CHAPTER 1 - INTRODUCTION

1.1 Background

Advancements in software and hardware infrastructure has made it easier to gather data from almost all aspects of our daily lives; from web user behaviors, sensor networks, body organ monitoring devices, astronomical monitoring to millions of conversations on social media. Data is no longer ordinary data as it has moved from gigabytes to petabytes. Data is now called Big Data. Since the advent of the big data era, the technological playing field has taken a new look. Almost all areas in the Information Communication and Technology have been massively impacted; some negatively and others positively. The data mining field has become increasingly important as companies, governments and big multinational organizations have been presented with the complication of analyzing tones of meaningless raw data. Most operations have moved from being centralized to distributed and the Internet has become an integral part of most operations. This has also brought about the invention of Cloud Computing which has gained popularity due to its mobility, huge availability and low cost. Data mining nowadays is in essence performed in the cloud.

1.2 Big Data

Big data is a relatively new term which refers to large volumes of data which comes from various sources at different velocity, different formats (text, audio, video, images) and with different layout or structure (structured, semi-structured and unstructured).

> According to BigDataPlanet.com [11], “Dataset whose volume, velocity, variety and complexity are beyond the ability of commonly used tools to capture, process, store, manage and analyze them can be termed as BIGDATA”

The size of Big data is ever-changing since the data is gathered continuously for instance from sensors used to gather climate information, posts to social media sites such as Facebook and Twitter, digital pictures and videos, retail transaction records, and mobile phone GPS signals just to name a few. Big data has become an integral part of most recent researches. The graph below outlines the various dimensions by which big data can be approached from.

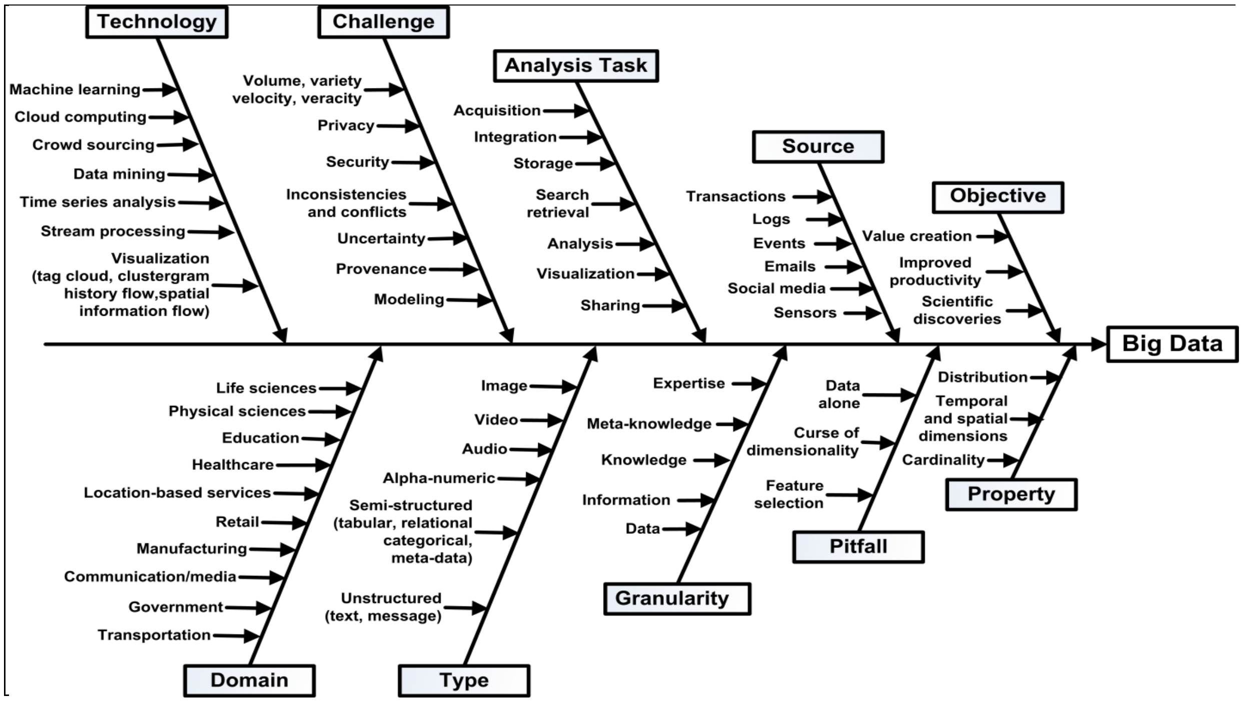


Figure 1: Dimensions of Big Data [13]

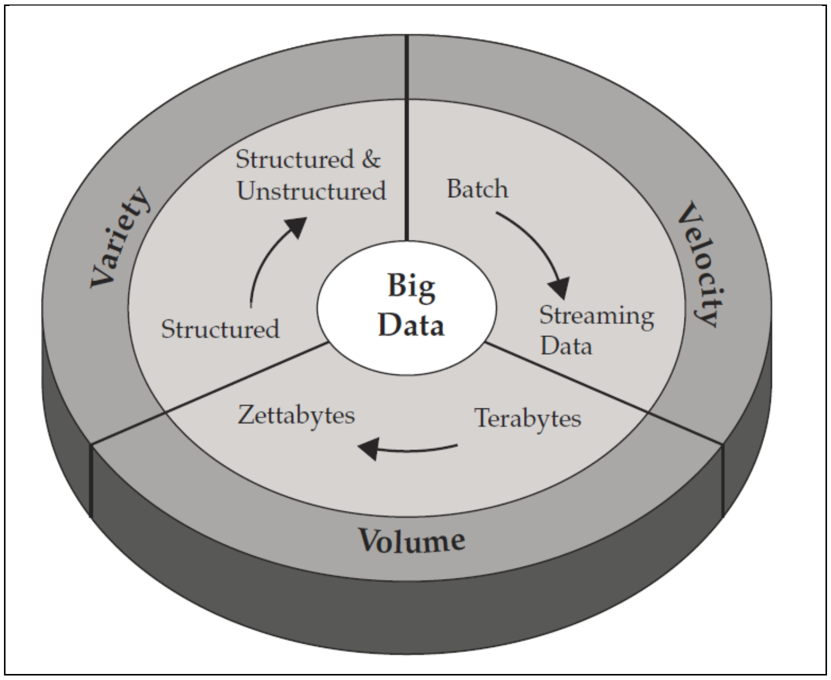


Figure 2:Characteristics of Big Data [13]

1.2.1 Characteristics of Big Data

Big data is characterized it by the following attributes:

1.2.1.1 Volume

This refers to the amount of data that is generated. Big Data is not measured in Gigabytes but rather in the order of Petabytes. Data may be generated from various sources and combined to increase the volume of data that has to be analyzed or stored. This data may grow to the tune of several Yotabytes, Exabyte and Zettabyte in the near future. The volume also includes numerous data sources each of which may be holding very large quantities of data which when combined contributes to substantial volumes of data. Conventional data storage methods can no longer cope with such enormous volumes.

1.2.1.2 Variety

This aspect refers to the diversity of data formats. Since data may come from various sources it is highly likely that the forms will be different. Traditionally data used to be structured and textual. This is no longer the case; data now includes audio, video, images and also text. The data may also be static or streams. In relation to the variety of data, Big Data can also be categorized as:

- Structured data: This type describes data which can fit well into a fixed relational schema within a standard SQL database. Structured data can be easily stored, manipulated and analyzed.
- Semi-structured data: This is a form of structured data that does not conform to an explicit and fixed schema. Examples include weblogs and social media feeds.
- Unstructured data: This type of data consists of formats which cannot easily be indexed into relational tables for manipulation such as social media interactions, audios and videos of recorded meetings, text documents, fax transfers, emails, medical information such as X-ray imagines and more [9

1.2.1.3 Velocity

The term ‘velocity’ refers to the speed with which the data is generated, gathered and processed to meet the demands of a particular domain. Many big data sources are dynamic and gather data continuously for instance sensor networks for weather monitoring. Below is a research carried out by Hewlett-Packard which shows the speed with which global data is growing per minute.

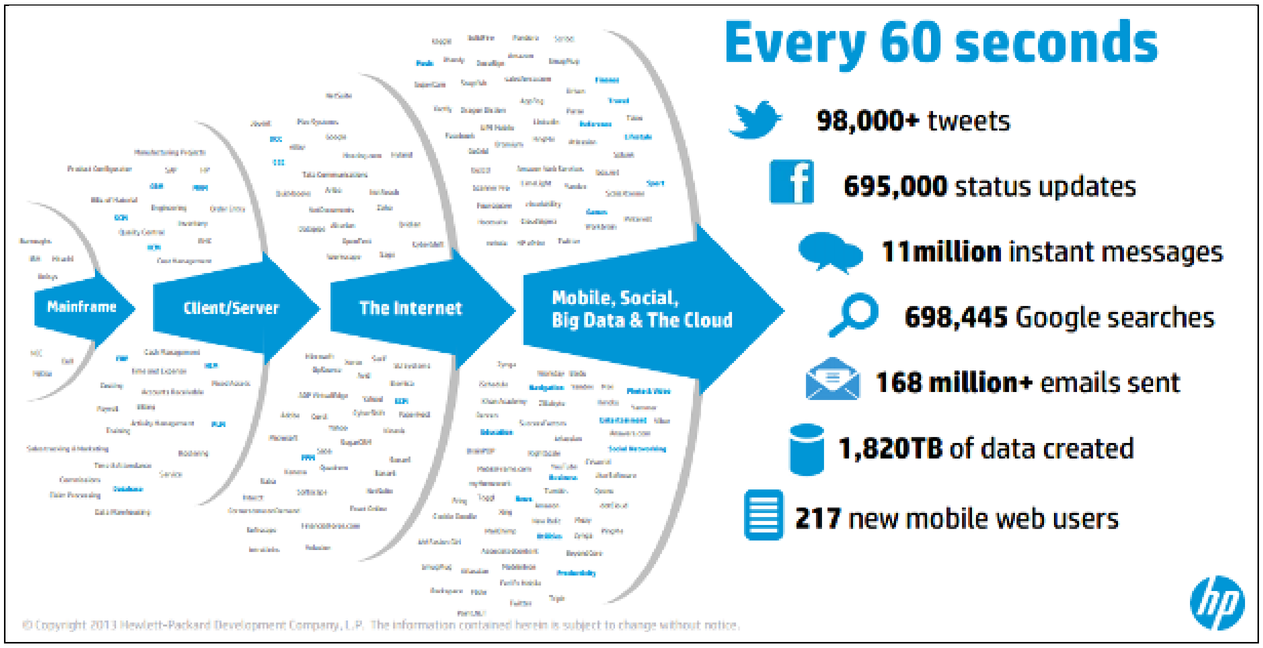

Every 60 seconds
98,000+ tweets
695,000 status updates
11million instant messages
698,445 Google searches
168 million+ emails sent
1,820TB of data created
217 new mobile web users
Mainframe
Client/Server
The Internet
Mobile, Social, Big Data & The Cloud
© Copyright 2013 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice.

Figure 3: Rate at which data is accumulating [11]

1.2.1.4 Variability

This refers to the inconsistency and unpredictability which can be shown by the data. Since data comes from several sources in so many different formats, it is most likely that the data will be inconsistent that is same type of data can be interpreted differently by different sources or may have some missing attributes.

1.2.1.5 Veracity

Since big data is collected from a diverse number of sources and/or sites the data quality being captured can vary greatly; this phenomenon is called veracity. The credibility and the reliability of the analysis largely depend on the veracity of the data sources.

1.2.1.6 Value

Data has become a valued asset for most companies. This is mainly because from the heaps of data they can derive some business intelligence which can give them a competitive edge over their competitors. Hence data now has a value attached to it. The quality of information that can be derived from the data depends on the quality of the data. The higher the quality, the higher the value.

1.2.2 Categories of Big Data

There are several categories of Big Data mainly differentiated by the sources. Some have been summarized in Table 1 below.

Table 1: Categories of Big Data

Categories	Example
Web and social media data	Facebook and Twitter posts.
Machine-to-machine data	Sensor networks, ad-hoc networks, Personal Area Networks (PAN)
Big transaction data	Health care claims, billing records, online product reviews.
Biometric data	Fingerprints, genetics, handwriting, retinal scans, X-rays, blood pressure, pulse and pulse-oximetry readings
Human-generated data	Unstructured and semi-structured data such as electronic medical records (EMRs), physicians’ notes, email, and paper documents

1.3 Data mining with big data

In essence, data mining is the process of discovering or finding some new, valid, understandable and potentially useful forms of data. The form of data refers to a discovered regularity among the data variables. If the detected regularity applies to all data, then it is about discovered model, if, however, the regularity can be correlated with the extent of data – it is a pattern or template. Data mining is carried out over large volumes of data in order to “pull“new information out of them that will be the basis for making (better) business decisions [14].

The most prominent challenge for Big Data applications is to search the enormous volumes of data and extract desired intelligence and patterns necessary for decision making. Before one can start mining big data he has to understand the characteristics of the data.

1.3.1 HACE Theorem [12].

According to [12] the HACE theorem explains the characteristics of big data that give birth to challenges in mining big data. Big Data starts with enormous data volumes, **H**eterogeneous; **A**utonomous sources which are usually distributed and do not have centralized control, and intends to explore **C**omplex and **E**volving relationships among data [12].

1.3.1.1 Heterogeneity

Big data does not emanate from a single source but rather collected by several data collectors who are different. Diverse information collectors prefer their own data structure or rules and procedures for data gathering. The design and structure of different applications also results in a variety data representation [12]. For example in collecting information about a patient; one might opt to use simple demographic information such as gender, race, age or family disease history others may go for biometric signatures. This presents different ways of capturing information about the same entity.

1.3.1.2 Autonomous Sources with Distributed and Decentralized Control

Distributed and autonomous data source is the norm in big data. Data sources usually have decentralised control; this implies that each data source is able to work independently and generate and collect information on its own without relying on any centralized control. This exposes the data to external attacks since the points of entry are increased.

1.3.1.3 Complex and Evolving Relationships

This challenge is brought about by the ever pouring streams of data. As the data volumes increase so does the complexity and relationships underneath. Data relations are no longer as straight forward as they used to be in traditional systems. In big data the heterogeneity and vastness of data makes it difficult to discover patterns.

1.3.2 Challenges in mining big data

For any system to successfully discover useful information from big data it has to meticulously address the characteristics outlined in the HACE Theorem. According to [12] a big data processing framework requires three tiers to produce desired outcome. These tiers are: Tier 1- data accessing and computing, Tier II- data privacy and domain knowledge and Tier III -Big Data mining algorithms. Each tier possesses its own challenges that need to be addressed in order to effectively mine the data.

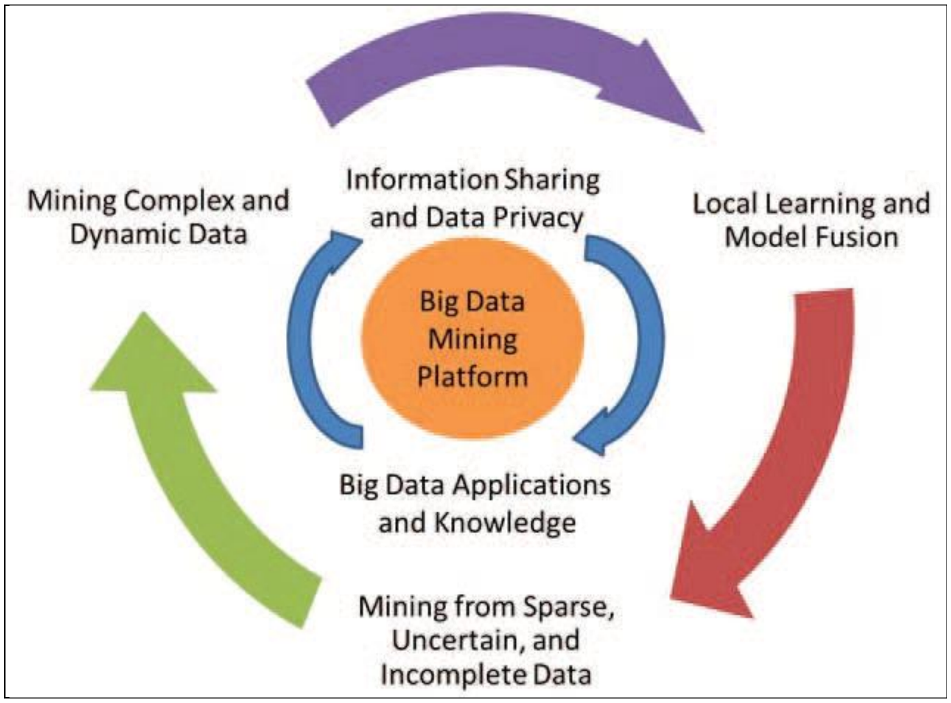

Mining Complex and Dynamic Data
Information Sharing and Data Privacy
Local Learning and Model Fusion
Big Data Mining Platform
Big Data Applications and Knowledge
Mining from Sparse, Uncertain, and Incomplete Data

Figure 4: A Big Data processing framework [12]

1.3.2.1 Big Data Mining Platform [12]

Big Data are usually distributed across various geographical locations and data volumes constantly grow. For a computing platform to be effective, it has to take into account distributed large-scale data storage. A clear technical barrier that exists for conventional data mining algorithms is that all data is required to be loaded into the main memory. Another prohibitive factor is that of moving data across different locations; this activity is expensive and risky. Sensitive data requires extra security for it to be transmitted across a network. [12]

Data involved in medium scale data mining tasks, are typically large and mostly distributed hence cannot be accommodated in main memory. Parallel computing and collective mining are the most common solutions to this challenge. They are used to sample and integrate data from different sources [12].

The data scale involved in Big Data Mining is far beyond the threshold capacity that a mainframe computer can handle, therefore Big Data processing framework normally rely on cluster computers connected on a high-performance computing platform. The clusters usually have some parallel programming tools, of which the far most popular is MapReduce. [12]

1.3.2.2 Big Data Semantics and Application Knowledge [12]

In Big Data, semantics and application knowledge refer to several aspects related to the rules and regulations, protocols and policies, user knowledge and domain information. Two issues of paramount importance at this tier are:

i. **Data sharing and privacy**: how data are maintained, accessed, and shared**;** [12]

Big data involves multiple parties which automatically imply information sharing. Information sharing brings data privacy and protection into play. As information is shared among users it has to be protected from unauthorized access and malicious attacks. Security measures such as certificates and other authenticating mechanisms have to be put in place. Anonymization of data fields such that data fields are not easy to pinpoint can also help to protect the data from attacks. The thrust behind data anonymization, is to introduce randomness into the data to achieve a number of privacy goals. Common anonymization approaches are to use perturbation, generalization, suppression and permutation to generate a modified version of the data. Once data is anonymized, it can be shared across several parties without hesitation or involving strict or restrictive access controls such as encryption.

ii. **Domain and application knowledge:** answers questions like “what are the underlying applications?” and “what are the knowledge or trends users intend to discover from the data?” [12].

1.3.2.3 Big Data Mining Algorithms

Data mining algorithms for big data have to be designed differently from traditional data mining algorithms. There are three fundamental challenges that big data mining algorithms have to overcome.

i. Local Learning and Model Integration for Multiple Information Sources

Since Big Data applications comprise of autonomous sources and decentralized controls, integrating dispersed data sources to a central workstation for mining is systematically a toll order because of the potential transmission cost and privacy issues [12]. Data mining activities can be carried out at each distributed site; but this result in a biased view of the data gathered at each individual site and consequently leads to biased decisions or models. A Big Data mining system should therefore allow for cost effective exchange of information and aggregation mechanism to ensure that all distributed information sources/sites can coordinate so as to achieve a global optimization goal. The global mining can however be carried out in two-steps namely local mining and global aggregation and correlation. This two-step process can be carried out at data, model, and knowledge levels.

ii. Mining from Sparse, Uncertain, and Incomplete Data

The most defining features for Big Data are Sparseness, uncertainty (inconsistent), and incompleteness.

- **Sparse:** Being sparse means that the number of crucial data points is not adequate for drawing reliable conclusions. This is normally as a result of data in a high-dimensional space failing exhibit clear trends or distributions. Normally data that are sparse significantly deteriorate the reliability of the findings derived from the data. To alleviate this dimension reduction or feature selection are commonly employed to reduce the data dimensions. Another way is to cautiously include additional samples to lessen the data scarcity. [12]

- **Uncertain:** Uncertain data means that each data field is no longer consistent and deterministic but is subject to some variable random or error distributions. The main cause is linked to domain specific applications with data readings and collections that are inaccurate [12]. Common solutions to this problem are to estimate model parameters.

- **Incomplete Data:** Data that are missing data field values for some samples are referred to as incomplete data. The missing values can be as a result of different realities, such as sensor node malfunction, or some systematic protocol to intentionally omit some data values (e.g., dropping some packets to save power for transmission) [12]. Solutions to this problem include; ignoring data fields with missing values and data imputation. Data imputation seeks to insert missing values to the data so as to produce improved models. Two major data imputation methods are; to append most frequently observed values and to build learning models to predict the best possible values for each data field, by adopting the observed values of a given instance.

iii. Extracting Complex and Dynamic Data

The rapid increasing of complex data and their evolution in volumes and in nature has fueled the rise of Big Data. The complexity of the data is in twofold; the structure of the data that is data is in numerous formats and the dependencies of the data in these structures.

- **Complex heterogeneous data types:** Data types available in Big Data include unstructured data, semi-structured data and unstructured data.

- **Complex inherent semantic associations in data.** Online news, comments on social media, pictures on Instagram, and video clips on YouTube may discuss about a similar event at the same time. There is obvious that these greatly diverse platforms contain information that is closely associated to each other. It is therefore important to be able to create a connection to these platforms and associate data fields from different data formats. Application system performance such as recommender systems and search engines will significantly be enhanced by the mining of these complex semantic associations across different data formats (“text-image-video”). However, with regards to Big Data, it is challenging to efficiently define semantic features and to build meaningful association models to eliminate the semantic gap of several heterogeneous data sources [12].

1.4 Structure of thesis

This thesis report is structured as follows:

- **Chapter1- Introduction:** this section seeks to introduce the area of study; in this case data mining of big data. It defines data mining and big data separately and then briefly outlines the challenges found in mining big data.
- **Chapter 2 -Literature Review:** this chapter provides a context for the proposed study and demonstrates why it is important and timely. This chapter outlines the already existing data mining techniques and algorithms. It also highlights the challenges faced in designing parallel/distributed algorithms.
- **Chapter 3- Problem Definition:** this part of the report explains the need for this project. It outlines the motivation behind the project, that is, what prompted us to propose the project in the first place. In this chapter we expose the gap that this project intents to bridge.
- **Chapter 4 – Proposed Methodology:** this chapter explains how we tackled the problem. It explains the original RDB-Miner algorithm and the strategy used to enhance it to Parallel RDB-Miner. In this chapter we also explain the experimental set up, the environment in which the algorithm was tested and the assumptions that were made for proposed algorithm.
- **Chapter 5 – Results and Analysis:** this chapter displays the observations made in the experiments and makes an explanation of each observation. Each result is accompanied with an analysis and a comparison to previous work. Each result is presented in a graphical manner.

- **Chapter 6 – Conclusion and Future work:** this is the final and last chapter of the thesis. It seeks to conclude the thesis and in this chapter make our own statement of how we feel about the project. We also outline the future work we intent to do, some further enhancements that could be done on the new proposed algorithm.

CHAPTER 2-LITERATURE REVIEW

2.1 Cloud Computing

It is difficult to talk about big data and not talk about Cloud computing. The Internet has played the catalyst role in big data. Most businesses are now carried out over the internet and hence this necessitated the need for business intelligence on the same platform. Cloud computing has enabled the businesses to have a flexible, portable, agile and cost effective platform for internet business intelligence. Cloud Computing can provide infrastructure to massive and complex data of data mining.

NIST (National Institute of Standards and Technology) defines Cloud Computing as a model that provides a ubiquitous, simple and on-demand network access to a shared set of resources (e.g. network resources, servers, data storage, applications and services) that can be readily available for use, or, if necessary, shut down, and all with minimal intervention of service providers [14].

Influential Gartner & Forrester provides the following definition: "CC is the area of computing in which scalable and highly resilient IT facilities provide in the form of services delivered via the Internet to numerous external customers." [14]

The cloud model is composed of five essential characteristics, three service models, and four deployment models. The essential characteristics of cloud computing are:

- On-demand self-service,
- Broad network access,
- Resource pooling,
- Rapid elasticity and
- Measured service. [15]

Providing CC services is divided into three elementary architectural models and different derivative combinations of the basic models. These three basic classifications are known as SPI: (Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS)) model.

- SaaS (software as a service) - a technology platform that allows access to applications via the Internet in the form of services that are hired as needed, instead of buying a separate software programs that must be installed on the user's computers; [15]
- PaaS (Platform as a Service) - model is a variation of SaaS structure that, as a service delivers environment development. Allows the user to build his own applications that run on the provider's infrastructure. Applications are delivered to users through the servers' interface accessible via the Internet. [15]
- IaaS (Infrastructure as a Service) - provides the ability to use computer infrastructure (mainly virtual platforms). Users do not buy servers, software, data storage or network equipment, but they buy these resources as an external service. [15]

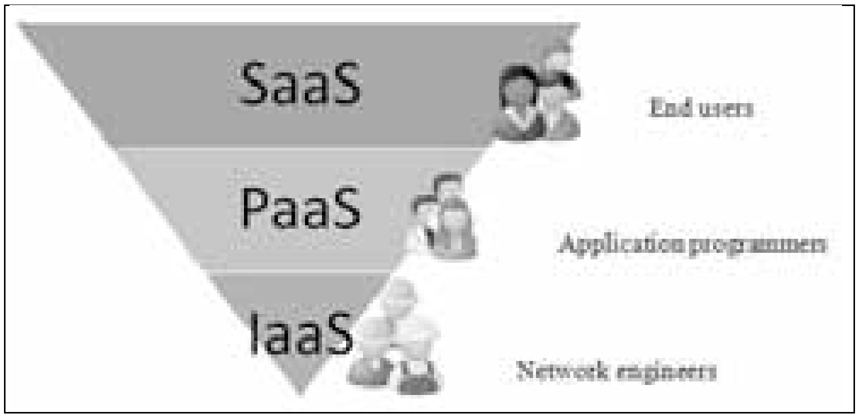


Figure 5: Classifications of cloud computing services

Regardless of the type of service delivery models (SaaS, PaaS, or IaaS), there are four basic models of implementing Cloud Computing services, including:

- Public Cloud - platform available and open to the public, regardless of whether they are individuals or organizations;
- Private Cloud - CC infrastructure accessible to only one organization. It can be managed by the organization itself or someone else who is doing that for the organization (outsourcing);
- Community Cloud - model of implementation that provides the ability for more organizations to share the same CC structure. Infrastructure supports special communities that have common interests, needs and security requirements;

- Hybrid Cloud - Model, which consists of two or more previously, discussed types of the establishment of CC structure which remain unique and independent entities, but with a certain kind of reciprocal link, in order to achieve mobility of data between them.

Advantages of Cloud Computing

i. Possibility for significant cost reduction
ii. Reducing the need for maintenance software support
iii. Reduction of IT departments in companies
iv. Scalability
v. Companies can now focus on their primary business
vi. Availability and independence of the unit
vii. Saving energy and contribution to the conservation of the environment

Disadvantages of Cloud Computing

i. Problem of availability (lack of availability) safety problem
ii. Management problem
iii. The possibility of sudden termination of the provider

The advent of cloud computing has brought about new technologies and techniques to aid storage, processing and analysis of large quantities of data that are produced. Cloud computing did not only provide users with a common parallel programming model and big data processing capacity, but also provides users with an open computing services platform. A series of cloud computing service platforms have been developed to provide data mining services for the public. Below are some of the major technologies that have come cloud computing;

2.1.1 Apache Hadoop

Apache Hadoop, an open source project, is seen as a framework for the development of distributed and scalable applications that work with very large amounts of data (measured in petabytes). It is based on Google's MapReduce algorithm and a special data management system HDFS (Hadoop Distributed File System), which also derived from Google's File System. Hadoop was developed in Java, so it is about a cross-platform product [16]. It is used by most cloud computing service providers because is a highly scalable analytics platform for processing large volumes of structured and unstructured data. It offers analytics at a low cost and high speed that some analysts say can't be achieved any other way. The low cost is as a result of using commodity hardware and the speed because of distribution of data and tasks.

Hadoop technology is not a single entity but rather, it consists of two main components namely Hadoop Distributed File System, HDFS and MapReduce.

Hadoop Distributed File System

HDFS (Hadoop Distributed File System. HDFS) is a File System designed for storing very large files with streaming data access patterns, running on clusters on commodity hardware. HDFS follows the master/slave architecture. The master node is called the Namenode which manages the file system namespace and regulates client accesses to the data. There are a number of worker nodes, called Datanodes, which store actual data in units of blocks. The Namenode maintains a mapping table which maps data blocks to Datanodes in order to process write and read requests from HDFS clients. It is also in charge of file system namespace operations such as closing, renaming, and opening files and directories. HDFS allows a secondary Namenode to periodically save a copy of the metadata stored on the Namenode in case of Namenode failure. The Datanode stores the data blocks in its local disk and executes instructions like data replacement, creation, deletion, and replication from the Namenode. The namenode is very essential for accessing files thus it is very important to make it resilient to failure as it constitutes a single point of failure [17]. Figure 6 gives the architecture of the HDFS.

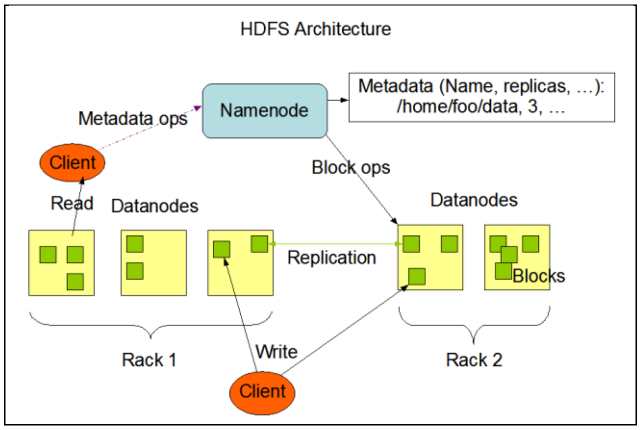


Figure 6: HDFS Architecture [6]

MapReduce

MapReduce is the programming paradigm/model allowing massive scalability and high performance in data processing. The MapReduce basically performs two different tasks i.e. Map Task and Reduce Task [17]. These are two separate but complimentary tasks. A map-reduce computation execution, Map tasks are given input from distributed file system. The map tasks produce a sequence of key-value pairs from the input and this is done according to the code written for map function. These value generated are collected by master controller and are sorted by key and divided among reduce tasks. The sorting basically assures that the same key values ends with the same reduce tasks. The Reduce tasks combine all the values associated with a key working with one key at a time. Again the combination process depends on the code written for reduce job [17].

2.1.2 Apache Hive

Hive is a data warehouse infrastructure built on the top of Hadoop framework and allows analyzing data and generating queries in a way similar to SQL queries in RDBMS [14]. Hive provides a means of running MapReduce job through an SQL-like scripting language, called HiveQL, that can be applied towards summarization, querying, and analysis of large volumes of data. HiveQL enables anyone already familiar with SQL to query the data.

Hive also allows programmers who are knowledgeable in the MapReduce framework to be able to plug in their custom mappers and reducers to perform more sophisticated analysis that may not be supported by the built-in capabilities of the language. Hive is best suited for batch processing of large amounts of immutable data (such as web logs). It is not appropriate for transaction applications that need very fast response times, such as database management systems. Hive is optimized for scalability (more machines can be added dynamically to the Hadoop cluster), extensibility (with MapReduce framework and other programming interfaces), and fault-tolerance. Latency is not a key design consideration. [19]

2.1.3 NoSQL Databases

Not Only Structured Query Language (NoSQL) is a class of “schema-less” database management systems that have been designed with more relaxed data models as compared to RDBMS. The term is not meant to discredit SQL but rather to indicate that these databases can provide additional solutions where relational models fall short. They neither use SQL nor relational data model. Such non-relational databases, do not support the ACID (Atomicity, Consistency, Isolation, and Durability) properties in full; actually they represent pure data warehouses with simple mechanisms of data control and transactions.

NoSQL concept relies on the following grounds: [14]

i. **Scalability** - ability to automatically respond (giving the required major resources) in accordance with the increase in the application;
ii. **Replication** - data in the case of distributed databases is stored in multiple nodes;
iii. **Partitioning Data** – means data sharing in a way that the different parts of the database are in different nodes. The goal of the partitioning data is to improve performance when reading and writing data

Compromises in terms of ACID properties are necessary in cloud computing environment because they can overcome certain limitations of relational databases and provide better performance in the following areas of application that is Storage and processing of very large amounts of semi-structured and unstructured data, all with low latency reading operations and automatic scalability. [14]
Although there is no single definition which defines what is included in the term NoSQL, in practice there are the following classes of NoSQL databases:

i. Key Value Stores

Provide a way of storing schema-less data by means of a distributed index for object storage. The key (data-type) will be displayed on the left and the corresponding value (actual data) on the right. Key/Value store is best applicable where write performance is of highest priority since its schema-less structure allows for fast storage of data.

ii. Column Oriented Databases

The data store provided by column oriented databases is similar to that of relational tables; the only difference is that column oriented also adds a dynamic number of fields/attributes to the model. The keys used in this model point to more than one table.

iii. Document Oriented Databases

In this model data is treated as autonomous objects whose attributes are stored as separate documents. A document contains information which uniquely identifies a single object. This model is flexible and agile because it has a schema-less structure and also read and write operations can be accomplished.

iv. Graph Databases

These are databases are more applicable to intelligent agencies as well as in social networks because they effectively outline relationships between objects and present a way to access overloaded sites through predominant reads. Their structure and representations are based on the graph theory. Data is stored in graph structure where nodes, edges and properties represent the data. The nodes represent entities in the database. Edges represent a connection or relationship between nodes/entities. The properties describe the nature of the nodes or relationship between nodes.

2.1.4 Apache Pig

Pig was originally developed at Yahoo Research around 2006 for researchers to have an ad-hoc way of creating and executing map-reduce jobs on very large data sets. In 2007, it was moved into the Apache Software Foundation. It is a platform designed for high levels of Hadoop, which is responsible for making MapReduce programs. Pig makes it easy to write MapReduce code introducing a special language - Pig Latin and the environment for the execution of such code. Pig translates the code from higher-level language (Pig Latin) into MapReduce code that is then executed in a cluster computer [14].

2.1.5 HBase

HBase is an open source, non-relational, distributed database modeled after Google's Big Table and written in Java. It is part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS in order to provide a fault-tolerant way of storing large quantities of sparse data (bits and pieces of information thrown in large pool of unimportant data) [20].

HBase features include compression, in-memory operation, and Bloom filters on a per-column basis. Tables in HBase can serve as the input and output for MapReduce jobs run in Hadoop, and may be accessed through the Java API but also through REST, Avro or Thrift gateway APIs.

HBase is not intended to replace classic SQL database, however its performance has massively improved, and it is now been used in several data-driven websites, including Facebook's Messaging Platform [20].

2.1.6 Hybrids of Hadoop

Hadoop has been the most successful technology to grace the big data era. However, it is not ideal for all types of data and all organisations. As such several extensions of Hadoop have been developed so that they bridge the gap that Hadoop failed to cover. Below are some of the most popular extensions or hybrids.

1. **HadoopDB** is a hybrid system that has the best features of both parallel databases and Map/Reduce. HadoopDB approaches parallel database in performance and efficiency. [16]

2. **Hadoop ++** is Hadoop with Trojan Index and Trojan Join. In terms of query processing Hadoop++ matches and improves the query run times of HadoopDB. Indexing capabilities can be achieved in Hadoop using Trojan index. [16]

The basic notion behind using Trojan index is that the schema and query workload is known already. The main features of Trojan index are non- invasiveness, no need to create a distributed SQL engine on top of Hadoop, optional index access path, possibility to create multiple index in the same split and seamless splitting.

The goal behind Trojan join is to avoid reduce phase since the data were already pre-partitioned. The algorithm access each input split and collects records of the same co-group.Both Trojan indices and joins are created during data loading. [16]

3. **Co-Hadoop** provides a solution for co-locating related files in Hadoop Distributed File System (HDFS). HDFS is extended to provide co-location at the system level. A new file property is used to identify related files and modify the placement policy of HDFS. This modification retains the benefits of Hadoop including load balancing and fault tolerance. The default block placement policy provides load balancing through even data distribution across the Data Nodes. [16]

4. **Trojan HDFS:** a new data layout called Trojan Layout is used which internally organizes data blocks intodifferent attribute groups according to the workload to improve the access performance. It creates the data layout without affecting the fault tolerance properties of Hadoop. Hadoop stores multiple replicas in different Data Nodes without changing the layout. Trojan HDFS creates different Trojan layouts for replicas and so the incoming requests can be forwarded to the replica with the most suitable layout. [16]

5. **Elastic Replica Management System for HDFS:** The data access patterns in HDFS vary heavily and the current replication strategy of creating a fixed number of replicas may be inadequate. Based on the data access patterns, data in Hadoop can be classified into different types. *Hot data* – data having a large number of concurrent access and high intensity of access, *cold data* - unpopular and rarely accessed data, *normal data* – rest of the data other than hot and cold. In a large and busy HDFS network, hot data will have concurrent and intense access. Replicating hot data only on three different nodes is not adequate to avoid contention. Also for cold data, three replicas may produce unnecessary

overhead. To tackle these issues ERMS is proposed which introduces an active/standby storage model which takes advantage of a high performance complex event processing engine to distinguish the real time data types and brings an elastic replication policy for the different types of data. [16]

6. **SQL-on-Hadoop**: technologies include a SQL layer or even a full SQL database over Hadoop. SQL-on-Hadoop solutions solve the data management issues of Hadoop and provide a scale-out alternative for traditional RDBMSs. They include, among others:

- ***HAWQ*** (Hadoop With Query) is a commercial product developed by Pivotal HD which enables enterprises to benefit from tried and tested Massively Parallel Processing (MPP) based analytic features and its query performance while harnessing the power of the Apache Hadoop stack [28].
- ***Spark SQL:*** Apache's Spark project is for real-time, in-memory, parallelized processing of Hadoop data. Spark SQL builds on top of it to allow SQL queries to be written against data.

Other tools that help support SQL-on-Hadoop include BigSQL, Apache Drill, Hadapt, H-SQL, Cloudera's Impala, JethroData, Polybase, Presto, Shark (Hive on Spark), Splice Machine, Stinger, and Tez (Hive on Tez).

2.2 An Overview of HadoopDB

HadoopDB is a project that was carried out in 2009 at Yale University. It aimed to combine the desired properties of MapReduce (scalability) with the performance and efficiency of parallel databases [3]. The thrust behind HadoopDB is to connect multiple single-node database systems using Hadoop as the task coordinator and network communication layer. Queries are parallelized across nodes using the MapReduce framework. HadoopDB runs on a shared-nothing architecture using commodity hardware. Any Open Database Connectivity (ODBC) compliant database can be used for example MySQL or PostgreSQL.

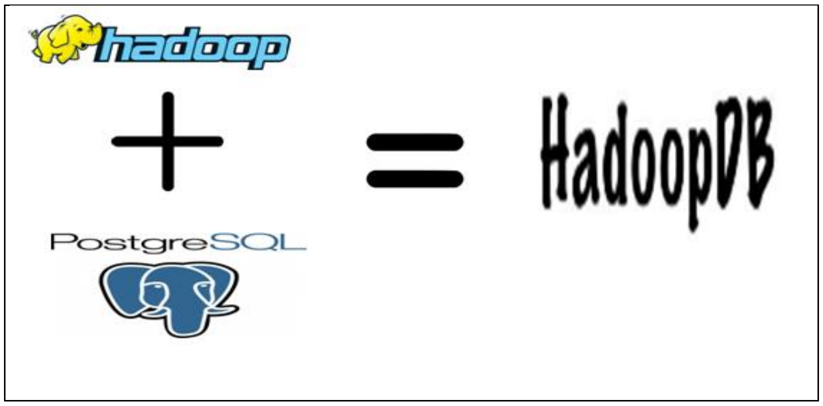


Figure 7: Idea behind HadoopDB [3]

2.2.1 HadoopDB Architecture

The figure below shows the architecture of HadoopDB. Shaded components are the modifications to the generic Hadoop architecture.

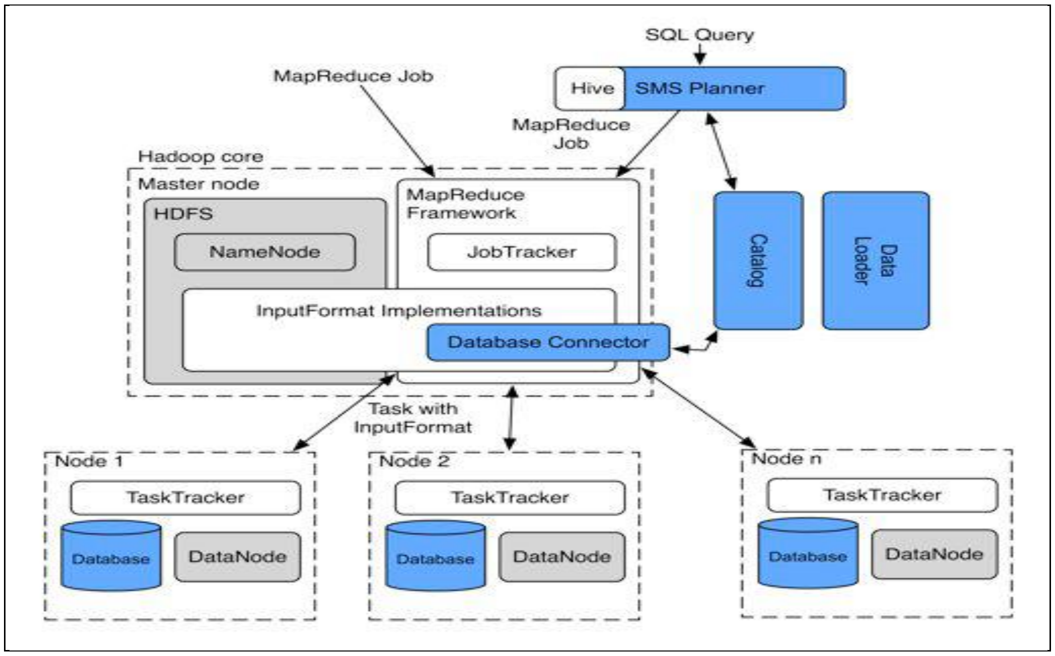


Figure 8: The architecture of HadoopDB [3]

2.2.1.1 Database Connector [3]

It is the interface between node database instances and TaskTrackers. It is responsable for the following:

a) Connect to the node database
b) Execute the SQL query
c) Return the results as key-value pairs

2.2.1.2 Catalog [3]

It is stored as an XML file in the HDFS. It is responsible for the following:

1. Maintain information about database
2. Database location, driver class
3. Datasets in cluster, replica or partitioning

2.2.1.3 SQL to MapReduce to SQL (SMS) Planner [3]

SMS planner is a parallel database front-end provided by HadoopDB to data analysts which enables them to execute SQL queries. The SMS planner is an extension of Hive. It accepts SQL queries and transforms them to MapReduce tasks and returns the results as SQL. It does this in the following steps:

i. The parser converts the query into an Abstract Syntax Tree.
ii. In order to retrieve the structure of the table(s) involved the Semantic Analyzer hooks up to the MetaStore. It also populates different data structures with meta-information such as the Deserializer and InputFormat classes required to scan the table and extract the necessary fields.
iii. The logical plan generator then creates a DAG of relational operators i.e. the query plan.
iv. The optimizer restructures the query plan to create a more optimized plan. For example, it pushes filter operators closer to the table scan operators.
v. Finally, the physical plan generator converts the logical query plan into a physical plan executable by one or more MapReduce jobs. The first and every other Reduce Sink operator marks a transition from a Map phase to a Reduce phase of a MapReduce job and the remaining Reduce Sink operators mark the start of new MapReduce jobs.

2.2.1.4 Node Database

This is any JDBC or ODBC compliant relational database that is set up on top of Hadoop. Single node databases are installed on all nodes. Open source database system are mostly used e.g. MySQL or PostgreSQL to reduce project costs.

2.3 Data Mining Techniques for Big Data

For any data mining technique to be able to successfully produce desired results, it has to address the three challenges outlined in point 1.3.1 HACE Theorem. Heterogeneity of data, autonomous and decentralised sources of data and the complexity of data relationships had to be considered when choosing a data mining technique for big data. In this section we look at the most common techniques used when mining big data.

2.3.1 Distributed Data Mining (DDM)

Distributed Data Mining (DDM) is a subdivision of the data mining field that offers a mechanism to mine dispersed data giving careful awareness to the distributed data as well as computing resources [21]. Distributed and autonomous data source is the norm in big data. Data sources usually have decentralised control; this implies that each data source is able to generate and collect information without involving or relying on any centralized control hence the most natural way to mine big data is to distribute the mining activities to all the sites. Traditional ways of collecting the data into a central location such as warehouse is no longer feasible. This is attributed to the fact that it is expensive to transmit data over the network and also the bottleneck involved during transfer, integration and extraction of the data.

Data mining environments consist of users (usually management or analysts), data (stored in a persistent storage which cannot be modified), hardware and the mining software. The mining software normally includes the mining algorithms and their associated programs. Distributed data mining tackles the impact of distribution of the aforementioned entities that make up a data mining environment on the actual data mining activity. There is general agreement that DDM is the process of extracting useful knowledge from data that has been fragmented into one or more geographically or physically distributed subsets [21]. There are two ways in which data can be distributed across sites; homogeneous and heterogeneous. Both approached assume the conceptual viewpoint that the data tables at each site are fragments of a single global table.

In the homogeneous case tables at each site have exactly the same attributes as the global conceptual table. These tables are regarded as sub tables of the global table, which can also be viewed as horizontal partitions of the global table. In the heterogeneous case sites do not have the similar attributes. The global table is vertically partitioned; each site contains a different collection of fields. However, each table instance at each site is assumed to contain a primary key or unique id to aid matching. It is not necessary to physically realize the global table; it is just a conceptual idea.

The bulk of DDM methods in the literature operate over an abstract architecture which includes multiple sites having independent computing power and storage capability. Local computation is done on each of the sites and either a central site communicates with each distributed site to compute the global models or a peer-to-peer architecture is used. In the latter case, individual nodes might communicate with a resource rich centralized node, but they perform most of the tasks by communicating with neighboring nodes by message passing over an asynchronous network. Below is the general framework for distributed data mining.

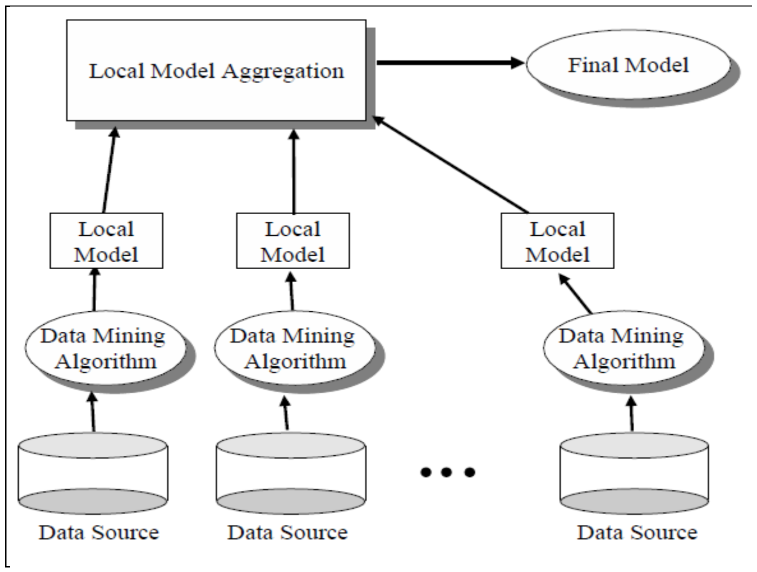


Figure 9: Distributed data mining framework [24]

According to [23] some features of a distributed scenario where DDM is applicable are as follows.

- The system consists of multiple independent sites of data and computation .which communicate only through message passing.
- Communication between the sites is expensive.
- Sites have resource constraints *e.g.* battery power.
- Sites have privacy concerns.

2.3.1.1 Software Agents in Distributed Data Mining

According to Shoham 1997; an agent is a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and processes. Agents are also defined in [21] as software or hardware entities that perform some set of tasks on behalf of users with some degree of autonomy. In order to work for somebody as an assistant, an agent has to include a certain amount of intelligence, which is the ability to choose among various courses of action, plan, communicate, adapt to changes in the environment, and learn from experience.

Consistent with the requirements of a particular problem, each agent might possess to a greater or lesser degree attributes like the ones [23]:

- **Reactivity:** the ability to selectively sense and act
- **Autonomy:** goal-directedness, proactive and self-starting behavior
- **Collaborative behavior:** can work in concert with other agents to achieve a common goal
- **“Knowledge-level” communication ability:** the ability to communicate with persons and other agents with language more resembling human like “speech acts” than typical symbol-level program-to-program protocols
- **Inferential capability:** can act on abstract task specification using prior knowledge of general goals and preferred methods to achieve flexibility; goes beyond the information given, and may have explicit models of self, user, situation, and/or other agents.
- **Temporal continuity:** persistence of identity and state over long periods of time

- **Personality:** the capability of manifesting the attributes of a “believable” character such as emotion
- **Adaptivity:** being able to learn and improve with experience
- **Mobility:** being able to migrate in a self-directed way from one host platform to another.

2.3.1.2 Architecture of Agents

An intelligent agent can be described as consisting of a *sensing* element that can receive events, a *recognizer* or *classifier* that determines which event occurred, a *set of logic* ranging from hard-coded programs to rule-based inferencing, and a *mechanism* for taking action. The agents’ intelligence can range from rudimentary sensor monitoring and data reporting, to more advanced forms of decision making and autonomous behavior.

Agent-based distributed data mining systems employ one or more agents to analyze and model local datasets, which generate local models. These local models generated by individual agents can then be composed into one or more new 'global models' based on different learning algorithms, for instance, JAM and BODHI.

BODHI is a Java and agent based distributed data mining system. BODHI also notes the importance of mobile agent technology. As all of agents are extensions of a basic agent object, BODHI can easily transfer an agent from one site to another site, along with the agent's environment, configuration, current state and learned knowledge. Figure 10 shows the BODHI architecture.

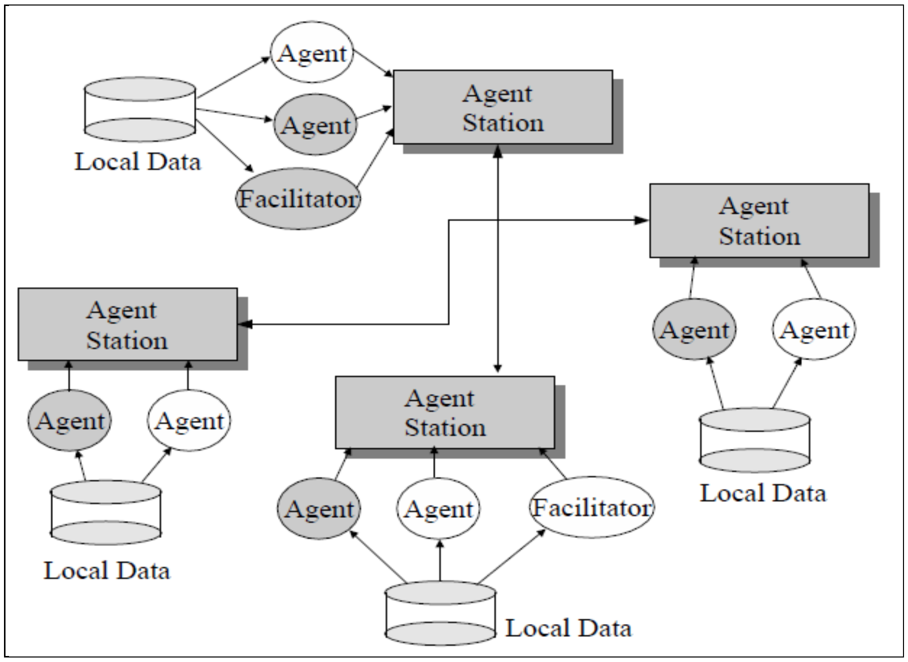


Figure 10: BODHI: Agent-based distributed data mining system [21]

2.3.2 Parallel Data Mining

Due to the large volumes of data associated with big data, traditional systems are no longer sufficient. Because data mining in big data is a compute-intensive exercise, the most obvious and natural means for enhancing performance and achieving scalability is parallelization. Parallelization involves the use of more than one processor to accomplish a particular task. A large task is broken down into smaller tasks and each processor assigned certain tasks.

There are two dominant approaches to parallelization namely; shared-memory and shared-nothing. In shared memory all processors access common memory [25]. Shared memory architecture has many desirable properties. Each processor has direct and equal access to all memory in the system. In distributed memory or shared-nothing architecture each processor has its own local memory that can only be accessed directly by that processor [25]. For a processor to have access to data in the local memory of another processor a copy of the desired data element must be sent from one processor to the other through message passing.

There are several approaches which can be used to distribute workload across the environment in question. According to [21] there are four major classes of parallel implementations. ***The figure below illustrates this distinction.***

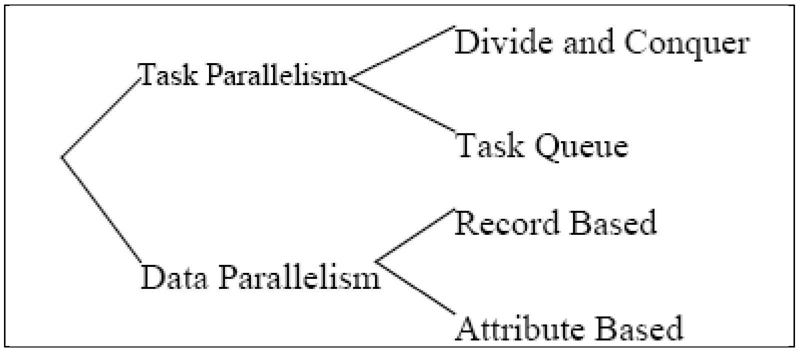


Figure 11: Methods of Parallelism [21]

i. **Task Parallelism:** In task-parallel algorithms portions of the search space are assigned to separate processors. The task parallel approach can be further divided into two groups namely;
   - *Divide and Conquer* strategy partitions the search space and assigns each fragment/partition to a specific processor.
   - *Task queue* dynamically assigns partitions of the search space to any available processor.

The main problem of task parallelism is that of load balancing which is caused by irregular distributions of records between nodes. The structure of the data set is the main determinant factor for the success of a task parallel implementation. The data set structure should be in a way that makes load balancing easy.

ii. **Data parallelism:** This approach distributes the data across available processors. This approach is further subdivided into two categories.
   - *Record Based Approach:* Non-overlapping sets of records are assigned to each of the processors.
   - *Attribute Based Approach:* In this approach, sets of attributes are assigned to each of the processors. The motivation behind attribute based approaches is the observation that many algorithms can be expressed in terms of primitives that consider every attribute in turn. If attributes are distributed over multiple processors, these primitives may be executed in parallel.

2.3.3 Algorithms for parallel and distributed data mining

In this project we narrow our scope to association rule mining (ARM) algorithms only. **Association rule mining** is a popular and well researched technique for discovering interesting relationships between variables in large data sets. It is anticipated to identify strong rules discovered in data sets using different metrics of interestingness. Two most popular interestingness metrics are support and confidence. They respectively reflect the usefulness and certainty of discovered rules. For example Rule 1: A=>B [support=2%, confidence=60%]; this means that 2% of all transactions under analysis show that A and B exist together; and 60% confidence means 60% of transactions containing A also contains B.

Most algorithms for association rule discovery mimic the same general modus operandi, based on the Apriori algorithm. The basic idea is to make multiple passes over the database, building larger and larger groups of associations on each pass. Thus, the first pass determines the "items" that occur most frequently in all the transactions in the database; each subsequent pass builds a list of possible frequent item tuples based on the results of the previous pass, and then scans the database, discarding those tuples that do not occur frequently in the database. The intuition is that for any set of items that occurs frequently, all subsets of that set must also occur frequently [21]. Apriori algorithm runs so many database scans in order to generate the rules; however if the database size is big the performance degrades significantly hence it has been parallelized to enhance performance when dealing with bigger datasets.

The basic approach to parallelizing association-discovery data mining is via database partitioning. Each available workstation in the networking environment is allocated a subset of the database records, and executes independently on that subset. Parallel data mining algorithms require some amount of intelligent communication among the independent network nodes to coordinate them [21].

2.3.3.1 Apriori Based Algorithms

2.3.3.1.1 Count Distribution:

This algorithm achieves parallelism by partitioning data. The data is horizontally partitioned such that each of N workstations gets 1/Nth of the database. Each workstation performs an Apriori-like algorithm on the subset. At the end of each iteration, the local counts will be summed up across all workstations into the global counts so that frequent itemsets can be found. Thus, this algorithm trades off I/O and duplication for minimal communication and good load-balance: each workstation must scan its database partition multiple times (causing a huge I/O load) and maintains a full copy of the (poor-locality) data structures used (causing duplicated data structure maintenance), but only requires a small amount of per-iteration communication (an asynchronous broadcast of frequency counts) and has a good distribution of work [21].

2.3.3.1.2 Data Distribution:

This algorithm is designed to minimize computational redundancy and maximize use of the memory bandwidth of each workstation. It works by partitioning the current maximal-frequency itemset candidates (like those generated by *Apriori*) amongst work stations. Thus, each workstation examines a disjoint set of possibilities; however, each workstation must scan the entire database to examine its candidates. Thus this algorithm trades off a huge amount of communication (to fetch the database partitions stored on other workstations) for better use of machine resources and to avoid duplicated work [21].

2.3.3.1.3 Candidate Distribution:

This algorithm is similar to data distribution in that it partitions the candidates across workstations, but it attempts to minimize communication by selectively partitioning the database such that each workstation has locally the data needed to process its candidate set. It does this after a fixed (small) number of passes of the standard data distribution algorithm. This trades off duplication (the same data may need to be replicated on more than one node) and poor load-balancing (after redistributing the data, the workload of each workstation may not be balanced) in order to minimize communication and synchronization. The effects of poor load balancing are mitigated somewhat, since global barriers at the end of each pass are not required [21].

2.3.3.2 Vertical Mining

The Eclat algorithm avoids most of the tradeoffs above by using an initial clustering step to pre-process the data before partitioning it between workstations. It thus achieves many of the benefits of candidate distribution without the costs. Little synchronization or communication is needed, since each node can process its partitioned dataset independently. A transformation of the data during partitioning allows the use of simple database intersections (rather than hash trees), maximizing cache locality and memory bandwidth usage.

To further optimize the Eclat algorithm it was further parallelized in [26]. It makes use of a vertical data layout by transforming the horizontal database transactions into vertical tid-lists of itemsets. By name, the tid-list of an itemset is a sorted list of ID's for all transactions that contain the itemset. Frequent *k*-itemsets are organized into mutually exclusive equivalence classes by common (*k*-1)-prefixes, in order that candidate (*k*+1)-itemsets can be generated by concatenating pairs of frequent *k*-itemsets from the same classes. The support of a candidate itemset can then be computed simply by intersecting the tid-lists of the two component subsets. Task parallelism is employed by dividing the mining tasks for different classes of itemsets among the available processes [26]. The equivalence classes of all frequent 2-itemsets are assigned to processes and the associated tid-lists are distributed accordingly. Each process then mines frequent itemsets generated from its assigned equivalence classes independently, by scanning and intersecting the local tid-lists. The steps for the parallel *Eclat* algorithm are presented below for distributed-memory multiprocessors [26].

1. Divide the database evenly into horizontal partitions among all processes;
2. Each process scans its local database partition to collect the counts for all 1-itemsets and 2-itemsets;
3. All processes exchange and sum up the local counts to get the global counts of all 1-itemsets and 2-itemsets, and find frequent ones among them;
4. Partition frequent 2-itemsets into equivalence classes by prefixes;
5. Assign the equivalence classes to processes;
6. Each process transforms its local database partition into vertical tid-lists for all frequent 2-itemsets;

7. Each process exchanges the local tid-lists with other processes to get the global ones for the assigned equivalence classes;
8. For each assigned equivalence class on each process, recursively mine all

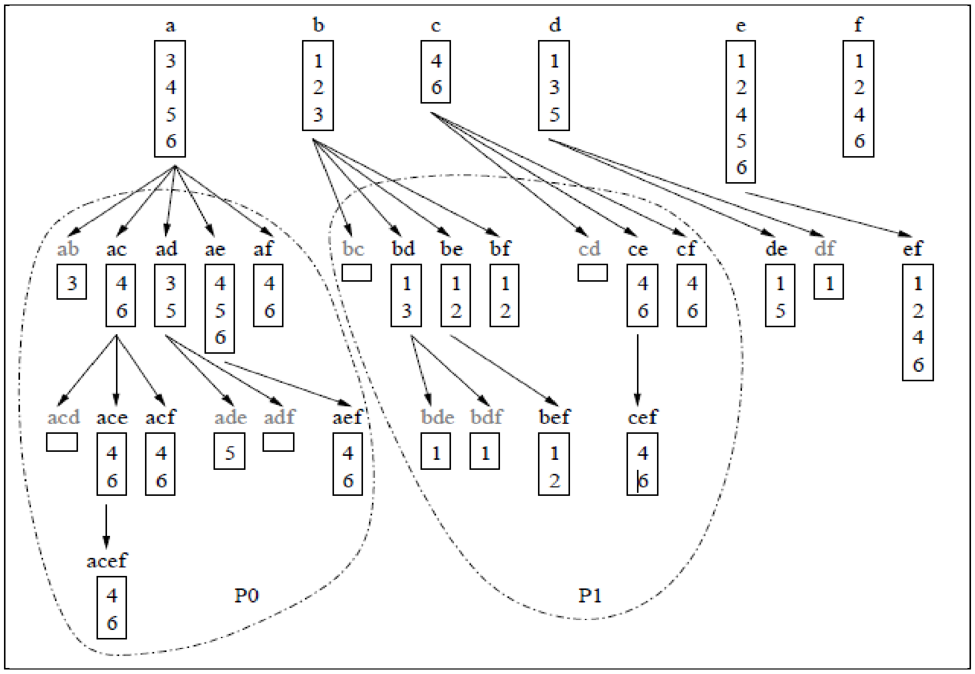


Figure 12: Mining frequent itemsets using the parallel Eclat algorithm.

2.3.4 Challenges in developing parallel algorithms for distributed environment

The development of parallel association rule mining algorithms for a distributed environment is graced by so many challenges. The following are some of them:

1. **Data distribution:** One of the major benefits of distributed/parallel data mining is "division of labor". This means that each node will be tasked to work with just a fraction of the total database. For each node to make noticeable independent progress the parallel algorithm must effectively distribute data across the underlined environment.

2. **I/O minimization:** Good data distribution must be supported with minimized I/O operations on the database. A lot of I/O operations significantly degrade the performance of an algorithm hence a good algorithm should minimize them.

3. **Load balancing:** In order to have the best results of parallelism, each workstation must be assigned approximately the same share of work to do. Normally if data distribution was carried out properly, it can help provide load-balancing. However some algorithms require periodic data redistribution to obtain excellent overall load-balancing.

4. **Avoiding duplication:** Ideally, no workstation should perform work that is already being executed by another node. This is in essence a waste of environmental resources (CPU time, memory, network etc).

5. **Minimizing communication:** A good parallel data mining algorithm should allow all workstations to smoothly and continuously operate asynchronously, without having to wait for global bureaucratic protocols or for communication delays.

6. **Maximizing locality:** A well designed data mining algorithms is one that fully harnesses performance potential of hardware. This involves maximizing locality for good cache behavior, utilizing as much of the machine's memory bandwidth as possible, etc.

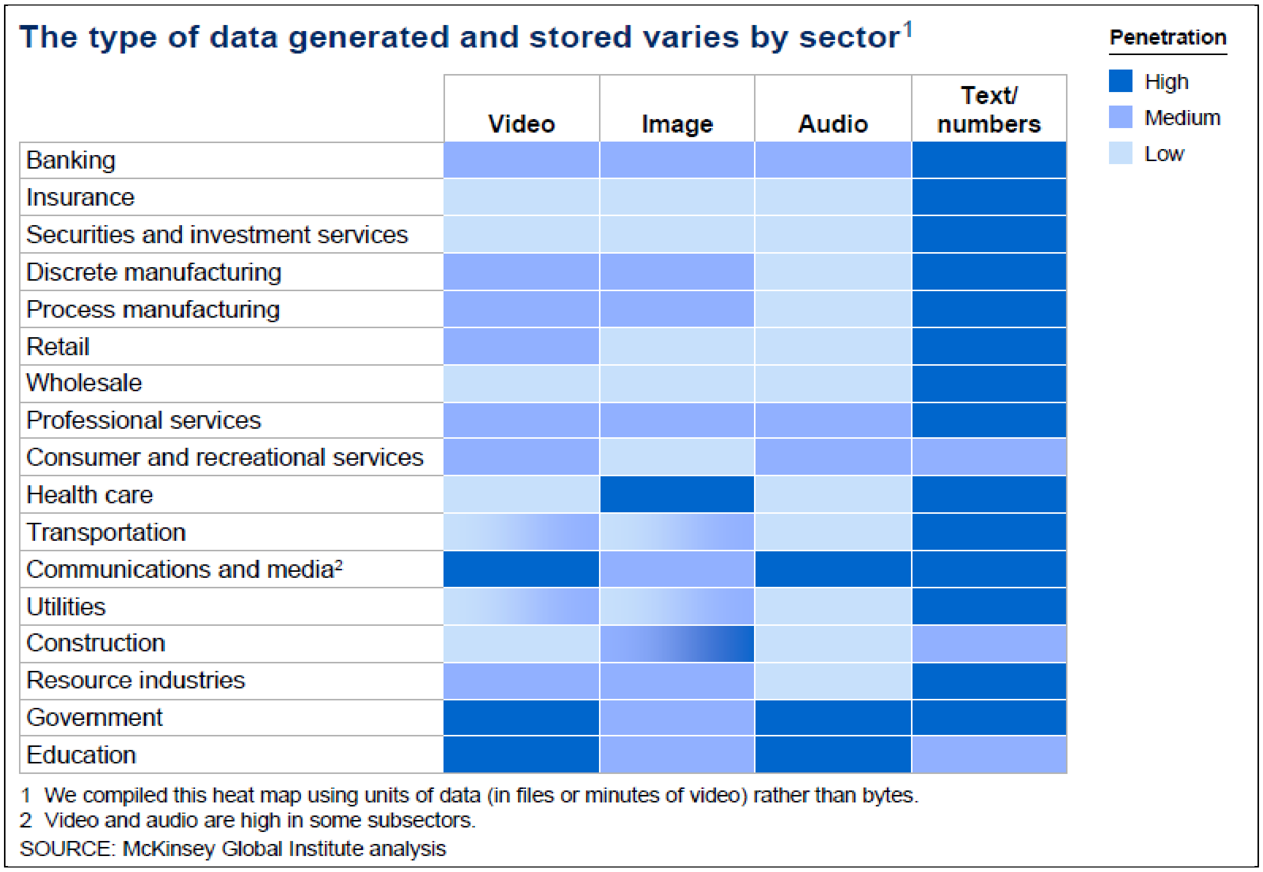
Achieving all of the above goals in one algorithm is nearly impossible, as there are tradeoffs between several of the above points. Existing algorithms for parallel data mining attempt to achieve an optimal balance between these factors.

CHAPTER 3 – PROBLEM DEFINITION

3.1 Motivation

3.1.1 Misconception of big data

As the world's view of data transitions from just data to big data, there is a false impression that all big data is unstructured or semi-structured. There is a notion that only unstructured data is on the rise and only it requires new handling techniques. This is not true. The heat map below shows that all sectors still produce more semi-structured or structured data than unstructured data in terms of the number of units.



The type of data generated and stored varies by sector[1]

	Video	Image	Audio	Text/ numbers
Banking	▨	▨	▨	■
Insurance	□	□	□	■
Securities and investment services	□	□	□	■
Discrete manufacturing	▨	▨	□	■
Process manufacturing	▨	▨	□	■
Retail	▨	□	□	■
Wholesale	□	□	□	■
Professional services	▨	▨	▨	■
Consumer and recreational services	▨	□	▨	▨
Health care	□	■	□	■
Transportation	□▨	□▨	□	■
Communications and media[2]	■	▨	■	■
Utilities	□▨	□▨	□	■
Construction	□	▨■	□	▨
Resource industries	▨	▨	□	■
Government	■	▨	■	■
Education	■	▨	■	▨

Penetration
■ High
▨ Medium
□ Low

1 We compiled this heat map using units of data (in files or minutes of video) rather than bytes.
2 Video and audio are high in some subsectors.
SOURCE: McKinsey Global Institute analysis

Figure 13: Heat Map for types of data generated by various sectors [12]

According to an article on IBM Data Magazine by Cristian Molaro et al [28], structured data still constitutes approximately 10% of the overall volume of big data. He also claims that this 10% is the one that matters the most because most of the 90% of the unstructured data are “cutie selfies” which in most cases have no business value. Most organizations, decision makers, and business analysts and other information consumers, can derive better business intelligence from big data–derived information when it is organized in a defined and structured manner rather than an unstructured one. This proves that as much as most of the big data (in terms of size) is unstructured, we cannot afford to ignore the structured data in big data. This 10% of big data requires new techniques and technologies because it is too big for traditional methods. There is therefore need for new platforms that can efficiently manage the enormous volumes of structured and semi-structured data being produced. The algorithms that are currently in existence are mainly intended to handle unstructured data and may not be as efficient in handling structured data. This has however motivated us to design a new algorithm by enhancing an already existing one.

3.1.2 Market Basket data format

Traditionally, research in the area of frequent itemset mining has focused on mining market basket data. Several algorithms and techniques have been introduced in the literature for mining data represented in basket data format. The primary objective of these algorithms has been to improve the performance of the mining process. Unlike basket data representation, no algorithms exist for mining frequent itemsets and association rules in relational databases that are represented using the formal relational data model. Typical relational data cannot be easily converted to basket data representation for the purpose of applying frequent itemset mining algorithms.

Therefore, a need arises for algorithms that can *directly* be applied to data represented using the formal relational data model and for a conceptual framework for mining such data. However Abdallah Alashqur et al [4] developed an algorithm called RDB-Miner which can be directly applied to relational data model. In this paper we propose to parallelize this algorithm such it can be applied on big data.

3.1.3 Results from previous work

In our previous work the results indicated that RDB-Miner algorithm can be implemented on HadoopDB. These experiments were carried out on a single cluster environment. The data was not distributed but still the performance was encouraging. These results have motivated us into proposing a parallel algorithm that will be implemented on a multi-cluster environment. It is most likely that the parallel algorithm will perform even better since there will be multi-tasking; work will be distributed over to more nodes.

The results below show that HadoopDB executes the algorithm faster than MySQL. Figures are in seconds.

Approximate No of Records	500 000	1 700 000
HadoopDB	8.3	15.6
MySQL	27.5	35.2

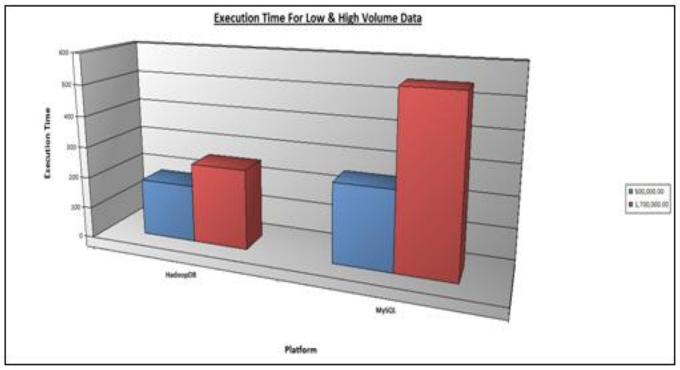


Figure 14: Results for Volume of Data

minSup	0.5	0.7	1.0
HadoopDB	15.6	10.3	9.3
MySQL	54.6	35.2	17.4

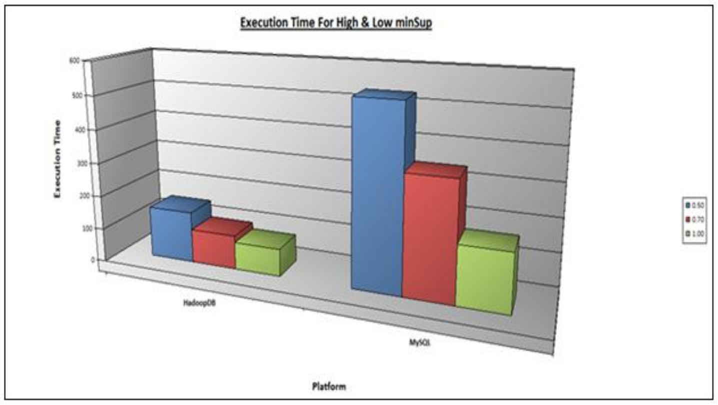


Figure 15: Results for various minSup

3.2 Problem Statement

This project proposes to enhance the RDB-Miner developed in [4]. We aim to parallelize the algorithm such that it can be applied on structured data in big data and implemented on a multi-cluster HadoopDB environment. The algorithm is intended to bridge the gap between unstructured data mining algorithms and structured data mining in big data. We aim to address the challenges outlined in section 2.3.4 of this report.

CHAPTER 4 – PROPOSED METHODOLOGY

4.1 Proposed Implementation Environment

In this paper we propose to design an algorithm that will be implemented on a multi-cluster HadoopDB that runs on a shared-nothing architecture. A shared nothing architecture (SN) is a distributed computing architecture in which each node is independent and self-sufficient, and there is no single point of contention across the system. More specifically, none of the nodes share memory or disk storage. [3] For simplicity reasons only three nodes will be used. Below is the proposed configuration.

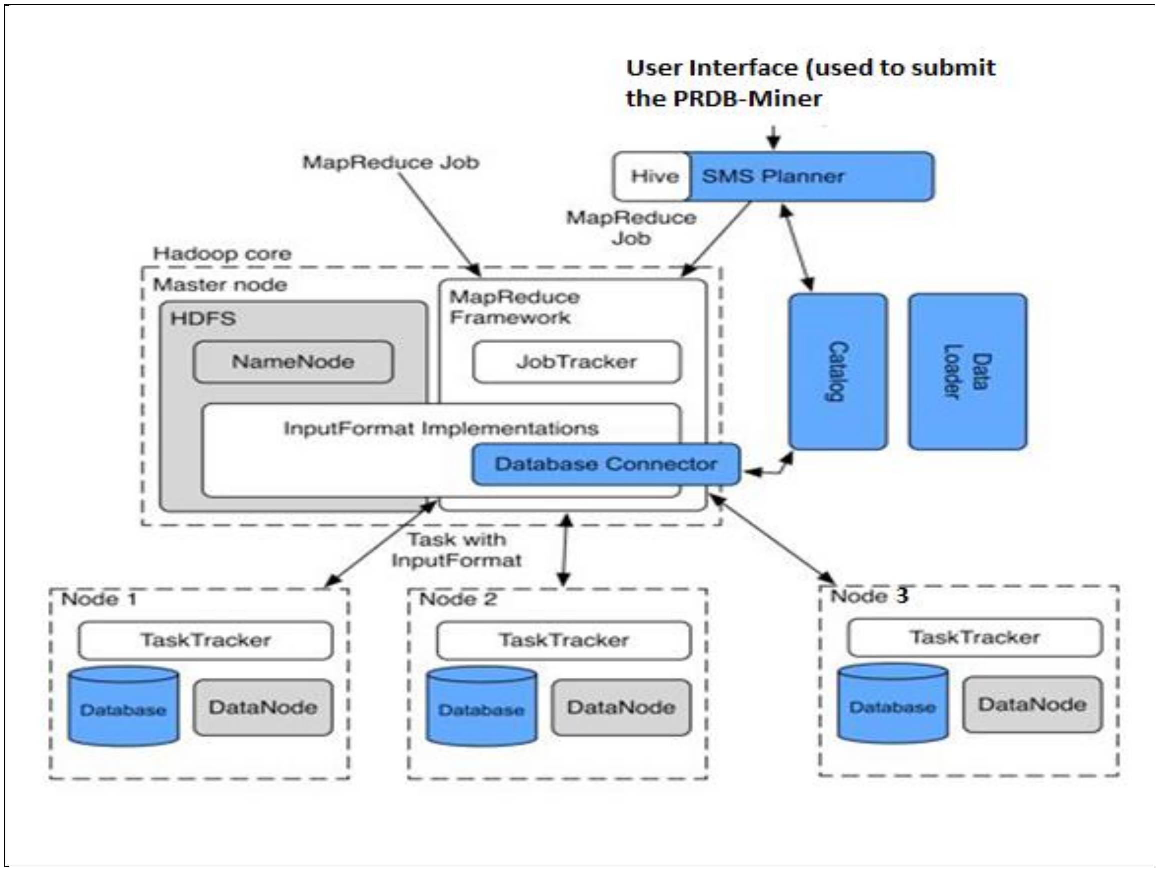


Figure 16: Proposed multi-cluster HadoopDB configuration

Since results will need to be consolidated at some point, one of the three nodes will act as a coordinator. The coordinator will be responsible for displaying the results collated from the other nodes. The coordinator will also be the point of entry for the query where the user submits the PRDB-Miner algorithm. The coordinator will also act as the master node which will host the NameNode and JobTracker which are important components for the Hadoop framework.

The user interface will be a web page designed using PHP. The user will be able to specify the minimum support from the proposed user interface. The proposed relational database to be used on top of Hadoop is MySQL because it is ODBC compliant and also open source. Below is the proposed network topology. The laptop will act as the master node from which the user of the algorithm will be submitting queries to the cluster.

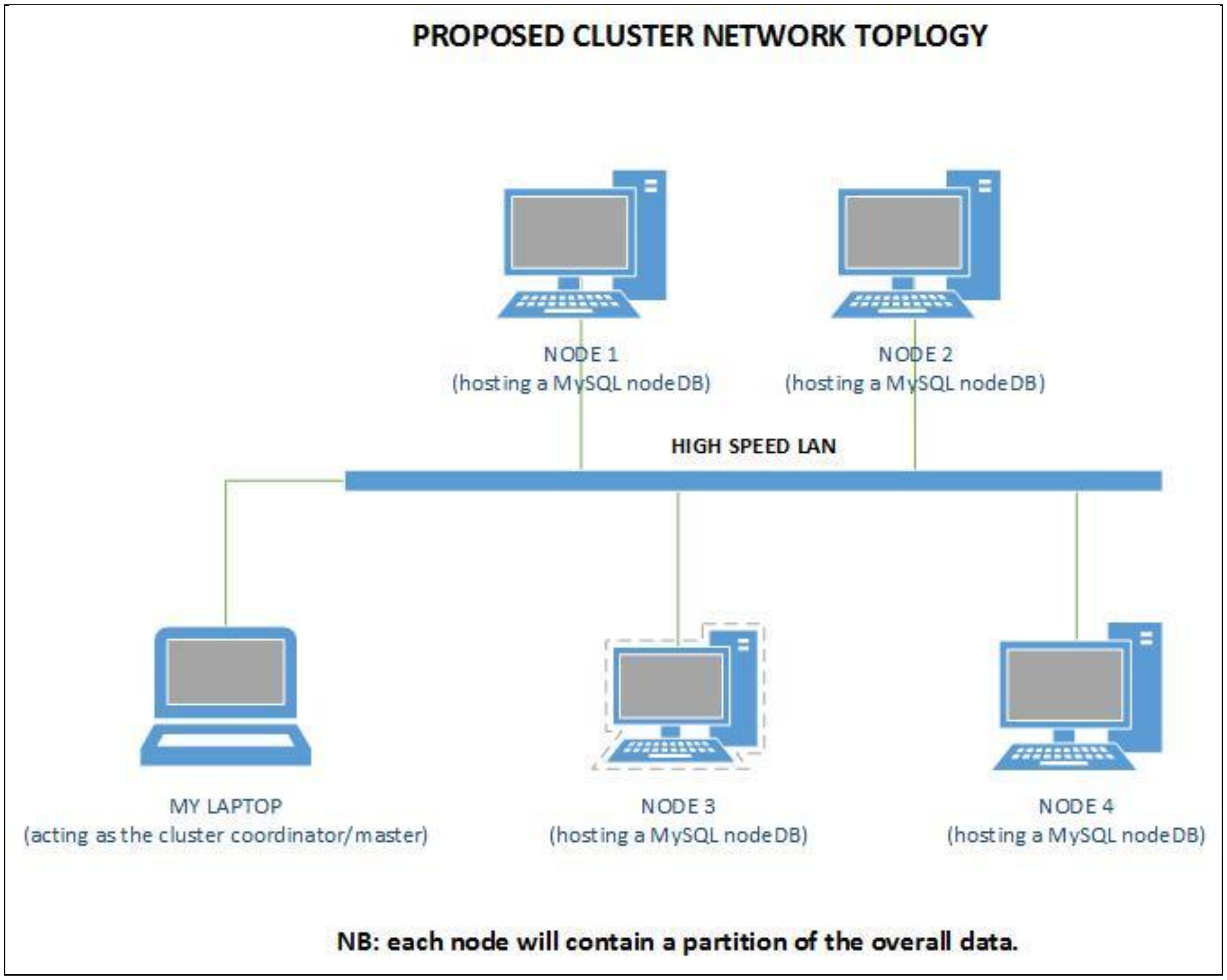


Figure 17: Proposed cluster network topology

There will be an agent at each node. The agent will be response for communicating with the coordinator. The agent will receive instructions to execute each iteration.

4.2 Proposed Parallel RDB-Miner Algorithm

4.2.1 The original RDB-miner algorithm

RDB-MINER is an algorithm for association rule mining on relational databases represented using the relational data model as opposed to basket data format. Therefore, this algorithm can be viewed as representing a new class of mining algorithms that is orthogonal to the class of algorithms represented by the Apriori algorithm. Viewed from a different perspective, we can think of RDB-MINER as an algorithm that performs *inter-attribute* frequent itemset mining, whereas existing algorithms perform *intra-attribute* mining. [4]

Algorithm *RDB-MINER [4]*

Input

```
R: a database relation
exclude_set: a subset of the attributes of R
0 Begin
1 Varchar SQL_str (512)
2 Compute_N (N, R, exclude_set)
3 Compute_PowerSet ( P (A) , R, exclude_set) ;
4 For c = 1 to N do
5 Extract_Ec (Ec , P (A) );

/* Ec ⊂  P (A) and each ISI ∈  Ec has a cardinality of c. */

6 For each itemset intension ISI ∈  Ec do
7 Generate_SQL (SQL_Str, ISI, Relation_Name);
8 Execute SQL_Str;
9 SQL_str = “”;
10 End
```

- Line 1 of the algorithm declares a variable called *SQL_str*, which is used to hold the SQL statement to be generated by the algorithm.
- Line 2 calls the procedure *compute_N*, that returns N, the number of attributes of relation R after excluding the attributes in *exclude_set*. The input arguments to *Compute_N* are relation R and *exclude_set*. *Exclude_set* is the set of attributes (normally primary key attributes) to be excluded from the computation of N. *Exclude_set* is left empty if all the attributes of R are to be included in the computation.
- Line 3 *Compute_powerset* procedure returns the power set, P (A), of relation R after excluding the attributes of *exclude_set*.
- Line 5 extracts the equi-cardinality subset *Ec* from P (A). For example, in the 2nd iteration of the for loop, where c=2, this step returns the subset *E2* = {{X,Y}, {X,Z}, {Y, Z} }, assuming we have the three attributes X, Y, and Z.
- The for loop between lines 6 and 10 extracts the *ISI*s (An *itemset intension* (*ISI*) is a subset of the attributes of a relation) from the equi-cardinality subset and generates and executes a SQL statement that computes the *support* of each such itemset. [4]

Below is the basic flow chart for RDB-Miner algorithm.

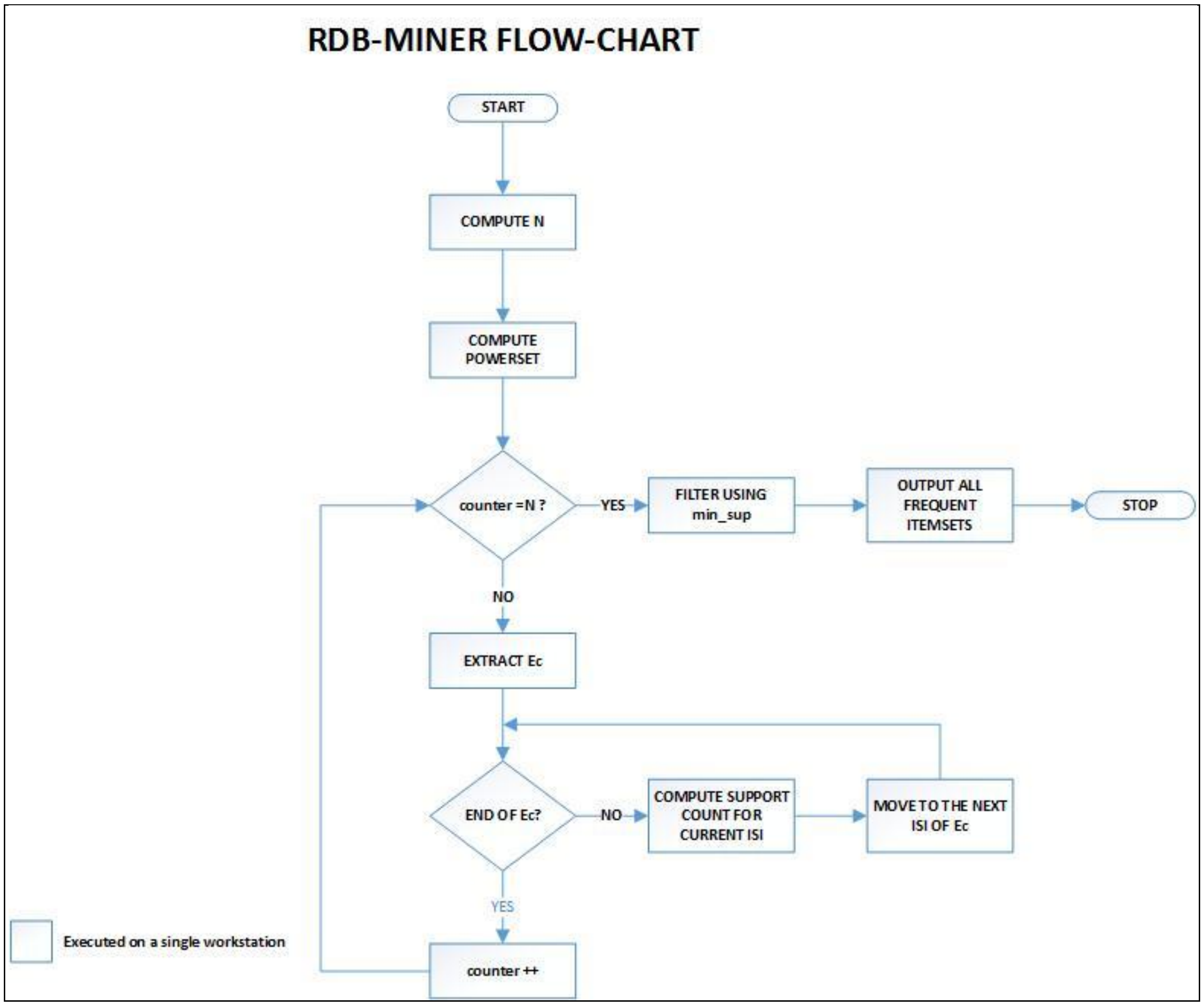


Figure 18: RDB-Miner Flow Chart

4.2.2 Improvement strategy

To parallelize the RDB-Miner algorithm we will follow the logic in the Count Distribution algorithm outlined in [26]. However the proposed algorithm will not broadcast the consolidated global frequent itemsets; instead each node will transmit its local frequent itemsets to the coordinator which in turn consolidates the local findings of each node to construct the global frequent itemsets.

Each iteration of the outer **for** loop that starts at line 4 extracts an equi-cardinality subset *Ec* from P (A). Each iteration of the inner **for** loop that starts at line 6 extracts an itemset intension *ISI* from *Ec* and generates and executes a SQL statement for that *ISI*. The generated SQL query computes the support count for a set of itemsets at once. Therefore, the net effect of the two nested for loops is to first compute 1-itemsets along with their *support count*. [4]

Since the transaction data is horizontally divided into N data subsets which are sent to N nodes. We seek to modify the second **for** loop of the algorithm; such that before it moves to ***k-itemsets*** it has to consolidate the ***(k-1)-itemsets*** from all the nodes. The logic is as follows:

1. Each node scans its data subset to generate the set of candidate itemset ***C p*** (candidate ***k-itemset***);
2. The candidate itemset ***C p*** is divided into ***N*** different partitions, which are sent to ***N*** nodes with their support count.
3. ***N*** nodes respectively accumulate the support count of the same itemset to produce the final practical support, and determine the set of frequent itemset ***Lp*** in the partition. At this point the nodes will not use the minSupport parameter to filter the frequent itemsets.
4. Each node then sends its computations to the coordinator for aggregation/consolidation.
5. The coordinator merges the output of the N nodes to generate the set of global frequent itemset ***Lp*** based on the set min_Sup.
6. Do the same for ***(k+1) –itemsets*** until c=N (number of attributes in the relation).
7. Output the final results.

4.2.3 Pseudo Code for the proposed PRDB-Miner Algorithm

Input at each node

R: a partition of database relation

exclude_set: a subset of the attributes of R

```
0  Begin
1  Varchar SQL_str (512)
2  Compute_N (N, R, exclude_set)
3  Compute_PowerSet ( P (A) , R, exclude_set) ;
4        c = 1
5        While c < (N+1) do
```

```
6              Extract_Ec (Ec , P (A) );
7              For each itemset intension ISI ∈  Ec do
8              Generate_SQL (SQL_Str, ISI, Relation_Name);
9              Execute SQL_Str = Cp;
10             Send_Results(Cp);
11             End For

12         If CheckNodeCount() = TRUE then ConsolidateResults();
13         c ++
14         GOTO 5
15     End While
16     OutputResult()
```

The improvements were put on lines 4 through 12. The explanations are as follows:

- Lines 1 to 3 remain the same but they are executed at the master/coordinator node,
- Line 4: a counter, c is initialized to 1,
- The commands in line 5 to line 10 are executed at the slave nodes,
- Line 5: the **while** loop checks to see if the counter c which counts the number of attributes in the relation/table has reached the total number of the attributes in the relation. If it true then it exits the loop and goes to line 16 and lets the master display the results,
- Lines 6 to 9 are the same as in the original algorithm,
- Line 10: ***SendResults()*** is a function that sends node results to the coordinator node. Its argument is the result generated by the ***Execute_SQL*** function in line 9.
- Line 12: ***CheckNodeCount*** function is executed by the coordinator only to check if all slave nodes have sent their results to it. It returns TRUE if all nodes have sent else it returns FALSE.
- Line 12: ***ConsolidateResults*** function is also used by the coordinator only to combine the results from all nodes and filters for frequent itemsets using minSup
- Line 13 increments the counter c

- Line 14 triggers the agent at each node to execute the mining function at each node. It passes the counter c as an argument.
- Line 16 outputs the results. The results are displayed at the master node.

The basic flow chart of the proposed algorithm is shown below.

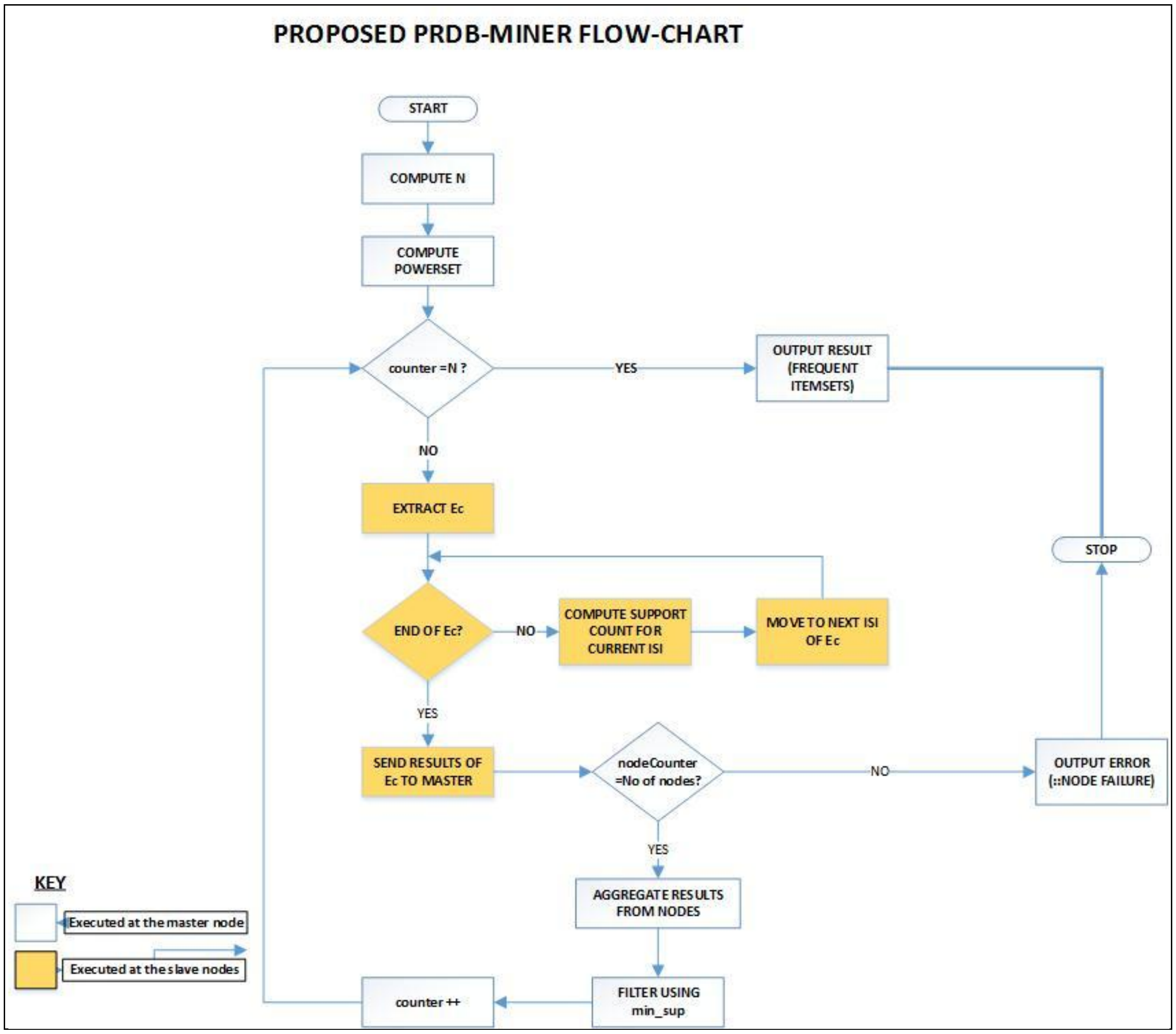

PROPOSED PRDB-MINER FLOW-CHART
START
COMPUTE N
COMPUTE POWERSET
counter =N ?
YES
OUTPUT RESULT (FREQUENT ITEMSETS)
NO
EXTRACT Ec
STOP
END OF Ec?
NO
COMPUTE SUPPORT COUNT FOR CURRENT ISI
MOVE TO NEXT ISI OF Ec
YES
SEND RESULTS OF Ec TO MASTER
nodeCounter =No of nodes?
NO
OUTPUT ERROR (::NODE FAILURE)
YES
AGGREGATE RESULTS FROM NODES
KEY
Executed at the master node
Executed at the slave nodes
FILTER USING min_sup
counter ++

Figure 19: Proposed PRDB-Miner Flow Chart

4.2.4 Assumption made for the proposed algorithm

The scope of this project is to enhance the RDB-Miner algorithm such that it works in a parallel environment. There are therefore some assumptions that had to be made in order to narrow the scope of the project. The assumptions are as follows:

1. The nodes in the cluster are connected by a high speed network and they are close to each other, that is, they are on the same local area network. This enables us to make a further assumption that there is no delay in relaying of information from one node to the other.
2. The high speed network connecting the nodes will not fail during the data mining activity. This allows us to ignore handling of 'dead' nodes. If one node in the cluster is down, the algorithm will output an error and terminates. The master node will be responsible for monitoring response time for the nodes. If a node does not respond within a given timeframe, the master will assume that the node has failed, hence output an error.

4.3 Test Data

In this paper we will be using processed Multi-Domain Sentiment Dataset which contains product reviews taken from Amazon.com from many product types (domains). We downloaded the data from (http://snap.stanford.edu/data/web-Amazon-links.html) the same data that was used by J. McAuley and J. Leskovec "Hidden factors and hidden topics: understanding rating dimensions with review text, RecSys, 2013" [27].

We used reviews for Amazon Instant videos, Automotive, Beauty and Baby categories. Some domains have hundreds of thousands of reviews; others have only a few hundred. We mixed the reviews such that each node can have a variety of reviews. The reviews contain star ratings 1 to 5 stars. The data is in a structured format and consists of four columns namely; ProductID – a string that uniquely identifies a product, ClientID – a string that uniquely identifies the reviewer, DateReviewed – date of review and the Rating – an integer between 1 and 5 which shows the review. The total number of records in the combined dataset is approximately 1.7million records.

4.4 Performance Factors to be considered

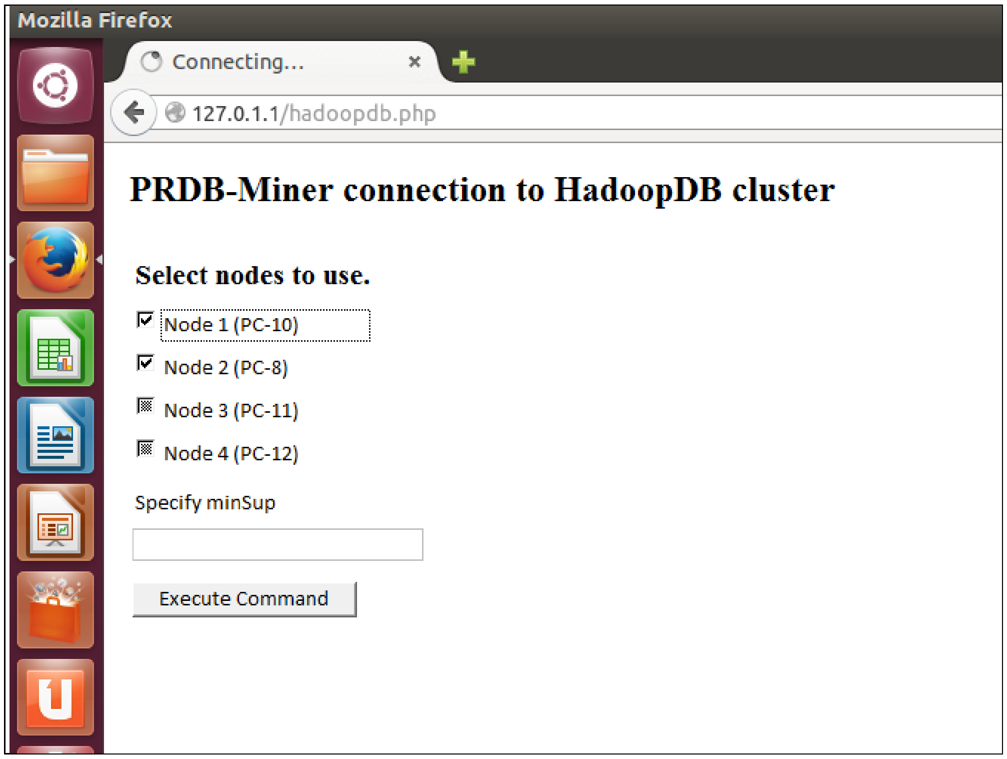
In this paper we will observe **Execution time** as a performance indicator. Execution time denotes the time taken (in seconds) by the algorithm to produce all association rules with the specified support. The execution time will be displayed and observed on the master node of the cluster. Three variables will be altered as the performance will be observed. The variables to be altered are ***minSup***, ***number of cluster nodes*** and ***volume of data*** being worked on.

4.5 Algorithm Implementation

In this paper we intend to use the proposed algorithm to develop a simple Recommender System. The algorithm will extract rules that allow the user to make the following conclusion:

Product X => Product Y, Product Z; this means that customers who liked Product X also liked products Y and Z. The minSup variable to be used is absolute support count. For instance if support for rule X=>Y is 5 it means that 5 customers who liked product X also liked product Y.

A simple interface was developed using PHP. The screenshot below shows the simple user interface developed. The user will be able to select the nodes which he wants to use as well as specify the minSup from the interface.



PRDB-Miner connection to HadoopDB cluster

Select nodes to use.

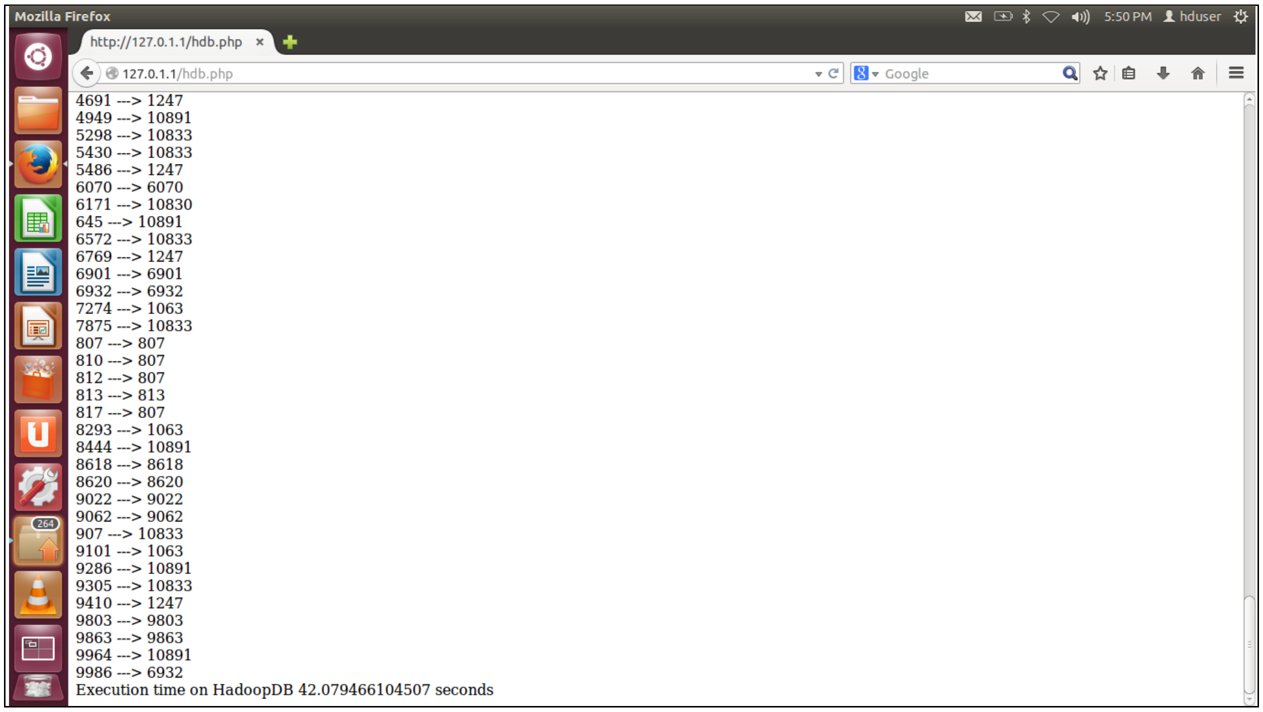
- [✓] Node 1 (PC-10)
- [✓] Node 2 (PC-8)
- [ ] Node 3 (PC-11)
- [ ] Node 4 (PC-12)

Specify minSup

Execute Command

Figure 20: User Interface

Some chunks of the algorithm will be installed on each node. These chunks will be invoked from the master node/laptop. Each node will extract local frequent item sets and store them into a table called **localAssociations.** Federated tables for each localAssociation table will be created at the master node. The master node will aggregate the frequent item sets from nodes and eventually display the results depending on the specified minSup. The execution time taken to generate the rules will then be displayed at the end of the list of rules. Below is a screenshot that depicts this.



```
4691 ---> 1247
4949 ---> 10891
5298 ---> 10833
5430 ---> 10833
5486 ---> 1247
6070 ---> 6070
6171 ---> 10830
645 ---> 10891
6572 ---> 10833
6769 ---> 1247
6901 ---> 6901
6932 ---> 6932
7274 ---> 1063
7875 ---> 10833
807 ---> 807
810 ---> 807
812 ---> 807
813 ---> 813
817 ---> 807
8293 ---> 1063
8444 ---> 10891
8618 ---> 8618
8620 ---> 8620
9022 ---> 9022
9062 ---> 9062
907 ---> 10833
9101 ---> 1063
9286 ---> 10891
9305 ---> 10833
9410 ---> 1247
9803 ---> 9803
9863 ---> 9863
9964 ---> 10891
9986 ---> 6932
Execution time on HadoopDB 42.079466104507 seconds
```

Figure 21: Rules and execution time

Below is a portion of the code for the proposed Parallel RDB-Miner algorithm:

```
<?php
require "database.php";
/** Miner class to perform data mining operations  */
class Miner {
   private $db;  // database object
   /*** Miner constructor    */
   public function Miner() {
      // get database object
      $this->db = Database::getDBO();
   } /** * select all the reviews for product X and count them (number of reviews =
NumOfProdReviews)   */
   public function process_reviews() {
        $sql = "SELECT productid, SUM(rating) as numOfReviews FROM reviews GROUP BY
productid";

        $result = $this->db->query($sql);

        if ($result->num_rows > 0) {          // output data of each row

            while ($row = $result->fetch_assoc()) {

                //echo "productid: " . $row["productid"] . " - Total Rating: " . $row["numOfReviews"]
. "<br>";

                //insert product reviews in the database

                $this->set_prod_reviews($row["productid"], $row["numOfReviews"]);

                //set rules

                $this->set_rules($row["productid"]);

            }

        } else {

            echo "0 results";

        }

    }    /** * insert productID & NumOfProdReviews into ProdReviews

     * @param varchar $productID

     * @param int $numOfReviews

         public function set_prod_reviews($productID, $numOfReviews) {

        $sql = "INSERT INTO prodreviews(productid,totalrating)
VALUES($productID,$numOfReviews)";

        if ($this->db->query($sql) === FALSE) {

            die("Error: " . $sql . "<br>" . $this->db->error);

        }

    }
    /**
     * find the most reviewed product (besides productX) by these users
     * (which product was reviewed the most by this set of users as a whole; lets call it Product Y)
     * insert into Rules values productX (LHS field) ,ProductY (RHS field)
     */
    public function set_rules($productid) {
        $sql = "SELECT productid, SUM(rating) as numOfReviews FROM reviews  WHERE
userid IN(SELECT userid FROM reviews WHERE productid=$productid)GROUP BY
productid ORDER BY numOfReviews DESC";
        $result = $this->db->query($sql);


        if ($result === FALSE) {
            die("Error: " . $sql . "<br>" . $this->db->error);
        }


        if ($result->num_rows > 0) {
            $row = $result->fetch_assoc();
            $sql = "INSERT INTO rules(lhs,rhs) VALUES($productid,{$row['productid']})";
            if ($this->db->query($sql) === FALSE) {
                die("Error: " . $sql . "<br>" . $this->db->error);
            }
        }
    }


}
```

CHAPTER 5 – RESULTS AND ANALYSIS

After the implementation of the PRDB-Miner algorithm, its performance was tested and compared to the results obtained from the previous experiment carried out for the original RDB-Miner. The execution time is the time taken by the simple application to generate the rules and display them on the user interface as illustrated in Figure 20 in the previous chapter. The results obtained are as follows:

5.1 Results for varying minSup

This experiment was carried out with 1.7 million records horizontally partitioned and uploaded into three cluster nodes. The minSup variable used is the absolute count. The graphical results are shown below.

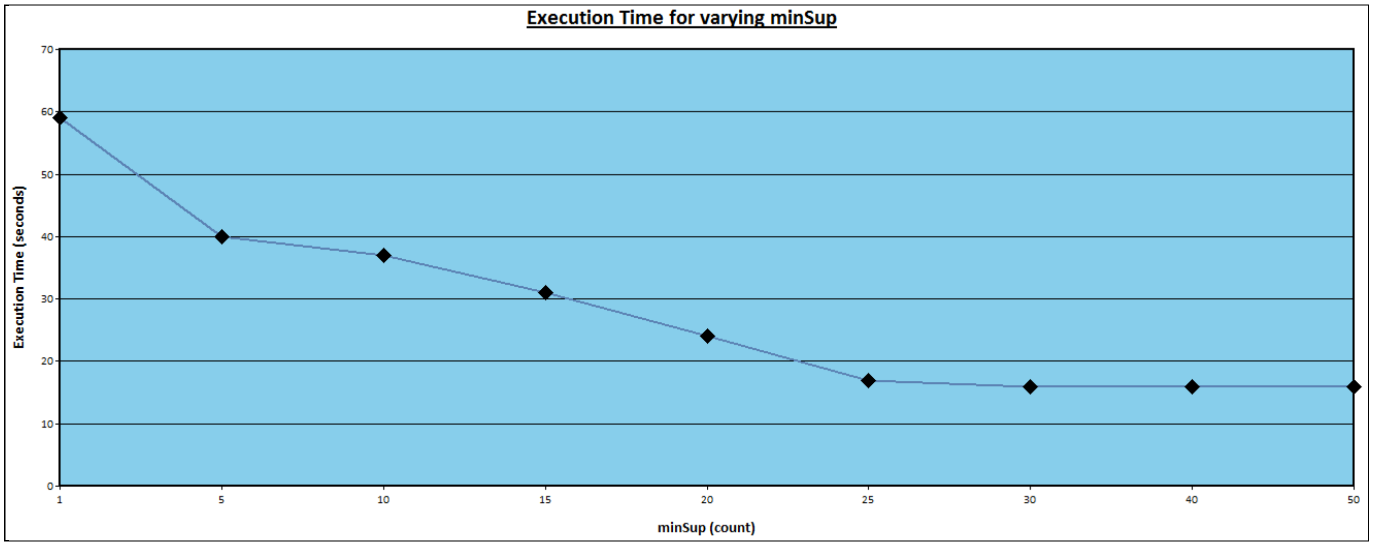


Figure 22: Results for varying minSup for PRDB-Miner

5.1.1 Observation

The execution time gradually declined as the minSup increased. The execution time had a steady decent from minSup of 0 to 25 but it eventually became constant as the minSup continued to increase.

5.1.2 Explanation

Low minSup value means that the algorithm will have to deal with a lot of data hence the longer execution times. The master node is responsible for filtering for frequent itemsets using the minSup value; if the value is low it means a lot of itemsets will qualify as frequent itemsets. If the number of records remains high for most cycles of the algorithm, it results in longer execution time. However as the minSup increases, it means that the algorithm filters out most of the records and disqualifies them as infrequent itemsets. As the algorithm continues to scan the node databases it will be left with just a few records to deal with hence it becomes faster to generate the rules.

As the minSup continues to increase, the execution time becomes constant because the effects of all big minSup values will be the same. For example if there is no frequent itemset with a support of 30 in the whole cluster, if we put a minSup of 31 and if we put 50, the results will the same because no frequent itemset will be found in both scenarios.

5.1.3 Comparison to RDB-Miner

Below is the graph from our previous project. Both RDB-Miner and PRDB-Miner behaves in a similar way; the execution time declined as the minSup values increased. The above explanation for PRDB-Miner fits well for the original RDB-Miner. However execution times on a single cluster environment are significantly higher than in a multi-cluster environment.

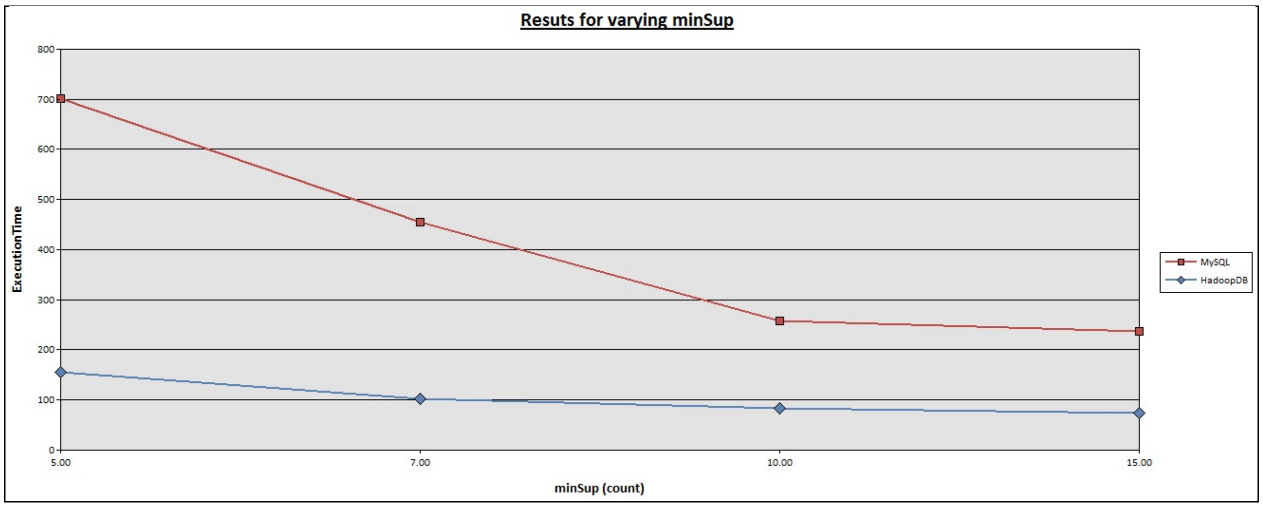


Figure 23: Results for varying minSup for original RDB-Miner

5.2 Results for varying data volumes

The algorithm was also tested for varying volumes of data. This experiment was carried out using three nodes. Each node was allocated an equal potion of the whole data. The records in each partition were equally increased by 1 000 000 records during each experiment. The total number of records used in the three experiments was, 1.2 million, 1.5 million and 1.8 million respectively. The minSup was set to default 5 during the whole experiment. Below is the graph for the execution time.

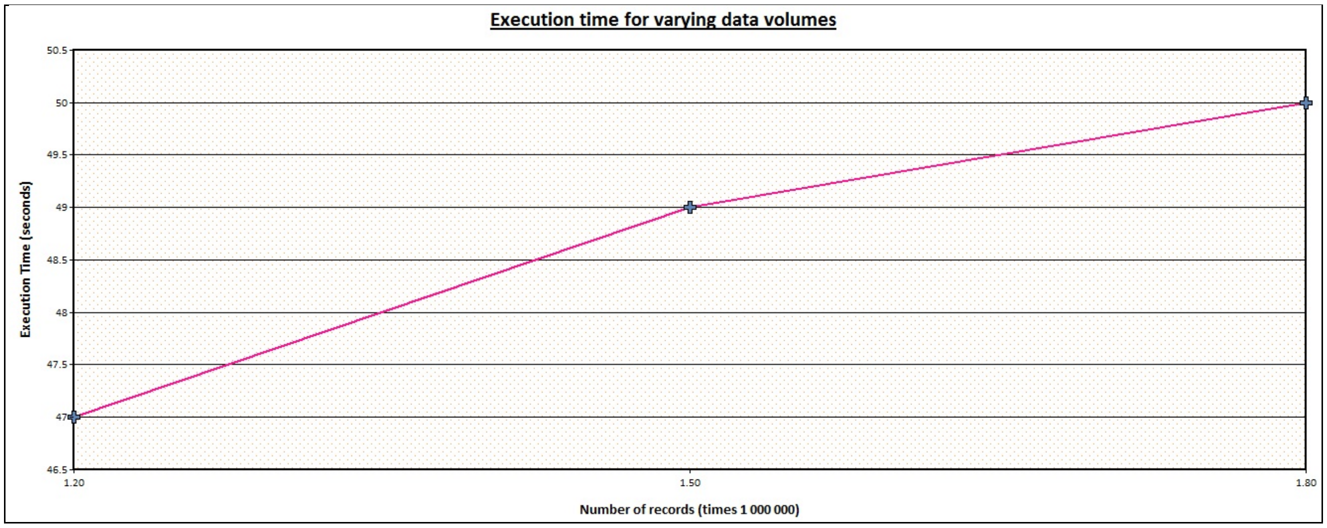


Figure 24: Results for varying volumes of data for PRDB-Miner

5.2.1 Observation

The execution time gradually increased as the volumes of data increased. However the increase was not very significant; an average of 2 to 3 seconds between experiments was observed.

5.2.2 Explanation

As the volume of data increases it means more data items and consequently more frequent itemsets. The more data there is the higher the execution time because the number of records remains high for most cycles of the algorithm. More records imply that large volumes of data will be exchanged between slave nodes and the master node resulting in some overheads. Also there will be strain on the master node as it will be having more data to filter and aggregate.

5.2.3 Comparison to RDB-Miner

Both RDB-Miner and PRDB-Miner behaves in a similar way when subjected to increasing volumes of data; the execution time increases as the data volumes increased. The above explanation for PRDB-Miner fits well for the original RDB-Miner. However execution times on a single cluster environment are significantly higher than in a multi-cluster environment. Below is the graph for RDB-Miner algorithm.

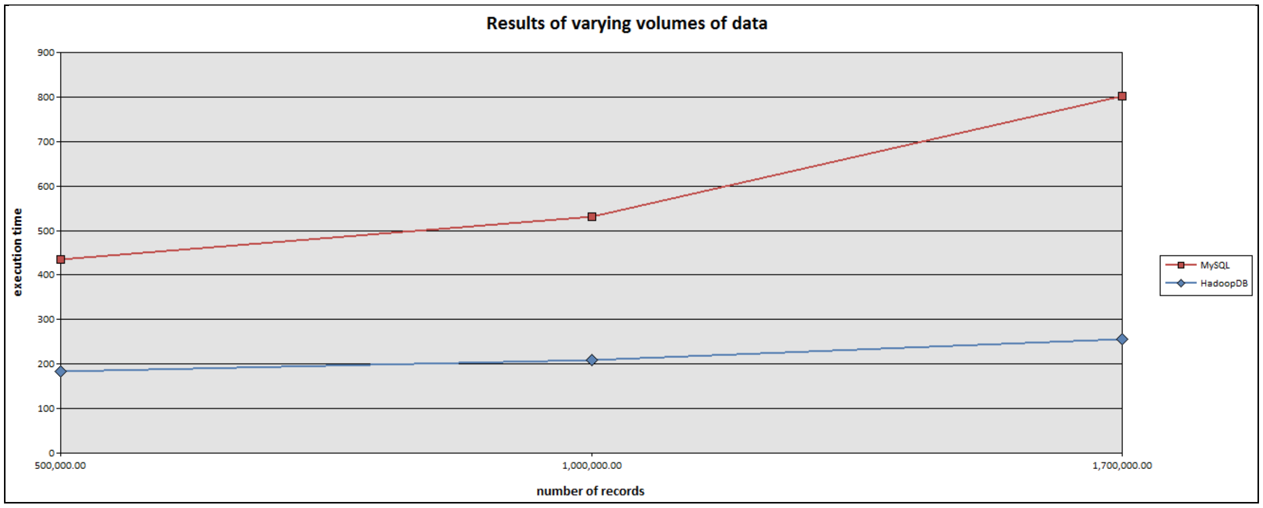


Figure 25: Results for varying data volumes for original RDB-Miner

5.3 Results for varying number of cluster nodes

The algorithm was also tested on varying number of nodes in the cluster. The maximum number of nodes we could use was four. The number of records and minSup remained constant in this experiment. 1.7 million records and minSup of 5 was used in this experiment. Below is the graph of results observed:

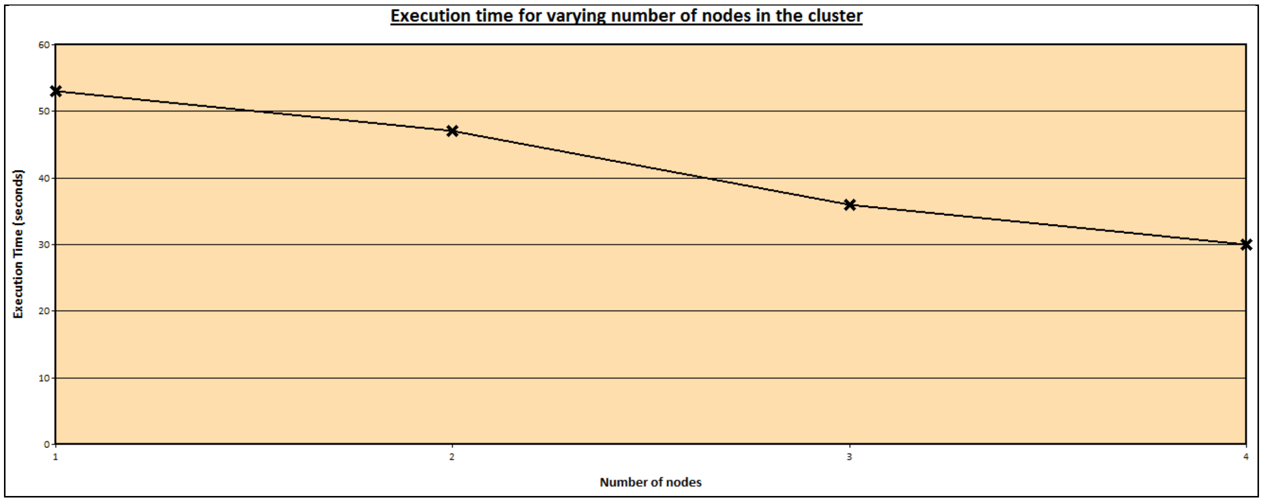


Figure 26: Results for varying number of cluster nodes for PRDB-Miner

5.3.1 Observation

The execution time gradually decreased as the number of nodes increased.

5.3.2 Explanation

The steady and gradual decrease of the execution time is as a result of increased "division of labor". Since the number of records remained constant as the number of nodes increased; it means each node received a smaller portion of the data than in the preceding run. Consequently this resulted in a shorter processing time for each node.

5.3.3 Optimum number of nodes

It is our hope that there exist an optimum number of nodes for the algorithm. The optimum point is where we can get the best performance of the algorithm. As the number of nodes continues to increase after reaching this point, it is most likely that the performance of the algorithm will start to fall because the data will be continually partitioned into smaller and smaller chunks such that each node will have very few records. Each node will have very little to do. The major task will be that of filtration and aggregation. This will result in a bottleneck at the master node since it will be faced with a lot of records to deal with. In such cases parallelization will be of no effect as one node will be left with much of the work. The optimum number of nodes is dependent on the number of records being acted upon. For fewer records the optimum number will be smaller. Below is an illustration of this scenario:

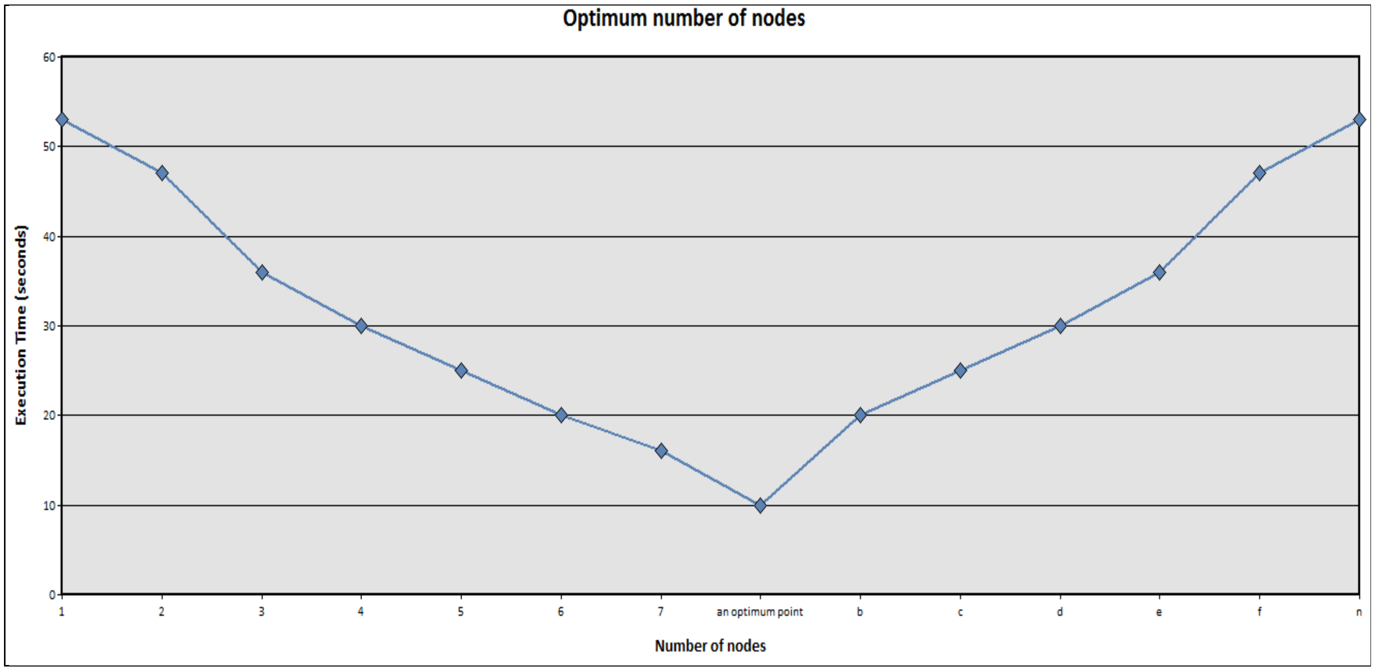


Figure 27: Optimum number of nodes

CHAPTER 6: CONCLUSION AND FUTURE WORK

The proposed algorithm, Parallel RDB-Miner performed remarkably and better than the original RDB-miner in the experiments carried out. It was also concluded that Parallel RDB-miner works well for relations that have a few number of attributes but have a large number of records. A high number of attributes results in a higher number of iterations thereby resulting in poor performance.

In future we hope to find a formula that calculates the optimum number of cluster nodes given the number of records. We also desire to further enhance the algorithm such that it becomes more resilient to node failure since this is a reality we cannot afford to ignore. Also we intend to test the algorithm on other open source SQL-on-Hadoop platforms such as Hive.

CHAPTER 7: PUBLICATIONS FROM THESIS

This chapter briefly highlights the publications that have been worked out during this research work.

7.1 Research paper accepted by journal

The research paper entitled "**ASSOCIATION RULE MINING FOR STRUCTURED DATA IN BIG DATA USING PARALLEL RDB-MINER ALGORITHM**" by Tererai Tinashe Maposa and Manoj Sethi, has been accepted for publication by the International Journal of Computer Science and Mobile Computing in Volume 4, Issue 6. The Paper ID is V4I6201599a64. The paper discusses the new algorithm designed in this thesis and the results obtained from the implementation.

REFERENCES

[1] ZHANG Yongliang ,QIN Jie, ZHENG Shiming, “Research On Distributed Mining Algorithm For Association Rules Oriented Mass Data”, Nanjing University of Posts and Telecommunications, Nanjing, China.

[2] Lingjuan Li, Min Zhang, “The Strategy of Mining Association Rule Based on Cloud Computing”, College of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China.

[3] Azza Abouzeid1, Kamil BajdaPawlikowski, Daniel Abadi1, Avi Silberschatz1, Alexander Rasin2, “HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads”**,** Yale University, Brown University.

[4] Abdallah Alashqur, “RDB-MINER: A SQL-Based Algorithm for Mining True Relational Databases” Faculty of Information Technology Applied Science University Amman, JDN

[5] Mingyuan An, Yang Wang, Weiping Wang, Ninghui Sun **—** “Integrating DBMSs as a Read-Only Execution Layer into Hadoop", University of Chinese Academy of Sciences Beijing, China.

[6] Hadoop. Web Page. hadoop.apache.org/core/.

[7] HadoopDB Project. Web page.db.cs.yale.edu/hadoopdb/hadoopdb.html.

[8] Daniel J. Abadi, “Data management in the cloud: limitations and opportunities,” Bulletin of the IEEE Computer Society Technical Committee on Data Engineering”

[9] Andrew Pavlo, Erik Paulson, Alexander Rasin, “A comparison of approaches to large-scale data analysis,” in Proc. SIGMOD’09, 2009, p. 165.

[10] Jaiwei Han and Micheline Kamber, “Data Mining Concepts and Techniques”, page 298.

[11] //bigdataplanet.com

[12] Xindong Wu, Fellow IEEE, Xingquan Zhu, Senior Member, IEEE, Gong-Qing Wu, and Wei Ding, Senior Member, IEEE, “Data Mining with Big Data”, IEEE Conference 2014.

[13] Marko Grobelnik, Jozef, “Big Data Tutorial”, Stefan Institute, Ljubljana, Slovenia, 2012.

[14] Robert Vrbić, “Data Mining and Cloud Computing”, University Vitez, Travnik, Bosnia and Herzegovina, 2012

[15] Ruxandra-Ştefania, “Data mining in Cloud Computing”, PETRE, Bucharest Academy of Economic Studies.

[16] Kala Karun, “A Review on Hadoop – HDFS Infrastructure Extensions”, Chitharanjan. K Sree Chitra Thirunal College of Engineering Thiruvananthapuram.

[17] Chen He, Derek Weitzel, David Swanson, “HOG: Distributed Hadoop MapReduce on the Grid”, Ying Lu Computer Science and Engineering University of Nebraska – Lincoln.

[18] Aysan Rasooli, “A Hybrid Scheduling Approach for Scalable Heterogeneous Hadoop Systems”, Department of Computing and Software, McMaster University, Hamilton, Canada.

[19] Hortonworks WebPage: https:// hortonworks.com/.

[20] Apache Software Web Page: https://apache.org/.

[21] Dr (Mrs). Sujni Paul, “Parallel and Distributed Data Mining”, Karunya University, Coimbatore, India.

[22] R. Hemamalini, Dr. L. F. Josephine Mary, “An Analysis on Multi-Agent Based Distributed Data Mining System”, Research scholar St. Peter’s University, Professor & HOD, MCA Dept. Sri Ram Engineering College, Veppampet.

[23] Josenildo C. da Silva, Chris Giannella, Ruchita Bhargava, Hillo Kargupta and Matthias Klusch, “Distributed Data Mining and Agents”, Department of Computer Science and Electrical Engineering University of Maryland Baltimore County, Baltimore, USA.

[24] Byung-Hoon Park and Hillol Kargupta, “Distributed Data Mining: algorithms, Systems and Applications”, Department of Computer Science, University of Maryland, Baltimore County.

[25] Dr (Mrs).Sujni Paul, Associate Professor, “An optimized distributed association rule mining algorithm in parallel and distributed data mining with XML data for improved response time”, Department of Computer Applications, Karunya University, Coimbatore, Tamil Nadu, India.

[26] Jianwei Li, Ying Liu, Alok Choudhary, “Parallel Data Mining Algorithms for Association Rules and Clustering”, DTKE Center and Grad. Univ. of CAS, Northwestern University.

[27] http://snap.stanford.edu/data/web-Amazon-links.html

[28] *http://ibmdatamag.com/.*