A Performance Comparison of Hadoop HBase and Cassandra

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Software Engineering

Submitted By

Ruchita Malik

(801331025)

Under the supervision of:

Dr. Seema Bawa

Professor, CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

July 2015

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*A Performance Comparison of Hadoop HBase and Cassandra*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Seema Bawa* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

(Ruchita Malik)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

15/7/15

(Dr. Seema Bawa)

Professor, CSED

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. S. Bhatia)
Dean (Academic Affairs)
Thapar University
Patiala

i

Acknowledgement

First of all I would like to thank the Almighty, who has always guided me to work on the right path of the life. It is a great privilege to express my gratitude and admiration towards my respected supervisor **Dr. Seema Bawa**, Professor, Computer Science & Engineering Department. She has been an esteemed guide and great support behind achieving this task. This work would not have been possible without the encouragement and able guidance. I also thank my supervisor for her time, patience, discussions and valuable comments. Her enthusiasm and optimism made this experience both rewarding and enjoyable. I am truly grateful to her for extending her total co-operation and understanding whenever I needed help and guidance from her.

I am also heartily thankful to **Dr. Deepak Garg**, Associate Professor and Head, Computer Science & Engineering Department and **Dr. Damandeep Kaur**, PG coordinator, for motivation and providing uncanny guidance and support throughout the preparation of the thesis report.

I will be failing in my duty if I do not express my gratitude to **Dr. S.S. Bhatia, Senior Professor and Dean of Academic Affairs**, for making provisions of infrastructure such as library facilities, computer labs equipped with net facilities, immensely useful for the learners to equip themselves with the latest in the field. I am also thankful to the entire faculty and staff members of Computer Science and Engineering Department for their direct-indirect help, cooperation, love and affection, which made my stay at Thapar University memorable.

Last but not the least, I would like to thank my family for their wonderful love and encouragement, without their blessings none of this would have been possible. I would also like to thank my close friends for their constant support.

Abstract

Databases are an important part of today's life. With the increase of data all around, it is important to find secure places to store it. One cannot afford to store all the data on one system, which is when data models come into role. The use of web has increased to a significant level. There is a vast variety of web and mobile applications in the present scenario. With this vast variety of data, the type of data also varies. The traditional database systems are not that capable to store the unstructured data. As a solution of this NoSQL data stores have come into existence. There are a number of NoSQL data stores for e.g. MongoDB, Cassandra etc. Data queries in these data stores are implemented quickly. The report is a performance comparison of Cassandra and HBase which has been done on a LINUX platform.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Technological advancement has greatly impacted the world of computing today. The introduction of numerous computing resources and their increasing application online has given rise to huge amount of data. This structured and unstructured data is called Big data. Big data is a buzzword used to describe data which is difficult to analyze due to its size. As the name suggests Big data may come across as a reference to the volume of data but it's majorly associated as a technology; a technology that helps in handling large of data and storage with tools and processes. Today, the major problem revolves around fast retrieval along with high performance. A mechanism to solve this problem is distributed systems. A distributed system is the one in which a network connects a group of autonomous computers, which share resources and coordinates activities in order to provide a computing facility and identify itself as a single unit. A distributed database is one example of a distributed system in which the storage devices are not connected to a single machine rather they are managed by a distributed database management system.

Hadoop is one such system which follows clustered approach and stores large data sets. It has a large throughput along with high fault tolerance; works on commodity hardware and separation of metadata from actual data is one of its key features. Google File System and Google's MapReduce have inspired Hadoop's file system architecture and data computational paradigm. Analysis, computation and transformation of huge amounts of data are done with the help of MapReduce paradigm. Since Hadoop is based on MapReduce paradigm it has gained importance over the years because of its scalability, reliability and ability to perform analysis and computation on enormous amounts of data. All the leading organizations like Google, Facebook and Yahoo are using it [1]. The largest cluster of Yahoo is of 3500 serves and there is a span of 25,000 servers which stores 25 petabytes of application data.

The Apache Cassandra database is used when scalability and high availability is required without compromising performance. It is used for mission-critical data as it has linear scalability and proven fault-tolerance on cloud infrastructure. It supports best replication

across multiple datacenters with lower latency. It also provides powerful built-in cashing along with convenience of column indexes which perform just as log updates.

Cassandra is used in over 1500 companies with large and active data set such as CERN, Facebook, eBay, Instagram, Reddit and The Weather Channel. With 75,000 nodes and storage of 10PB of data, its largest production deployments are at Apple. Netflix (2,500 nodes, 420) and eBay (over 100 nodes, 250 TB) are other large Cassandra installations.

1.1 Rise of Big data

Big data embraces the fact that there is a lot of information around us than ever before which can be put to phenomenal new uses. Big data is not same as the internet although the internet helps in collection and sharing of huge amounts of data easier. Big data encircles around the face that a lot of knowledge can be derived from data when dealing with large body of information than what we can learn with smaller amounts.

During the third century BC, the collection of human knowledge was believed to be at the Library of Alexandra. Whereas today, with the current data present in the world; if each living person is given an amount of information, it'll be 320 times more as predicted to be in Alexandra library- estimated to be 1200 Exabyte [1]. Also, it'll take five stacked up piles of CDs which would all reach to the moon if all this data is written on CDs.

This rise of data wasn't there till 2000, a quarter of the information of the world was digital back then and the rest of the information was kept recorded on paper and analog Medias. The explosion of data happened as digital data expands with a speed of doubling itself every three years. Now, less than 2% of stored information is nondigital.

The enormous size of data misled sometimes just in terms of size. Data nowadays is rendered into aspects which haven't been quantified as of now. This is called datafication. It is a technological trend which transforms computerized data into new forms of value. For example, location was initially associated with longitude and latitude, and now with GPS satellites. Even relationships and "likes" are being datafied cause of Facebook and other social medias[2] . This data is now put in to brilliant use now. Now a computer is not taught how to do things, such as translate a language or drive a car but is

now fed with enough data to check with the probability of substituting a more appropriate word in case of translation and also if the traffic light is green/red in case of driving. This is made possible with the help of powerful processors and smart algorithm which uses math based on statistics [3].

1.1.1 Characteristics of Big data

Big data is described by the following characteristics:

a. Volume: It is the size of data which helps in discovering the potential and value of data in hand and if can be considered Big Data or not.
b. Variety: It is the category the data under consideration belongs. This is an important fact which the data analysts need to know.
c. Velocity: The data generation speed or how rapidly the data is generated and processed.
d. Variability: It is the inconsistency which a data may possess. This factor can cause problems to the ones analyzing the data.
e. Veracity: It refers to the quality of data as it depends on how the data is captured. Accuracy of analysis is dependent of the veracity of data.
f. Complexity: When large amount of data comes from different sources, data management becomes complex. The data is then connected and correlated so that the information can be extracted from the data in hand.

Huge Volumes and different varieties of data have led to discovery of new types of databases called NoSQL databases (NoSQL indicates Not Only SQL). Examples of NOSQL databases are Mongo DB, Dynamo DB, Oracle NOSQL, Cassandra, Hadoop etc. These databases store the unstructured data in an efficient manner. The retrieval of the data using queries is easy and fast. The data here is determined by the combination of a key and a corresponding value. A NoSQL database handles unpredictable and unstructured data; to simply state it doesn't follow the typical relational database management system (RDBMS) structure. Fast growing companies are now switching to NoSQL database from traditional RDBMS as its structure is not limited to that of tables

and it provides a fault tolerant and distributed architecture without succumbing to ACID properties.

1.2 Challenges with traditional databases

Following are some of the challenges faced while working with traditional databases:

a. It is not suitable for huge volume of data (petabytes of data) which have mixed data types such as images, text, videos etc.
b. It is not able to scale large volumes of data. For example, the Government of India's AADAR project has 15-20 petabytes of data which can't be scaled using traditional databases [6].
c. Scale out: Operations such as 'Read' and 'Write' are cache dependent.
d. Scale up: It has limited CPU (Memory and Processing) capabilities.
e. RDBMS model is complex as it includes parsing, locking, logging etc.
f. Satisfying to ACID properties causes hindrance for scaling which is much relaxed with NoSQL databases.

When referring to NoSQL databases understanding of CAP theorem should be done. CAP Theorem states the three basic requirements which should be considered while designing application in a distributed system. These requirements are Consistency, Availability and Partition Tolerance. Consistency insures that the data after an execution operation is consistent. For example, after an update operation is performed, all the clients see the same data [7]. Availability guarantees service with no downtime. Partition Tolerance means if communication between servers is unreliable the system continues to function. Theoretically, when dealing with distributed system it's not feasible to achieve all the three requirements. Hence, NOSQL database uses combinations of these requirements. For example, HBase fulfills C and P and Cassandra fulfills A and P.

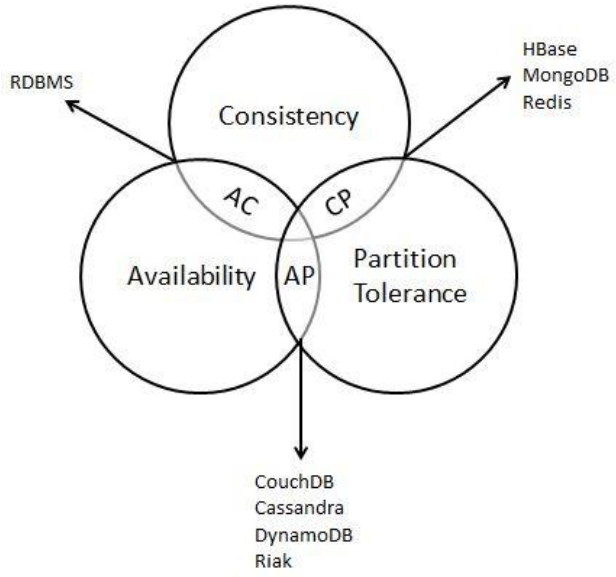

Figure 1 CAP Theorem

NoSQL database is BASE (Basically Available, Soft State, Eventual Consistency) and not ACID system. Basically Available ensures that the system guarantees availability within the terms of CAP theorem. Soft State indicated that even with input the state of the system may change over time. This is due to Eventual Consistency which indicates that if a system doesn't receive input it will become consistent during that time. All NoSQL databases are schema-less unlike relational databases which have strong schemas. Migration in schema-less database needs to be carefully done as implicit schema accesses the data [7]. A version-controlled sequence helps in data migration of databases with strong schemas by saving every schema change.

In this growing world of big data, Google, a company who have their hands full with large quantities of data, have come up with a technology called Bigtable in 2006 [8]. Bigtable is not described as a database rather a persistent multidimensional map which is sparse and distributed; designed to run on commodity hardware and store petabytes of data. Bigtable partitions data using uniquely indexed row keys for distribution in a cluster whereas columns can be defined within rows accordingly making it schema-less. The two Apache database projects, Cassandra and HBase are based much on the original Bigtable definition. HBase is described as an open-source Bigtable implementation whereas Cassandra is descended from Amazon's Dynamo and Bigtable.

1.3 Hadoop HBase

Hadoop is an open-source Java based framework and a project of Apache Software Foundation, led by Yahoo, which processes and distributes huge data sets across that distributes and processes across clusters of servers. [9] High fault tolerance and reliability provided by Hadoop has allowed the movement from single server to multiple servers. Since it provides software capabilities enabling them to handle failures at the application level, it makes Hadoop a tough and power platform. It also helps in providing business insights by analyzing large amounts of structured and unstructured data gathered from various sources over the internet. Hadoop is able to process data present in different forms such as xml format, file format, database tables and extract the meaningful and significant information to as help in business analytics [10]. It is now becoming an industry standard framework to handle huge data volume.

1.3.1 Core Components of Hadoop

a. MapReduce: The algorithm was initially developed by Google. Hadoop implements the same algorithm which is used to process large scale data. MapReduce is a programming model which consists of JobTracker and TaskTracker. The jobs are submitted to the JobTracker by clients. JobTracker then assigns the appropriate work to the TaskTracker which further process that work to provide a useful result.
b. HDFS: Hadoop Distributed File System (HDFS) as the name suggests provides distributed storage of data. It comprises of NameNode, DataNode and Secondary NameNode. [11] Multiple replicas of data are created and are distributed over the multiple computers in the cluster.
c. YARN: Yet Another Resource Negotiator (YARN) The job tracking and job scheduling tasks which have been performed by MapReduce are divided into two: ResourceManager and ApplicationMaster. ResourceManager is a main component of YARN along with NodeManager and JobHistoryServer.

1.3.2 Hadoop Architecture

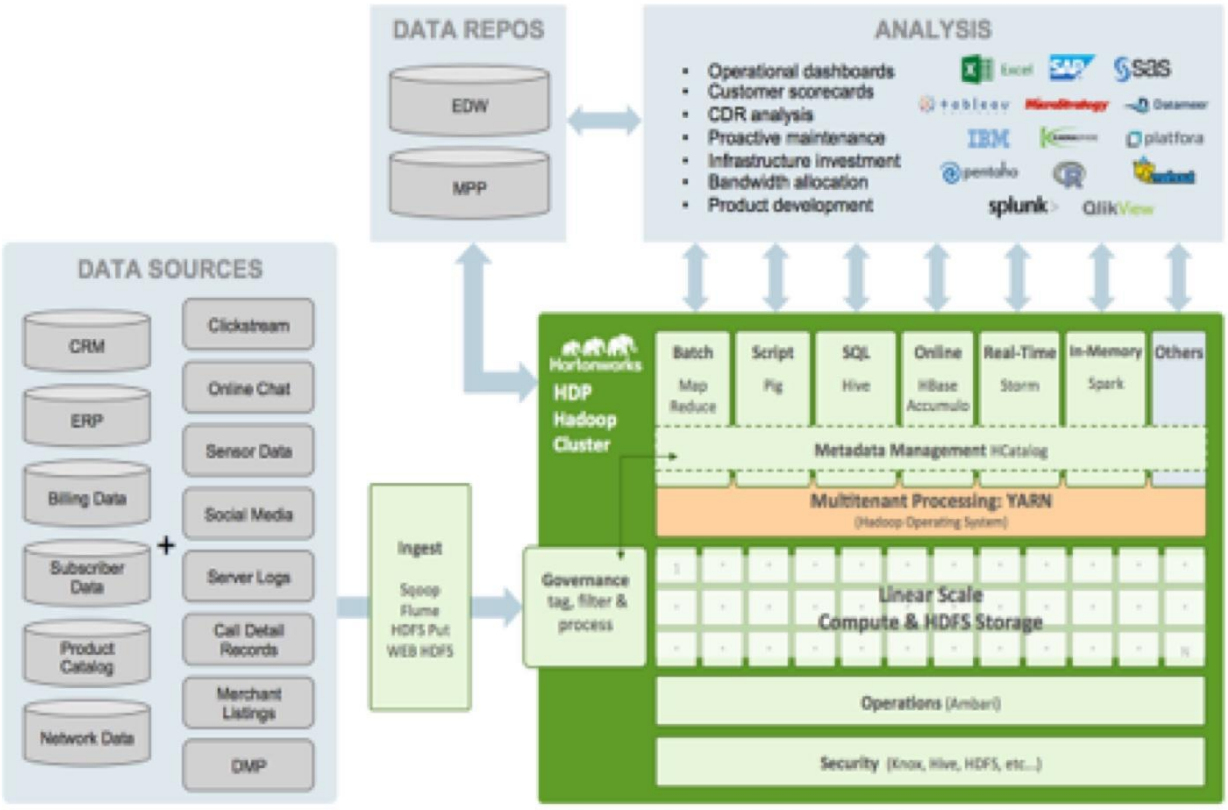

Figure 2 Layered Architecture of Hadoop [9]

Hadoop has a layered architecture where at the lowest level lies the Hadoop distributed file system (HDFS). In HDFS, all the data gathered from the various resources is stored. The layer above HDFS is Yet Another Resource Negotiator (YARN) which is responsible for allocation and planning of resources. YARN is responsible for the smooth functioning of MapReduce at it has divided the job scheduling and job tracking into two parts. MapReduce works over YARN and executes the job or work in two phases along with other components of Hadoop such as Pig, Hive etc. The programming model in Hadoop is MapReduce. These three, HDFS, YARN and MapReduce together form the core of Hadoop.

There are two types of nodes present in Hadoop namely master node and many slave nodes. Heartbeat communication is used by the master node to control the slave nodes along with giving them instructions. *MapReduceJobTracker* and HDFSNameNode are present in master node and *MapReduceTaskTracker* and *HDFS DataNode* are present

in slave nodes. DataNodes and TaskTrackers are present in master node for data locality. Workload of the slave should be different and isolated from the master node as the slave node required much more maintenance as compared to the master node and hence it is recommended that master and slave nodes work separately.

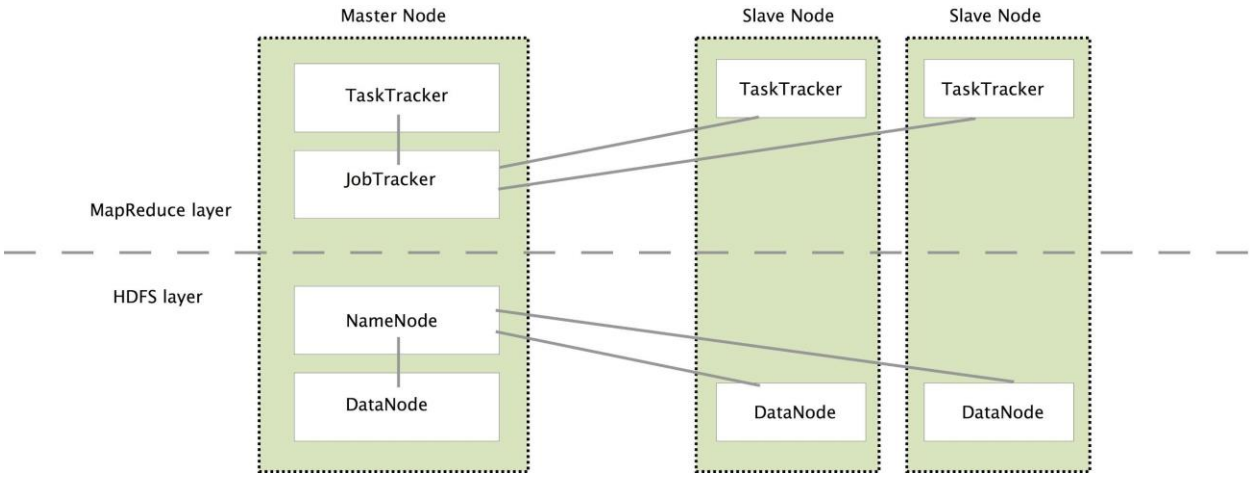

Figure 3 High Level Architecture of Hadoop

YARN splits job tracking and job scheduling in Hadoop 2.0 into two separate processes. This job tracking and scheduling is done by MapReduce. ResourceManager and NodeManager have replaced JobTracker and TaskTracker respectively. In Hadoop 2.0, the slave comprises of *YARN NodeManager* and *HDFS DataNode* and the master node have *YARN ResourceManager* and *HDFS NameNode*. For data locality, the Node Managers and Data Nodes are co-located [9].

1.3.3 Advantages and Disadvantages of Hadoop

Hadoop provides a lot of advantages over Cassandra in different respects such as:-

a. The Computation system is a Distributed type system. The network traffic used is less in amount due to computation of local data.
b. Partial failures are handled easily as the tasks performed are independent in Hadoop [12].
c. A rather simple model of programming.

d. Hadoop is easily Scalable system. A very minimal amount of code refactoring is required in order to run it on tens to thousands of machines [13].
e. Hadoop Distributed File System is a reliable, simple and robust system.
f. Hadoop Distributed File System can be used to store large chunks of data and provides faster access to data.
g. The system has high Fault Tolerance as data is duplicated at numerous amounts of nodes and can be recovered automatically easily.
h. MapReduce programs are written in Java which is a common programming language.
i. It can be used as an on-demand service.

Some of the major disadvantages of Hadoop are:-

a. It is hard to achieve Cluster management in Hadoop and is also very typical [14].
b. It is dependent on a single node (master) which can halt the complete system and hence requires more care.
c. The model of programming is a little confining in case of Hadoop and also very typical.
d. Hadoop does not handle small sized files efficiently.

1.3.4 Hadoop HBase

Hadoop is a distributed platform which has high reliability and scalability. Hadoop is a general name for several subsystems [15]. Apache HBase is a NoSQL column oriented distributed database management system which helps in real time read/write access to Big Data. HBase works on top of HDFS (Hadoop Distributed File System). HBase utilizes Hadoop's MapReduce programming model and the distributed processing paradigm of HDFS [16]. It hosts large tables containing numerous rows and columns which run across the commodity hardware. HBase on its own is a powerful database which provides capabilities to perform real-time queries with the speed of batch processing and key value store through MapReduce [17]. Hence, along with aggregate analytic reports HBase also allows to query for individual records.

HBase defines a four-dimensional data model with four coordinates defining each cell as shown in Figure 4. The following are four coordinates:

a. Row Key: A unique row key which doesn't have a datatype is associated with each row. This row key is treated as a byte array.
b. Column Family: Row data is organized into column families where every row has a set of similar column family but same column qualifies are not needed by column families across rows. Changes to column families are difficult and they need to be defined beforehand as they are as their own data files.
c. Column Qualifier: Column qualifiers are the actual columns of the column families. They are actually columns themselves.
d. Version: Data can be accessed for a specific version of the column qualifies and each column can have a set configurable number of versions.

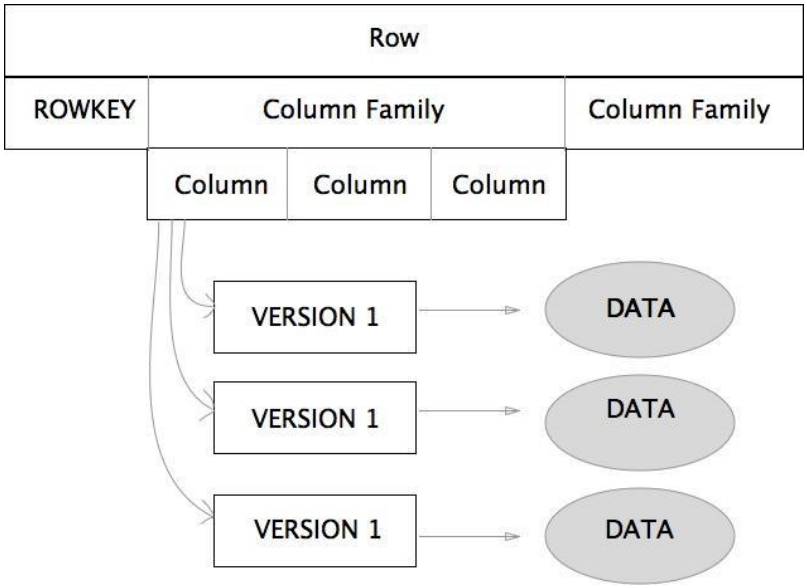

Figure 4 HBase Four-Dimensional Data Model

As shown in Figure 4, row key helps in accessing an individual row which contains one or many column families. Every column families further may have one or more columns/column qualifiers [30]. These columns have version. Thus, accessing data requires the knowledge of row key, column family along with the column and its version.

HBase data model is designed according to how the data will be accessed. To access data in HBase following are the two ways:

a. With the help of row key or scanning the table for a range of keys.
b. Or the data can be accessed with map-reduce in a batch manner.

HBase's main strength is cause of this dual approach as data can be accessed offline for batch analysis as well as it can be accessed in real-time using key values. For real-time access key/value store is used where the key is the row key and collection of column family is the value as seen in Figure 5.

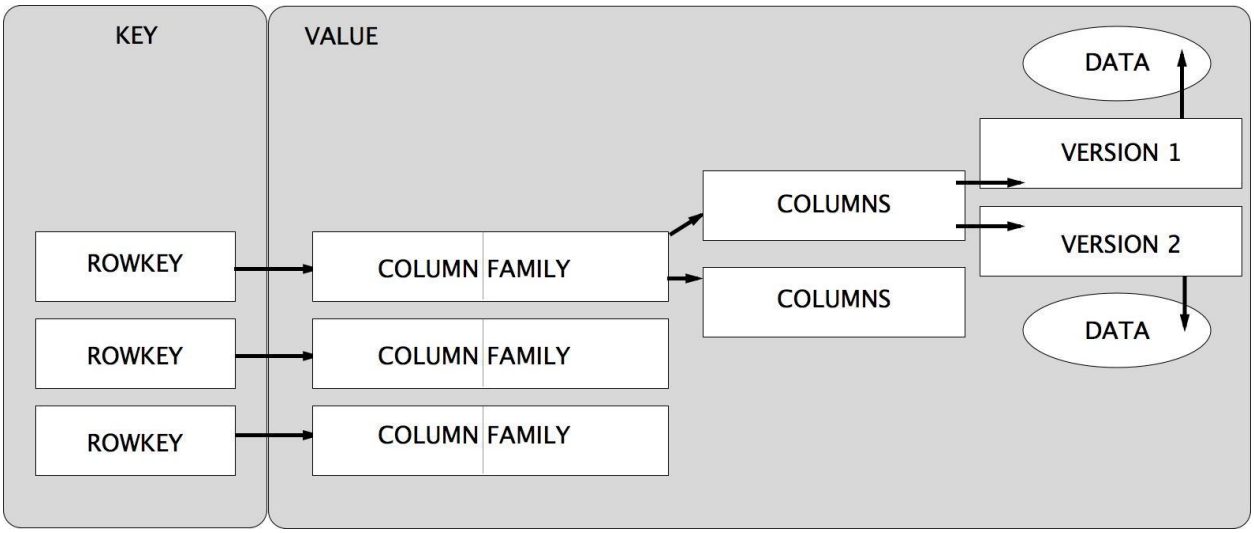

Figure 5 HBase as a Key/Value Store

Rows associated with the row keys can be retrieved. Also, a table scan be performed which is nothing but a set of rows which is retrieved by mentioning the starting and ending row keys. Values of columns cannot be queried in real-time; hence design of the row key is greatly significant. The major two reasons for which the design of row key is important are:

a. Row key control the real-time access which can be done in HBase as table scans operates with row keys.
b. Data is distributed across HDFS on the basis of row keys. Row keys should be different in order to distribute the data as if they start with the same tag, then majority of data will be stored on a single isolated node.

Hence how data is to be accessed is greatly dependent on the design of row keys. This data is stored across HDFS which is a core component of Hadoop.

1.3.5 HDFS

HDFS provides large scale reliable storage and is a Java-based distributed file system. The interface of HDFS is patterned on the UNIX file system. HDFS has shown massive data storage which has been up to 200PB along with providing scalability to 40,000 servers, where single cluster can be scaled to 4500 servers. The major difference of HDFS with distributed file systems is deployed on any type of hardware along with providing fault tolerance. It is recommended to be used where high throughput access to large data set is required.

HDFS also follows the master slave architecture where the master machine, *NameNode*, controls the access rights and manages namespace of the file system. The slave machines, DataNodes which is responsible for storing the data and regulating the data attached to the nodes and using it to perform tasks. Linux OS is used to run NameNode and DataNodes. Also, NameNode assisted by Secondary NameNode are software which runs on machines those who have Linux operating system. It also has one *Secondary NameNode* which assists *NameNode* for monitoring purposes

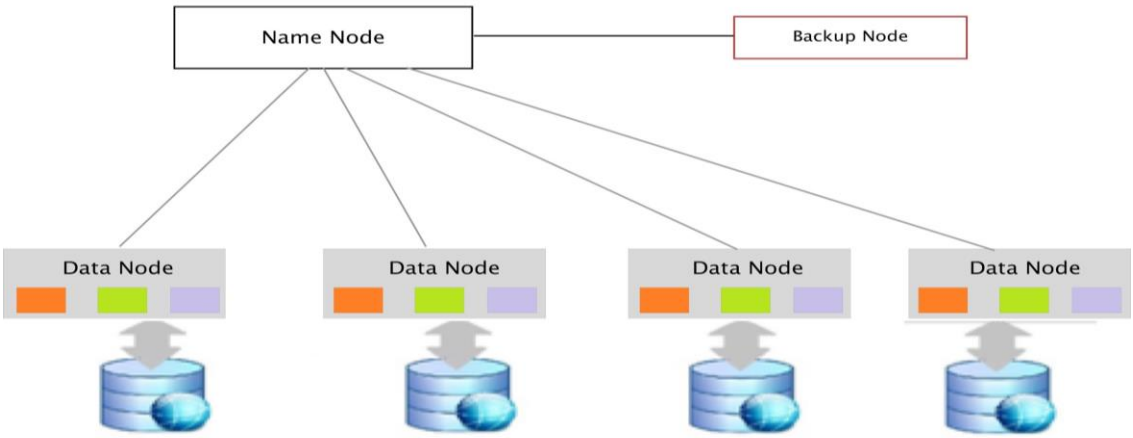

Figure 6 HDFS Architecture

NameNode: It is a system which is multithreaded in order to efficiently handle multiple clients making multiple requests. Execution of various functions such as opening and closing is done by it. The system and handles requests from multiple clients at a time. It allows the execution of functions like opening, closing, renaming of files and directories.

DataNode: A single NameNode handles multiple DataNodes in HDFS. DataNodes are responsible for storing data blocks which are created by splitting a file into multiple blocks. Read and write requests by clients are also handled by the DataNode. They are instructed by NameNode to create, delete and replicate blocks.

Secondary NameNode: A log file *edits* keeps the log of all the operations done in a file. This log is kept by the NameNode and all the changes are appended making the size of the log file large in case of busy cluster. The HDFS state is read from image file knows as fsimage by a NameNode and changes are done through edits. To keep the size of log within limits, Secondary NameNodes mergers the edits and fsimage periodically. Also, in case of failure of a NameNode the metadata and edit logs are recovered from Secondary NameNode making the system robust and fault tolerant.

1.3 Cassandra

Apache Cassandra™ is an open source NoSQL database which is greatly scalable. All forms of data be it structured, unstructured or semi-structured can be managed using Cassandra across multiple data servers and cloud. Its dynamic data model delivers maximum flexibility and fast response time along with being available, scalable and operationally simple across various servers with no single point of failure. Due to its architecture, any authorized user can access data by connecting itself to any node in the data center using Contextual Query Language (CQL) which has a very similar syntax to SQL (Sequel Query Language) [33]. The easiest way to interact with Cassandra is by CQL shell, cqlsh. Tables, keyspaces, insert and query tables and many more operations can be done using cqlsh. DataStaxDevCenter can be used if a graphical tool is convenient [34]. In production, a number of drivers are supplied by DataStax such that CQL statements are passed between client and cluster. OpsCenter helps in accomplishing administrative tasks.

Automatic data distribution to all nodes in a database cluster or ring is provided by Cassandra. No codes or programs are required from the developer or administrator to distribute this data across the cluster as data is partitioned transparently to all the nodes in the ring.

Cassandra also stores redundant copies of data throughout the nodes present in the ring through built-in and customizable replication [35]. This helps in providing availability such that even if one node goes down, the other copies of the data of that node will be present on other machines in the ring. Configuration can be done to achieve replication across multiple cloud availability zones.

1.2.1 Cassandra Architecture

Figure 7 Cassandra Write Data Flow

The designing of Cassandra is done in such a way that it is able to handle big data workloads across multiple nodes without having to come across a single point of failure. The architecture is done understanding the fact that failures in system and hardware may and do occur [36]. This problem of failure is addressed by deploying across homogeneous nodes a peer-to-peer distributed system where data is distributed across the various nodes present in the cluster [33]. Every second the information is exchanged within nodes. Data durability is ensured by sequentially writing commit log on every node to capture the write activity. Then in an in-memory structure the data is indexed and written which is called a memtable that is similar to a write-back cache [37]. The data is

written to disk once the memory structure is full in an SSTable data file. The partition and replication across the cluster is done automatically to all writes. Cassandra uses a process known as compaction to discard obsolete data and periodically consolidate SSTables. Cassandra is a row-oriented database but from the perspective of CQL the database consists of tables. CQL can be accessed by the developer through cqlsh or drivers for application languages. A cluster typically consists of one keyspace per application. Developers can access CQL through cqlsh as well as via drivers for application languages.

Read and write requests can be sent by the client to any node in the ring. A node acts as a coordinator for a client operation when a client connects to that node to make a request. This coordinator node behaves like a proxy between the nodes that have the data and the client application and it also decides based on the cluster configuration as to which node will get the request in the cluster.

1.2.2 Key structures

a. **Node**: It is the basic infrastructure component where the actual data is stored in Cassandra.
b. **Data center**: A data center can be physical or virtual collection of related nodes. Depending on the workloads the data center should be used that may be physical or virtual. Data center is set by replication. Replication factor decides to how many multiple data centers the replication will be done [33]. The use of different data centers allows Cassandra transaction to not be affected by the other workloads and keep requests closer with lower latency. Data centers should however never spam physical locations.
c. **Cluster:** A cluster can span physical locations and contains one or more data centers.
d. **Commit log:** For durability the data is first written on the commit log. The data can be archived, recycled or deleted only it has been written down to SSTables.
e. **Table:** A table is a set of ordered columns fetched by a row. A row comprises of columns and has a primary key whose first part is a column name.

f. **SSTable:** Cassandra writes memtables periodically to SSTable (Sorted String Table). It is an immutable data file which is append only and stores sequentially on disk and is maintained for all the Cassandra tables.

1.2.3 Data distribution and replication

Data distribution and replication is an important feature of Cassandra. Data is stored in tables is identifies using a primary key which helps in determining the node at which the data is stored. Copies of these rows are called replicas.

Factors which influence replication:-

a. **Virtual nodes:** Virtual nodes are also referred to as Vnodes and it simplifies many tasks such as assignment data ownership to physical machines. Calculation and assignment of tokens to each node is done by Vnodes. While adding or removing a node rebalancing a cluster is not required due to Vnodes. A node takes up the responsibility of a portion of data from other nodes when it joins the cluster [34]. In case of failure, the load is distributed evenly to other nodes. Therefore, rebuilding a dead node is easier as it involves the other nodes in the cluster. Vnodes also improves the utilization of heterogeneous machines. A proportional number of Vnodes are assigned to larger and smaller machines.

b. **Partitioner:** A partitioner determines the distribution of data across the nodes including the replicas in a cluster and partitions the data across the cluster. Partitioner is a function which derives token which represents a row by its partition key. The data of row is then distributed across the nodes by the token value. The following partitioners are offered by Cassandra:

   i. **Murmur3Partitioner**: It is the default partitioner which based on MurmurHash hash values, uniformly distributes data throughout the cluster.

   ii. **RandomPartitioner**: Based on MD5 hash values it uniformly distributes data across the cluster.

   iii. **ByteOrderedPartitioner**: It lexically keeps an ordered distribution of data by their key bytes.

The Murmur3Partitioner and RandomPartitioner both uses tokens which helps them in assignment of equal data portions to nodes and distribute data from all the nodes across the cluster and other groupings like keyspace. Even if different partition keys are used by the tables like usernames or timestamps, then also the partition is successfully done. Load balancing is simplified as the hash range gets on an average equal number of rows. Read/write requests are also evenly distributed over the cluster.

c. **Replication strategy:** It helps in determination of replicas for each data row. Multiple nodes contain replicas to ensure reliability and fault tolerance. The place where the replicas will be stored is decided by the replication strategy. The number of replicas across the cluster is called replication factor. If a node contains only one copy of each row on a node its replication factor will be 1. When a replication factor is 2, two copies are present on each on a different node of each row [34]. There is no master replica or primary replica; all the replicas hold equal importance. It is preferred that the replication factor is under the number of nodes present in the cluster. But, replication factor can be increase and new nodes can be subsequently added. There are two replication strategies which are followed:
    i. Simple Strategy: This strategy is used for only single data center. The partitioner protocol picks the first node for first replica and places the second replica on the very next node.
    ii. Network Topology Strategy: It is widely used as single data centers can be expanded to multiple data centers and hence network topology strategy can be deployed. It lets one decide the replicas per data center.
d. **Snitch:** The topological information to place replicas is determined by snitch. These replicas are determined by the replication strategy and the data centers and racks nodes on which these replicas are belonged to are determined by snitch. Information about the network topology is provided to the Cassandra by snitches so that routing of request is efficient and replica distribution by grouping machines to data centers and racks is smooth [37]. Information provided by snitch also allows the replication strategy to place the replicas. Cassandra tries to ensure that only one replica is on the same track.

1.2.4 Security

Cassandra supports built-in security features such as password authentication and authorization. System authentication tables contain user credentials and privileges internally [37]. Other security features are listed below:

- Client-to-node encryption: - Cassandra provides an option to make the communication from client machine to the database secure. Data which is transferred back and forth from client machines is ensured to be secured and not compromised by the client to server SSL.
- Authentication based login: -CREATE USER command can be used by administrators to authenticate new users to Cassandra database. Cassandra itself manages internally all the user accounts and access rights to the cluster using passwords [35]. Changes to the user accounts can be done using CQL.
- Object permission management: After the internal authentication, permission management is done by GRANT/REVOKE security paradigm to determine the authorization capabilities or object permissions of the user. It basically determines what the user does once it is inside the database.
- Enabling JMX authentication: JMX (Java Management Extensions) by default is accessible only through localhost in Cassandra. To enable remote JMX connections the settings of LOCAL_JMX in Cassandra-env.sh is changed to enable authentication or the changes are done to the SSL. Also, the tools which utilize JMX like nodetool and DataStax OpsCenter should be configured as per the authentication.

1.2.5 Internal Database Storage

Managing data

Storage structure same as a Log-Structured Merge Tree is used by Cassandra, unlike a B-Tree which is used by typical relational database [37]. In a large distributed system, read before write can produce stalls in the read performance helps Cassandra avoids reads before writes. Increase in input/output requirement along with corrupt cache can also be caused by reading before writing. Suppose there are two clients reading at the same time,

one makes and update X and the other overwrites and make and update Y, removing X. Storage engine thus sequentially writes updated parts in append mode to a row so as to avoid a read-before-write condition. Cassandra doesn't re-writes/reads data which is already present, and doesn't overwrite the rows in place.

A log-structured engine uses sequential IO, avoids overwrites and updates data is important for writing data to HDD (hard disk) and SSD (solid-state disks). Writing randomly than sequentially on HDD involves high seeks operations. This seek penalty can be substantial. Write amplification and disk failure is avoided by sequential IO; accommodating inexpensive consumer SSDs. Table storage provided by Cassandra on disk is finely controlled

Cassandra storage basics

The key concepts of how data is stored in Cassandra are based on the hinted handoff feature and how Cassandra follows ACID properties. Hinted handoff is a feature which ensures high write availability when consistency is not that important. It is important to know how Cassandra stores data in order to manage and access it efficiently. Consistency in Cassandra is concerned with how up-to-date and synchronized data row is with its replicas [37]. Data is processed at several stages of the write path. Client utilities and APIs are available for developing applications for data storage.

Transactions and concurrency control

Cassandra does not succumb to RDBMS ACID properties and doesn't deal with transactional rollback or locking mechanisms but offers eventual consistency along with atomic, isolated, and durable transactions. It allows the user to decide how consistent a transaction need to be is.

Concept of joins and foreign key is not supported by Cassandra as it is a non-relational database. Isolation and atomicity is supported by Cassandra at row-level but for high availability and better write performance sometimes transactional isolation and atomicity is compromised.

Data consistency

When a row of Cassandra is up-to-date and synchronized with all its replicas, it is said to be in a consistent state. In Cassandra tunable consistency is offered in which for any read/write operation, the client decides the how much consistency is needed. The concept of eventual consistency is hence extended by Cassandra.

1.2.6 Backup and Restoring

Snapshots of on-disk data files are taken by Cassandra in order to back up. These files are basically SSTable files stored in the data directory. Snapshots can be taken of individual keyspaces or single table when the system is online. Command like nodetool snapshot takes snapshots per node. Snapshot of the entire cluster can be taken by pssh (parallel ssh tool) which provides consistent backup eventually [37]. At the time when the snapshot is taken, no node guarantees consistency but a restored snapshot soon gets consistent using the built-in consistency mechanisms of Cassandra.

When snapshots are taken of the entire system, incremental backups are done on each node to backup data that has been added or changed from the last snapshot. A hard link is copied into a backup every time a SSTable is flushed. This backup is done in the subdirectory of the data directory which is provided when the JNA (Java Native Access) is enabled.

Chapter 2: Literature Review

The rise of big data along with increasing demand of data-intensive computing has led to the rise NoSQL databases such as Cassandra and HBase. The review of the various methods suggested by researchers to handle these increasing demands and to enhance the performance of such databases is done along with meeting the various challenges and barriers.

2.1 Evolution of Big Data

M. Mridulm *et al*. [2] performed analysis of big data and said that many sources such as business processes, social networking sites, tractions produced big data which is both structured and unstructured. Business application being developed is majorly web oriented data intensive and large scale. These applications can be accessed through various devices including mobiles. Big data sizes vary greatly from dozens of terabyte to many petabytes all in a single set of data. It is difficult to capture, manage and process such huge data.

K. K. Reddi and D. Indira [3] defined big data as a combination of a lot of data sets such as structured, semi - structured, unstructured homogeneous and heterogeneous data. They presented a Nice Model in which transfers are committed to very less demand periods wherever there is a lot of idle bandwidth available under this model. This bandwidth can then be restructured for bigger amount of data transmission without actually affecting the other users inside the system. A store and a forward approach are used by the Nice model by utilizing the staging servers. The differences in the time zones and variations in the bandwidth are accommodated by the model. To solve problems like compression, security and routing algorithms, they have made suggestions for newer algorithms.

A. Bifet *et al.* [4] proposed that to obtain useful knowledge about the system and allowing organizations to respond quickly when problems arise or detect so that to increase the levels of performance, streaming data analysis in real time is becoming the quickest and the most efficient way these days. Large chunks of data are built every single day called as “big data”. Tools such as apache Hadoop, apache pig, cascading,

scribe, storm, apache HBase, apache mahout, etc. are used for mining of big data. Thus, he said that our capability to handle many Exabyte of data is primarily dependent on the presence of rich amount of datasets, techniques and software frameworks.

B. Purcell *et al.* [5] stated that Big Data consists of large chunks of data sets that can't be handled by the traditional database systems. The technique used by them for storage of big data which includes structured, semi - structured and unstructured data is object based and multiple clustered NAS (Network Attached Storage). Hadoop allows processing of this big data through MapReduce which helps in locating and selection of data which satisfies the query. They also discussed the various challenges the big data has posed to current businesses.

2.2 Inefficiency of traditional databases

S. V. Phaneendra and E. M. Reddy [6] illustrated how RDBMS tools are facing difficulties in handling huge data or big data. They elaborated how big data differs from regular data in terms of the five dimensions which are volume, variety, velocity, value and complexity along with describing the Hadoop architecture which consists of NameNode, DataNode and EdgeNode. The architecture makes Hadoop efficient to handle big data and log management of this by scalable algorithm which is used in various industries such as health-care, insurance etc. Also, the challenges faced by handling big data like search analysis and data privacy has also been discussed by them.

S. Agarwal *et al.* [7] presented BlinkDB, a parallel query engine which runs interactive SQL queries on large sets of data. BlinkDB was developed around two ideas :(1) a framework which is adaptive and which works on stratified samples from original data, which are multi-dimensional in nature. It maintains and builds these samples which have been collected over time. (2) A sample selection strategy which is dynamic in nature and helps in selecting the proper sized sample according to the query response time and accuracy requirements.

2.3 Hadoop and MapReduce

The increasing demand of data-intensive computing has led to the rise distributed computing clusters. To make such clusters inexpensive and highly utilized, software frameworks and application programming model is needed. For easy writing of scalable parallel applications which can handle and exploit huge clusters for data-intensive computation, MapReduce programming model is being implemented [8]. MapReduce is designed in such a way that it provides scalability and allows every node to handle its respective slice of dataset while loosely coordinating with other nodes. The programming model helps in improving the performance of application as the increase in number of computer nodes increases the parallelism which can be exploited. This programming model of MapReduce is implemented by Hadoop [9].

Hadoop was chosen as one of the framework for this research for various reasons. Firstly, its popularity and wide use across Internet service companies like Amazon and Yahoo. Secondly, due to its large scale deployment like the one at Yahoo with 4000 nodes in a cluster [10]. Third, since it is designed and deployed on commodity hardware it lowers the cost to build research clusters. A good number of networked commodity hardware is needed for data-intensive computing than specialized supercomputers. Fourthly, being an open source framework Hadoop is easier to obtain, profile, and modify. Lastly, the basic design of Hadoop is similar to other distributed frameworks [11, 12].

J. Lin *et al.* [13] suggested that even though Hadoop is widely used in large scale data analysis but there are certain classes of algorithm which are not entirely suitable to MapReduce programming model. He described this with the analogy that Hadoop is like a “hammer” but all the algorithms are not like “nails”. He focuses on the face that alternative non-iterative algorithms can be found to provide simple solution to the problems. He has described the standard MapReduce and that every iteration of pagerank signifies to a MapReduce job. Also, the use of iterative graph, gradient descent can be done to check for convergences and set up iterations.

K. Lee *et al.* [14] carried a MapReduce survey which was intended to assist the database and open source organizations in a better understanding of the various technical aspects

of the MapReduce framework. In the survey, features of MapReduce framework have been characterized and discussed along with its inherent advantages and disadvantages. Introduction of its optimization strategies reported in the recent literature has been done. The open problems and the challenges being faced have also been mentioned by the author of parallel data analysis with MapReduce.

C. Jermaine *et al*. [15] proposed a system model for OLA (Online Aggregation) over MapReduce in a distributed environment of large scale. They implemented their model in Hyracks which is parallel data platform which runs jobs that are data-intensive in nature over a cluster of shared-nothing machines. They used a Bayesian framework for estimating and producing confidence bounds of the model. The model produces usable and accurate estimates efficiently. Since MapReduce jobs majorly run on cloud, OLA can help a user to monitor accuracy online and save computation when the desired accuracy is achieved.

T. Condie *et al*. [16] pipelined intermediate data between operators, and proposed a modified MapReduce architecture while preserving fault tolerance and programming interfaces. A pipelining version of Hadoop was developed by them, known as Hadoop Online Prototype (HOP) to validate the design. Continuous queries are supported by HOP helping the MapReduce programs to be written for event monitoring and stream processing. Output of MapReduce jobs copied to disk before being used, this process simplifies fault tolerance. Early returns from job can be achieved while it's computed as modified version supports online aggregation.

The Hadoop ecosystem consists of various other components which provides support in specific area where it's not suitable to use MapReduce paradigm and may face performance issues. The most common ones are HBase Hive, Pig and Zookeeper [17-18].

- **Pig** gives a high-level platform to create MapReduce programs. It is a high level data flow language with the infrastructure to analyze high -level programs which help in accessing large data sets. It's got two layers, one is the language layer which comprises of a textual and easy to use language called Pig Latin. The

second layer is the infrastructure layer having a compiler for MapReduce program compilation.

- **Hive**: It a system that analyzes data and ad-hoc queries along with performing data summarization. It is a data warehouse which uses a HiveQL to make data queries which is on the lines of SQL.
- **HBase**: It is Hadoop Database which is column oriented and performs pointed queries as well as batch computation. It runs over HDFS and uses it for storage.
- **ZooKeeper**: It is responsible for maintaining configuration information, distributed synchronization and naming. It is a centralized service and is required for functioning of HBase.

Pig, Hive, HBase and ZooKeeper have been widely used around several research projects around cloud computing, indexing, DBMS and scheduling [19].

Y. He *et al*. proposed RCFile [20] which combines the features of horizontal with vertical partitioning and avoids unnecessary decomposition of data thereby avoiding wastage in terms of processing. Initially the data is partitioned horizontally and these partitions are further partitioned vertically. The advantage of this approach is that it avoids network access as columns which are a part of same rows are located on the same node together. Also, it provides access privilege to only those columns which are in the query as compression can be done within each vertical partition. RCFile also employs PAX which is very much similar to Cheetah, however the primary difference is that there is no use of block-level compression. Facebook proposed the RCFile and nowadays beings used extensively in popular systems such as Pig and Hive.

Data layouts of Trojan [21] most of the time follow those of the spirit of PAX; however, data layout inside a block can have any type of data layout. There are different Trojan Layouts that are created for each replica for exploiting the replicas made by Hadoop for fault-tolerance, thus enabling us to use the most appropriate layout which is dependent on the query at hand. It has been proved that queries that involve almost all the attributes uses less with row level layout of the database system whereas selective queries should be using column layout. PAX-based layouts in Hadoop performance are not good as compared to Trojan data layouts.

HDFS (Hadoop Distributed File System) [22] is majorly used on client-server architecture to analyze large data sets through indexed searching. While the client in HDFS contain web applications, the server on the other hand contain a master-slave system. The data is initially stored on NAS (Network Access Storage) and after that analyzed by ETL (Extract, Transform and Load). The data is loaded into the HBase table after being extracted from NAS and transformed for operational activities. Amount of data for analysis is reduced by data filtering and a search through index pattern. HBase composition helps in the data sharing rate along with ease in analysis.

2.4 HBase

V.K. Konishetty *et al.* [23] proposed implementation of operations in a scalable way over HBase. These operations were done on set data structure and mainly included union, intersection and difference. They discussed the limitations and methods to optimize these three operations over the Hadoop ecosystem. C. Zhang and H. Sterch [24] proposed a new Hadoop component CloudBATCH which functions like a regular batch job queueing system but with additional feature like cluster resource management. This management of clusters allows the discovery of hybrid needs of legacy applications and MapReduce for computing. It runs on HBase and includes tables which stores resource management information which can be accessed across all the nodes to manage metadata for resources and jobs.

M.F. Husain [25] suggested to process RDF (Resource Description Framework) data with the help of MapReduce directly on Hadoop although arguably for semi structured RDF data HBase is better with MapReduce. Update operations of RDF application may face limitations as Hadoop cannot be modifying randomly. HBase uses row keys to access data. Performance issues can appear for certain kind of queries if extra index structures are not added. Weiss et al. [26] proposed a structure which includes six indexes (PSO, POS, SPO, SOP, OPS and OSP) for storing RDF triples. All possible combinations of RDF triples are covered by these indexes. The changes are done to the HBase tables to build the structure rather than building these indexes.

X. Gao *et al.* [27] proposed a Lucene index solution which provides an interactive real-time search for all the text data which is stored in HBase. HBase is built on top of HDFS with distributed architecture. It can access huge amount of data in a reliable and efficient manner but doesn't provide a mechanism to search field values and text efficiently. Better scalability and high flexibility for the indexing system can be achieved along with high performance by hosting Lucene indices with HBase tables.

Rows are automatically deleted in HBase when they reach the expiration time. For every column family the minimum and maximum number of versions of row can be set and during compactions all the excessive versions can be removed [28]. It is recommended that if old values don't hold much importance, the maximum version number should not be set high as it greatly increases the size of stored data. This is useful if concurrency control is supported by version identifier. This recommendation is relevant when the version identifier is used to support concurrency control. Since distribution of data in HBase is according to row-key, for a range of row-keys only specific HBase region server is responsible. Range queries therefore are handled exceptionally well as neighboring keys are mostly stored on the same server.

HBase helps in executing user-level code by providing library and runtime environment with HBase region servers. This framework is inspired by Google's BigTable [8]. The performance is improved dramatically by pushing the computation on the server to operate on the data directly. Also, communication overheads are reduced by transferring data from region servers to client [29]. HBase being a data-centric programming model [30] helps in improving the performance of the system by parallel query processing. Partitioning the data is necessary along with a well-designed schema to extract the benefits of this framework.

Hadoop++ [31] is a database system which uses User Defined Functions (UDF's) that provides the functionality of indexing for data stored in HDFS, i.e., without actually changing framework of Hadoop at all. The Trojan Indexes or also called as the indexing information is inserted into logical input splits and it also serves as a cover index for the chunk of data inside the split. Moreover, there is no imposition in overhead time in query processing as the index is created at the load time. There is another feature of Hadoop++

which supports data joins by co-allocating and co-partitioning data at load time. Intuitively, inside the network it has led to expensive data transfer/shuffling, this has enabled the data join to be processed at the map side, rather than at the reduce side. Hadoop++ in comparison to HadoopDB performs better.

HAIL (Hadoop Aggressive Indexing Library)[32] by taking advantage of the n replicas (typically n=3) which is the default value maintained by Hadoop for fault tolerance provides performance improvements for the long index creation times of Hadoop++, and also by building a totally different clustered index for each duplicate entry. At the time of query, the most suitable index to the query is selected, and the particular duplicate of the data is scanned during the map phase. As a result, HAIL leads to a lot of improvement substantially in the performance of MapReduce processing, since the probability of identifying an appropriate index for efficient data access is increased. HAIL has been shown in for the improvement of the index creation times and the performance of Hadoop++. HAIL and Hadoop++ both support joins with an improved efficiency.

2.5 Cassandra

Two engineers at Facebook, Lakshman and Malik, [33] designed Cassandra and open sourced it to Apache community. Cassandra is described by them as a distributed storage system which is highly available without a single point of failure and able to handle large amount of data across commodity servers. Cassandra was mainly designed to solve the inbox search problem of Facebook which lets user search their inbox recursively. The aim was to keep the time interval between request and response as low as possible along with distributing data across data centers with different location as well as within itself. Cassandra was inspired by Amazon Dynamo, which is used in shopping cart on Amazon. Even though it's quite similar but it's not as limiting like Amazon Dynamo.

Cassandra is based on a key-value data storage system, which means that for every set of data that exists there is a unique value of key connected with it. Whenever the data is retrieved or used for any purpose, the value of the key is used to identify the particular data set associated with it. One of the major downsides of the database design is that there may exists stored duplicates of data sets as the key is the only thing to identify the

connected data (example - like postal codes in the databases). The availability, flexibility and IO-speed compensate very well for the above design flaw. Also since these days since disk size has become cheaper, it has become less of an issue to store some duplicates. Nowadays the database systems never consume their disk space completely, as the disk space often is increased whenever required to an “infinite” number of Megabytes.

S. Fukuda *et al.* [34] proposed that to improve the average response time of the searching queries in the database systems; we can use scheduling search queries and parallel execution of the range queries together. Under the condition that both the range and single queries are mixed together, it was realized that the scheduling methods was getting appropriate priority to each query under the situation that both single and range queries are mixed. This was implemented into Cassandra and evaluated as well. As a result of this, with the comparison of the original Cassandra, a reduction was seen in the average response time of single queries. In addition to this, due to scheduling and Parallel execution of range queries, the average response time of range queries was also reduced.

H.M.L. Dharmasiri and M.D.J.S. Goonetillake [35] have successfully managed to produce and implement a federated NoSQL system with three different types of NoSQL solutions. By implementing a federated system between MongoDB, CouchDB and Cassandra, this task has been achieved. They proved by the various types of tests and evaluations it can be finally exclaimed that the NoSQL federation is a feasible solution and it will actually take off a lot of time and effort taken in large scale data migrations. Since these choice options have a very large impact on the final-end performance of NoSQL systems, careful selection must be done.

M. Chalkiadaki and K. Magoutis [36] posted a method for QoS-aware implementation of Cassandra groups on the basis of SLAs application. For evaluating and predicting server capacity requirements, the evaluation method is effective given that simple application workload descriptions. The process is simple due to the scalability and elasticity mechanisms built into NoSQL systems such as Cassandra; therefore they think that the work done by them is more applicable to such systems on a broader scale.

Chapter 3: Problem Statement: An Analytical Comparison

In the current scenario of the industry where data is increasing at an exponential rate and due to the Business Intelligence these days, importance of big data has increased to such an extent inside the market that we have to store and manage the data in a well-built manner so that we can use it for data analysis services and build large scale applications that are helping users throughout the world.

In today's world application are made with the specific needs of users, they need real time changes within click of hands. The industry is so rapidly is changing that the need for faster and effective databases has become the need of today's world.

With data becoming more and more complicated day by day, the need for handling it effectively is also increasing at a rapid pace. Data that is stored in systems are of different types and hence classified as unstructured data consisting of images, texts, audio, video etc.

Unstructured Data also known as Unstructured Information refers to data or information that does not have a pre-defined **model of data** or is not organized. **Unstructured** information is heavy text which contains data of other types as well.

3.1 Gap Analysis

Performance evaluation between Cassandra and HBase is needed in order to assess the differences in performance such that a business information systems or application is able determine whether the system requirements are being met correctly with the proper data set ("Big Data").

It is important to use an effective and quick system for the handling of data these days. There have been a lot of theoretical comparisons made between databases in order to evaluate a better database according to available data set but no practical implementation has been done.

3.2 Problem Statement

The rise of big data has led to the immergence of NoSQL databases such as HBase and Cassandra and has made the selection of appropriate database for an application difficult. An analytical and performance comparison using real query processing implementation helps in understanding the efficiencies of such databases. The comparison made on CRUD principle (Create, Read, Update and Delete) are basic to any type of data set and application system. Hence these queries provide us with real data and help make a decision in order to choose the perfect database for an application.

3.3 Big data and Small data

There are certain things that cannot be overlooked when dealing with data. Best practices must be instituted for the care of big data just as they have long been in small data. Before enjoying big data's amazing analytical features, you must first get it under control---with tools that are up to the challenge of implementing best practices in a big data world.

There are a various tools and practices that a database system should keep in check with, some of them are as follows.

- **Availability** :- Ensure that the platform is always on 24/7
- **Management**: - Monitor and administer all resources and jobs.
- **Disaster Recovery** :- Rapidly restore the platform to operational status when service is interrupted
- **Provisioning**:-Scale up and out rapidly and cost effectively.
- **Optimization**:-Tune the performance of all applications and processes to meet requirements.
- **Backup and Restore**:-Rapidly and reliably backup and restore data.
- **Security**:-Authentication, permission, encryption, time-stamping, and non-repudiation of access.

- **Governance**: - Define and enforce controls over creation development and usage of all data.
- **Auditing**: Audit and log all network, system, application, usage and other events.
- **Replication**: Replicate all the types of data bidirectionally, reliably and with high throughput.
- **Virtualization**: Administer, access and utilize all resource via a unified interface.
- **Archiving**: Archive any and all data for eDiscovery, query and analysis.

3.4 Selection criteria of Databases

When designing a new application, we must select the database according to the application's requirement. The different criteria which should be considered are:

- **Development criteria:** It includes the drivers which are available, query functionality, data consistency and data model. The factor affects the functionality of the application along with the time needed to build it.
- **Operational criteria**: This criterion includes performance, scalability, high availability, data center awareness, security, management and backups. During the lifetime of the application, the operational costs greatly affect the project's TCO (Total Cost of Ownership). Therefore, these cost enable to adhere to SLAs which decreasing administrative overheads.
- **Commercial criteria**: It includes pricing, licensing and support. The database which is chosen should be available and aligned with the business.

3.5 Migration from relational databases to NoSQL

A NoSQL, which is also interpreted as Not only SQL, allows the storage and retrieval of data by providing a mechanism which helps in modelling data in other forms than tabular relations as used in relational databases. NoSQL databases have various applications across the globe.

It doesn't utilize SQL for manipulation of data nor is it built on tables. Even though it doesn't provide ACID properties it still manages to deliver a fault tolerant and distributed architecture.

Currently, all major developing companies whose data is rapidly increasing are now switching from traditional databases to NoSQL databases such as Hadoop and Cassandra. Big Data is majorly generalized to data analytics technologies such as Hadoop. Hadoop on the other hand focuses majorly on data storage than analytics. Scaling data storage effectively has actually given rise to data analytics of Big Data. In the global database market, only 1/4th is analytics and rest is operational databases. Administrative work is a part of generally all the databases but Cassandra being a single server minimizes the operational overheads. Also, Cassandra is comparatively easy to understand due to its homogeneous node configuration. Hence we are comparing HBase and Cassandra to measure which has better performance in terms of CRUD operations. CRUD operations are Create, Read, Update, and Delete and are significant criteria for persistent storage of data.

3.5.1 Key features of NoSQL

- It is a scaled-out architecture, capable of running on a large number of nodes together.
- In order that real time data reading does not conflict with real time data write, a non-locking concurrency control mechanism is used.
- ·It's a very scalable replication and distribution system.
- In comparison to the traditional SQL- based databases, the NoSQL architecture provides much higher per-node performance.

NoSQL data models are mainly grouped into four categories even though they may incorporate more than one model at once.

- **Key-Value Databases:** Key Value store is the simplest data model. Basically it is a distributed persistent associative array where the key is a unique identifier for a value, which can be any data application needs to store. This model is also the

fastest way to get data by known key, but without the flexibility of more advanced querying. It may be used for data sharing between application instances like distributed cache or to store user session data. For example, Amazon Simple DB and Redis.

- **Document Data Store Databases:** For storing the semi structured Document store is a data model for storing semi-structured document object data and metadata. The JSON format is normally used to represent such objects. For example, CouchDB and MongoDB.
- **Columnar NoSQL Databases:** The data model of these is a column family. These are used for organizing data based on individual columns where actual data is used as a key to refer to whole data collections. It is similar to a relational database index; however a column family may be an arbitrary collection of columns. There are more complex aggregation structures like super columns and super column families to allow access to the data by several keys. This particular approach is used for very large scalable databases to greatly reduce time for searching data. It is rarely used outside of enterprise level applications. For example, HBase and Cassandra.
- **Graph Databases**: The data model for Graph Databases allows the objects to form a link and get linked by a lot of other objects hence building a graph structure. There are additional properties related to the links in order to describe the relations between different objects. Graph databases are faster for high associative data set types and graph queries and map more directly to object oriented programming models. Also they typically support ACID transaction properties in the same way as most RDBMS. For example, InfoGrid and Infinite Graph

3.6 Analytical comparison of databases

Both Cassandra and HBase are NoSQL database which means that they cannot be manipulated with SQL. Cassandra implements CQL (Cassandra Query Language), which has a similar syntax to SQL.

In order to perform a proper comparison, the understanding of differences of the two databases is required. The understanding of differences further requires understanding the similarities between the two.

- They both manage large data and are designed according to that.
- Distributed storage is done in both of them. Data can be stored and accessed by any client connects to the node in the cluster.
- Safeguarding of data loss is done by both of them through replication.
- Similar write paths are implemented by both starting with logging write operations which ensures durability.
- Command-line shells are implement in JRuby and both uses Java greatly.

The Basic differences of HBase and Cassandra

Even though with the above parallel, there are a lot of differences between them.

- Even though both are symmetrical which means clients can access through any node, the symmetry provided is not complete. Cassandra requires seed nodes whereas HBase requires master nodes.
- For internode communication Cassandra utilizes Gossip protocol whereas HBase relies on Zookeeper for the same.
- Real transactions are not supported by them but some level of consistency control is provided by them.
- The primary index in both Cassandra and HBase is the row key but Cassandra lets you create secondary indexes which are not supported by HBase.
- As Cassandra has CQL, HBase has “coprocessors” which executed user code in context to HBase processes.

The real work is to decide a cluster for a particular application. The selection requires a solid understanding of size of data along with complexity of creating and handling a multi-node cluster. Thus, this thesis tries to answer the problem by analytical comparisons between the two such that appropriate choices can be made by the users when selecting a database. These choices will help in proper handling of the large set data present.

Chapter 4: Implementation Details

Focus of this thesis is on the Performance Comparison of Hadoop (HBase) and Cassandra database systems. For this the implementation has been done on Linux (Ubuntu 12.04) system. All the implementation work is done in Eclipse IDE using Java framework. For the database systems - Hadoop (HBase) and Cassandra, respective libraries have been used in order to test a certain set of data for a performance evaluation between the two databases. The evaluation of this data is done using graphs and tables for time taken on different set of operations. Finally it has been concluded that Cassandra takes less time than Hadoop HBase. The data set considered for the following was unstructured data (consisting of different type) taken for the reason as NoSQL deals better with such type of data.

The implementation starts with installation of different sets of libraries used in the project which are as follows:-

Hadoop: Version - 2.6.0
HBase: Version - 0.94.8
Cassandra: Apache - 1.2.16
Java: Version - 1.7

4.1 Hadoop

4.1.1 Installation Steps for Hadoop

a. First we need to Java and install it on our computer system.

b. Then we need to create a SSH certificates required by Hadoop for its proper functioning.

c. Further we downloaded hadoop-2.6.0 from the Hadoop website and installed on our system using terminal.

d. Before actually running Hadoop on our system we need to make some changes in the configuration files of the system directory and make directories according to the requirements of our use.

e. All of the Hadoop variables in the .bashrc file for our use.
f. After this we need to set the Java_Home in the Hadoop configuration file hadoop-env.sh.
g. Then we need to define the ports we are going to use in Hadoop core-site.xml file where the HDFS will connect.
h. Further we have to make directories for the data and name on our computer system where Hadoop will store the data we are going to use along with giving the path of the same in hdfs-site.xml.
i. After completing these steps of installation og Hadoop we can start Hadoop by running "start-dfs.sh" command in the sbin folder.
j. Now after successfully installing Hadoop we can see the NAME NODE and DATA NODE running on different ports by typing "jps" command.

4.2 HBase

4.2.1 HBase Installation

a. We have configured java and generated SSH certificates during Hadoop Installation so we don't need to repeat those steps again.
b. Now we need to download HBase and install it on our computer system.
c. Before running the HBase we have to change its properties in the configuration files and make some directories for data storage.
d. We have to define the variables such as Java_Home in HBase-env.sh so that they can be used by Java.
e. Further we need to define the HBase variables in the .bashrc file in our computer system.
f. We will need to update the HBase-site.xml along with setting important properties in the file.
g. After that we have to set HBase.rootdir property, HBase.zookeeper.quorum and set the path of data directory for the zookeeper data storage.
h. Further we need to set the rootdir property value to the port that we have set in Hadoop core-site.xml.

i. Now after all these steps we can start HBase, for this Hadoop must be in running state already in the back. To start HBase run the command “start-HBase.sh” in bin folder.
j. Now all the nodes such as HMaster and HRegionServer are started on different ports that we can verify by using the command “jps”.
k. After HBase is configured and installed successfully and we are ready to connect it from our Eclipse IDE to proceed with the implementation.

4.2.2 HBase Implementation

a. For further development we will need Eclipse IDE (For Java code compilation). Download and Install it from its website.
b. After successful installation of Eclipse IDE create a project for Hadoop to proceed with the implementation.
c. Import all the necessary jars that are required for the development.
d. Using the Java API we make the connection with HBase and create the tables and datasets necessary of around 1GB using multimedia data. All the data in HBase is stored as bytes.
e. Now certain set of operations to be implemented on the data set is begun.
f. We start with the process of data insertion into the database system and record the time taken to do that (in nanoseconds).
g. After this we process the database with data selection process on the same data set and recorded the results for it as well.
h. The third set of operation that is performed on this data set is deletion.
i. We repeat these three operations 10 times each and record the results in form of time (nanoseconds) to complete the analysis.

4.3 Cassandra

4.3.1 Cassandra Installation

a. We have already configured java and generated SSH certificates during Hadoop Installation so we don’t need those steps again.

b. Now for the analysis of the other Database (Cassandra) we follow the following steps.
c. First we download Cassandra and install it on our computer system.
d. Before starting Cassandra we have to update configuration files and create directories to store data in our system.
e. We have to define Cassandra variables in .bashrc file of our system.
f. Further we need to also define the variables such as Java_Home in Cassandra conf file.
g. Next we have to create directories to store logs and data of Cassandra.
h. After this we are ready to start Cassandra. To do so run the command "sudo sh Cassandra" inside the bin folder.
i. Cassandra is now running on our system. We can see its connection status by running command "sudo sh Cassandra-cli".
j. We can also check the Cassandra running port by typing command "jps".

4.3.2 Cassandra Implementation

a. We already have installed Eclipse IDE for Java code compilation, hence no need to repeat.
b. Now we create the java project for Cassandra to proceed with the implementation.
c. Further import all the necessary jars that are required for the process of development.
d. By Using the Java API we make the connection with the Cassandra database using Cluster and session.
e. Then after establishing a successful connection, we need to create the tables and built the dataset of around 1GB using multimedia data similar to what we used in HBase.
f. We use blob data type to store the multimedia data.
g. After this we perform the exact three sets of data operations (Insertion, Selection and Deletion) similar to HBase DB.
h. Record all the values for these operations after performing these for 10 times each and not them for Graphs and Tables.

The implementation screenshots have been attached further along with the mentioned steps for a better understanding.

4.4 Implementation Snapshots

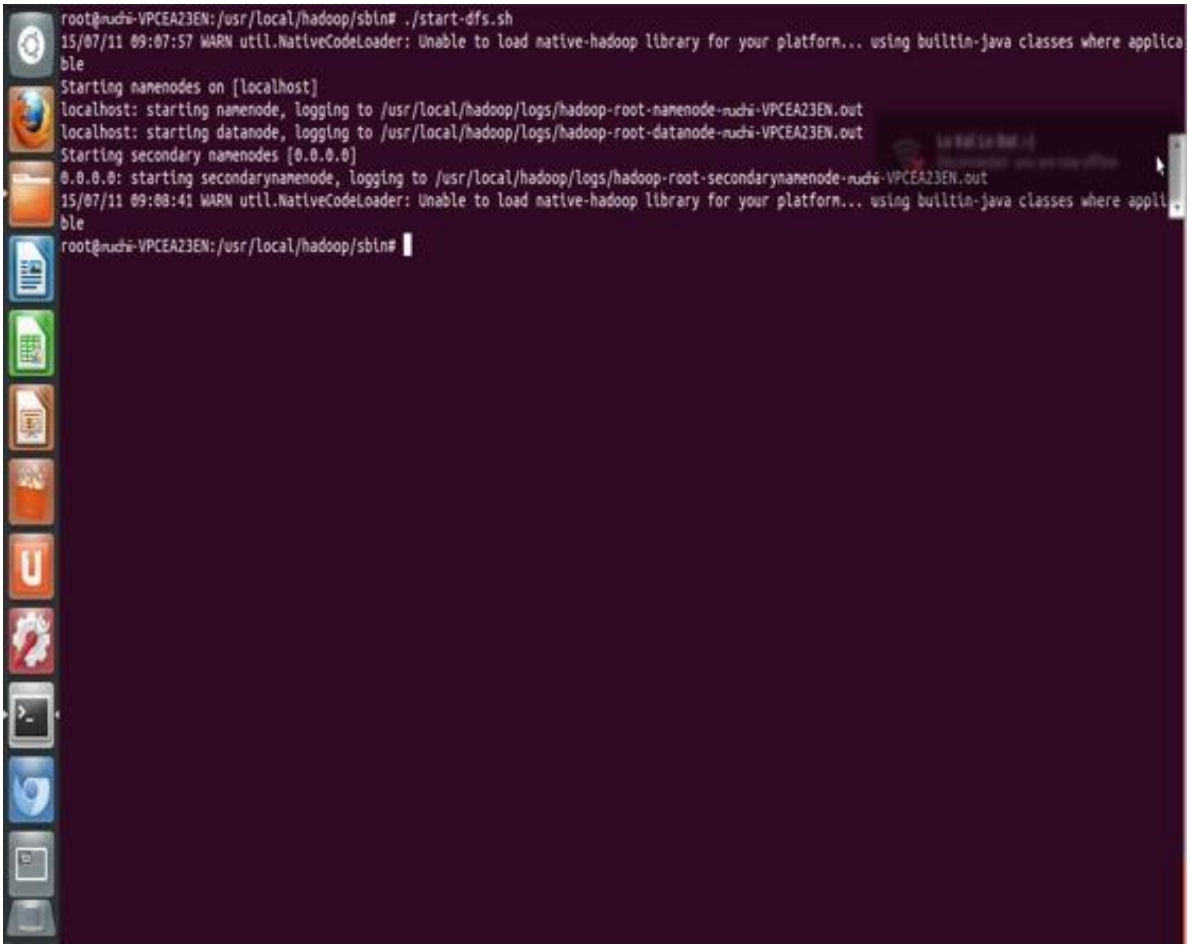

Figure 8 Starting Hadoop

As shown in this figure, Hadoop is started by typing command "**start-dfs.sh**" in the sbin folder of Hadoop. This will lead to creation of different nodes on our machine as seen above.

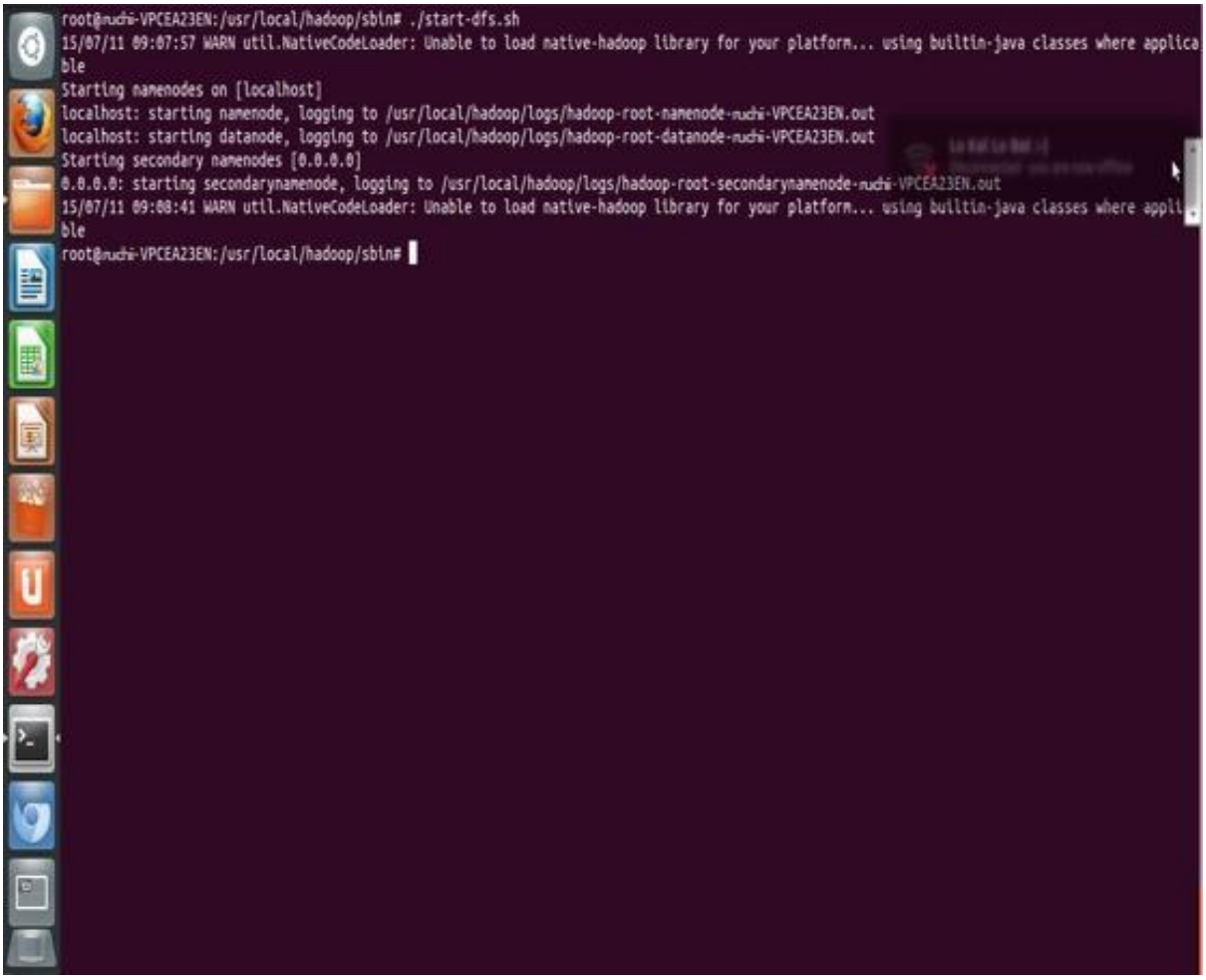

Figure 9 Running State of Hadoop

The above figure shows the running state of Hadoop. By typing the command “jps”, it shows all the different nodes that are running on our system. We can see the NAMENODE and the DATANODE etc. running on our machine.

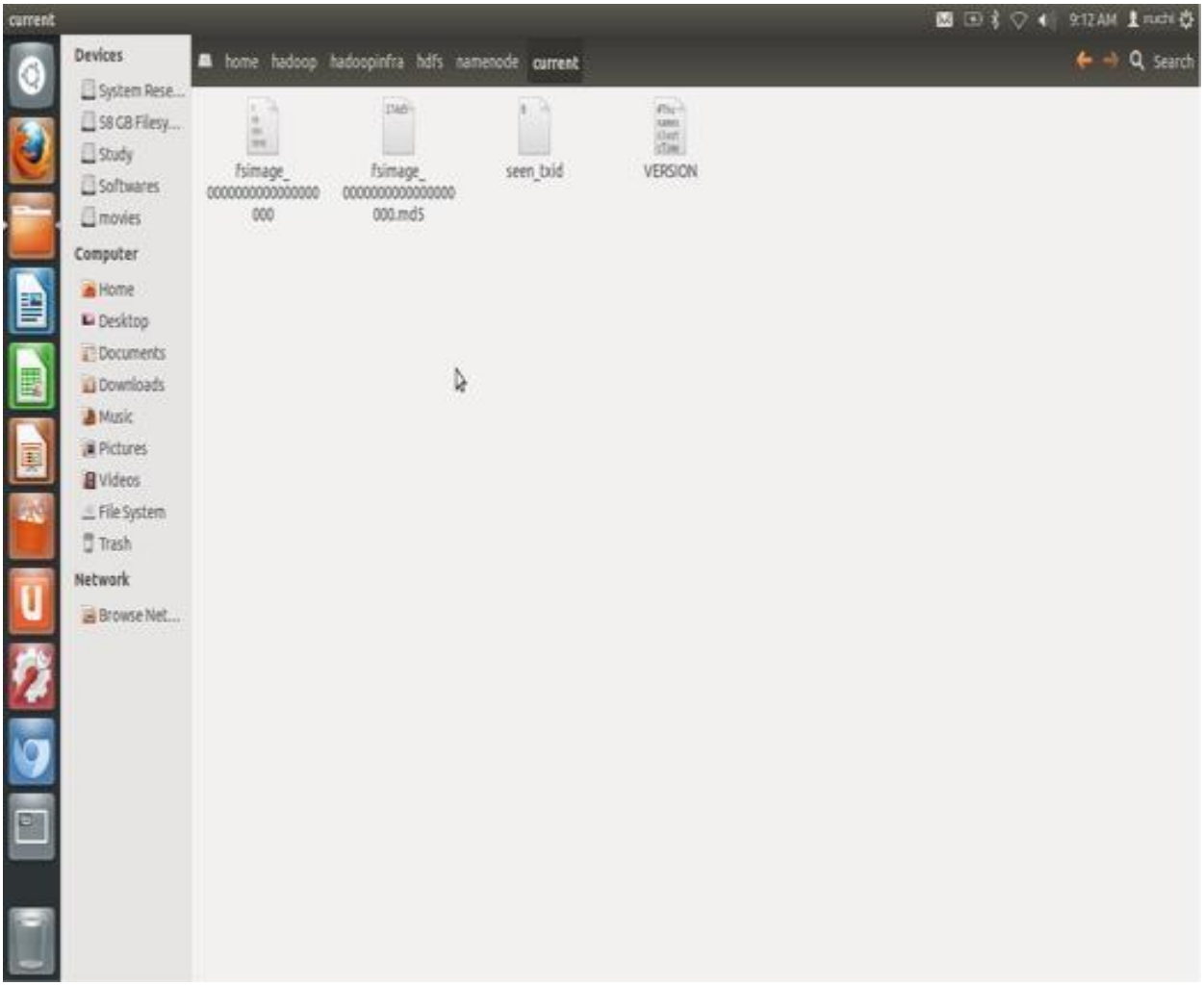

Figure 10 Hadoop's Storage Directory

As shown in the above figure, the Hadoop NAMENODE storage directory is show Hadoop stores all the information regarding name node in this directory.

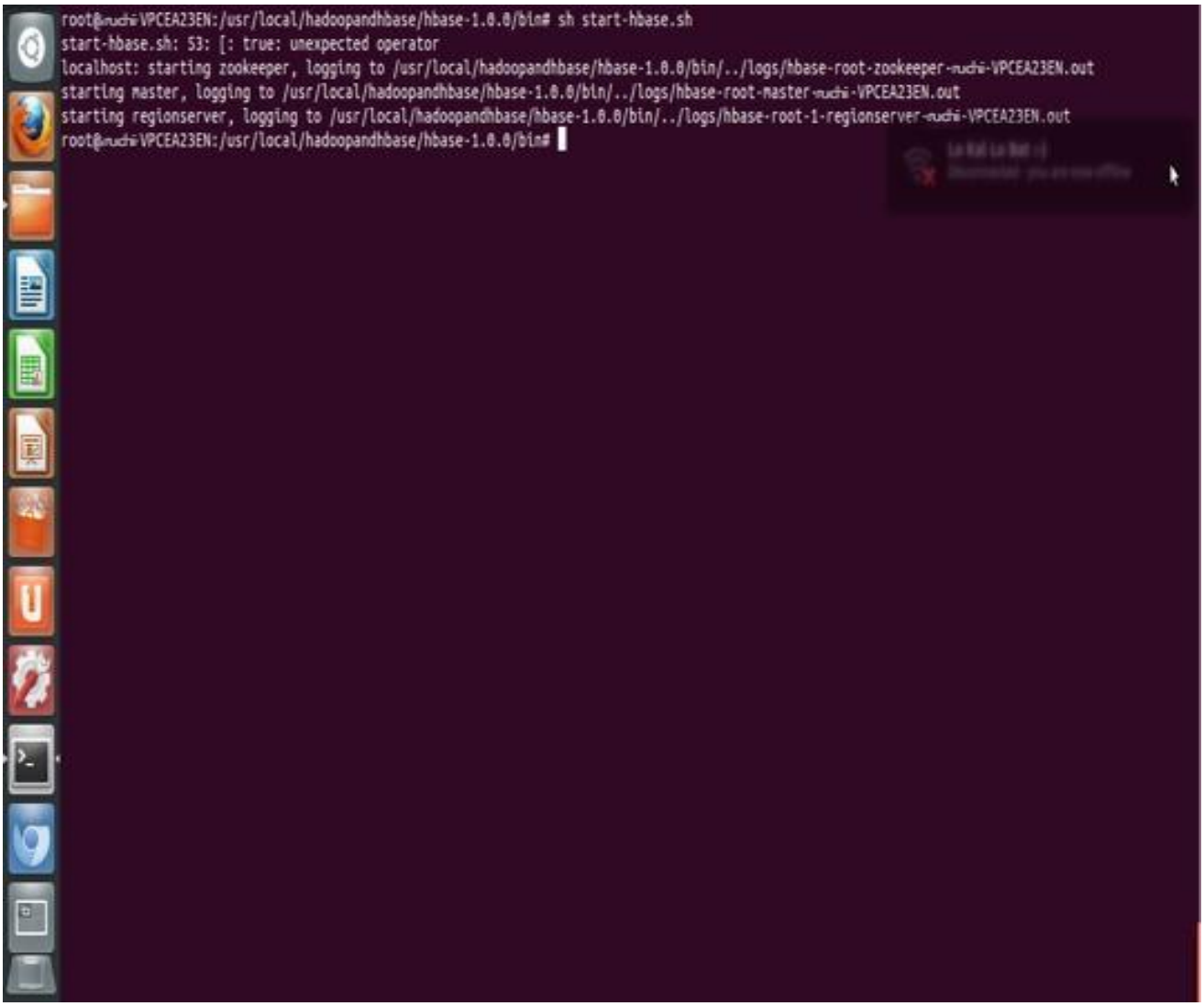

Figure 11 HBase starting state

As shown in the above figure, HBase is started by typing "start-HBase.sh" command. After this HBase will start Zookeeper, Hmaster and Region Server node.

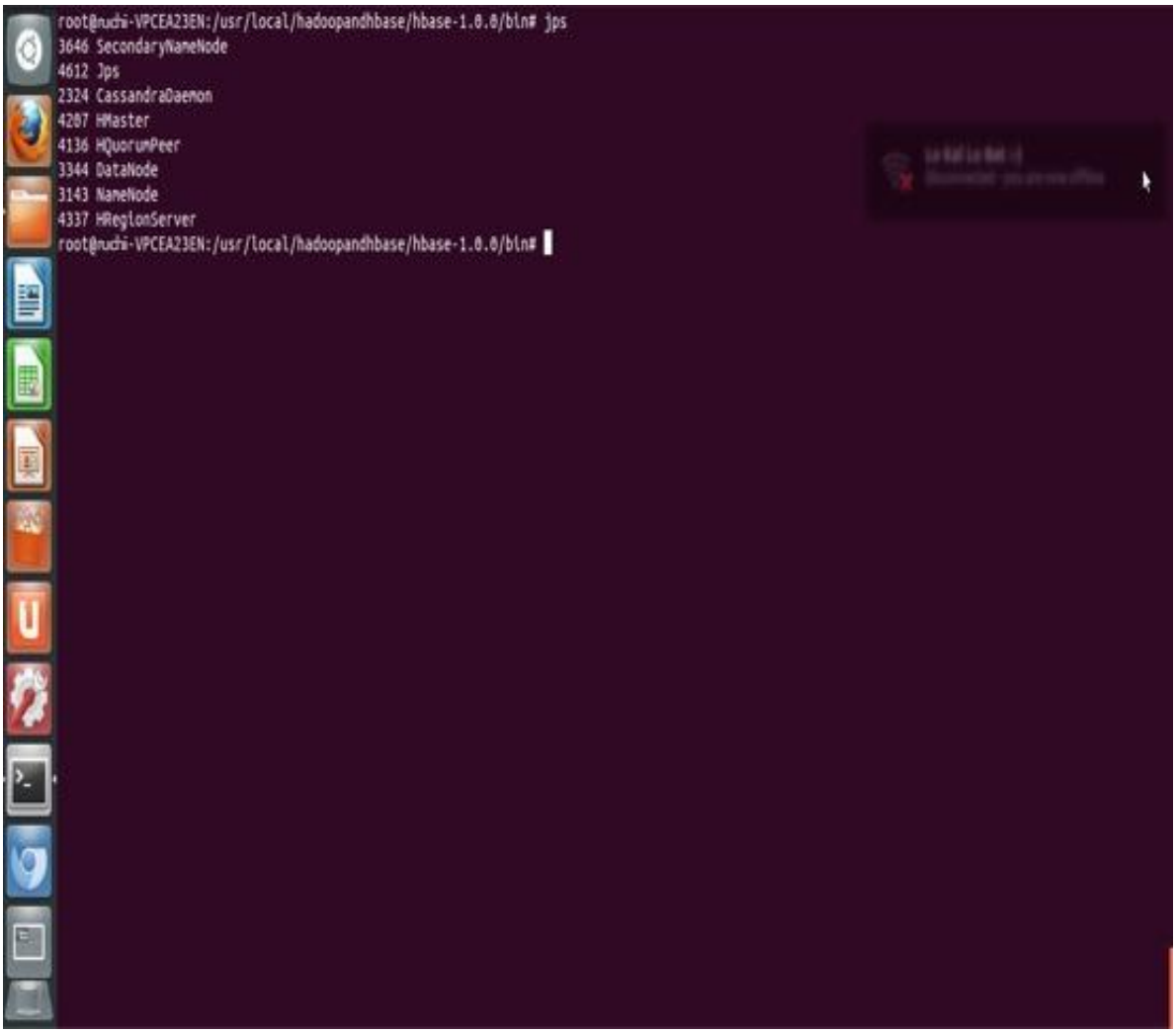

Figure 12 HBase running state

As shown in the above figure, HBase is running. We can see which node is started at which port by typing “jps” command.

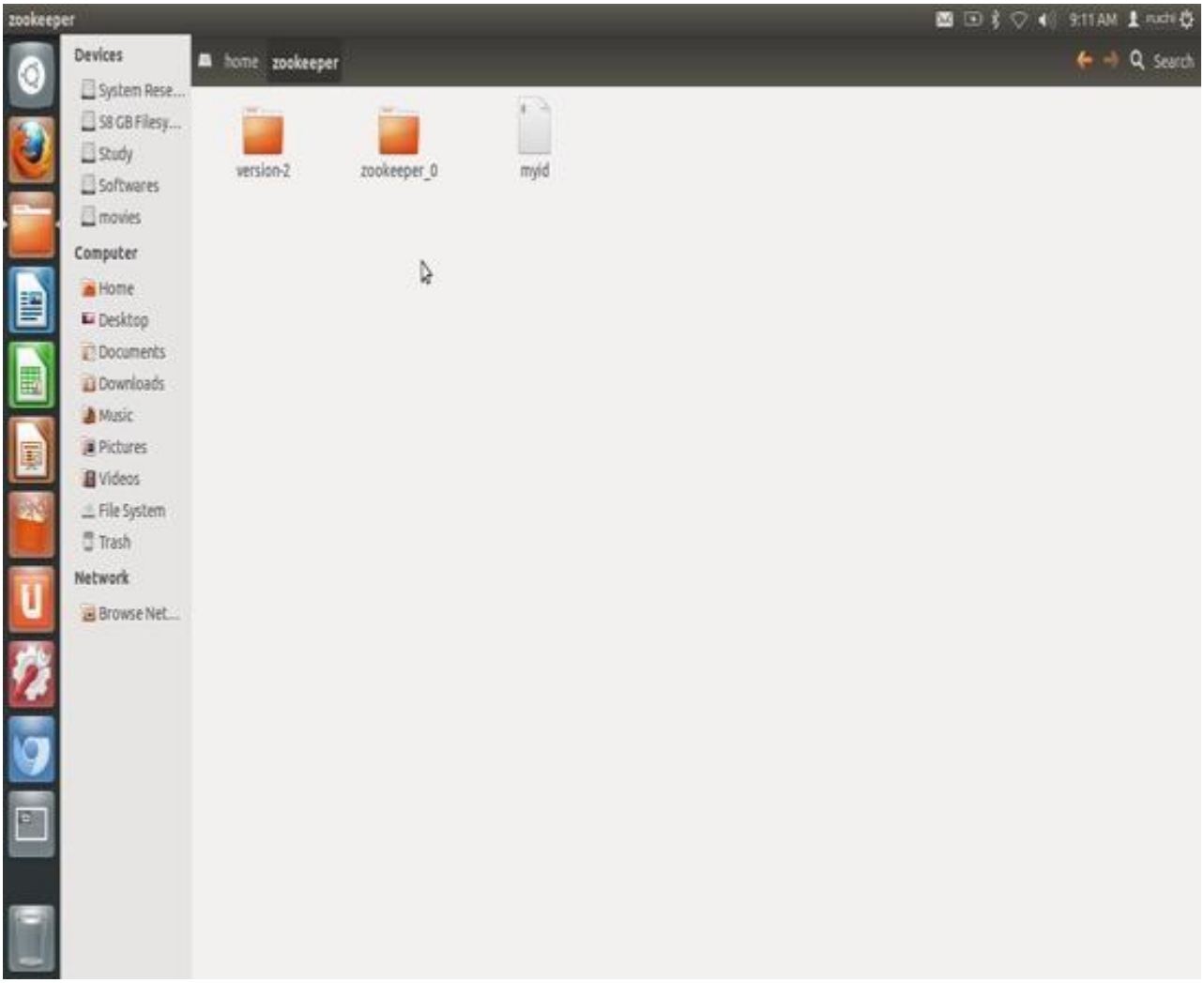

Figure 13 HBase Zookeeper Directory

As shown in the above figure, Zookeeper storage directory known. Here Zookeeper will store all the data related to the database.

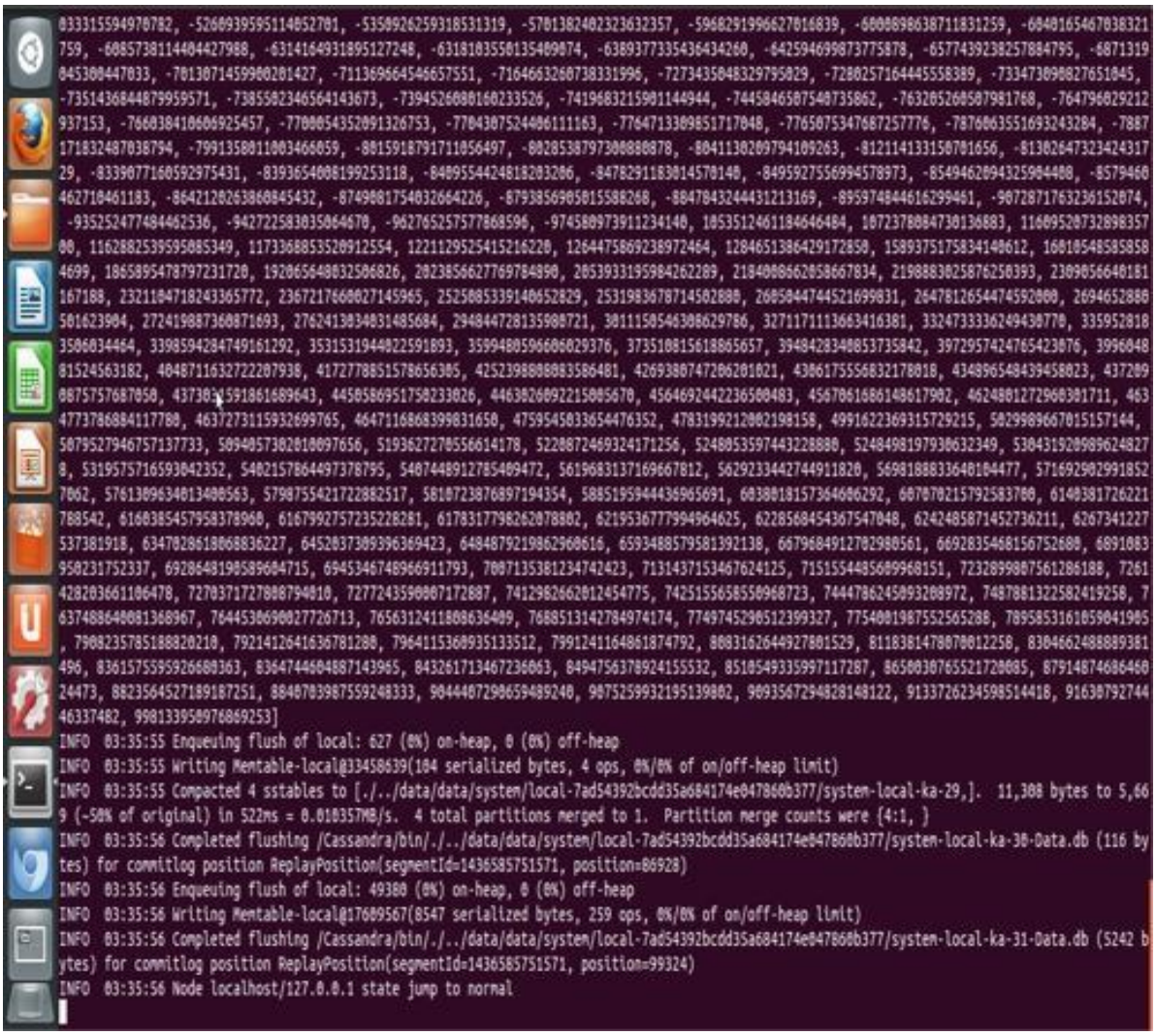

Figure 14 Cassandra's starting state

As shown in the above figure, Cassandra is started by typing “sh cassandra” command in the bin folder. It will run the Cassandra at a port on our system.

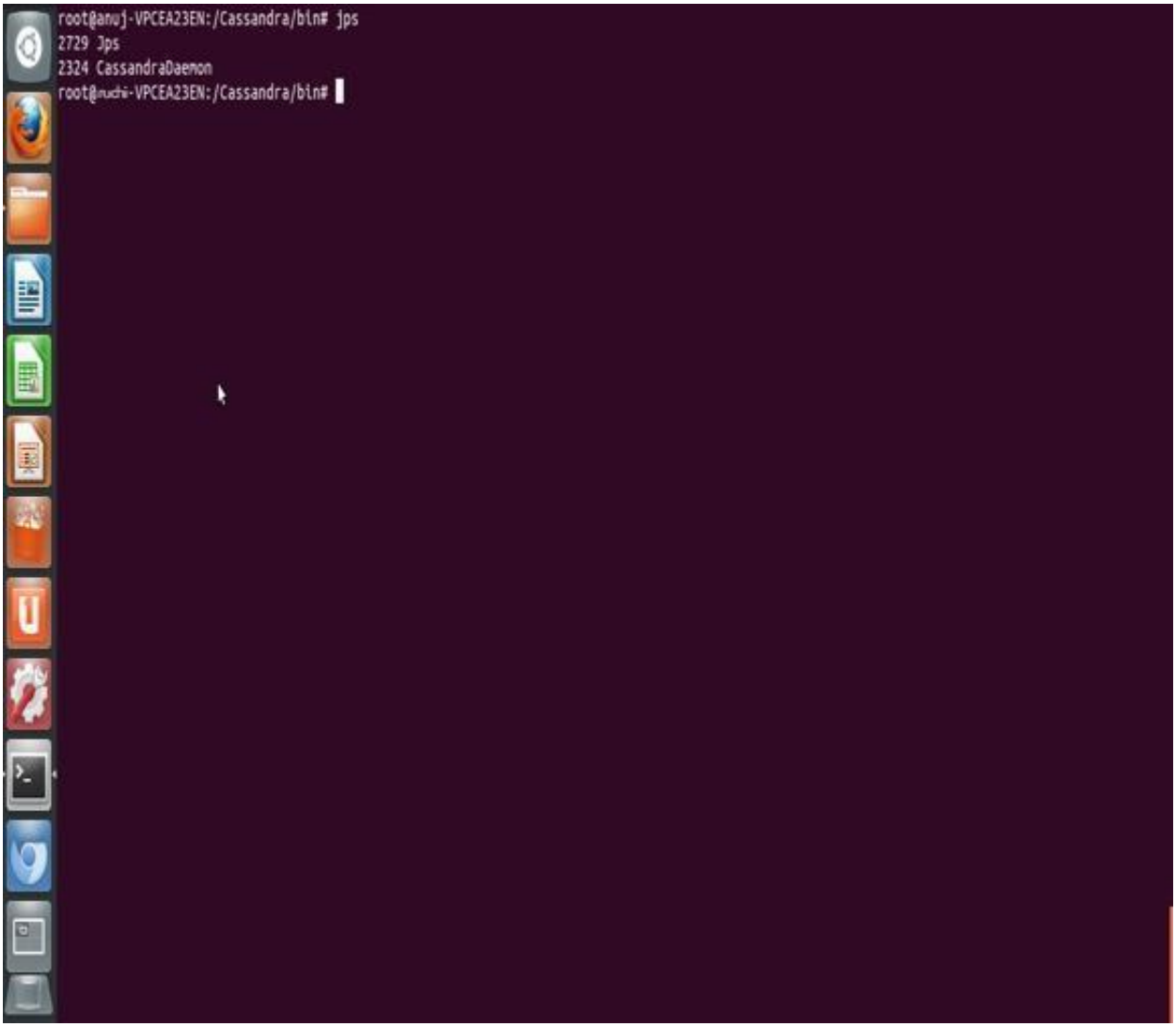

Figure 15 Cassandra running

As shown in the above figure, We can see that Cassandra is running on our machine. We can see this by typing “jps” command on the terminal window.

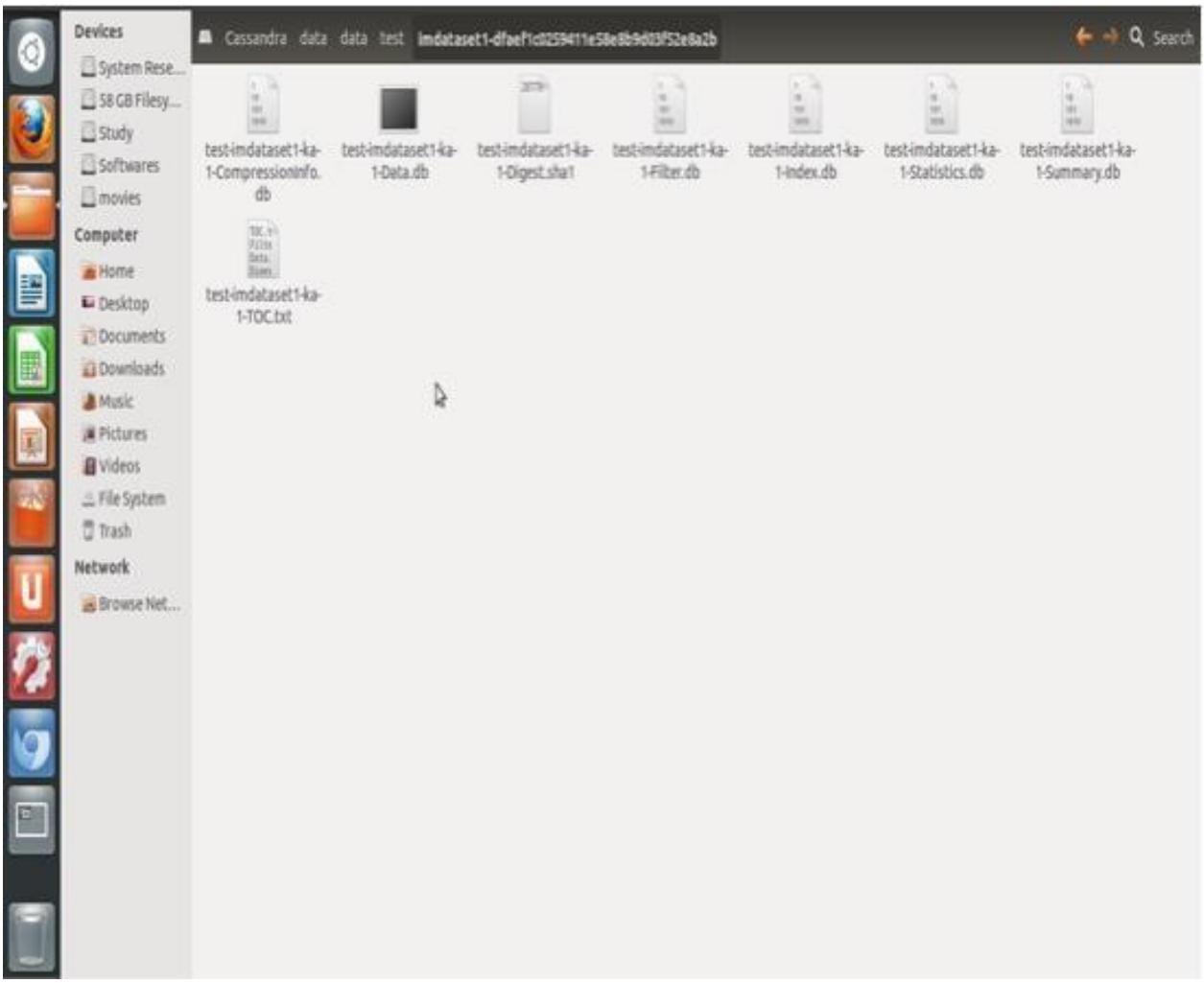

Figure 16 Cassandra's storage directory

As shown in the above figure, Cassandra stores all its data in this directory. Here Test is the name of cluster where all the data we created is stored.

Chapter 5: Results and Comparative Analysis

The implementation has been done in LINUX (Eclipse IDE). The code has been written in Java framework. The Cassandra and Hadoop java libraries have been used to make the connection and perform all the required operations for the comparison. As NoSQL deals better with unstructured data, we were motivated to make our comparisons taking data sets of image data. Insertion, Selection and Deletion operations have been performed and the results have been compared on the basis of time taken by both the databases to perform these operations.

5.1 Analytical Comparison of Hadoop HBase and Cassandra

Below are the graphs and tables as recorded by each of the databases (HBase and Cassandra) for performance evaluation.

Table 1 Data Insertion Time (in nanoseconds)

Cassandra	Hbase
72928547	167615419
72834803	168624108
72861542	171860373
75215421	168748957
74025157	172526637
76651151	169224633
79124527	171726469
75959754	166163288
75017956	167871024
77541278	166454863

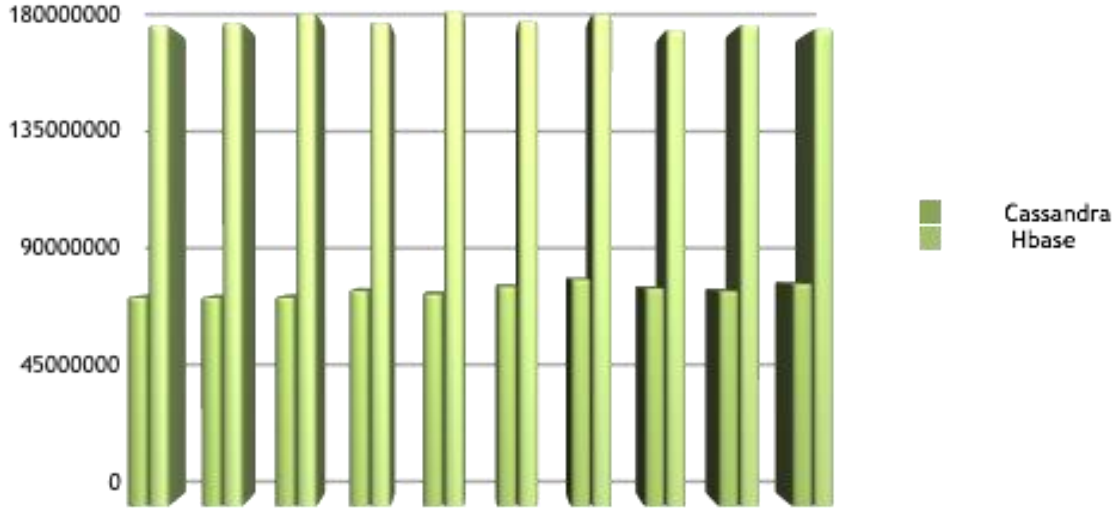

Figure 17 Graph representation of Insertion Time

In this graph **x-axis** is showing the no of iterations and **y-axis** is showing the time taken in nanoseconds. From this graph it is clear that the HBase has taken more time than Cassandra for inserting data into the table inside the database.

Table 2 Retrieval Time (in nanoseconds)

Cassandra	Hbase
3263251	7752065
2848516	7277829
3033208	6987547
3664085	7835999
2993733	7148921
2949523	7106403
2972214	6989845
3276895	7209290
3494110	7143010
2792688	7086218

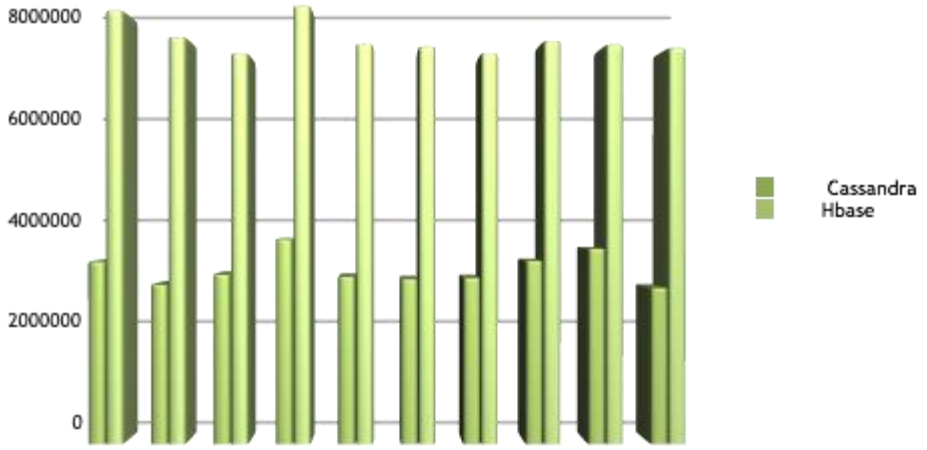

Figure 18 Graph representation of Retrieval Time

In this graph x-axis is showing the no of iterations and y-axis is showing the time taken in nanoseconds. From this graph it is clear that the Cassandra has taken less time to retrieve the data from the database tables.

Table 3 Deletion Time (in nanoseconds)

Cassandra	Hbase
2576733	6351899
2114901	5965707
2098626	6085375
1829405	6017782
2007903	6314057
2321771	6020404
2745514	6027365
2838331	6012989
2963564	5989135
2561478	5942727

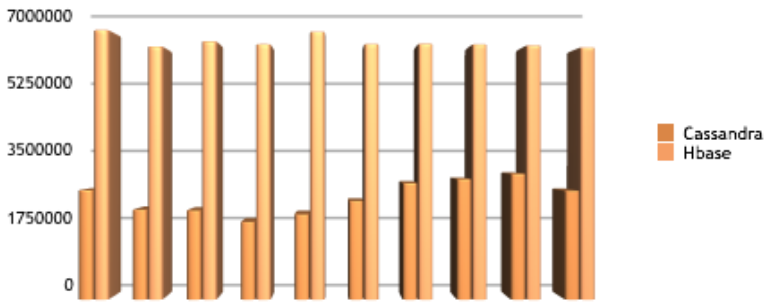

Figure 19 Graph representation of Deletion Time

In this graph x-axis is showing the no of iterations and y-axis is showing the time taken in nanoseconds. From this graph it is clear that the Cassandra has taken less time than Hadoop for deletion operation in the database system.

5.2 Comparative study

A comparative study of HBase and Cassandra is shown in Table 4.

Table 4 Comparative Study of HBase and Cassandra

Point	HBase	Cassandra
Foundations	HBase is based on BigTable (Google)	Cassandra is based on Dynamo DB (Amazon). Initially developed at Facebook by former Amazon engineers. This is one reason why Cassandra supports multi data center.
Infrastructure	It uses the Hadoop Infrastructure (Zookeeper, NameNode and HDFS). Organizations that will deploy Hadoop anyway may be comfortable with leveraging Hadoop knowledge by using HBase	Cassandra started and evolved separate from Hadoop and its infrastructure and operational knowledge requirements are different than Hadoop. But for analytics, many Cassandra deployments use Cassandra+Storm (which uses Zookeeper) or Cassandra + Hadoop.
Infrastructure Simplicity and Single Point Of Failure(SPOF)	The HBase-Hadoop Infrastructure has several "moving parts" consisting of Zookeeper, Name Node, HBase Master, and Data	Cassandra uses a single Node-type. All nodes are equal and perform all functions. Any node can act as a coordinator, ensuring no

	Nodes, Zookeeper are clustered and naturally fault tolerant. Name Node needs to be clustered to be fault tolerant.	coordinator, ensuring no SPOF. Adding Storm or Hadoop, of course, adds complexity to the infrastructure.
Read Intensive Use Cases	HBase is optimized for reads, supported by single-write master, and resulting strict consistency model, as well as use of Ordered Partitioning which supports row-scans. HBase is well suited for doing Range based scans.	Cassandra has excellent single-row read performance as long as eventual consistency semantics are sufficient for the use-case. Cassandra quorum reads which are required for strict consistency and is slower than HBase reads. Cassandra does not support Range based row-scans which may be limiting in certain use-cases. Cassandra is well suited for supporting single-row queries, or selecting multiple rows based on a Column-Value index.
Multi-Data Center Support and Disaster Recovery	HBase provides for asynchronous replication of an HBase Cluster across a WAN. HBase clusters cannot be set up to achieve zero RPO, but in steady-state HBase should be roughly failover-equivalent to any other DBMS that relies on asynchronous replication over a WAN	Cassandra Random Partitioning provides for row-replication of a single row across a WAN, either asynchronous or synchronous. Cassandra clusters can therefore be set up to achieve zero RPO, but each write will require at least one wan-ACK back to the coordinator to achieve this capability.
Ordered Partitioning	It supports only Ordered Partitioning. The rows for a CF are stored in RowKey order in HFiles, where each Hfile contains a "block" or "shard" of all the rows in a CF.	It supports Ordered Partitioning, but user doesn't use it due to the "hot spots" it creates and the operational difficulties they cause. Random Partitioning is the only recommended.

RowKey Range Scans	Because of ordered partitioning, HBase queries can be formulated with partial start and end row-keys, and can locate rows inclusive-of, or exclusive of these partial-rowkeys. The start and end row-keys in a range-scan need not even exist in HBase.	Because of random partitioning, partial rowkeys cannot be used with Cassandra. RowKeys must be known exactly. Counting rows in a CF is complicated. It is highly recommended that for these types of use-cases, data should be stored in columns in Cassandra, not in rows.
Linear Scalability for large tables and range scans	Due to Ordered Partitioning, HBase will easily scale horizontally while still supporting rowkey range scans.	If data is stored in columns in Cassandra to support range scans, the practical limitation of a row size in Cassandra is 10's of Megabytes. Rows larger than that causes problems with compaction overhead and time.
Atomic Compare and Set	HBase supports Atomic Compare and Set. HBase supports transaction within a Row.	Cassandra does not support Atomic Compare and Set. Counters require dedicated counter which because of eventual-consistency require that all replicas in all natural end-points be read and updated with ACK.
Read Load Balancing - single Row	HBase does not support Read Load Balancing against a single row. A single row is served by exactly one region server at a time. Other replicas are used only in case of a node failure.	Cassandra will support Read Load Balancing against a single row. However, this is primarily supported by Read.ONE, and eventual consistency must be taken into consideration.
Secondary Indexes	HBase does not natively support secondary indexes, but one use-case of Triggers is that a trigger on a "put" can automatically keep a secondary index up-to-date, and therefore not put the burden on the application (client).	Cassandra supports secondary indexes on column families where the column name is known. (Not on dynamic columns).

Simple Aggregation	SUM, MIN, MAX, AVG, STD are supported. Other aggregations can be built by defining java-classes to perform the aggregation	Aggregations in Cassandra are not supported by the Cassandra nodes - client must provide aggregations. When the aggregation requirement spans multiple rows, Random Partitioning makes aggregations very difficult for the client.

5.3 Testing

Testing is an important aspect of life cycle of a program. It helps in achieving accurate results with the removal of unnecessary bugs. In order to make the comparison, the same data set was passed through both HBase and Cassandra 10 times to achieve accurate results. The sample dataset is of size 1 GB and consisted of text and images. The storage in database is dynamic in nature.

Software testing is usually a layered process of testing, it includes various type of layers such as the user interface (UI) layer, the business layer, the data access layer and the database itself. The UI layer is responsible for the interface design of the database system, along with the business layer which includes the databases supporting business strategies.

After the execution, results are noted and compared and plotted on bar graph. Testing results show that there's a slight difference in the time (in nanoseconds) every time an operation is performed. An average of these results is used to draw conclusion.

5.3.1 Types of testing and processes

There are basically two types of database testing techniques that are black-box testing and white-box testing.

a. **Black Box Testing in database testing:** Black box testing involves testing interfaces and the integration of the database, which includes:

   - Mapping of data (including metadata).

- Verifying incoming data.
- Verifying outgoing data from query functions.

With the help of these techniques, the functionality of the database can be tested thoroughly.

b. **White Box Testing in database testing:** White box testing mainly handles the internal structuring of the database. The specification details are not shown to the user.

    - It involves the testing of database logical views and triggers which are going to support database rebasing.
    - It performs module testing of database functions, triggers, views, SQL queries etc.
    - It validates database tables, data models, database schema etc.
    - It selects default table values to check on database consistency.

The main advantage of white box of black box testing in database system testing is that coding error are detected, so that the internal bugs/errors generated in the database can be removed. The major limitation of white box testing is that SQL statements are not covered. The major drawback of black box testing is that it is unknown how much of the program is being tested. Also, certain errors cannot be detected.

For any application in a system Unit and Integration Testing can be implemented. This type of testing is used in order to effectively produce the results of the data set. This also helps in preventing any data getting corrupted. The more you test you application database the more effective it becomes. This helps in performing the CRUD operations quickly and efficiently.

Unit and Integration Testing for Cassandra is easy to perform and quick to run. It is deterministic and is not brittle in performance.

Chapter 6: Conclusions and Future Work

There are many common features of distributed databases: Cassandra and HBase and at the same time there are certain differences also. HBase and Cassandra are NoSQL based database systems. In our thesis we have discussed various other features of these database systems and have tried to understand why these databases are used so widely everywhere in the world and why companies are migrating the organizational data from their previous databases to these databases.

Both these databases are used for managing excessively "Big Data". It solves the basic problems faced by database systems in handling big data sets. In our thesis we have implemented Cassandra and HBase on image datasets as well as text data. The data taken is dynamic and unstructured in nature and hence more useful. Cassandra uses CQL (Cassandra Query language) which uses parts of SQL and performs better for dynamic data sets.

Table 5 Average time taken by HBase and Cassandra

	Insertion Time	**Deletion Time**	**Retrieval Time**
Cassandra	**0.075 seconds**	**0.002 seconds**	**0.003 seconds**
HBase	**0.169 seconds**	**0.006 seconds**	**0.007 seconds**

6.1 Conclusion

The following conclusions can be drawn from the results obtained.

i. Cassandra takes lesser time than HBase in regard of operations such as insertion, deletion as well as retrieval of dynamic data.
ii. The time taken by HBase to perform these operations is more than double of that taken by Cassandra for the same.
iii. Cassandra outperformed HBase for the given data set.

6.3 Future Work

We already have concluded that Cassandra is better in performance analysis in comparison to HBase. These databases are used for Business Intelligence that leads to store data for analysis purpose. So as future work, integration of these databases with ETL (Extract, Transform and Load) tools can be done to analyze the live data that has real time behavior such as data of social networking sites and real time applications. ETL is a process in data warehousing which helps in extraction of data from various homogeneous and heterogeneous resources along with transforming the data into proper format for analysis and querying. Performance analysis of these databases on such "Big Data" projects can then be done.

References

[1] K. N. Cukier and V.M. Schoenberger, "The Rise of Big Data," *in Foreign Affairs*, May/June 2013.

[2] M. Mridul , A. Khajuria, S. Dutta, N Kumar," Analysis of Bigdata using Apache Hadoop and Map Reduce" *in International Journal of Advanced Research in Computer Science and Software Engineering,* Volume 4, Issue 5, May 2014 .

[3] K.K. Reddi and D. Indira, "Different Technique to Transfer Big Data: survey," *in Int. Journal of Engineering Research and Applications*, vol.3, issue 6, pp. 708-711, Nov-Dec 2013.

[4] A. Bifet, "Mining Big Data in Real Time," [Online] Available: http://ailab.ijs.si/dunja/TuringSLAIS-2012/Papers/Bifet.pdf , 2012.

[5] B. Purcell, "The emergence of "big data" technology and analytics," *Journal of Technology Research,* 2013.

[6] S.V. Phaneendra and E.M. Reddy, "Big Data-solutions for RDBMS problems-A survey," *in International Journal of Advanced Research in Computer and Communication Engineering,* vol. 2, issue 9, September 2013.

[7] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden and I. Stoica, "BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data", *in EuroSys'13 Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 29-42, New York, ACM, 2013

[8] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, Article 4, June 2008.

[9] Hadoop. http://hadoop.apache.org, 2008.

[10] J. Dean and S. Ghemawat,"MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM - 50th anniversary issue: 1958 - 2008,*vol. 51, no. 1,pp. 107-113, January 2008.

[11] S. Ghemawat, H. Gobioff, and S. Leung,"The Google file system," *SOSP '03: in Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 29–43, New York, USA, 2003.

[12] J. Piernas, J. Nieplocha and E. J. Felix,"Evaluation of active storage strategies for the lustre parallel file system," *in SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing* , pp. 1–10, New York, USA, 2007.

[13] J. Lin, "MapReduce is Good Enough?" [Online] Available: http://arxiv.org/pdf/1209.2191.pdf , September 2012

[14] K. Lee, Y. Lee, H. Choi, Y. D. Chung, B. Moon, "Parallel Data Processing with MapReduce: A Survey" *in ACM SIGMOD Record*, vol. 40, no. 4, pp. 11-20, December 2011.

[15] N. Pansare, V. Borkar, C. Jermaine and T. Condie, "Online Aggregation for Large MapReduce Jobs," [Online] Available: http://www.vldb.org/pvldb/vol4/p1135-pansare.pdf.

[16] T. Condie, N. Conway, P. Alvaro, J. Gerth, J. Talbot, K. Elmeleegy, R. Sears and J. M. Hellerstein,"Online Aggregation and Continuous Query support in MapReduce," *in SIGMOD'10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp. 1115-1118, ACM New York, USA, 2010.

[17] Y. Tanimura, A. Matono, S. Lynden and I. Kojima, " Extensions to the Pig data processing platform for scalable RDF data processing using Hadoop," *in 26th International Conference on Data Engineering Workshops,* IEEE, pp. 251-256, March 2010.

[18] I. Elghandour and A. Aboulnaga, "ReStore: reusing results of MapReduce jobs," in *Proceedings of the VLDB Endowment,* vol. 5, issue 6, pp. 586-97, ACM, February 2012.

[19] C. Olston, G.Chiou, L.Chitnis, F. Liu, Y. Han, M. Larsson, A. Neumann, V.B. Rao, V Sankarasubramanian, S. Seth, C. Tian, T, ZiCornell and X. Wang, ”Nova: continuous Pig/Hadoop workflows,” in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, SIGMOD ‘11*,pp. 1081-90, ACM 2011.

[20] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu. ,”RCFile: a fast and space-efficient data placement structure in MapReduce-based warehouse systems,” *In Proceedings of International Conference on Data Engineering (ICDE),* IEEE, pp. 1199-1208, 2011.

[21] A. Jindal, J.A. Quianie-Ruiz and J. Dittrich, “Trojan data layouts: right shoes for a running elephant,” in *SOCC '11 Proceedings of the 2nd ACM Symposium on Cloud Computing*, article 21, USA 2011.

[22] J. Lee and S. Un. “ Digital Forensic as a Service : A Case Study of forensic Indexed Search ”. *in Proceedings of International Conference on ICT Convergence (ICTC)* , pp. 499--503, Jeju Island, IEEE, October 2012.

[23]V. K. Konishetty, K.A. Kumar, K.Voruganti and G.V.P. Rao, “Implementation and evaluation of scalable data structure over HBase,” *in Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ACM, pp. 1010-1018, New York, USA 2012.

[24] C. Zhang and H. Sterck, “CloudBATCH: A Batch Job Queuing System on Clouds with Hadoop and HBase,” *in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference*, pp. 368-375, December 2010.

[25] M.F. Husain, P. Doshi, L. Khan and B. Thuraisingham, "Storage and retrieval of large RDF graph using Hadoop and MapReduce," *in Proc. 1st Int. Conf. on Cloud Computing (CloudCom 09),* pp. 680-686, Springer-Verla, 2009.

[26] C. Weiss, P.Karras and A. Bernstein, "Hexastore: sextuple indexing for semantic web data management," in Proceedings of the VLDB Endowment, vol.1, issue 1, pp. 1008-1019,ACM, August 2008.

[27]X. Gao,V. Nachankar and J. Qiu, "Experimenting Lucene Index on HBase in an HPC Environment" in *HPCDB '11 Proceedings of the first annual workshop on High performance computing meets databases*, pp. 25-28 , ACM, 2011.

[28] D. Borthakur, J. Gray, J. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash et al. "Apache hadoop goes realtime at facebook," *in Proceedings of the 2011 international conference on Management of data, SIGMOD*, vol. 11, pp. 1071–1080, 2011.

[29] H. Vashishtha and E. Stroulia, "Enhancing query support in HBase Via An Extended Coprocessors Framework,"*in ServiceWave'11 Proceedings of the 4th European conference on Towards a service-based internet*,pp. 75–87, ACM Berlin, 2011.

[30] K. Pingali, D. Nguyen, M. Kulkarni, M. Burtscher, M. Hassaan, R. Kaleem, T. Lee, A. Lenharth, R. Manevich, M. Mendez-Lojo et al., "The tao of parallelism in algorithms," *in Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, pp. 12-25,USA 2011.

[31] J. Dittrich, J.-A. Quianie-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad, "Hadoop++: making a yellow elephant run like a cheetah (without it even noticing)," *in Proceedings of the VLDB Endowment (PVLDB)* , pp:518- 529, 2010.

[32] J. Dittrich, J.-A. Quianie-Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad, "Only aggressive elephants are fast elephants," *in Proceedings of the VLDB Endowment (PVLDB)* , pp:1591-1602, 2012.

[33] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.* ,pp. 35–40, April 2010.

[34] S. Fukuda, R. Kawashima, S. Saito and H. Matsuo "Improving Response Time for Cassandra with Query Scheduling" *in First International Symposium on Computing and Networking*, IEEE Computing Society, pp. 128-133, USA 2013.

[35] H. M. L. Dharmasiri and M.D.J.S. Goonetillake "A Federated Approach on Heterogeneous NoSQL Data Stores", *International Conference on Advances in ICT for Emerging Regions* , IEEE, pp: 234 - 239, 2013.

[36] M. Chalkiadaki and K. Magoutis, "Managing Service Performance in the Cassandra Distributed Storage System," *IEEE International Conference on Cloud Computing Technology and Science*, pp. 64-71, December 2013.

[37] Apache Cassandra. http://cassandra.apache.org/.

[38] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P\. Vosshall, and Werner Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.* ,pp. 205–220, October 2007.

[39] W. Fan and A. Bifet "Mining big data: current status and forecast to the future," *in ACM SIGKDD Explorations Newsletter*,vol. 14, Issue 2,pp. 1-5, December 2012.

[40] C. He, Y. Lu and D. Swanson, "Matchmaking: A New MapReduce Scheduling, " *in CLOUDCOM '11 Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pp. 40-47, USA, 2011.

.

List of Publications

[1] R. Malik and S. Bawa, "Cassandra and Hadoop HBase: A Performance Evaluation," in 1st International Conference on Next Generation Computing Technologies, IEEE. [Communicated]

Video Presentation

[1] The video presentation is present on YouTube under the name “Performance Comparison of HBase and Cassandra” and is available at :

http://youtu.be/z8K2J2JPJGM