter presents the idea used in this thesis. It discusses research problem, objectives, goals and motivation for the research. Justification for the research problem is outlined, together with an explanation of the research methodology used. The next chapter describes the literature survey and relevant background work done till date in context of this thesis.

CHAPTER 2

LITERATURE REVIEW

This Chapter first discusses the technologies and algorithms used in detail with their applications and challenges. In last section, the overview of the work done on this field is discussed.

2.1 Association Rules

The building blocks of a market basket analysis are the items that may appear in any given transaction. Groups of one or more items are surrounded by brackets to indicate that they form a set, or more specifically, an itemset that appears in the data with some regularity. Transactions are specified in terms of itemset, such as the following transaction that might be found in a typical grocery store:

$$\{\text{bread, peanut butter, jelly}\}$$

The result of a market basket analysis is a collection of association rules that specify patterns found in the relationships among items he itemsets. Association rules are always composed from subsets of itemsets and are denoted by relating one itemset on the left-hand side (LHS) of the rule to another itemset on the right-hand side (RHS) of the rule. The LHS is the condition that needs to be met in order to trigger the rule, and the RHS is the expected result of meeting that condition. A rule identified from the example transaction might be expressed in the form:

$$\{\text{peanut butter, jelly}\} \rightarrow \{\text{bread}\}$$

In plain language, this association rule states that if peanut butter and jelly are purchased together, then bread is also likely to be purchased. In other words, "peanut butter and jelly imply bread." Developed in the context of retail transaction databases, association rules are not used for prediction, but rather for unsupervised knowledge discovery in large databases.

Because association rule learners are unsupervised, there is no need for the algorithm to be trained; data does not need to be labelled ahead of time. The program is simply unleashed on a dataset in the hope that interesting associations are found. The downside, of course, is that there

isn't an easy way to objectively measure the performance of a rule learner, aside from evaluating them for qualitative usefulness—typically, an eyeball test of some sort.

Although association rules are most often used for market basket analysis, they are helpful for finding patterns in many different types of data. Other potential applications include:

1. Searching for interesting and frequently occurring patterns of DNA and protein sequences in cancer data
2. Finding patterns of purchases or medical claims that occur in combination with fraudulent credit card or insurance use
3. Identifying combinations of behavior that precede customers dropping their cellular phone service or upgrading their cable television package

Association rule analysis is used to search for interesting connections among a very large number of elements. Human beings are capable of such insight quite intuitively, but it often takes expert-level knowledge or a great deal of experience to do what a rule learning algorithm can do in minutes or even seconds. Additionally, some datasets are simply too large and complex for a human being to find the needle in the haystack.

2.2 Hadoop

Hadoop is an open source tool from the ASF – Apache Software Foundation. Open source project means it is freely available and even its source code can be changed as per the requirements. If certain functionality does not fulfil our requirement, we can change it according to our need. Most of Hadoop code is written by Yahoo, IBM, Facebook, Cloudera. It provides an efficient framework for running jobs on multiple nodes of clusters. Cluster means a group of systems connected via LAN. Hadoop provides parallel processing of data as it works on multiple machines simultaneously.

It is inspired by Google, which has written a paper about the technologies it is using like Map-Reduce programming model as well as its file system (GFS). Hadoop was originally written for the Nutch search engine project when Doug cutting and his team were working on it but very soon, it became a top-level project due to its huge popularity.

Hadoop is an open source framework which is written in Java. But this does not mean you can code only in Java. You can code in C, C++, Perl, python, ruby etc. You can code in any language but it is recommended to code in java as you will have lower level control of the code. It efficiently processes large volumes of data on a cluster of commodity hardware. Hadoop is developed for processing of huge volume of data. Commodity hardware is the low-end hardware, they are cheap devices which are very economic. So, Hadoop is very economic.

Hadoop can be setup on a single machine (pseudo-distributed mode), but the real power of Hadoop comes with a cluster of machines, it can be scaled to thousand nodes on the fly i.e., without any downtime. We need not make any system down to add more systems in the cluster. To learn installation of Hadoop on a multi-node cluster, follow this installation guide.

Hadoop consists of three key parts – Hadoop Distributed File System (HDFS), Map-Reduce and YARN. HDFS is the storage layer, Map Reduce is the processing layer and YARN is the resource management layer.

Why Hadoop

Hadoop is not only a storage system but is a platform for data storage as well as processing. It is scalable (more nodes can be added on the fly), Fault tolerant (Even if nodes go down, data can be processed by another node) and Open source (can modify the source code if required).

Following characteristics of Hadoop make is a unique platform:

1. Flexibility to store and mine any type of data whether it is structured, semi-structured or unstructured. It is not bounded by a single schema.
2. Excels at processing data of complex nature, its scale-out architecture divides workloads across multiple nodes. Another added advantage is that its flexible file-system eliminates ETL bottlenecks.
3. Scales economically, as discussed it can be deployed on commodity hardware. Apart from this its open-source nature guards against vendor lock.

Hadoop works in master – slave fashion. There is a master node and there are n numbers of slave nodes where n can be 1000s. Master manages, maintains and monitors the slaves while slaves are the actual worker nodes. Master should be deployed on good configuration hardware and not just any commodity hardware as it is the centerpiece of Hadoop cluster.

Master just stores the meta-data (data about data) while slaves are the nodes which store the data. Data is stored distributed in the cluster. The client connects with master node to perform any task.

How Hadoop Works

Step1: Input data is broken into blocks of size 128 Mb (by default) and then blocks are moved to different nodes.

Step 2: Once all the blocks of the file are stored on data nodes, a user can process the data.

Step 3: master, then schedules the program (submitted by the user) on individual nodes.

Step 4: Once all the nodes process the data, output is written back to HDFS

Hadoop Flavor's

Below are the various flavors of Hadoop.

- Apache – Vanilla flavour, the actual code is residing in apache repositories.
- Hortonworks – Popular distribution in the industry.
- Cloudera – It is the most popular in the industry.
- MapR – It has rewritten HDFS and its HDFS is faster as compared to others.
- IBM – Proprietary distribution is known as Big Insights.

All the databases have provided native connectivity with Hadoop for fast data transfer. For example, to transfer data from Oracle to Hadoop, you need a connector. Figure 3 shows the ecosystem of Hadoop.

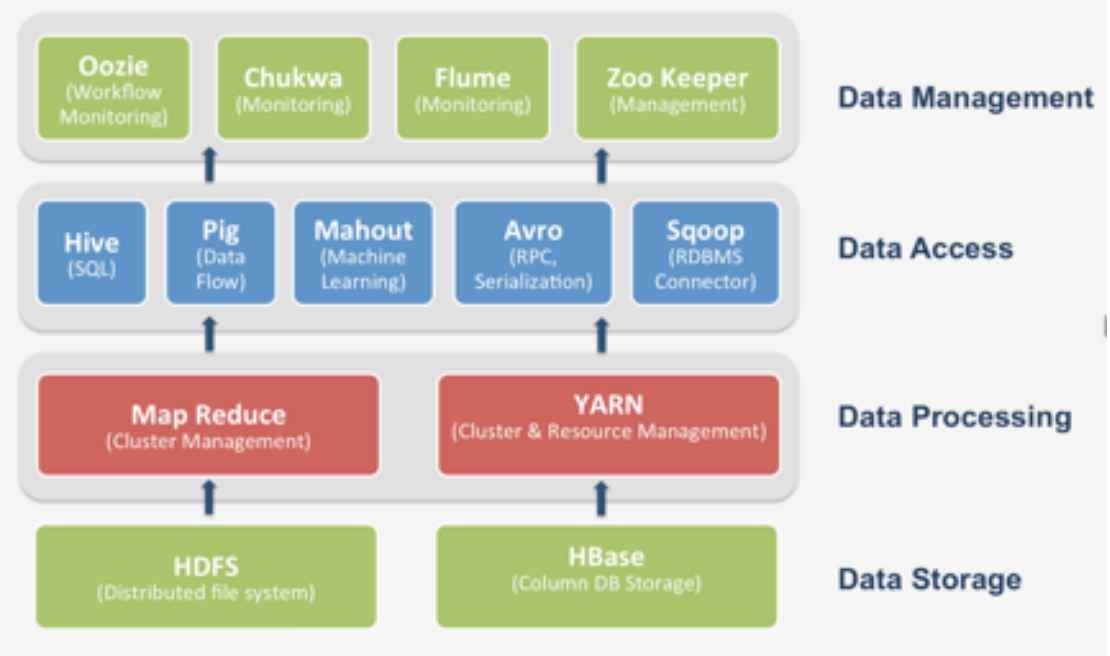

Fig 3. Hadoop Ecosystem

Hadoop Features and Characteristics

Apache Hadoop is the most popular and powerful big data tool, Hadoop provides world's most reliable storage layer – HDFS, a batch Processing engine – MapReduce and a Resource Management Layer – YARN.

1. Open-source – Apache Hadoop is an open source project. It means its code can be modified according to business requirements.
2. Distributed Processing – As data is stored in a distributed manner in HDFS across the cluster, data is processed in parallel on a cluster of nodes.
3. Fault Tolerance – By default 3 replicas of each block is stored across the cluster in Hadoop and it can be changed also as per the requirement. So, if any node goes down, data on that node can be recovered from other nodes easily. Failures of nodes or tasks are recovered automatically by the framework. This is how Hadoop is fault tolerant.
4. Reliability – Due to replication of data in the cluster, data is reliably stored on the cluster of machine despite machine failures. If your machine goes down, then also your data will be stored reliably.
5. High Availability – Data is highly available and accessible despite hardware failure due to multiple copies of data. If a machine or few hardware crashes, then data will be accessed from another path.

6. Scalability – Hadoop is highly scalable in the way new hardware can be easily added to the nodes. It also provides horizontal scalability which means new nodes can be added on the fly without any downtime.
7. Economic – Apache Hadoop is not very expensive as it runs on a cluster of commodity hardware. We do not need any specialized machine for it. Hadoop provides huge cost saving also as it is very easy to add more nodes on the fly here. So, if requirement increases, you can increase nodes as well without any downtime and without requiring much of pre-planning.
8. Easy to use – No need of client to deal with distributed computing, the framework takes care of all the things. So, it is easy to use.
9. Data Locality – Hadoop works on data locality principle which states that move computation to data instead of data to computation. When a client submits the MapReduce algorithm, this algorithm is moved to data in the cluster rather than bringing data to the location where the algorithm is submitted and then processing it.

Limitation of Hadoop

- Issues with small files
- Processing speed.
- High latency
- Supports only batch processing
- Vulnerable by nature

2.3 HDFS

Hadoop Distributed Filesystem (HDFS) is the world's most reliable storage system. HDFS is a Filesystem of Hadoop designed for storing very large files running on a cluster of commodity hardware. HDFS is designed on principle of storage of less number of large files rather than the huge number of small files. It provides fault tolerant storage layer for Hadoop and its other components. Replication of data helps us to attain this feature. It stores data reliably even in the case of hardware failure. It provides high throughput access to application data by providing the data access in parallel.

Hadoop works in master-slave fashion, HDFS also has 2 types of nodes that work in the same manner. There is name node(s) and data nodes in the cluster.

1. Master node (Also called Name node) – As the name suggests, this node manages all the slave nodes and assign work to slaves. It should be deployed on reliable hardware as it is the centerpiece of HDFS.
2. Slave node (Also called data node) – Data nodes are the slaves which are deployed on each machine and provide the actual storage. They are the actual worker nodes. These are responsible for serving read and write requests from the clients. They can be deployed on commodity hardware. If any slave node goes down, name node automatically replicates the blocks which were present at that data node to other nodes in the cluster.

Data storage in HDFS

Whenever any file has to be written in HDFS, it is broken into small pieces of data known as blocks. HDFS has a default block size of 128 MB which can be increased as per the requirements. These blocks are stored in the cluster in distributed manner on different nodes. This provides a mechanism for MapReduce to process the data in parallel in the cluster.

Multiple copies of each block are stored across the cluster on different nodes. This is a replication of data. By default, HDFS has a replication factor of 3. It provides fault tolerance, reliability, and high availability.

A Large file is split into n number of small blocks. These blocks are stored at different nodes in the cluster in a distributed manner. Each block is replicated and stored across different nodes in the cluster.

Rack Awareness in Hadoop HDFS

Hadoop runs on a cluster of computers which are commonly spread across many racks. Name Node places replicas of a block on multiple racks for improved fault tolerance. Name Node tries to place at least one replica of a block in each rack, so that if a complete rack goes down then also system will be highly available Optimizing replica placement distinguishes HDFS from most other distributed file systems. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization.

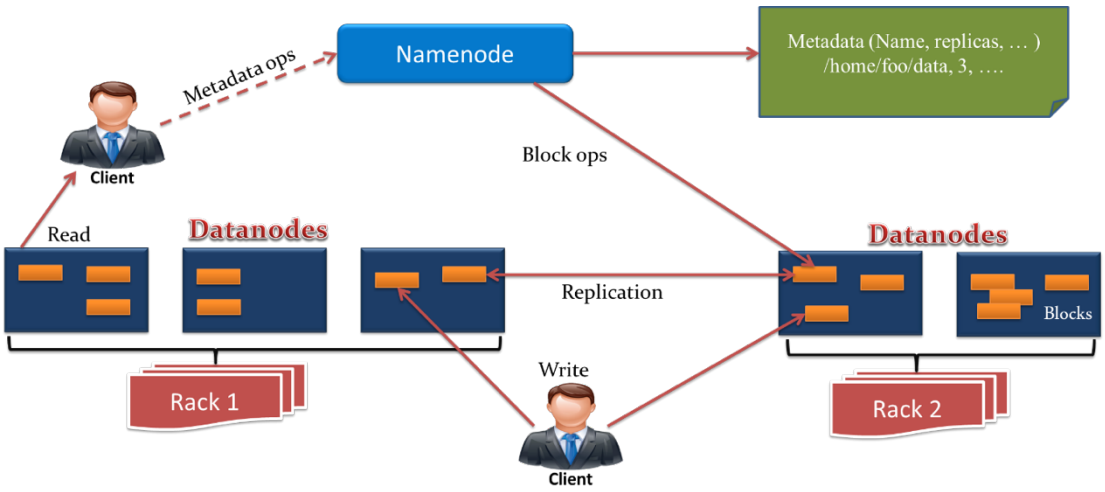

Fig 4. Hadoop Distributed Filesystem

There is a single name node which stores metadata and there are multiple data nodes which do actual storage work. Nodes are arranged in racks and Replicas of data blocks are stored on different racks in the cluster to provide fault tolerance. To read or write a file in HDFS, the client needs to interact with Name node. HDFS applications need a write-once-read-many access model for files. A file once created and written cannot be edited. There are several data nodes in the cluster which store HDFS data in the local disk. Data node sends a heartbeat message to name node periodically to indicate that it is alive. Also, it replicates data to other data node as per the replication factor.

Features of HDFS

- Distributed Storage – Data is stored in distributed manner
- Blocks – Data is split into blocks
- Replication – Blocks are replicated at different nodes
- High Availability – Data is highly available due to replication
- Data Reliability – Data is stored reliably in HDFS
- Fault tolerant – Data replication provides fault tolerance feature
- Scalability – Nodes in HDFS cluster can be increased on the fly
- High throughput access to application – Parallel processing provides high throughput access to application

2.4 Hadoop MapReduce

Map-Reduce is the data processing layer of Hadoop. Map-Reduce is a product system for effortlessly composing applications that process the vast amount of structured and unstructured data stored in the HDFS. It processes the huge amount of data in parallel by dividing the job (submitted job) into a set of independent tasks. By this parallel processing, speed and reliability of cluster is improved. We just need to put the custom code (business logic) in the way map reduce works and rest things will be taken care by the engine.

MapReduce programs are composed in a specific style influenced by useful programming builds, specifically figures of speech for processing data. Here in map reduce we get input as a list and it changes over it into yield which is again a list. It is the heart of Hadoop. Hadoop is so much intense and productive because of map reduce function as parallel handling of data is carried out.

Hadoop Map-Reduce is exceedingly versatile and can be utilized across numerous PCs. Numerous little machines can be utilized to process jobs that ordinarily couldn't be processed by a huge machine. Conceptually, Map-Reduce programs transform lists of input data elements into lists of output data elements. A Map-Reduce program will do this twice, using two different list processing idioms

1. Map
2. Reduce

Basic Terminologies used in Map Reduce are

Job – A "full program" – an execution of a Mapper and Reducer across a data set. It is an execution of 2 processing layers i.e. mapper and reducer. A Map-Reduce job is a work that the client desires to be performed. It comprises of the input data, the Map-Reduce Program, and configuration info. So, client needs to submit input data, he needs to write Map Reduce program and set the configuration info.

Task – An execution of a Mapper or a Reducer on a piece of data. It is additionally called Task-In-Progress (TIP). It implies processing of data is in progress either on mapper or reducer.

Task Attempt - A specific example of an endeavor to execute a task on a node. There is a possibility that anytime any machine can go down. For example, while processing data if any node goes down, framework reschedules the task to some other node. This rescheduling of the task cannot be infinite. There is an upper limit for that as well. The default value of task attempt

is 4. If a task (Mapper or reducer) fails 4 times, then the job is considered as a failed job. For high priority job or huge job, the value of this task attempt can be increased as well.

Working of MapReduce

Map-Reduce divides the work into small parts, each of which can be done in parallel on the cluster of servers. A problem is divided into a large number of smaller problems each of which is processed independently to give individual outputs. These individual outputs are further processed to give final output. Figure 5 shows the working outline of MapReduce.

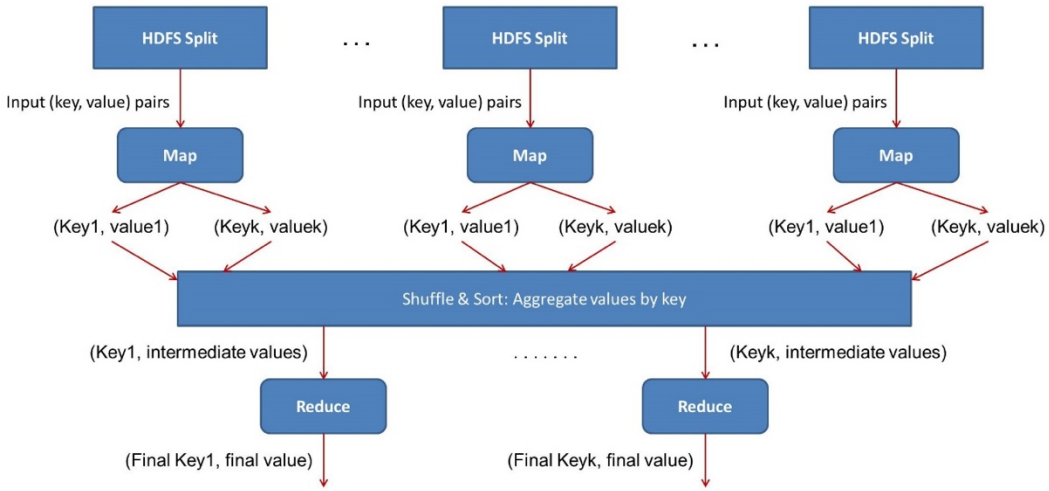

Fig 5. Working of MapReduce

Mapper:

Mapper task processes each input record and it generates a new <key, value> pairs. The <key, value> pairs can be completely different from the input pair. In mapper task, the output is the full collection of all these <key, value> pairs. Before writing the output for each mapper task, partitioning of output take place on the basis of the key and then sorting is done. This partitioning specifies that all the values for each key are grouped together.

Map-Reduce frame generates one map task for each InputSplit generated by the InputFormat for the job. Mapper only understands <key, value> pairs of data, so before passing data to the mapper, data should be first converted into <key, value> pairs.

Reducer

The output of the mapper is processed by the Reducer. After processing the data, it produces a new set of output, which will be stored in the HDFS.

Reducer takes a set of an intermediate key-value pair produced by the mapper as the input and runs a Reducer function on each of them. This data (key, value) can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Reducer first processes the intermediate values for particular key generated by the map function and then generates the output (zero or more key-value pair). One-one mapping takes place between keys and reducers. Reducers run in parallel since they are independent of one another. The user decides the number of reducers. By default, number of reducers is 1.

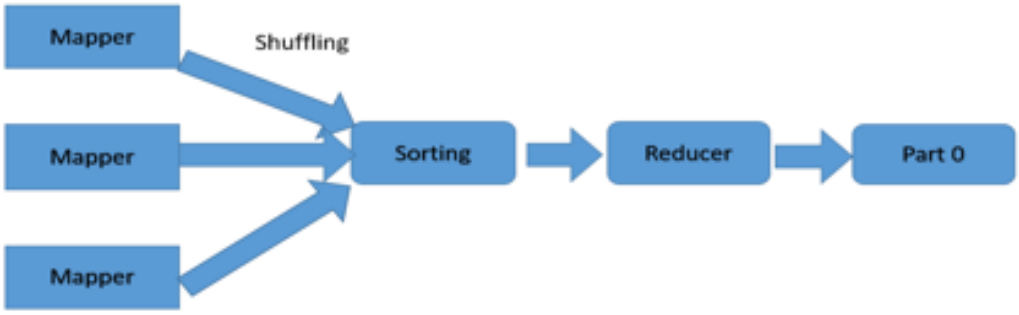

Fig 6. Reducer in Hadoop

Phases of Reducer:

1. Shuffle Phase: In this phase, the sorted output from the mapper is the input to the Reducer. In this phase, with the help of HTTP, the framework fetches the relevant partition of the output of all the mappers.
2. Sort Phase: In this phase, the input from different mappers is again sorted based on the similar keys in different Mappers. The shuffle and sort phases occur concurrently.
3. Reduce Phase: In this phase, after shuffling and, sorting, reduce task aggregates the key value pairs. By OutputCollector.collect(), the output of the reduce task is written to the File-system. Reducer output is not sorted.

Key Value Pair Generation:

InputSplit – It is the logical representation of data. It describes a unit of work that contains a single map task in a Map-Reduce program.

Record Reader- It communicates with the InputSplit and it converts the data into key value pairs suitable for reading by the Mapper. By default, it uses TextInputFormat for converting data into key value pair. RecordReader communicates with the InputSplit until the file reading is not completed.

In Map-Reduce, map function processes a certain key-value pair and emits a certain number of key-value pairs and the Reduce function processes values grouped by the same key and emits another set of key-value pairs as output. The output types of the Map should match the input types of the Reduce as shown below:

Map: (K1, V1) -> list (K2, V2)

Reduce: {(K2, list (V2}) -> list (K3, V3)

Generation of key-value pair depends on the data set and the required output. In general, the key-value pair is specified in 4 places: Map input, Map output, reduce input and Reduce output.

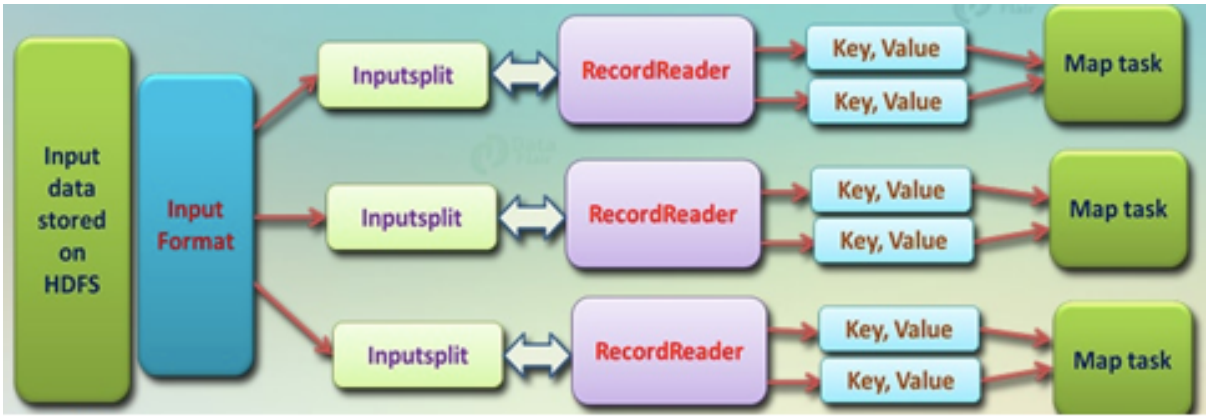

Fig 7. Key Value Pairing in Hadoop MapReduce

1. Map Input: Map-input by default will take the line offset as the key and the content of the line will be the value as Text. By using custom input format, we can modify them.
2. Map Output: Map basic responsibility is to filter the data and provide the environment for grouping of data based on the key.

   Key – It will be the field/ text/ object on which the data has to be grouped and aggregated of the reducer side.

   Value – It will be the field/ text/ object which is to be handled by each individual reduce method.

3. Reduce Input: The output of Map is the input for reduce, so it is same as Map-Output.
4. Reduce Output: It depends on the required output.

Partitioner

Partitioning of the keys of the intermediate map output is controlled by the Partitioner. By hash function, key (or a subset of the key) is used to derive the partition. According to the key value each mapper output is partitioned and records having the same key value go into the same partition (within each mapper), and then each partition is sent to a reducer. Partition class determines which partition a given (key, value) pair will go. Partition phase takes place after map phase and before reduce phase.

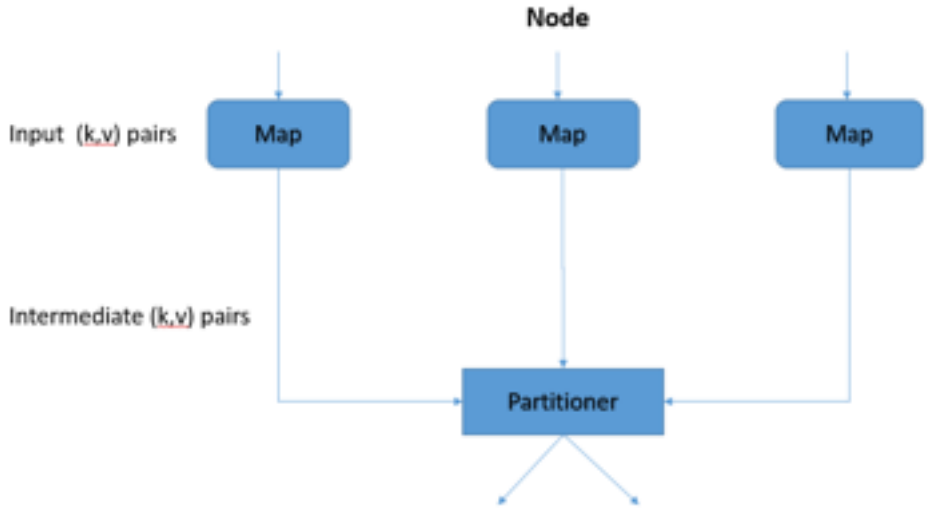

Fig 8. Partitioner in Hadoop

Map-Reduce job takes an input data set and produces the list of key value pair which is the result of map phase in which input data is split and each task processes the split and each map, output the list of key value pairs. Then, the output from the map phase is sent to reduce task which processes the user-defined reduce function on map outputs. But before reduce phase, partitioning of the map output take place on the basis of the key and sorted.

This partitioning specifies that all the values for each key are grouped together and make sure that all the values of a single key go to the same reducer, thus allows even distribution of the map output over the reducer. Partitioner in Hadoop Map-Reduce redirects the mapper output to the reducer by determining which reducer is responsible for the particular key.

Combiner

On a large dataset when we run Map-Reduce job, so large chunks of intermediate data are generated by the Mapper and this intermediate data is passed on the Reducer for further processing, which leads to enormous network congestion. Map-Reduce framework provides a function known as Combiner that plays a key role in reducing network congestion.

The combiner in Map-Reduce is also known as 'Mini-reducer'. The primary job of Combiner is to process the output data from the Mapper, before passing it to Reducer. It runs after the mapper and before the Reducer and its usage is optional

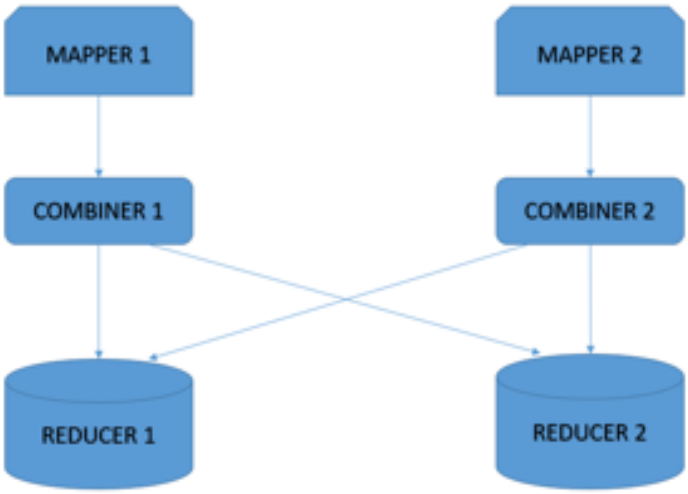

Fig 9. Combiner in Hadoop.

The working of combiner in Hadoop is shown in figure 10 & 11. In the Figure 10, no combiner is used. Input is split into two mappers and 9 keys are generated from the mappers. Now we have (9 key/value) intermediate data, further mapper will send directly this data to reducer and while sending data to reducer, it consumes some network bandwidth (bandwidth means time taken to transfer data between 2 machines). It will take more time to transfer data to reducer if the size of data is big.

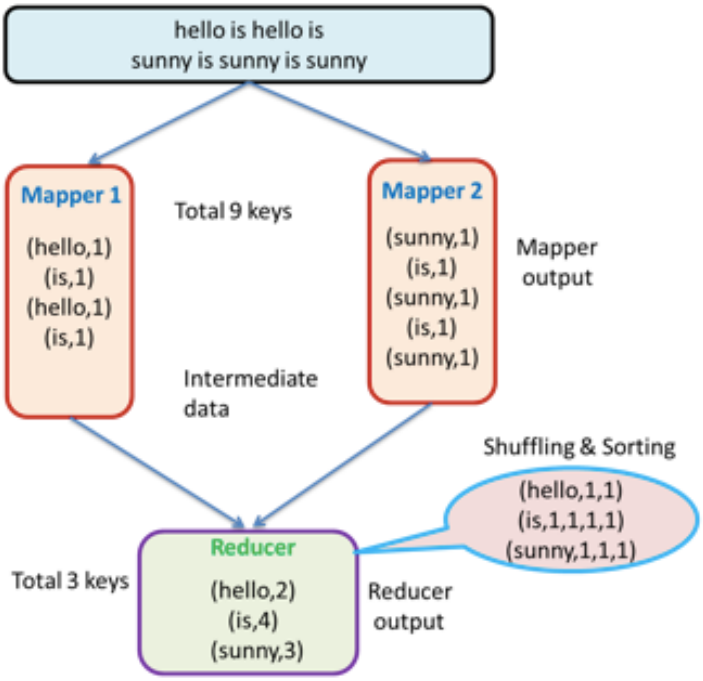

Fig 10. Working of MapReduce in Hadoop with combiners

Now in between mapper and reducer if we use a combiner, then combiner shuffles intermediate data (9 key/value) before sending it to reducer, and generates 4 key/value pair as an output.

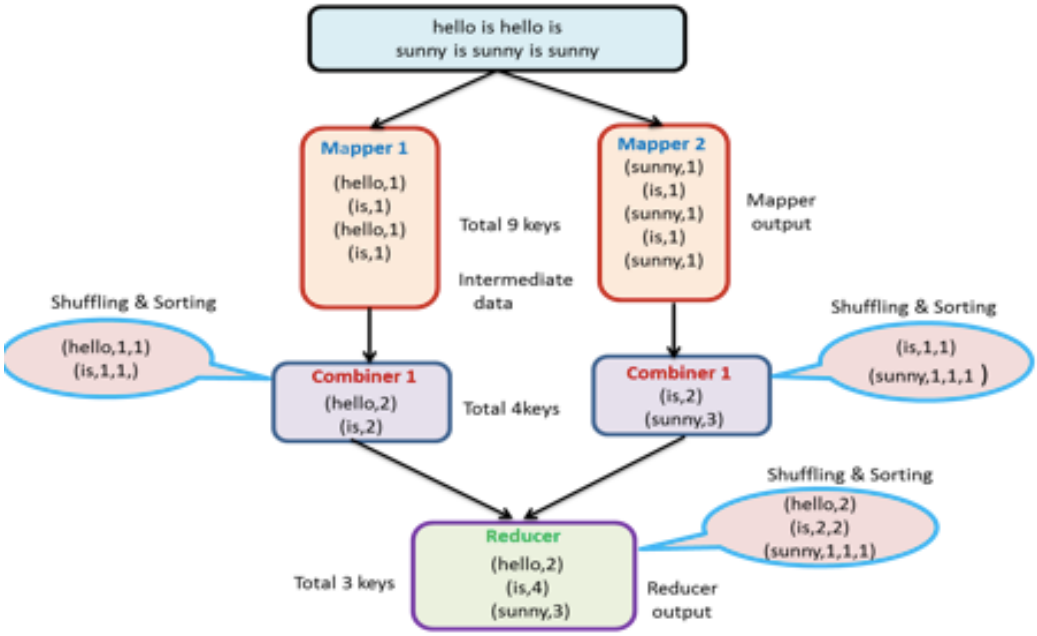

Fig 11. Working of MapReduce in Hadoop with combiners

Reducer now needs to process only 4 key/value pair data which is generated from 2 combiners. Thus, reducer gets executed only 4 times to produce final output, which increases the overall performance.

Shuffle & Sort

Shuffle phase in Hadoop transfers the map output from Mapper to a Reducer in Map-Reduce. Sort phase in Map-Reduce covers the merging and sorting of map outputs. Data from the mapper are grouped by the key, split among reducers and sorted by the key. Every reducer obtains all values associated with the same key. Shuffle and sort phase in Hadoop occur simultaneously and are done by the Map-Reduce framework.

The process of transferring data from the mappers to reducers is known as shuffling i.e. the process by which the system performs the sort and transfers the map output to the reducer as input. So, shuffle phase is necessary for the reducers, otherwise, they would not have any input (or input from every mapper). As shuffling can start even before the map phase has finished so this saves some time and completes the tasks in lesser time.

The keys generated by the mapper are automatically sorted by Map-Reduce Framework, i.e. Before starting of reducer, all intermediate key-value pairs in Map-Reduce that are generated by mapper get sorted by key and not by value. Values passed to each reducer are not sorted; they can be in any order. Learn Hadoop Map-Reduce job optimization and performance tuning techniques.

Sorting in Hadoop helps reducer to easily distinguish when a new reduce task should start, thus saves time for the reducer. Reducer starts a new reduce task when the next key in the sorted input data is different than the previous. Each reduce task takes key – value pairs as input and generates key-value pair as output.

Note that shuffling and sorting in Hadoop Map-Reduce are not performed at all if you specify zero reducers (setNumReduceTasks(0)). Then, the Map-Reduce job stops at the map phase, and the map phase does not include any kind of sorting (so even the map phase is faster).

2.5 YARN

YARN is Yet Another Resource Negotiator, the cutting-edge computation technology and cluster administration innovation. YARN gives a stage to build/run numerous distributed applications in Hadoop. YARN was released in the Hadoop 2.0 version. In 2012, denoting a noteworthy change in Hadoop design. YARN delegates and parts up the obligation into different daemons and accomplishes better execution and adaptation to non-critical failure.

Because of YARN, Hadoop, which could work only as a batch process, can now be designed to process interactive and real-time processing systems. This is a huge advantage as many systems, machines, sensors, and other sources generate huge data continuously streaming and YARN can process this data. YARN architecture is extremely scalable, fault tolerant, and processes data faster as compared to MapReduce 1.x. YARN focuses on high availability and utilization of resources in the cluster. YARN architecture has the following three components:

Resource Manager

In YARN, Resource Manager is the master process manager responsible for resource management among the applications in the system. Resource Manager has a scheduler, which only allocates the resources to the applications and resource availability which Resource Manager gets from containers that provide information such as memory, disk, CPU, network, and so on.

Node Manager

In YARN, Node Manager is present in all the nodes, which is responsible for containers, authentication, monitoring resource usage, and reports the information to Resource Manager. Similar to Task Tracker, Node Manager sends heartbeats to Resource Manager.

Application Master

Application Master is present for each application, responsible for managing each and every instance of applications that run within YARN. Application Master coordinates with Resource Manager for the negotiation of the resources and coordinates with the Node Manager to monitor the execution and resource consumption of containers, such as resource allocations of CPU, memory, and so on.

2.6 Spark

Apache Spark is a general-purpose & lightning fast cluster computing system. It provides high-level API. For example, Java, Scala, Python and R. Apache Spark is a tool for Running Spark Applications. Spark is 100 times faster than Bigdata Hadoop and 10 times faster than accessing data from disk. Spark is written in Scala but provides rich APIs in Scala, Java, Python and R. It can be integrated with Hadoop and can process existing Hadoop HDFS data. Apache Spark was introduced in 2009 in the UC Berkeley R&D Lab, later it becomes AMP Lab. It was open sourced in 2010 under BSD license. In 2013 spark was donated to Apache Software Foundation where it became top-level Apache project in 2014.

The reason why spark came into picture while Hadoop is performing well is because, in the industry, there is a need for general purpose cluster computing tool as:

1. Hadoop MapReduce can only perform batch processing.
2. Apache Storm / S4 can only perform stream processing.
3. Apache Impala / Apache Tez can only perform interactive processing
4. Neo4j / Apache Giraph can only perform to graph processing

Hence in the industry, there is a big demand for a powerful engine that can process the data in real-time (streaming) as well as in batch mode. There is a need for an engine that can respond in sub-second and perform in-memory processing. Apache Spark is a powerful open source engine that provides real-time stream processing, interactive processing, graph processing, in-memory processing as well as batch processing with very fast speed, ease of use and standard interface.

Features of Spark

Spark has numerous features and capabilities worth mentioning, as follows:

1. Runs 100 times faster than MapReduce when running in-memory and 10 times faster when running on disk.
2. Can process iterative and interactive analytics.
3. Many functions and operators available for data analysis.
4. DAG framework to design functions easily.
5. In-memory based intermediate storage.
6. Easy to use and maintain.

7. Written in Scala and runs in JVM environment; applications using Spark can be written in Scala, Java, Python, R, Clojure.
8. Runs in environments such as Hadoop and Mesos, or standalone, or in cloud.

Limitations of Spark

1. Problem with small file
2. No File management system
3. Expensive
4. Manual optimization
5. Iterative processing
6. Latency
7. Window Criteria
8. Less number of algorithms
9. Does not support real-time processing.
10. Back pressure handling.

2.7 Flink

Apache Flink is an open source platform which is a streaming data flow engine that provides communication, fault-tolerance, and data-distribution for distributed computations over data streams. Flink is a top-level project of Apache. Flink is a scalable data analytics framework that is fully compatible to Hadoop. Flink can execute both stream processing and batch processing easily.

The development of Flink is started in 2009 at a technical university in Berlin under the stratosphere. It was incubated in Apache in April 2014 and became a top-level project in December 2014. Flink is a German word meaning swift/Agile. The logo of Flink is a squirrel, in harmony with Hadoop ecosystem.

The key vision for Apache Flink is to overcome and reduces the complexity that has been faced by other distributed data-driven engines. It is achieved by integrating query optimization, concepts from database systems and efficient parallel in-memory and out-of-core algorithms, with the MapReduce framework. As Apache Flink is mainly based on the streaming model, Apache Flink iterates data by using streaming architecture. The concept of an iterative

algorithm is tightly bounded into Flink query optimizer. Apache Flink's pipelined architecture allows processing the streaming data faster with lower latency than micro-batch architectures (Spark). An apache Flink ecosystem is shown in figure 12 and working of Flink is shown in figure 13.

Features of Flink

1. High performance
2. Low latency
3. Lightning fast speed
4. Fault Tolerance
5. Stream processing
6. Scalable

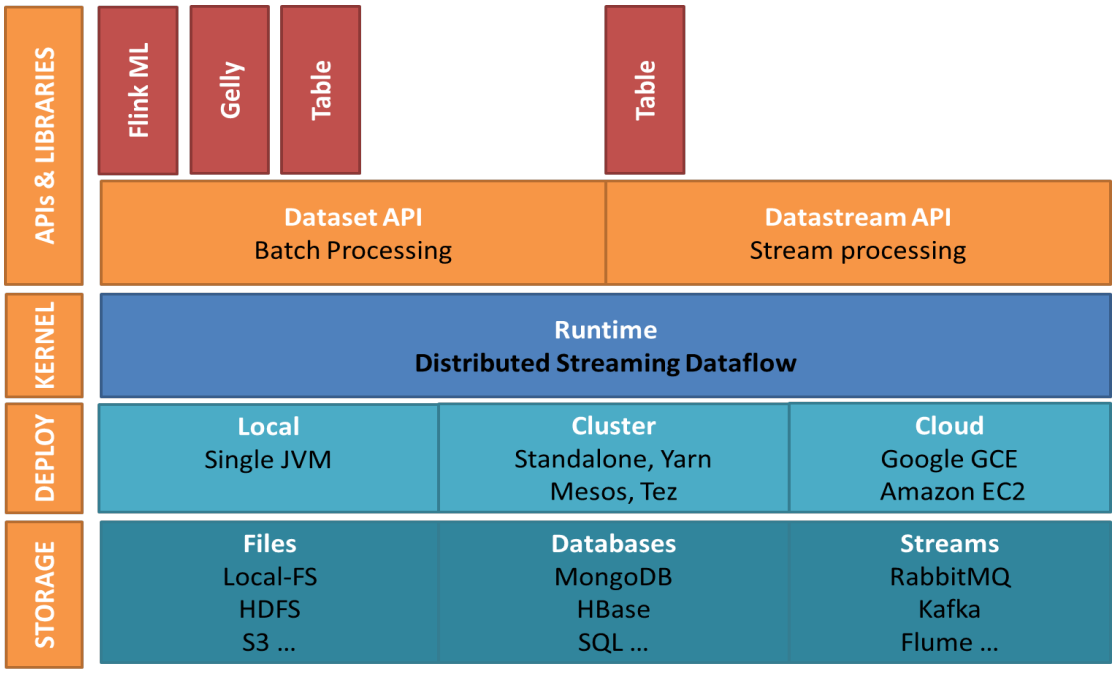

Fig 12. Apache Flink Ecosystem

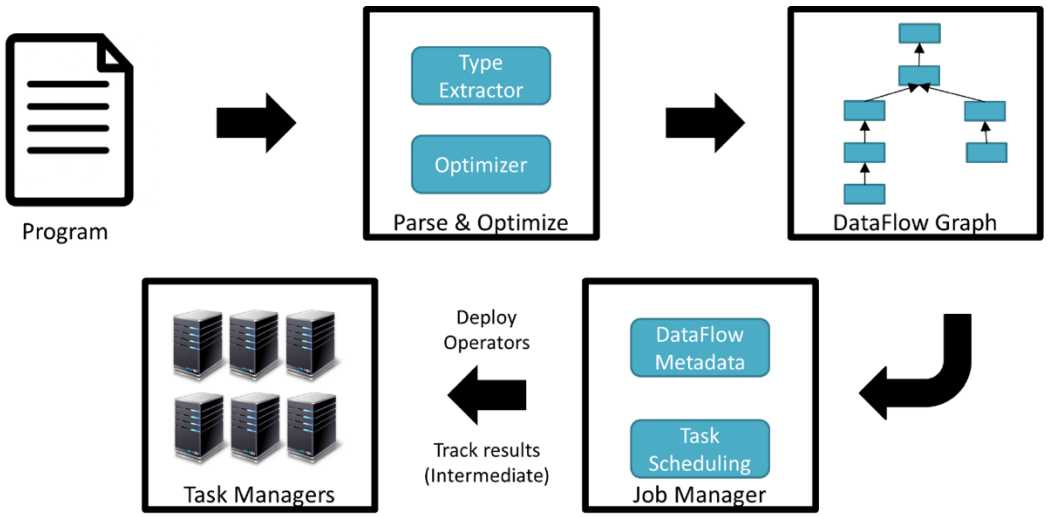

Fig 13. Flink Execution Model

2.8 Apriori Algorithm

Just as it is challenging for humans, transactional data makes association rule mining a challenging task for machines as well. Transactional datasets are typically extremely large, both in terms of the number of transactions as well as the number of items or features that are monitored. The problem is that the number of potential itemsets grows exponentially with the number of features. Given k items that can appear or not appear in a set, there are 2^k possible itemsets that could be potential rules. A retailer that sells only 100 different items could have on the order of 2^100 = 1.27e+30 itemsets that an algorithm must evaluate—a seemingly impossible task.

Rather than evaluating each of these itemsets one by one, a smarter rule learning algorithm takes advantage of the fact that, in reality, many of the potential combinations of items are rarely, if ever, found in practice. For instance, even if a store sells both automotive items and women's cosmetics, a set of {motor oil, lipstick} is likely to be extraordinarily uncommon. By ignoring these rare (and, perhaps, less important) combinations, it is possible to limit the scope of the search for rules to a more manageable size.

Much work has been done to identify heuristic algorithms for reducing the number of itemsets to search. Perhaps the most-widely used approach for efficiently searching large databases for

rules is known as Apriori. Introduced in 1994 by Rakesh Agrawal and Ramakrishnan Srikant, the Apriori algorithm has since become somewhat synonymous with association rule learning. The name is derived from the fact that the algorithm utilizes a simple prior (that is, a priori) belief about the properties of frequent itemsets.

Strengths	Weaknesses
• Is capable of working with large amounts of transactional data • Results in rules that are easy to understand • Useful for "data mining" and discovering unexpected knowledge in databases	• Not very helpful for small datasets • Requires effort to separate the true insight from common sense • Easy to draw spurious conclusions from random patterns

Table 1. Strengths and Weakness of Apriori Algorithm

The Apriori algorithm employs a simple a priori belief to reduce the association rule search space: all subsets of a frequent itemset must also be frequent. This heuristic is known as the Apriori property. Using this astute observation, it is possible to dramatically limit the number of rules to be searched. For example, the set {motor oil, lipstick} can only be frequent if both {motor oil} and {lipstick} occur frequently as well. Consequently, if either motor oil or lipstick is infrequent, any set containing these items can be excluded from the search.

Now let's consider a simple transaction database. The following table shows five completed transactions in an imaginary hospital's gift shop:

Transaction number	Purchased items
1	*{flowers, get well card, soda}*
2	*{plush toy bear, flowers, balloons, candy bar}*
3	*{get well card, candy bar, flowers}*
4	*{plush toy bear, balloons, soda}*
5	*{flowers, get well card, soda}*

By looking at the sets of purchases, one can infer that there are a couple of typical buying patterns. A person visiting a sick friend or family member tends to buy a get-well card and flowers, while visitors to new mothers tend to buy plush toy bears and balloons. Such patterns

are notable because they appear frequently enough to catch our interest; we simply apply a bit of logic and subject matter experience to explain the rule.

In a similar fashion, the Apriori algorithm uses statistical measures of an itemsets "interestingness" to locate association rules in much larger transaction databases. In the sections that follow, we will discover how Apriori computes such measures of interest and how they are combined with the Apriori property to reduce the number of rules to be learned.

Measuring Support & Confidence

Whether or not an association rule is deemed interesting is determined by two statistical measures: support and confidence measures. By providing minimum thresholds for each of these metrics and applying the Apriori principle, it is easy to drastically limit the number of rules reported, perhaps even to the point where only the obvious or common-sense rules are identified.

The support of an itemset or rule measures how frequently it occurs in the data. For instance, the itemset {get well card, flowers}, has support of 3 / 5 = 0.6 in the hospital gift shop data. Similarly, the support for {get well card} → {flowers} is also 0.6. The support can be calculated for any itemset or even a single item; for instance, the support for {candy bar} is 2 / 5 = 0.4, since candy bars appear in 40 percent of purchases. A function defining support for the itemset X can be defined as follows:

$$\text{support}(X) = \frac{\text{count}(X)}{N}$$

Here, N is the number of transactions in the database and count(X) is the number of transactions containing itemset X.

A rule's confidence is a measurement of its predictive power or accuracy. It is defined as the support of the itemset containing both X and Y divided by the support of the itemset containing only X:

$$\text{confidence}(X \rightarrow Y) = \frac{\text{support}(X, Y)}{\text{support}(X)}$$

Essentially, the confidence tells us the proportion of transactions where the presence of item or itemset X results in the presence of item or itemset Y. Keep in mind that the confidence that X leads to Y is not the same as the confidence that Y leads to X. For example, the confidence of {flowers} → {get well card} is 0.6 / 0.8 = 0.75. In comparison, the confidence of {get well card} → {flowers} is 0.6 / 0.6 = 1.0. This means that a purchase involving flowers is accompanied by a purchase of a get-well card 75 percent of the time, while a purchase of a get-well card is associated with flowers 100 percent of the time. This information could be quite useful to the gift shop management.

2.9 FP-Growth Algorithm

FP-tree based frequent itemset mining technique, called FP-Growth, created by Han et al accomplishes high proficiency, in comparison to Apriori-like approach. The FP-Growth technique embraces the divide-and-conquer system, utilizes just two full I/O scans of the database, and keeps away from iterative candidate generation. Frequent pattern mining consists of two steps:

1. Building a compact data structure, FP Tree (frequent pattern tree), which stores more data in less space.
2. Second is building of a FP-tree based pattern growth (FP-Growth) strategy to reveal every frequent pattern recursively.

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

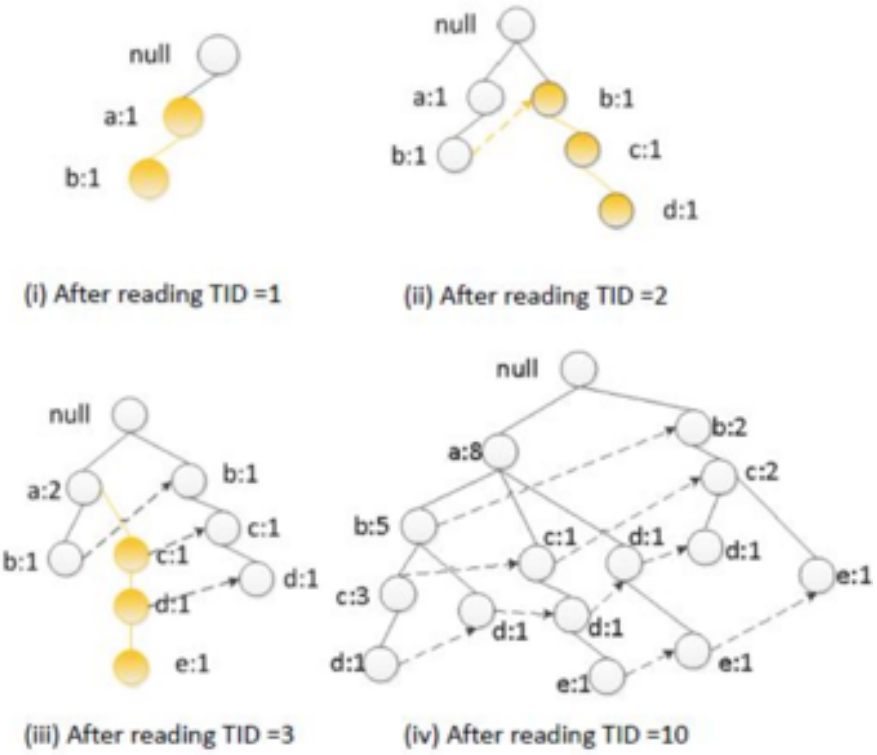

Fig 14. Construction of FP Tree

Constructing of FP-tree involves two scans on transaction database. The primary scan aggregates the support of every item and afterward chooses items that fulfil minimum support. This strategy produces frequent 1-itemsets and after that stores them in frequency descending order. The second scan builds FP-tree.

The FP-Tree is a compressed representation of the input. While reading the data source each transaction t is mapped to a path in the FP-Tree. As different transaction can have several items in common, their path may overlap. With this it is possible to compress the structure.

First a transaction t is read from the database. The algorithm checks whether the prefix of t maps to a path in the FP-Tree. If this is the case the support count of the corresponding nodes in the tree are incremented. If there is no overlapped path, new nodes are created with a support count of 1. Figure 15 shows the corresponding activity diagram using an UML (Unified Modelling Language) activity diagram.

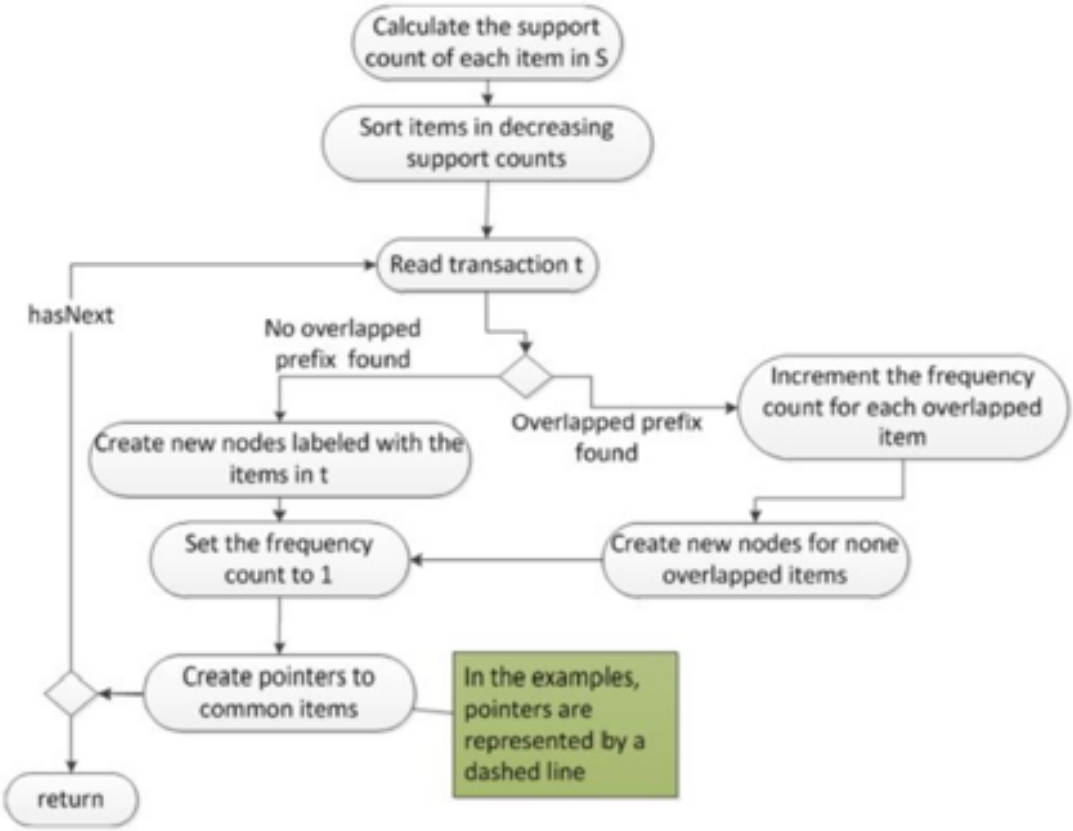

Fig 15. Activity Diagram

Additional a FP-Tree uses pointers connecting between nodes that have the same items creating a singly linked list. The corresponding FP-Tree is used to extract frequent item sets directly from this structure. Each node in the tree contains the label of an item along with a counter that shows the number of transactions mapped onto the given path.

In the best-case scenario, there is only a single node, because all transactions have the same set of items. A worst-case scenario would be a data source where every transaction has a unique set of items. Usually the FP-tree is smaller than the uncompressed one, because many transactions share items.

As already mentioned the algorithm has to scan the data source twice.

- Pass 1: The data set is scanned to determine the support of each item. The infrequent items are discarded and not used in the FP-Tree. All frequent items are ordered based on their support.
- Pass 2: The algorithm does the second pass over the data to construct the FP-tree.

The following example shows how the algorithm works

According to Figure 14 the first transaction is {a,b}. Because the tree is empty, two nodes a and b with counter 1 are created and the path null→a→b is created.

After {b, c, d} was read, three new nodes b, c and d have to be created. The value for count is 1 and a new path null→b→c→d is created. Because the value b was already in transaction one, there is a new pointer between the b's (dashed lines).

The transaction {a, c, d, e} overlaps with transaction one, because of the a in the first place. The frequency count for a will be incremented by 1. Additional pointers between the c's and d's are added.

After each transaction was scanned, a full FP-Tree is created. Now the FP-Growth algorithm uses the tree to extract frequent item sets.

Extract frequent item sets

A bottom-up strategy starts with the leaves and moves up to the root using a divide and conquer strategy. Because every transaction is mapped on a path in the FP-Tree, it is possible to mine frequent item sets ending in a particular item, for example e or d. So according to Figure 16, the algorithm first searches for frequent item sets ending with e and then with d, c, b and a until the root is reached. Using the pointers, each the paths can be accessed very efficient by following the list. Furthermore, each path of the tree can be processed recursively to extract the frequent item sets, so the problem can be divided into smaller subproblems. All solutions are merged at the end. This strategy allows to execute the algorithm parallel on multiple machines.

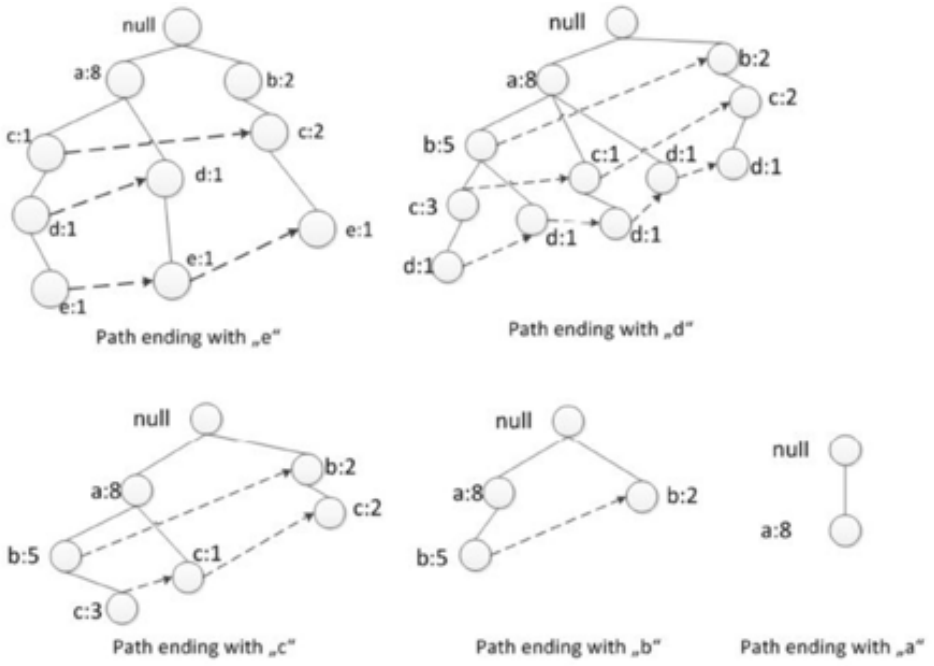

Fig 16. Extraction of frequent items from FP tree

The FP-Growth algorithm finds all item sets ending with a specified suffix using the divide and conquer strategy. Assume the algorithm analyses item sets ending with e. To do so, first the item set e has to be frequent. This can be done using the corresponding FP-Tree ending in e. If it is frequent, the algorithm has to solve the subproblem of finding frequent item sets ending in de, ce, be and ae. These subproblems are solved using the conditional FP-Tree. The algorithm for generating FP tree and extracting the frequent itemset are as follows:

1: INPUT: A file D consisting of baskets of items, a support threshold σ, and an item prefix I, such that $I \subseteq J$.
2: OUTPUT: A list of itemset $F[I](D,\sigma)$ for the specified prefix.
3: $F[i] \leftarrow \{\}$
4: for all $i \in J$ occurring in D do
5: $F[I] \leftarrow F[I] \cup \{I \cup \{i\}$
6: # Create D_I
7: $D_I \leftarrow \{\}$
8: $H \leftarrow \{\}$
9: for all $j \in J$ occurring in D such that $j > i$ do
10: if support $(I \cup \{i,j\}) \geq \sigma$ then
11: $H \leftarrow H \cup \{j\}$
12: for all $(tid, X) \in D$ with $I \in X$ do
13: $D_I \leftarrow D_I \cup \{(tid, X \cap H)\}$
14: # Depth-first recursion
15: Compute $F[I \cup \{i\}](D_I,\sigma)$
16: $F[I] \leftarrow F[I] \cup F[I \cup \{i\}]$

2.10 Related Work

She Xiangyang [6] presents an Apriori enhanced calculation of parallel association rules based on MapReduce. The strategy accomplishes its parallelization through the MapReduce structure, streamlines unique database to recreate the transaction record database and produces the frequent itemsets, and requests in rising the frequent item sets as per the support degree, at that point mines frequent item sets in the cluster.

Dachuan [7] Huang proposes new upgrades to the MapReduce usage of FIM calculation by presenting a cache layer and a particular online analyzer. They assessed the adequacy and productivity of Smart Cache by means of broad trials on four open datasets. Smart Cache can lessen by and large 45.4%, and up to 97.0% of the aggregate execution time compared with the state-of-the-art solution.

Feng Gui [8] proposed DPBM, an appropriated framework construct pruning calculation situated in light of Spark, which manage FIM (frequent Itemsets mining). DPBM incredibly decrease the measure of candidate itemset by presenting an innovative pruning strategy for matrix-based frequent itemset mining algorithm, an enhanced Apriori calculation which just needs to check the input data once. What's more, every PC node lessens enormously the memory utilization by implementing DPBM under a most recent distributed-environment Spark, which is an exceptionally quick distributed computing. The exploratory outcomes demonstrated that DPBM have preferable execution time over MapReduce-based calculations on frequent itemset mining as far as speed and scalability is concerned.

Jian Guo [9] presents CMR-Apriori calculation which depends on the conventional Apriori calculation that consolidates Map/Reduce parallel execution, with Map/Reduce programming model and related encoding operation. Through twice Map/Reduce process, CMR-Apriori calculation enormously decreases the running time of the algorithm, tackling issues utilizing proficient and exact calculations.

Yihua Huang [10] proposed YAFIM (Yet Another Frequent Itemsets Mining), a parallel Apriori calculation in light of the Spark RDD structure an uncommonly composed in-memory parallel processing model to bolster iterative calculations and intuitive information mining. Experimental results demonstrate that, contrasted with the calculations implemented with MapReduce, YAFIM accomplished 18× speedup in normal for different benchmarks.

Sheng-Hui Liu [11], introduced an enhanced reformative Apriori calculation that uses the length of every transaction to decide the extent of the most extreme candidates itemset. By reducing the creation of low frequency itemset in Map function, memory depletion is enhanced, incredibly enhancing execution effectiveness.

Run-Ming Yu [12] changed the conventional Apriori calculation by enhancing the execution productivity. Since the single-phase calculation just utilized only one MapReduce operation, it will produce unnecessary candidates itemset and result in deficient memory. He outlined and implemented a proficient algorithm: FPM (Frequent Patterns Mining) Algorithm solely based on MapReduce Framework (FAMR).

Ning Li [13] implemented a parallel Apriori calculation in light of MapReduce, which is a structure for handling tremendous datasets on specific sorts of distributable issues utilizing countless number of computer nodes. The test comes about which exhibits that the proposed calculation can scale well and proficiently handle substantial large datasets on commodity hardware.

Xueyan Lin [14] presented the MapReduce programming model of Hadoop platform and Apriori calculation of data mining, proposes the detailed steps of MR-Apriori calculation. Theoretical and experimental results indicated MR-Apriori calculation make a sharp increment in proficiency.

Zhuobo Rong [15] utilizes the possibility of MapReduce parallel programming, the great Apriori and FP-Growth calculation are relocated to the MapReduce environment keeping in mind the end goal to effectively take care of the current issues of Apriori and FP-Growth calculation in the conventional usage techniques, and address the needs of large-scale data association rules mining.

Manoj Sethi [16] proposes a new algorithm for frequent itemset mining called “Sandwich Apriori” which is a combination of Apriori and Reverse-Apriori with new improved pruning technique. The evaluation results showed that proposed approach is efficient in terms of execution time and number of candidate itemsets generated, then traditional Apriori.

Table 2 shows the summary of the techniques studied in related work.

Technique	Platform	Algorithm improved	Achieved	Remark
K-map Apriori	MapReduce	Apriori	Scalability and works in only K phase	High waiting time between two phases
MR-Apriori	MapReduce	Apriori	Scalability and works in only two phases	Perform insignificantly if generation of k-frequent itemset is huge, each node takes insignificant amount of time
IPARBC	MapReduce	Apriori	Performance is better	Permutation Process brings large complexity
DPBM	MapReduce	Apriori	Is more efficient	Scope for further Improvement
IMR-Apriori	MapReduce	Apriori	Perform well and much scalable	
SeaRum	MapReduce	Apriori	Parallelization of association rule extraction phase and provide SaaS platform	_
PRAMA	MapReduce	Apriori	Near-Linear speed-up, High scalability, reduce duplicates, Extract rules Directly	Combines Random sampling and Parallelization

YAFIM	Spark RDD	Apriori	Faster computation	Faster computation than MapReduce
NIMBLE	NIMBLE	Apriori	Portable, support rapid prototyping	Designed for fast and efficient implementation of MLDM algorithms
PEMA	MapReduce	Apriori	reduced the response time and communication cost	used only for homogeneous DARM environment
R-Apriori	MapReduce	Apriori	improved performance as size of the dataset and no. of items increases	
FIM using distributed computing	MapReduce	Apriori	Gives better time complexity and space complexity	
DPA	MapReduce	Apriori	Reduce Processor idle time	Have certain limitation

Table 2. Summary of Techniques

2.11 Chapter Summary

This chapter identified the features and limitations of MapReduce frameworks and the famous Apriori and FP-growth algorithms. The various studies proposing advanced models of these two algorithms has been discussed in related work section.

CHAPTER 3

SYSTEM PARADIGM

This chapter illustrates the approach that helps in understanding the behavior of the Apriori and FP-growth algorithm on different MapReduce frameworks while working on variety of Datasets. We also compared both the algorithms on different platforms. Section 3.1 gives an overview of the research undertaken. Section 3.2 portraits the architectural view of the proposed paradigm. Section 3.3 describes each module of the system and how it works. Lastly, Section 3.4 gives the summary of the chapter.

3.1 Proposed Framework

Due to the increase in Web services and use of computers in most of the businesses, the amount of data available online and offline has changed drastically in terms of volume as it has become a global source of useful information. Analyzing such an amount of data manually is impossible, so the researchers has proposed various algorithms to analyses and for mining the patterns available in data to predict the current trends in the society going on. Some platforms have also been developed to ease this process for organizations. But each organization has different type of data in terms of size, rate of increase of data, number of attributes, structured or unstructured data. Not every algorithm or platform is suitable in all the circumstances.

Use of association rules in understanding the patterns in data is increasing day by day. Association rules are if then statements, which relate two or more terms by analyzing the frequent occurrence of the more than two articles together. Association rules has revolutionized the Advertising industry, how to make a customer buy a product has become a lot easier than earlier times. Earlier, the products were mainly available at the brand's store, so when a person goes to buy a product, say toothpaste, by applying the association mining by the owner, the chances of buying the mouth freshener or tooth brush are more than combination of toothpaste and room freshener. So, to increase the sale of products the items were arranged in such a

manner that if person came to buy one thing, his chances of buying other thing rises. This method of rising the sales is association rule mining, but was applied manually.

These days, a lot of people do shopping online, so now to analyses that huge data we need some code which will automatically detect the frequent items that has been sold together. A lot of algorithms has been proposed, the most famous ones are Apriori and FP-growth. In digital world, when a customer selects an item to add to his cart, the association rule mining is done to suggest the items that they might brought together as indicated by the stored data. For example, while purchasing a kurta, the matching lagging will be shown as in recommended section, and the earrings of the same color are shown, as it was deducted from previous sales data, that a person usually brought matching legging with the kurta. Similarly, when we select a pizza for home delivery, it starts suggestions regarding adding cold-drink and other sides like garlic bread, but it will not show pasta in the recommendations, as not many people purchase pizza and pasta together. So, to upraise the business association rule based algorithms for frequent item set mining algorithms are used by most of the organizations these days.

To understand the applicability of MapReduce Frameworks, Apriori and FP-Growth were implemented on three different datasets of different sizes on Apache Hadoop, Apache Spark and Apache Flink. The results of these two algorithms are compared on all three platforms on three different dataset conditions to recognize the conditions suitable for each algorithm in different situations.

In addition to the obvious value to the advertising industry, the research community has long sought mechanisms to effectively disseminate new scientific discoveries and technological breakthroughs so as to advance our collective knowledge and elevate our civilization.

3.2 Architectural View

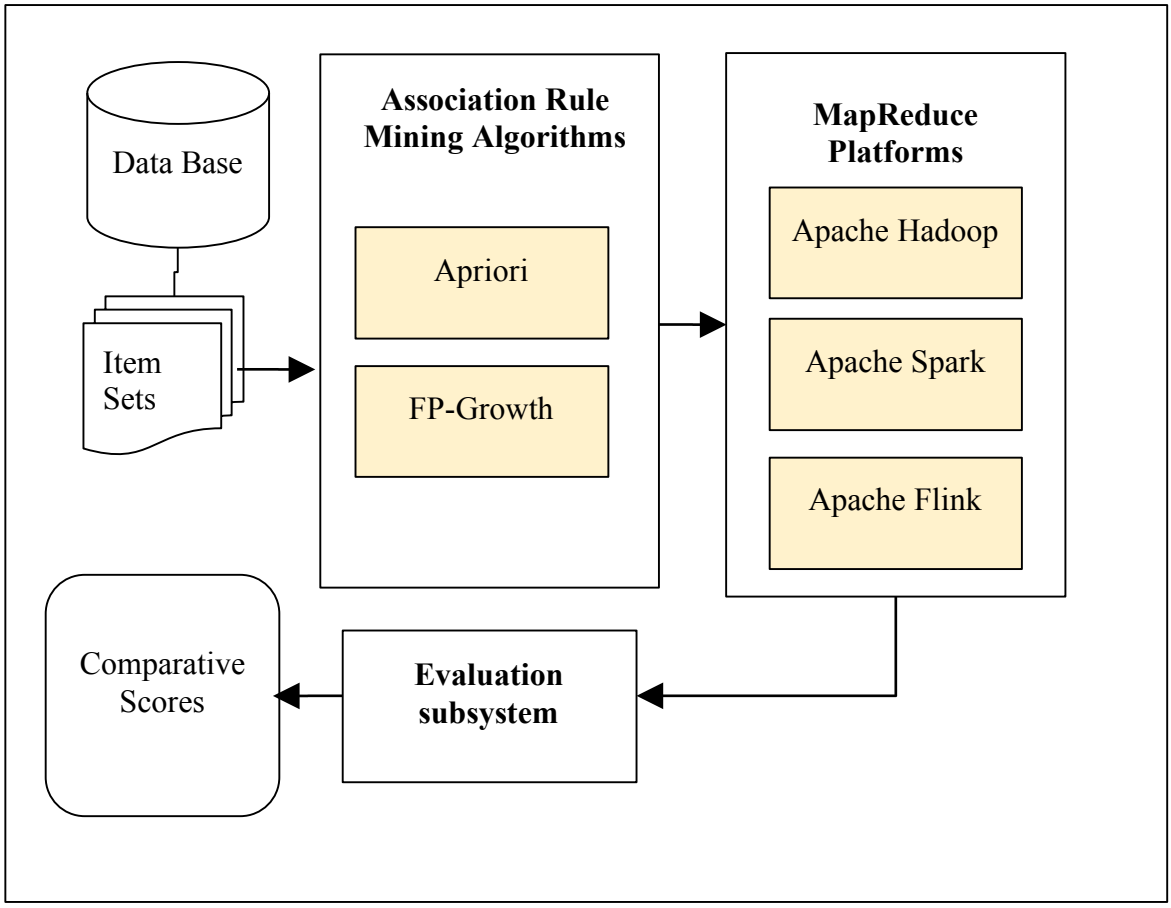

Fig 17. Pictorial view of System paradigm

The system firstly retrieves the data from online stores. The data is converted into item sets, each item set represents items brought together. Both the association rule mining algorithms are run on these three datasets separately on each of the MapReduce platforms. The time taken to generate the frequent items are recorded for each of the six implementations during the three trails for different data sets. The average of the three iterations are averaged to conclude the algorithm that performs good in all the conditions. Figure shows the overview of the system proposed in this research.

3.3 Chapter Summary

This chapter explains the work done in this project.

CHAPTER 4

IMPLEMENTATION

In this chapter, we will discuss the experimental setup of the research work done. First section will discuss the data set followed by programming tools used for programming. In the next section, a case study is discussed to elaborate the working. In the last section summary of the chapter is given.

4.1. Data Set

The data set used in this research is collected from SPMF- An open source data mining library. It is a repository which specialized in pattern mining. We have taken three data sets from this data mining library of different size. The first data set is named as 'Food Mart', it is a dataset of customer transactions from a retail store, it contains 4141 entries. Entries are regarding 1554 different items. Second dataset is T1014D100K which contains 870 number of items and 100000 number of transaction. Third dataset is termed as 'Online Retail', it is transformed from the Online retail dataset, it contains 541909 transaction entries of 2603 items.

4.2. Programming Tool

The three MapReduce frameworks- Apache Hadoop, Apache Spark and Apache Flink are used. The coding of two algorithms in these three platforms is done in java and python.

4.3. Evaluation Methods

To evaluate the performance of the different MapReduce frameworks 'time' is taken as the evaluation metric to fetch the frequent itemset.

4.4. Programming Tools and software used

Operating System: Windows 10

Language used: Python, Java

Dependencies: Maven, Apache Commons Language 3.4

Mining Tool: Hadoop, Spark, Flink

CHAPTER 5

RESULTS & ANALYSIS

In this chapter, we show the results obtained by our work. Section 5.2 analyze the results to understand which algorithm perform best in which framework.

5.1 Output

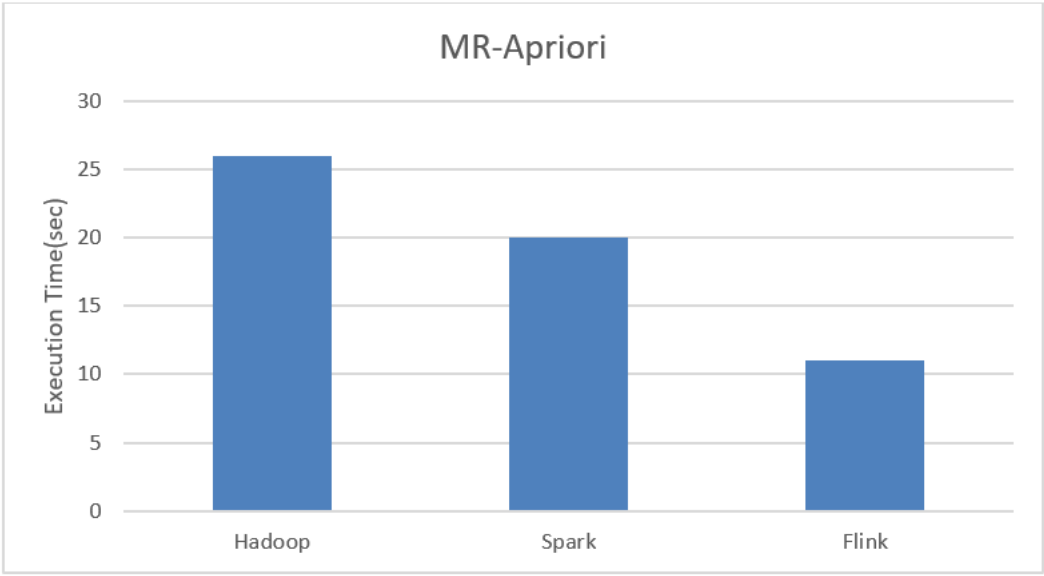

Figure 18. Food Mart with Minimum Support 0.1%

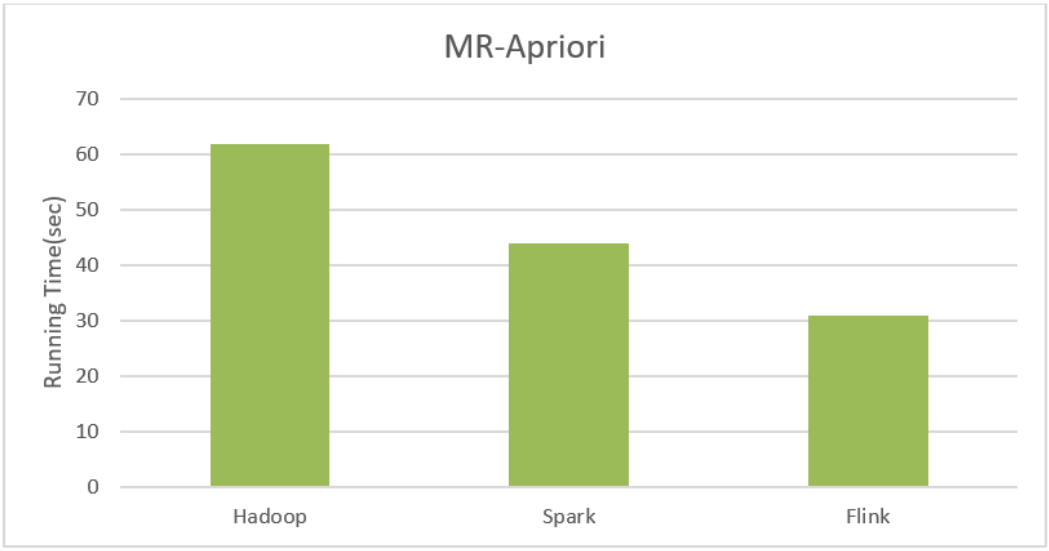

Figure 19. T1014D100K with Minimum Support 0.3%

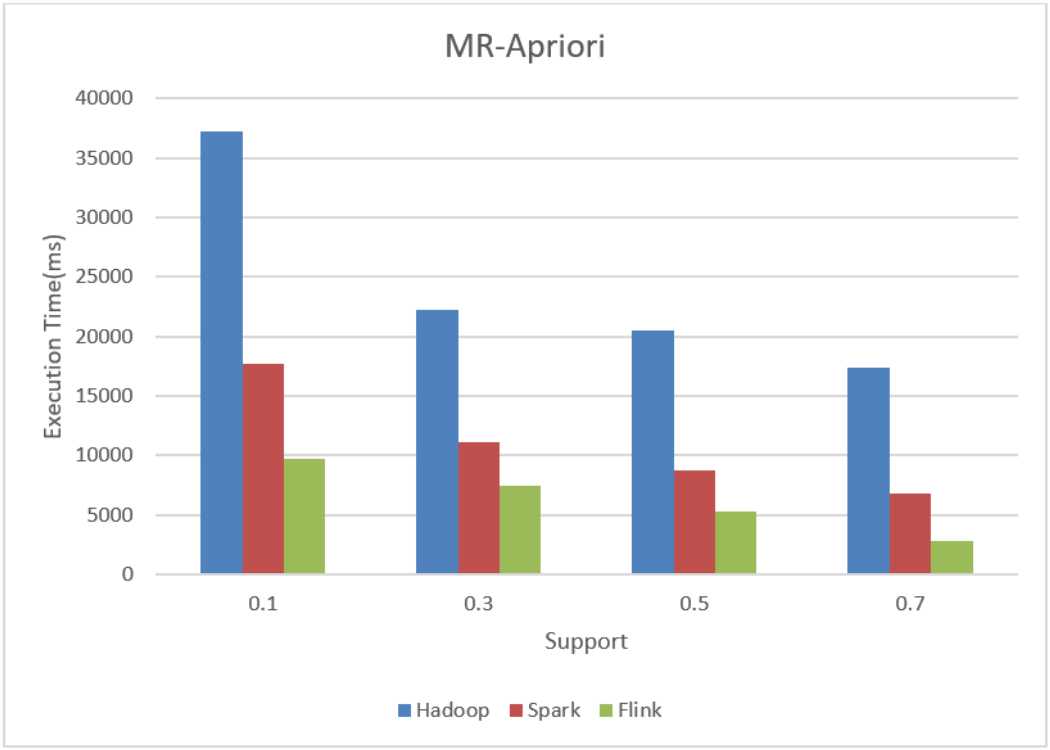

Figure 20. Online Retail with Minimum Support 0.5%

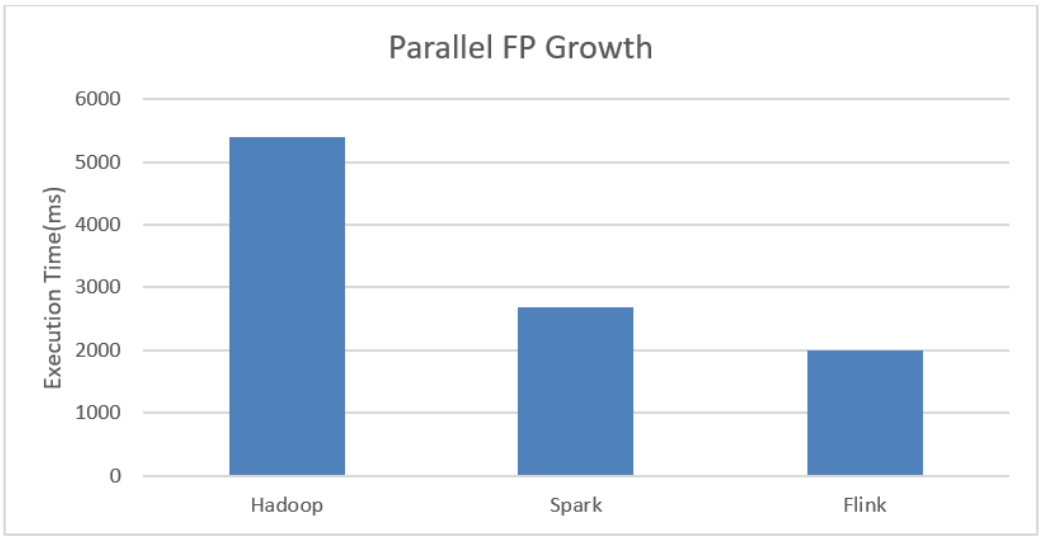

Figure 21. Food Mart with Minimum Support 0.3%

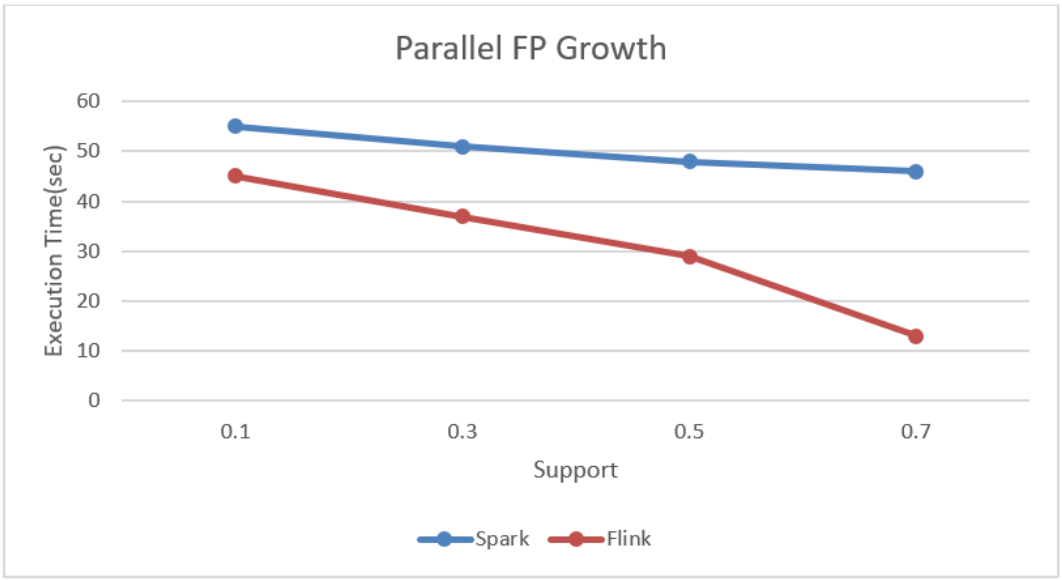

Figure 22. T1014D100K with Minimum Support 0.5%

5.2 Analysis

For Food Mart dataset with around 4000 transaction and minimum support of 0.1%, Hadoop takes approximately 26 seconds whereas Spark and Flink takes 20 and 11 seconds, respectively. For T1014D100K dataset with around 100000 transaction and minimum support of 0.3%, Hadoop takes approximately 61 seconds whereas Spark and Flink takes 44 and 31 seconds, respectively.

Figure 20 shows the scalability of MR Apriori algorithm on three different platforms for the Online Retail dataset with around 500000 transaction and minimum support of 0.5%. Figure 21 compares the performance of Parallel FP Growth algorithm on three different platforms. Figure 22 analyses the scalability of PFP on Spark and Flink on dataset T1014D100K with minimum support of 0.5%.

CHAPTER 6

CONCLUSION

This chapter concludes the contributions made by this thesis. Also figure out the limitation of the work done and briefly discuss the future scope of the research.

6.1 Research Summary

The study in this work presents Flink based MR Apriori and Parallel FP Growth which is applied to mine frequent patterns from extensive datasets. It utilizes essential Apriori requirement that an itemset must be frequent if only all its non-empty subset is frequent. It is executed on Apache Flink, Apache Spark and Apache Hadoop which gives parallel and distributed processing condition. Flink is most appropriate for Apriori in light of the fact that Apache Flink have local support for iterative calculation and Apriori is based upon iterative calculation. Flink's pipelined design enable us to begin another Apriori iteration when few results of previous iteration are available. Delta cycle usefulness of Flink makes Apriori exceptionally parallel and powerful calculation for colossal datasets. In Summary, we have presented an execution of Apriori and FP Growth on Hadoop, Spark and Flink and tried to compare with various datasets. We also demonstrated that Flink based Apriori is equipped for dealing with extensive transactional-based datasets effortlessly.

6.2 Limitations

In this research experiments were conducted in controlled environment and virtual machine were used to simulate the behavior of distributed environment. Also, one node cluster was used for results.

6.3 Future Scope

Since Flink is the one of the most recent advancement in the field of Big data, not much work has been conducted to see how it performs with other distributed platforms. Also, not even a single algorithm in the field of association rule mining is introduced till date. Our work can be extended to cover large computer clusters dataset with more than one tera bytes. Additionally, we can apply this parallel version of Apriori and FP Growth to various application domain such as weather data, internet traffic, medical information etc. We can also use these algorithms to generate different and interesting association rule faster and effectively.

REFERENCES

[1] Apache Hadoop, http://hadoop.apache.org/

[2] Apache Spark, http://spark.apache.org/

[3] Apache Flink, https://flink.apache.org/

[4] http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

[5] http://fimi.ua.ac.be/data/

[6] She Xiangyang, Zhang Ling "Apriori Parallel Improved Algorithm Based on MapReduce Distributed Architecture" Published in: Instrumentation & Measurement, Computer, Communication and Control (IMCCC), 2016 Sixth International Conference on July 2016 DOI: 10.1109/IMCCC.2016.59

[7] Dachuan Huang, Yang Song, Ramani Routray, Feng Qin "Smart Cache: An Optimized MapReduce Implementation of Frequent Itemset Mining" Cloud Engineering (IC2E), 2015 IEEE International Conference on March 2015 DOI: 10.1109/IC2E.2015.12

[8] Feng Gui, Yunlong Ma, Feng Zhang, Min Liu, Fei Li, Weiming Shen, Hua Bai "A distributed frequent itemset mining algorithm based on Spark" Computer Supported Cooperative Work in Design (CSCWD), 2015 IEEE 19th International Conference on May 2015 DOI: 10.1109/CSCWD.2015.7230970

[9] Jian Guo, Yong-gong Ren "Research on Improved A Priori Algorithm Based on Coding and MapReduce" Published in: Web Information System and Application Conference (WISA),on Nov.2013 DOI: 10.1109/WISA.2013.62

[10] Hongjian Qiu, Rong Gu, Chunfeng Yuan, Yihua Huang "YAFIM: A Parallel Frequent Itemset Mining Algorithm with Spark" Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International Conference on May 2014 DOI: 10.1109/IPDPSW.2014.185

[11] Sheng-Hui Liu, Shi-Jia Liu, Shi-Xuan Chen, Kun-Ming Yu "IOMRA - A High Efficiency Frequent Itemset Mining Algorithm Based on the MapReduce Computation Model " Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on Dec 2014 DOI: 10.1109/CSE.2014.247

[12] Run-Ming Yu, Ming-Gong Lee, Yuan-Shao Huang, Shi-Xuan Chen "An efficient Frequent Patterns Mining Algorithm based on MapReduce

Framework" Published in Software Intelligence Technologies and Applications & International Conference on Frontiers of Internet of Things 2014, International Conference on Dec. 2014 DOI: 10.1049/cp.2014.1525

[13] Ning Li, Li Zeng, Qing He, Zhongzhi Shi "Parallel Implementation of Apriori Algorithm Based on MapReduce" Published in Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on Aug. 2012 DOI: 10.1109/SNPD.2012.31

[14] Xueyan Lin "MR-Apriori: Association Rules algorithm based on MapReduce" Published in Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on June 2014 DOI: 10.1109/ICSESS.2014.6933531

[15] Zhuobo Rong, Dawen Xia, Zili Zhang "Complex statistical analysis of big data: Implementation and application of Apriori and FP-Growth algorithm based on MapReduce" Published in Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on May 2013 DOI: 10.1109/ICSESS.2013.6615467

[16] Tarinder Singh; Manoj Sethi, "Sandwich-Apriori: A combine approach of Apriori and Reverse-Apriori 2015 Annual IEEE India Conference (INDICON) Year: 2015 Pages: 1 - 4, DOI: 10.1109/INDICON.2015.7443786

[17] R. Agrawal, T. Imielinski and A. Swami, “Mining Association Rules Between Sets of Items in Large Databases,” in ACM SIGMOD Conf. Management of Data, Washington, D.C., pp. 207–216, (1993).

[18] Kitchenham and Charters, “Guidelines for performing Systematic Literature Reviews in Software Engineering”, 2007, Elsevier

[19] Honglie Yu, Jun Wen and Hongmei Wang. An Improved Apriori Algorithm Based On the Boolean Matrix and Hadoop. In International Conference on Advanced in Control Engineering and Information Science (CEIS), pp.1827-1831, 2011.

[20] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In Proc. OSDI. USENIX Association, 2004.

[21] J. Han, H. Pei and Y. Yin. Mining Frequent Patterns without Candidate Generation. In Proc. Conf. on the Management of Data (SIGMOD’00, Dallas, TX), ACM Press, New York, NY, USA 2000.

[22] Lan Vu and Gita Alaghband. Novel Parallel Method for Mining Frequent Patterns on Multi-core Shared Memory Systems. In ACM conference , Denver USA , 49-54, 2013.

[23] Li N., Zeng L., He Q. & Shi Z. Parallel Implementation of Apriori Algorithm Based on MapReduce. In Proc. of the

[24] 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD '12), Kyoto, IEEE: 236 – 241, 2012.

[25] Li N., Zeng L., He Q. & Shi Z. Parallel Implementation of Apriori Algorithm Based on MapReduce. In Proc. of the 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD '12), Kyoto, IEEE: 236 – 241, 2012.

[26] Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara and Wei Li. New algorithms for fast discovery of association rules. Technical Report 651, Computer Science Department, University of Rochester, Rochester, NY 14627. 1997.

[27] Yang X.Y., Liu Z. & Fu Y. MapReduce as a Programming Model for Association Rules Algorithm on Hadoop. In Proceedings of the 3rd International Conference on Information Sciences and Interaction Sciences (ICIS '10), Chengdu, China, IEEE: 99 – 102, 2010.

[28] Yeal Amsterdamer, Yeal Grossman, Tova Milo and Pierre Senellart. CrowdMiner: Mining association Rules from the crowd.In Proceedings of VLDB Endowment, 2013.

[29] Zahra Farzanyar and Nick Cercone. Efficient Mining of Frequent Itemsets in Social Network Data based on Mapreduce Framework. In 2013 IEEE/ACM International Conference on Advances in Social Network Analysis and Mining, 1183-1188, 2013

[30] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang, "Pfp: Parallel fp-growth for query recommendation," in Proceedings of the 2008 ACM Conference on Recommender Systems, ser. RecSys '08. New York, NY, USA: ACM, 2008, pp. 107–114. [Online]. Available: http://doi.acm.org/10.1145/1454008.1454027

[31] L. Zhou, Z. Zhong, J. Chang, J. Li, J. Huang, and S. Feng, "Balanced parallel fp-growth with mapreduce," in Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference on, Nov 2010, pp. 243–246.

[32] I. Pramudiono, K. Takahashi, A. KH Tung, and M. Kitsuregawa, “Procssing Load Prediction for Parallel FP-Growth,” in Proc. 16th Institute of Electronics, Information and Communication Engineers Data Engineering Workshop (DEWS2005), 2005.

[33] Rini Joy; K. K. Sherly "Parallel frequent itemset mining with spark RDD framework for disease prediction" 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT) Year: 2016 Pages: 1 - 5, DOI: 10.1109/ICCPCT.2016.7530360

[34] Xiaoting Wei; Yunlong Ma; Feng Zhang; Min Liu; Weiming Shen "Incremental FP-Growth mining strategy for dynamic threshold value and database based on MapReduce" In Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD) Year: 2014 Pages: 271 - 276, DOI: 10.1109/CSCWD.2014.6846854.