JADAVPUR UNIVERSITY

MASTER DEGREE THESIS

Efficient Storage Management for Big Data in Healthcare Domain

A thesis submitted in fulfillment of the requirements for the degree of
Master of Technology in Distributed & Mobile Computing
in the

School of Mobile Computing And Communication

by
SWASTIK MUKHERJEE
University Roll Number: 001730501011
Examination Roll Number: M4DMC19014
Registration Number: 141108 of 2017-2018

Under the Guidance of
Dr. NANDINI MUKHERJEE
Professor, Department of Computer Science and Engineering
Faculty of Engineering and Technology
Jadavpur University
Kolkata-700032

2019

Declaration of Originality and Compliance of Academic Ethics

I hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, i.e. me, as part of Master of Technology in Distributed & Mobile Computing studies.

All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name	: SWASTIK MUKHERJEE
Class Roll No.	: 001730501011
Examination Roll No.	: M4DMC19014
Registration No.	: 141108 of 2017-2018
Thesis Title	: Efficient Storage Management for Big Data in Healthcare Domain

Signed: ____

Date: ____

To whom it may concern

This is to certify that the work in this thesis entitled "Efficient Storage Management for Big Data in Healthcare Domain" has been satisfactorily completed by Swastik Mukherjee, University Roll Number: 001730501011, Examination Roll Number: M4DMC19014, Registration Number: 141108 of 2017-2018. It is a bona-fide piece of work carried out under my supervision at Jadavpur University, Kolkata-700032, for partial fulfillment of the requirements for the degree of Master of Technology in Distributed & Mobile Computing from the School of Mobile Computing And Communication, Jadavpur University for the academic session 2017-2019.

Dr. Nandini Mukherjee
Professor
Department of Computer Science & Engineering,
Jadavpur University
Kolkata-700032.

DIRECTOR
School of Mobile Computing
And Communication,
Jadavpur University
Kolkata-700032.

Prof. Pankaj Kumar Roy
Dean, Faculty of Interdisciplinary
Studies, Law and Management
Jadavpur University
Kolkata-700032.

Certificate of Approval

(Only in case the thesis is approved)

This is to certify that the thesis entitled "Efficient Storage Management for Big Data in healthcare Domain" is a bona-fide record of work carried out by Swastik Mukherjee, University Roll Number: 001730501011, Examination Roll Number: M4DMC19014, Registration Number: 141108 of 2017-18, in partial fulfilment of the requirements for the award of the degree of Master of Technology in Distributed & Mobile Computing from the Department of Computer Science and Engineering, Jadavpur University for the academic session 2017-2019. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

(Signature of the Examiner)
Date:

(Signature of the Examiner)
Date:

Jadavpur University

Abstract

Faculty of Interdisciplinary Studies, Law and Management, Jadavpur University
School Of Mobile Computing And Communication
Master of Technology in Distributed & Mobile Computing

Efficient Storage Management for Big Data in Healthcare Domain

by
SWASTIK MUKHERJEE
University Roll Number: 001730501011
Examination Roll Number: M4DMC19014
Registration Number: 141108 of 2017-18

The technological advancements in various sectors has led into generating huge amount of data that can bring a great value to the business if this data is managed properly. Healthcare is one of the most important sector where data can revolutionize the conventional way of medication, decease detection and emergency protocol following. Various treatment policies, prescription, patient history, patient family history, test report, reports from tracking devices help the consultants to examine the progress of the patient's health properly. However, conventional Electronic Health Record (EHR) systems are not capable to handle this enormous amount of data. Besides, different format of maintaining health data in different organizations restricts them from exchanging valuable information. An efficient storage and retrieval platform for health data is thus very necessary to resolve this issues. Hadoop is one of the most popular choice for distributed Big data processing and storage with its distributed file system and MapReduce programming model. However, there exists some issues in Hadoop which is restricting it to use it's full potential for high performance computing. The blind partitioning of huge dataset leads toward the random fragmentation and distribution of data in hundreds or thousands of cluster nodes which creates a performance gap. This thesis intends to carry out a study on issues and performance gap in Hadoop for storing and handling Big data and further proposes the architecture of efficient big data storage and retrieval platform in healthcare domain. The proposed architecture is capable of handling vertical and horizontal fermentation with guided MapReduce deployment.

Acknowledgements

On the submission of "Efficient Storage Management for Big Data in healthcare Domain", I wish to express gratitude to the Department of Computer Science & Engineering for sanctioning a thesis work under Jadavpur University under which this work has been completed.

I would like to convey my sincere gratitude to **Dr. Nandini Mukherjee**, Professor, Department of Computer Science & Engineering, Jadavpur University for her valuable suggestions throughout the project duration. I am really grateful to her for her constant support which helped me a lot to fully involve myself in this project and develop new approaches in the field of Big Data Engineering.

I would like to express my sincere, heartfelt gratitude to Himadri Sekhar Roy, Ph.D Scholar, Department of Computer Science & Engineering, Jadavpur University, Kolkata, for suggestions guidance and constant Support.

I would also wish to thank Prof. Punyasha Chatterjee, Director of the School Of Mobile Computing Communication, Jadavpur University and Prof. Pankaj Kumar Roy, Dean, Faculty of Interdisciplinary Studies, Law and Management, Jadavpur University for providing me all the facilities and for their support to the activities of this research.

Lastly I would like to thank all my teachers, classmates, guardians and well wishers for encouraging and co-operating me throughout the development of this project. I would like to especially thank my parents whose blessings helped me to carry out my project in a dedicated way.

Regards,
SWASTIK MUKHERJEE
University Roll Number: 001730501011
Examination Roll Number: M4DMC19014
Registration Number: 141108 of 2017-18
School of Mobile Computing And Communication
Jadavpur University

Signed: ______________________________

Date: ______________________________

Contents

List of Figures

List of Abbreviations

NoSQL	**N**ot **O**nly **SQL**
SQL	**S**tructured **Q**uery **L**anguage
CAP	**C**onsistency **A**vailability **P**artition Tolerance
BASE	**B**asically, **A** vailable **S**oft state **E**ventual cnsistency
ACID	**A**tomicity **C**onsistency **D**urability **I**solation
GFS	**G**oogle **F**ile **S**ystem
HDFS	**H**adoop **D**istributed **f**ile **S**ystem
YARN	**Y**et **A**nother **RDDR**esource **N**egotiator
RDD	**R**esilient **D**istributed **D**ataset
RPC	**R**emote **P**rocedure **C**all
HDD	**H**ard **D**isk **D**rive
RPC	**R**emote **P**rocedure **C**all
API	**A**pplication **P**rogram **I**terface

Chapter 1

Introduction

1.1 Introduction

Management of healthcare has become one of the most challenging sector converging towards wide range of areas like patient report generation, patient monitoring, patient record accumulation, habits and personalized medications, drug development, patient similarity prediction, patient history and reports conservation. This extensive amount of actions produce enormous amount of data which should be captured, cleaned, stored and processed. Complex data analysis algorithms and predictive analysis tools must be used for extracting right and real time information from this. This vast amount of data includes medical imaging like ECG reports, scan images, other body parameters for any disease, textual formats like prescription, blood and any other test reports, consultants notes, insurance claims, previous health issues, unstructured information from wearables and mobile phones special apps, behavioural reports by tracking devices. It is very challenging to store these vast amount and variety of data in conventional Electronic Healthcare Record (EHR) systems [1] that too in a cost effective way. The sensory devices and mobile phones being smart enough to capture data of patient on the daily basis, various treatment protocols for excessive case of Diabetes or heart failure or elder care are being possible by using Machine Learning tools which need a huge amount of cleaned and trustworthy patient data. Analysis of these huge amount of data provides the potentiality of identifying side effects of medicine, root cause of early infection, allergic reactions, responsibility of food and daily habits which further leads to the need of data intensive applications.
In today's era of advanced 5G technologies, sensory and smartphone devices and social media platforms, massive amount of generated data is revolutionizing every sector like business, marketing, healthcare, agriculture and education. This huge amount of data needs to be managed properly and cost

efficiently irrespective of their variety and volume. This scenario is triggering to emerge many big data technology platforms to store and process according to the business need. Hadoop, under Apache open sourcing umbrella, has gained a massive popularity for its distributed architecture and horizontal scalability with cost efficiency. The HDFS is the core component for efficient storage of huge amount of data distributed across the cheap commodity cluster nodes while MapReduce processes the chunks of data in parallel by moving computation to data itself. This features of Hadoop makes it very different for processing big data comparing to other conventional technologies. However, file fragmentation variety and key based partitioning of dataset are not available in the hadoop as it does blind partitioning of dataset. Column wise fragmentation or guided deployment of MapReduce is also an issue that are not resolved in the hadoop. These problems are restricting Hadoop to perform better based on the business need.

1.2 Motivation

The huge amount of data generated in the health sector are very essential for patient care and accurate real time medication. However, the massive amount of generated data are either unstructured or redundant or dropped due to proper management or cleanliness of data. Data exchange in various hospitals are also rare as different organization maintains different formats that creates a great bottleneck for information exchange. Some studies find column family based processing platforms performing better for some scenario whereas some studies find clustered processing of data as better performing tool while horizontal fragmentation of data has also become very relevant for Hadoop.The study [3] shows performance issues in hadoop which can degrade performance for healthcare data analysis. It has been also observed that guided MapReduce deployment in Hadoop cluster is not supported as it does blind partitioning and distribution of big data fragmented in smaller chunks leading towards the random distribution of data among the hundred or thousands of Hadoop cluster nodes. Key-based file fragmentation is also not supported by Hadoop framework that can improve clusterization of data on the basis of business need. These scenario leads to need of an efficient big data storage and retrieval platform resolving all these issues.

1.3 Objective

The main objective of the thesis is to propose a highly scalable distributed health data storage and retrieval platform which is capable to store and process enormous amount of big data. The thesis intends to propose the architecture to -

- Support key-based fragmentation of big data files.
- Enable guided MapReduce deployment for data analysis.
- Clustered storage of the data based on the feature selected by the end user depending on the business need.
- Impose different use of replication factor without any data loss.
- Imply vertical and horizontal fragmentation features.
- Implement web based interactive user Interface to use Distributed platform.
- Import data from external sources and storing them into user specified cluster nodes.

1.4 Structure of The Thesis

The thesis is divided into seven chapters discussing the different aspects in details as of below.

- Chapter 2 provides an in depth theoretical background on big data.
- Chapter 3 discusses the feature and architectural overview of Apache Hadoop
- chapter 4 summarises a study on the performance issues observed in Hadoop distributed framework.
- chapter 5 provides a brief overview on the proposed architecture.
- chapter 6 describes the tools and techniques used to implement the framework.
- chapter 7 includes the discussion, limitations and future research directions for the proposed architecture.

Chapter 2

BIG DATA

2.1 Introduction

The rapid technological advancements in sensory technologies, Internet of things, Artificial Intelligence, social media platforms, Internetworking and many other alike fields are yielding gigantic amount of data every second, every day. The communication platforms being more open to general people and World Wide Web being the backbone of the same, the data intensivity has increased so dramatically that the world is tending towards a new data driven world where data is the new power. This huge amount of data, if manipulated properly, can bring a great impact with statistical analysis and mathematical tools which is already evident at the fields like business analysis, stock marketing, health care, gnomic research, astronomy, gov. sectors , agricultural industry and many other fields. To harness the merit from these enormous amount of data, it needs to be captured, cleaned, stored and processed to further analyze, share, transfer, query or visualize. However these enormous amount of data is ubiquitously known as "Big Data" to generalize the challenges associated with this.

2.2 Definition

Big Data usually refers to the data set with size beyond the ability of handling by commonly used software tools that is they are difficult to capture, manage, process and analyze the data within a tolerable elapsed time. An American information technology research [2] and advisory firm updated its definition as follows "Big Data is high volume, high velocity, high variety information assets that requires new form of processing to enable enhance decision making, insight discovers and process optimization".

2.3 Characteristics of Big Data

The term 'Big Data' refers to the characteristics and challenges embedded in harnessing the knowledge from it. It can be elaborated [5] as

- **Volume –** The continuous generation of data is growing exponentially and becoming larger and larger. The accuracy of the data analytics or analysis increases with the increase in the size of the data set. However, these much amount of data should be cleaned, stored and processed efficiently. This is one of the major challenges of big data whose size is converging towards petabytes to zettabytes or larger than that. It is quite evident that this challenge of big data should be managed efficiently to manipulate it properly.

- **Velocity –** The sudden rise in the sensory technologies, smart devices and scaling advancements of various communication platforms have been resulted in the generation of data at high rate. This data should be captured properly so that the computation can take place on the fly and the data can be manipulated in real time. But if the rate of data generation is beyond the capturing capability of the existing conventional systems then it is called the problem of velocity for big data.

- **Variety –** The data being generated in different fields from different source are of different kinds and of different shape i.e. format. The generated data from various sources can be of structured, unstructured or semi-structured. These generated data, irrespective of their forms or variety, should be accumulated in the data pool efficiently to harness the knowledge from it.

- **Veracity –** Veracity of data refers to the trustworthiness or the verifiability of the same generated from a source. These is a major problem of data gathering. As all of the business decision and logics depends on the statistical analysis of dataset whose accuracy fully depends on the quality of the data being gathered from various sources.

- **Value –** All the challenges embedded with big data are mainly focused to solve and extract a great value for business which is solely depended on the right methodologies to apply and implement accurately leading towards more efficient business decisions.

2.4 Solving big data problems

The emergence of distributed systems, cloud technology and internet technology has made lot of alternatives and new technologies to rise for storing and processing big data. Open sourcing of software, usage of cheap commodity hardware and advanced communication platforms has leaded to revolt various new techniques for handling, managing, storing and processing big data. These techniques are discussed below.

2.4.1 Server

The continuous integration of cheap hardware technologies and rapid popularity of open source GNU/Linux Os has made the availability and management of servers very easy and cost effective. The amalgamation of servers with different kinds of services and advancement in cheap network devices create a flexibility for better management of Big data application.

2.4.1.1 Virtualization

The virtualization [6] of hardware and software with high degree of abstraction makes very flexible environment and performance enhancement for processing big data. The degree of concurrency and full utilization of underlying hardware technologies makes the concept of virtualization very effective for processing big data.

2.4.1.2 Platform for managing data

The exponential growth of volume of big data sometimes causes very challenging situations for implicit managing and processing the same for real time data intensive applications leading towards the need of explicit scheduler and job managing software modules for high performance. The scheduling, managing and cleaning platforms are very suitable for automatic resource management and allocation.

2.4.2 Open sourcing

The open sourcing of various technologies like hadoop makes it very easier to develop and integrate modules suitable for case specific, data specific or application specific scenario rather than implementing everything from scratch. The flexibility of open source software also helps to allied with other technologies to create efficient and cost effective platforms.

2.5 Big data kindret technologies

The advancement of technologies has emerged in various big data processing platforms and frameworks. New type of databases are being highlighted with optimal performances. The case specific or scenario based performance of the application varies for which the work is being carried out to find out the issues. The following subsections depict various big data allied technologies that has gained a great popularity over the years.

2.5.1 NoSQL database

Not **o**nly **S**QL is a paradigm of data management suitable for distributed datasets which are very large in size. It is different from the techniques and concepts used for the relational model that uses tabular approach. Although the name NoSQL creates confusion of terminating SQL but it may or may not support the SQL functionalities.

2.5.2 Basic Paradigm of NoSQL

As stated earlier, NoSQL databases donot prohibit the usage or advantages of SQL. The NoSQL system which are designed to be completely non-relational tends to avoid partial functionalities provided by the tabular structured query language. NoSQL paradigm depicts three main principle of NoSQL [4]

- **The Base Theorem** Base model is totally diverse concept of ACID model. The BASE model refers to the Basically Available, Soft state and Eventual Consistency which signifies in the high availability of data. Soft state depicts the span or period of time when the system can be non synchronous and at the last time data should be consistent.
- **The CAP Theorem** One of the famous concept referring to the theorem of Brewer′s CAP theorem [16 himadri]. Consistency, availability and tolerance of network partition. The main idea is not to satisfy all three criteria of CAP at the same instance but to achieve at least two at any instance of time. The tendency to orient a system on NoSQL paradigm is CP, CA, or AP.

- **The Eventual Consistency Theorem** This is one of the consistency model that is very popular for the paradigm of parallel programming. It particularly implies a sufficiently long period of time over which no changes are sent, all updates can be expected to propagate eventually through the system and all the replicas will be consistent.

2.6 Column family based databases

Column family based databases [7] are gaining popularity for storing the unstructured data in the following column oriented model. The concept of schema turns into the concept of key-space for the column family based storage techniques where a column family holds one or more rows.

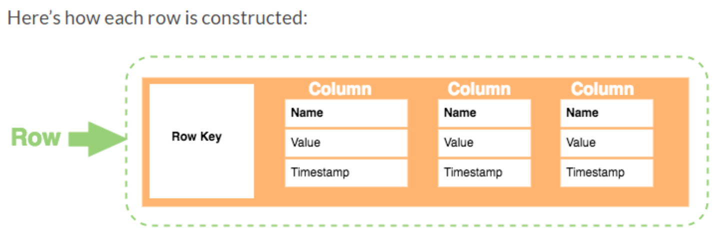


FIGURE 2.1: Column family based databases

Varying number of columns can be stored in each row in column family based databases. Column stores are very efficient as it uses data compression techniques for space efficiency. The scalability of column family based databases is very high and these databases perform great for aggregation queries. The main feature of column family based databases is that they are very fast to be loaded and processed.

2.7 Hadoop

Hadoop [8] under apache open source licence has gained a massive popularity as it can process huge volume of data in parallel and store them in the HDFS which is distributed over the cluster nodes made with cheap commodity hardware.

It was developed with the concept of Google File System (GFS) [9] and MapReduce [10] model is an added advantage with high degree of concurrency. The cheap commodity cluster nodes used in the HDFS is scalable to hundreds or thousands of nodes with fault tolerance and cost efficiency. Lots

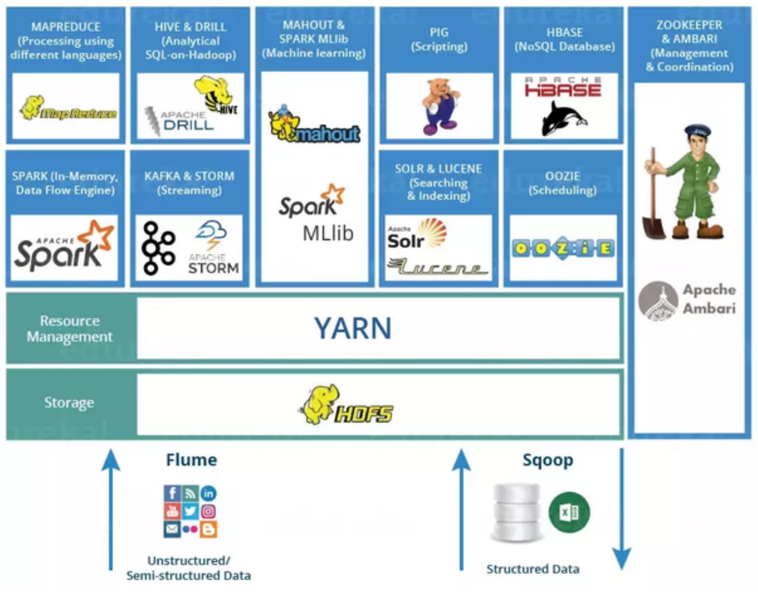

MAPREDUCE (Processing using different languages)
HIVE & DRILL (Analytical SQL-on-Hadoop)
MAHOUT & SPARK MLlib (Machine learning)
PIG (Scripting)
HBASE (NoSQL Database)
ZOOKEEPER & AMBARI (Management & Coordination)
SPARK (In-Memory, Data Flow Engine)
KAFKA & STORM (Streaming)
SOLR & LUCENE (Searching & Indexing)
OOZIE (Scheduling)
Resource Management
YARN
Storage
HDFS
Apache Ambari
Flume
Unstructured/ Semi-structured Data
Sqoop
Structured Data

FIGURE 2.2: Hadoop stack for managing big data

of work have been carried out to improve the performance of hadoop. Many software facilities of databases and other utilities are being added. Yet Another resource Negotiator (YARN) [11] is a resource manager added with hadoop that makes it more reliable, fast and fault tolerant.

2.8 Spark

Apache spark is an open source distributed cluster computing framework that supports both the real time and historical data analysis. It relies on the Resilient Distributed Dataset (RDD) [12] which is nothing but a read-only multiset of distributed data items accumulated by cheap commodity cluster nodes.

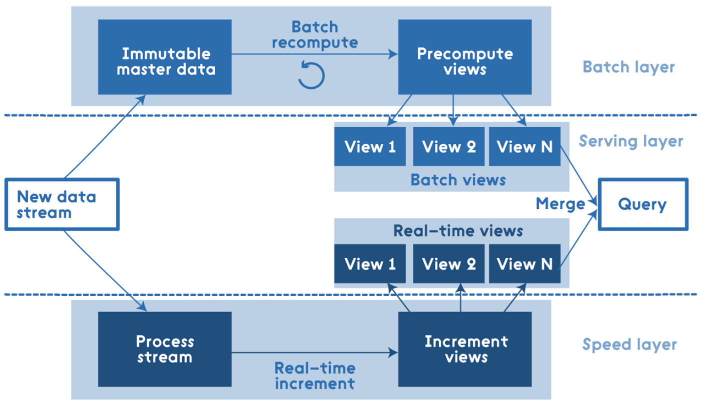


FIGURE 2.3: Lambda architecture of Apache spark

The concept of RDD facilitates to overcome the limitations and restrictions found in MapReduce paradigm. It also supports many machine learning libraries and GraphX [13], the graph processing library.

Chapter 3

Apache Hadoop

3.1 Introduction

The distributed data processing and storage is a prior choice for capturing, storing, managing and analysing huge amount of data which also provides reliability, recovery, file management, parallel processing with consistency. The main feature of distributed systems is scalability. In the distributed data processing platform horizontal scaling adds an added advantage of affixing any number of nodes at will. These advantages helps to remove the shortage of space much needed for big data and also adds more number of computations with parallel processing and high throughput. Vertical scaling refers to adding more power to existing computing machines which somehow increases cost making the application more expensive. However, horizontal scaling gives the flexibility to increase the number of nodes depending on the need of application requirement or business needs. Thus horizontal scaling removes the shortage of space required to store and process data at lower cost and little maintenance complexity.

Distributed Big data processing frameworks should have the capability to store the data and read or write it based on the application scenarios. Data management, data cleaning, data read or write, data integration, data analysis or predictive analysis is taken place very frequently for business case study. These further leads to the need of a file system that is efficient enough to allow read-write operations on the data distributed over many nodes with the capability of parallel processing. These requirements led google to form a distributed file system named as Google File System (GFS) that allows user to use cheap commodity hardwares to store big data which further led to form google Big table based on the concept of cloud computing. Yahoo on the other hand, introduced HDFS based on the ideology of GFS and got instant attention of big data engineers and researchers as well.

3.2 MapReduce

MapReduce refers to the programming model working on the [Key,Value] pairs, mainly implemented aiming to process and hatch large data sets. The MapReduce model works on the basis of map function and reduce function. Key-Value pairs, regulated by the user, are processed by the map function to further generate intermediate Key-Value pairs. The intermediate Key-Value pairs are further fed to reduce function for merging all the values for each key. Parallel execution in distributed cluster nodes requires communication

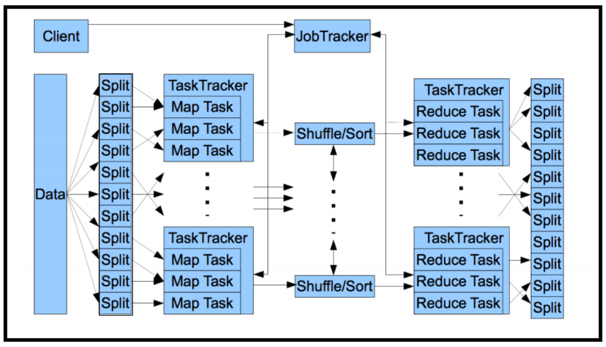


FIGURE 3.1: MapReduce working mechanism

between machines, splitting of input data, scheduling the program execution among nodes, data failure or machine failure recovery, synchronization among all the nodes and consistency among read-write operations for every node. All these challenges are handled automatically in the MapReduce programming model to increase the transparency and required to have a little or no experience of distributed systems for using or manipulating the same. These features of MapReduce model bring a great flexibility to process terabytes of data. It is also scalable to thousands of commodity cluster nodes working in parallel and producing high throughput for enormous chunk of data.

3.2.1 Execution Overview

The input data gets automatically splitted into a set of N parts for the invocation of map function among all the distributed nodes. Multiple splits offers the ideal environment for processing partitioned inputs in parallel. The intermediate key space is partitioned into R pieces with the help of partitioning function e.g. [h(key) Mod R] where h refers to a hash function. The partitioning of intermediate key is aimed to invoke the reduce function. The number of partitioning function and partitions are regulated by the user. The calling of MapReduce function generates sequence of steps to flow the execution of Map and Reduce function.

The input files, at first, splitted into N pieces by the MapReduce library each of whose size differs generally 16 megabytes (MB) to 64 MB. However, the size of the input splits are totally controlled by the users. The program then starts copying itself to be executed among all the cluster nodes. Among all the copies of the program one copy gets a special treatment for being the server and the rest being the slave or clients assigned by the server. The server selects the otiose clients for N map tasks and R reduce tasks to assign the same. The map tasks and reduce tasks differ in the responsibilities. The client assigned with map tasks read the records of the input split fed to it for parsing the Key-Value pair and passes each pair to user defined map functions while the intermediate Key-Value pairs are buffered in the memory. This buffered data are partitioned into R parts and being written into the disks periodically. The location of these written pairs are accumulated only to be forwarded to the server which in turns sends the same to reduce clients. Being proclaimed by the server about the addresses of partitioned data, the reduce client uses Remote Procedure Call (RPC) to read the buffered data. All the intermediate data are then gets sorted by the intermediate keys in a way that all the occurences of the same type of key are batched together. The type of sorting i.e. external sorts or internal sorts solely depends on the size of the intermediate data. If the size of the intermediate data is too large then the external sorting technique is used extensively. The sorted intermediate data are then iterated over by the reduce client. The key is passed depending on the encounter with unique intermediate key.The output of the Reduce function is attached to a final output file for this reduce partition. The server wakes up just after the completion of all the map and reduce tasks and The user program call of MapReduce returns back to the user code. The successful execution of MapReduce results in generating R number of output files which further may be treated as input file of another MapReduce functions.

For each map and reduce tasks the state of each task i.e. idle, in-progress or completed is tracked in server data structure. The location of intermediate file regions are converged from map to reduce client via server. That is why the size and location of the R is also stored in the server data structure.The MapReduce library is highly scalable from hundred to thousands cluster nodes. So, it is obvious that it needs to be fault tolerant for more reliable execution of whole task.

3.3 HDFS

The Hadoop Distributed File System (HDFS) [14] is one of the most popular distributed file systems which runs on the cheap commodity cluster nodes without replacing the core file system provided by the Operating System of the same. HDFS is scalable from hundreds to thousands of cluster nodes with the feature of high fault tolerance at lower cost and great reliability. The high throughput feature of HDFS makes it suitable for hatching large dataset of applications.

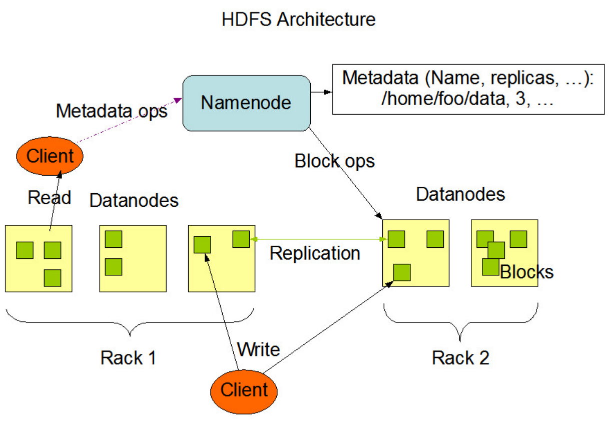


FIGURE 3.2: HDFS architecture

The hardware failure in HDFS is very normal as it is made of hundreds or thousands of server machines connected with network switches. Each of the server machines, being made of cheap commodity hardwares, holds part of the HDFS. The huge number of components makes it regular with the non-trivial probability of failing components or nodes any time. Thus discovery of failure and fast and automatic recovery from failure is must for reliability of the whole architecture. HDFS supported applications need streaming access to the data set as they are not the general purpose applications running on regular file systems. The HDFS is typically designed for batch processing than interactive use. The main motto of the architecture is high throughput data access. Being scaled to hundreds of nodes the HDFS is supposed to support large data sets ranging from gigabytes to exabytes and even larger than that. High aggregated data bandwidth and hundreds of nodes is supported in single cluster further supporting tens of millions of files for individual instances. HDFS follows simple approach of Read-Many-Write-Once file access policy to maintain the tenacity. A file once updated and closed need not to be changed. This approach makes it easy to maintain the coherence among multiple nodes accessing single file. The data being huge in size gets distributed among hundred to thousands nodes and thus taking computation near data rather than bringing data near computation is cost effective in terms of network communication, bandwidth utilization and high throughput generation for the system. Portability is another factor that makes HDFS so special as it is easily portable from one platform to another.

3.4 Namenode and Datanode

HDFS is totally based on the server-slave architecture with one server server node named as Namenode which manages the file system namespace and access control to files by clients. The rest of the individual cluster nodes are termed as Datanodes. They together form a storage space HDFS running on it. A file system namespace is revealed by the HDFS to store the user data in the file system. The big data file is splitted into one or more fixed size blocks and stored in the Datanodes available in the cluster. The file system namespace operation like open, close, renaming of file and directories are executed by the Namenode along with the mapping of blocks to Datanode. However, the read and write operations are served by the Datanodes. Block creation, replication and deletion is handled by the Datanodes but only on the requests from the Namenode. The Datanode and Namenodes are cheap

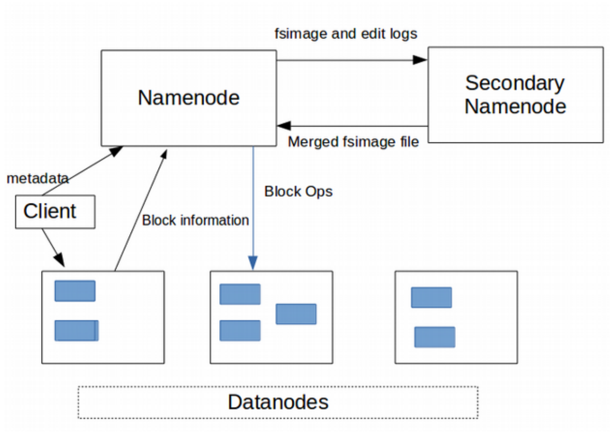


FIGURE 3.3: Namenode and Datanode overview

commodity hardware machines with GNU/Linux OS running on it. The builtin language supported by the HDFS is java which is highly portable and can be extended to any platforms. Every task is regulated from Namenode which in general configured on more reliable machines. No two Datanodes communicate with each other to avoid network complexity and architectural bottlenecks.

3.5 File System Namespace

HDFS traditionally supports hierarchical order of file organization. The application or users can create directories and store files inside these directories. The file system namespace hierarchy is same as most of the other existing file systems. The files can be created, read, renamed and moved from one directory to another.HDFS does not support hard links or soft links. However, the HDFS architecture does not repel implementing these features. The total information of file system is maintained by the Namenode which consistently keep tracks of the changes taken place on any instance of time. Even, the number of copies of a particular file is kept in list for failure recovery. The number of copies of file is terms as replication factor. The main advantage of HDFS is that it doesn't replace the core operating system running on top of the commodity clusters and each and every node of the cluster holds a part of the each part of HDFS. Though the HDFS runs on the unreliable cheap hardwares it is highly available, consistent and reliable to fetch data any number of time. Beside the horizontal scaling factor of HDFS also brings the flexibility to add any number of node to the cluster without affecting the capability of HDFS. The application using HDFS can accumulate data by generating a new file and appending or writing back the data into the newly created file. The bytes that are written to the file can't be altered or removed. The updation of the file is allowed only after reopening the file and appending or deleting the bytes and closing it. This helps to achieve the consistency of HDFS. HDFS follows the approach of 'Single Writer, Multiple Reader' (SWMR) approach.

A fixed time lease for the opened file by the HDFS client is generated. During the lease time, the file gets locked for others client leading towards the consistent change in the file. If the client which has already been granted with the lease required to extend the same then it has to send a heartbeat signal to the namenode and renew the lease period of updating the file. The lease period is strictly bounded with soft limit or hard limit. The client that is granted the lease of a file starts updating it with write operation until the lease has been expired. If the soft limit of lease expires and the lease granted client somehow fails to close the file, the Namenode waits for the hard time limit for response of that client after which it is assumed that the client has exempted from updating the file. The Namenode then automatically revokes the lease and close that file itself on behalf of exempted client. Meanwhile any other client may preempt the lease period of that file. HDFS doesn't guarantee of making the data visible to the reader till the write operation to that particular file is finished and the file has been closed.

The bytes written by the client are first buffered in the temporary storage only to be pushed in the pipeline formed by the Datanodes when the buffer size gets filled. The buffered data are pushed into the pipeline as a form of packet of bytes. Every packet sent, is supposed to get the acknowledgement after which the next packet is dispatched.

3.6 File Fragmentation

In general the dataset in big data environment has a huge size ranging from gigabyte to exabyte and even more than that. It is also evident that this much amount of data cannot not be accommodated in individual Datanodes leading towards the need of accumulation of many Datanodes forming a cluster and the big data file is fragmented into fixed size chunks. This fixed size chunks are called blocks. Though the size of the chunks can be controlled by the user, it generally varies from 64 to 128 MB. Size of all the blocks kept same except for the last block. The smaller size of block increases the number of file fragments which further increases complexity for data locality and data fetching capacity. The larger size of chunks results in the more page replacement and forms complexity in management. All the fragmented chunks or blocks of the big data file is distributed among the cluster node which further gets stored in the Hard Disk Drive (HDD) or secondary memories of Datanodes as a form of file. Generally the conventional secondary data storage formats are divided into tracks and sectors. The intersection point of every track and sector is called a block whose general size is 512 byte. All the blocks of HDD is byte addressable by the file system provided by the underlying OS of Datanodes. The HDFS blocks are much bigger than that of underlying OS, though a block of HDFS is actually combination of multiple blocks of OS at the lower level of storage. Given a source of big data file Hadoop starts buffering the records of the file till it reaches to the specified size of say 64 or 128 MB ad then it sends the buffered data to a datanode available in the cluster as a form of packet and store it there.

Failure recovery is must for HDFS as any Datanode can fail at any point of time. Thus multiple copies of same block of data is kept in various Datanodes. The number of copies of a block can be regulated by the user. However 3 copies are kept in general for automatic and quick failure recovery. The Namenode is mainly responsible for taking decision about replication maintenance and failure recovery. That is why the Namenode sends Blockreport and Heartbeat signal periodically. The Heartbeat signal from a datanode signifies that the Datanode is alive. If for certain period of time the Heartbeat signal is not received that means the Datanode has died. Blockreport from a Datanode signifies the list of all the blocks on that particular Datanode.

3.7 Replica Placement

The reliability and performance of HDFS totally depends on the placement of the replicas among the Datanodes as it signifies the fault tolerance. Optimality in the placement of replica needs lots of attention and experiments as it can make the system lot more reliable than usual.It also increases the bandwidth utilization and data availability.

The HDFS being an accumulation of hundred or thousands of Datanodes, the communication configuration is done via switches. The nodes under one switch is called as rack. One node of one rack is local to another node in the same rack. The network bandwidth inside one rack is higher than outside of that rack. The communication between two nodes of different rack passes by the switches. Every Datanode starts determining its rack number during their the start up time with the help of HDFS API to register with the rack id in Namenode list. The very simplistic approach is to place the replicas in unique racks so that failing of a network switch can easily recover from failure by using data of another rack. However it increases the cost of writes as every updation of any blocks must be reflected in others copies of the same blocks.

To improve the write cost of data blocks with the replication factor three, one copy is placed on one Datanode and another copy is placed on another node of the same rack while the third copy is placed on a different node of different rack. The probability of node failure is much higher than that of rack failure. Fetching of copies from a local rack is fast as the networks bandwidth in the same rack is higher. Selection of replica is done based on the nearest distant node from the reader. If the reader requests to read a block, the nearest copy of that block is served.

A metadata of all the blocks and read-write updates is kept in the name node. The standard TCP/IP protocol is used for communication. Datanode to Namenode and RPC call is used to invoke MapReduce functions to the Datanodes.

Chapter 4

A study On Performance issues in Hadoop

4.1 Introduction

Hadoop has become one of the most advanced and preferable choice for processing and analyzing big data. However, it obviously has some bottlenecks. These shortcoming became a great bottleneck for scalability and performance. IBM highlighted that with this architecture can be scalable to 5000 node and 40000 jobs. However, YARN has been introduced to overcome this problem which can scale to 32000 to 40000 nodes and 26 millions job.

In spite of having this problem there are other problems too that should be resolved to improve Hadoop performance a lot. The big data file which is fragmented into many parts starts a blind partitioning resulting in the random distribution of blocks. The blind partitioning of data is only helpful for faster fragmentation but causes to deploy the MapReduce functions in an unguided manner. This further leads to communication and computation overhead leading towards the poor performance of Hadoop MapReduce model. The fragmentation of big data does not happen according to the user choice based on key. Besides, the replication is also not used intelligently. The replication factor is only used for recovery and fault tolerance.

4.2 Performance Issues

The main performance degradation of Hadoop is due to constant deficiency of high disk performance I/O and bandwidth of network. The impact of bottlenecks affected by disk environments is more obvious than that caused by CPU or memory performance for applications which is data intensive. There are multiple causes for this I/O performance problem bottleneck. The performance gap between processors and I/O systems in a clusters is rapidly amplifying. As an example, processor performance has seen an annual increase of approximately 60% for the last two decades, while the overall performance improvement of disks has been hovering around an annual growth rate of 7% during the same period of time. Second, the heterogeneity of various resources in clusters makes the I/O bottleneck problem even more pronounced.

HDFS always places the first block replica onto the writer node if the node is in the cluster, and on a node with highest proximity otherwise. This scheme makes the cluster very unbalanced in terms of data placement in case the write node does not leverage MapReduce. All the first block replicas would be placed on the writer node. Over a long course of run, this makes the writer node a hotspot. Also, analyzing the MapReduce layer it can be seen that, it tries to execute application copies on cluster nodes that have required data locally available. As it is generally the case that disk I/O is faster than network I/O, moving computation is cheaper than moving data. Since Hadoop has been into its use and prominence for quite a long time, hardware components have evolved since the early stages of Hadoop. The general implementation of HDFS does not take the gap in generation of hardware components into account. Considering disk I/O, an application copy running on a recent disk generation, like Solid State Disks or faster SATA, with twice as much I/O speeds shall be able to transfer, approximately, twice as much as data blocks on an older generation. If the gaps in hardware technologies are given significant weightage, it would surely play an important role in the overall performance of the cluster. If the idea of storing related data can be extended till the hardware disk implementation on the underlying OS level then the locality of reference would bring a great performance enhancement for big data storage and retrieval.

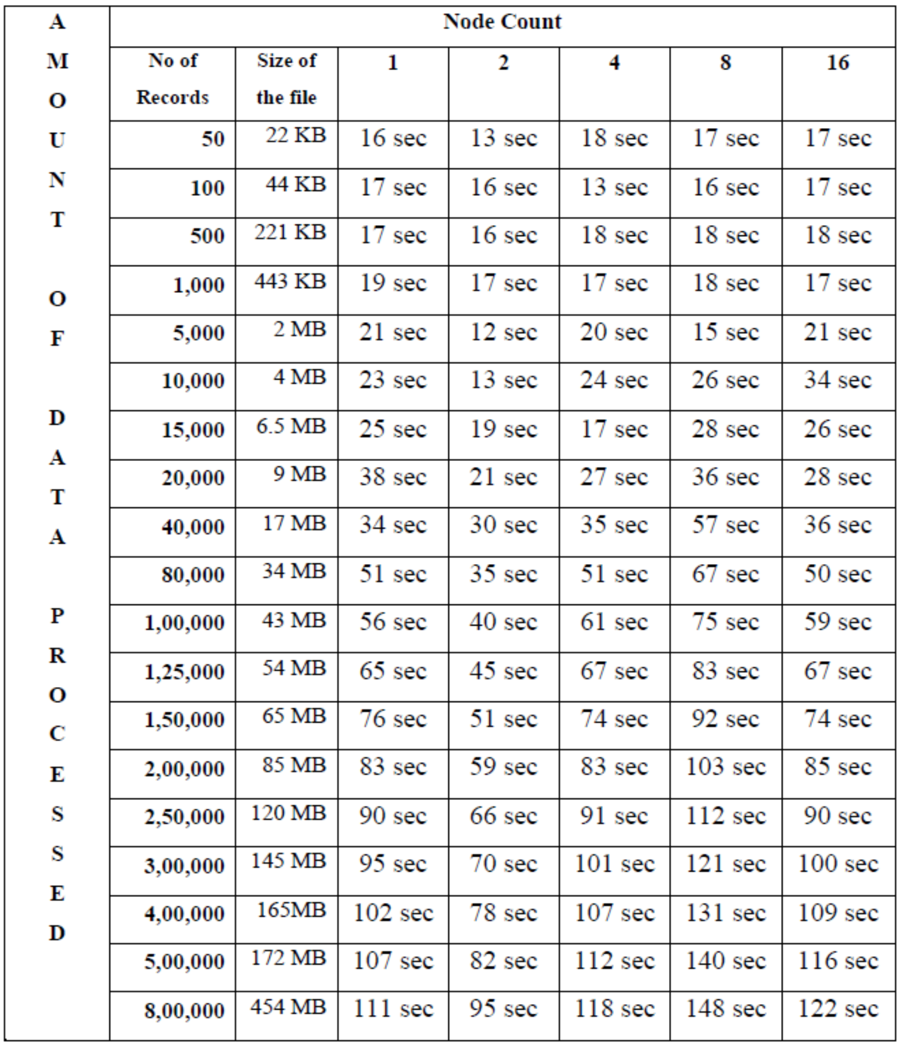
The study [3] proposes a data model and executes some simple queries to examine the performance of Hadoop for each query. The queries are given below.

- Find all doctors names and their details with specialization string is 'cardiologist'.
- Find all information of patients like demographic info, religion, food habit etc, with patient's name 'xyz';
- Find all ECG report of a particular patient with name 'x';

- Find history of 'Diabetes' of a particular patient with name 'x';
- Find sensor observed data of a patient for one hour;
- Patient treatment case history with name 'B';

Every query results in the formation of MapReduce codes to achieve the outcome. The volume of the data flowing the proposed model varies from 8 lac records to only 50 records with 1 to 16 commodity cluster nodes. Each experimentation was done 3 times and the average of the 3 execution time is taken.

Query 1: Find all doctors names and their details with specialization string is 'Cardiologist'.



AMOUNT OF DATA PROCESSED	Node Count						
	No of Records	Size of the file	1	2	4	8	16
	50	22 KB	16 sec	13 sec	18 sec	17 sec	17 sec
	100	44 KB	17 sec	16 sec	13 sec	16 sec	17 sec
	500	221 KB	17 sec	16 sec	18 sec	18 sec	18 sec
	1,000	443 KB	19 sec	17 sec	17 sec	18 sec	17 sec
	5,000	2 MB	21 sec	12 sec	20 sec	15 sec	21 sec
	10,000	4 MB	23 sec	13 sec	24 sec	26 sec	34 sec
	15,000	6.5 MB	25 sec	19 sec	17 sec	28 sec	26 sec
	20,000	9 MB	38 sec	21 sec	27 sec	36 sec	28 sec
	40,000	17 MB	34 sec	30 sec	35 sec	57 sec	36 sec
	80,000	34 MB	51 sec	35 sec	51 sec	67 sec	50 sec
	1,00,000	43 MB	56 sec	40 sec	61 sec	75 sec	59 sec
	1,25,000	54 MB	65 sec	45 sec	67 sec	83 sec	67 sec
	1,50,000	65 MB	76 sec	51 sec	74 sec	92 sec	74 sec
	2,00,000	85 MB	83 sec	59 sec	83 sec	103 sec	85 sec
	2,50,000	120 MB	90 sec	66 sec	91 sec	112 sec	90 sec
	3,00,000	145 MB	95 sec	70 sec	101 sec	121 sec	100 sec
	4,00,000	165MB	102 sec	78 sec	107 sec	131 sec	109 sec
	5,00,000	172 MB	107 sec	82 sec	112 sec	140 sec	116 sec
	8,00,000	454 MB	111 sec	95 sec	118 sec	148 sec	122 sec

FIGURE 4.1: the time for processing to search details of the doctors

The doctor's name and details with the string 'Cardiologist' is returned by the query. The node wise performance graph is shown in following fugure.

The following graph shows the node performance with data volume respect to time.

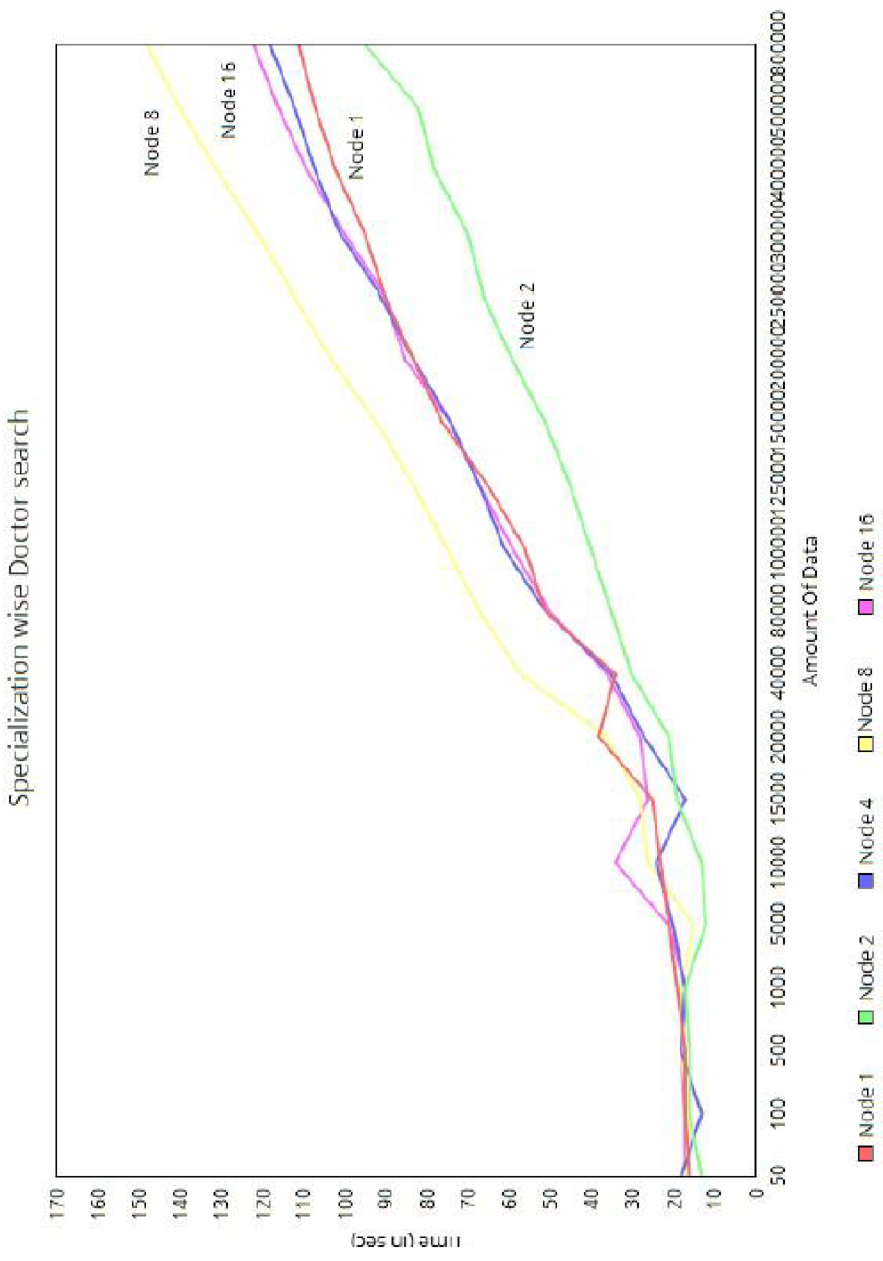
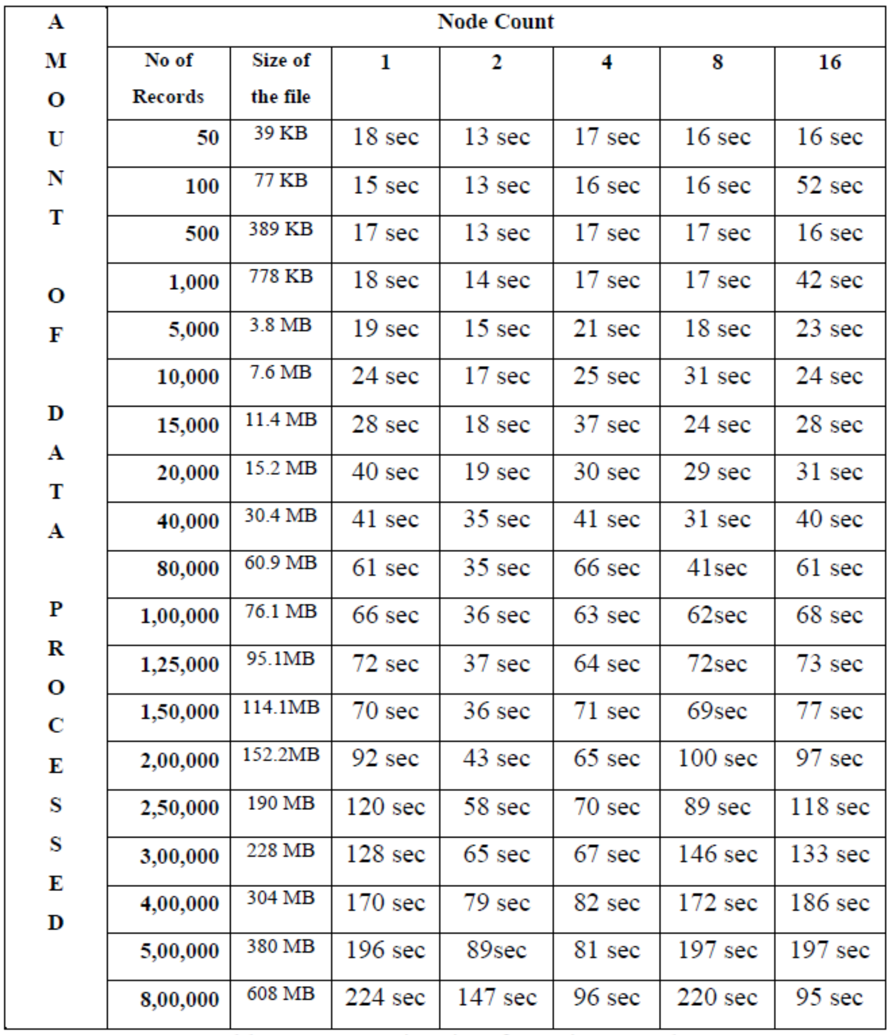


FIGURE 4.2: Node search performances

Query 2: Find all information of patients like demographic info, religion, food habit etc, with patient's name 'xyz'.



AMOUNT OF DATA PROCESSED	Node Count						
	No of Records	Size of the file	1	2	4	8	16
	50	39 KB	18 sec	13 sec	17 sec	16 sec	16 sec
	100	77 KB	15 sec	13 sec	16 sec	16 sec	52 sec
	500	389 KB	17 sec	13 sec	17 sec	17 sec	16 sec
	1,000	778 KB	18 sec	14 sec	17 sec	17 sec	42 sec
	5,000	3.8 MB	19 sec	15 sec	21 sec	18 sec	23 sec
	10,000	7.6 MB	24 sec	17 sec	25 sec	31 sec	24 sec
	15,000	11.4 MB	28 sec	18 sec	37 sec	24 sec	28 sec
	20,000	15.2 MB	40 sec	19 sec	30 sec	29 sec	31 sec
	40,000	30.4 MB	41 sec	35 sec	41 sec	31 sec	40 sec
	80,000	60.9 MB	61 sec	35 sec	66 sec	41sec	61 sec
	1,00,000	76.1 MB	66 sec	36 sec	63 sec	62sec	68 sec
	1,25,000	95.1MB	72 sec	37 sec	64 sec	72sec	73 sec
	1,50,000	114.1MB	70 sec	36 sec	71 sec	69sec	77 sec
	2,00,000	152.2MB	92 sec	43 sec	65 sec	100 sec	97 sec
	2,50,000	190 MB	120 sec	58 sec	70 sec	89 sec	118 sec
	3,00,000	228 MB	128 sec	65 sec	67 sec	146 sec	133 sec
	4,00,000	304 MB	170 sec	79 sec	82 sec	172 sec	186 sec
	5,00,000	380 MB	196 sec	89sec	81 sec	197 sec	197 sec
	8,00,000	608 MB	224 sec	147 sec	96 sec	220 sec	95 sec

FIGURE 4.3: time taken to search patient details

The query returns the result with specified node search performance shown in followingfig.

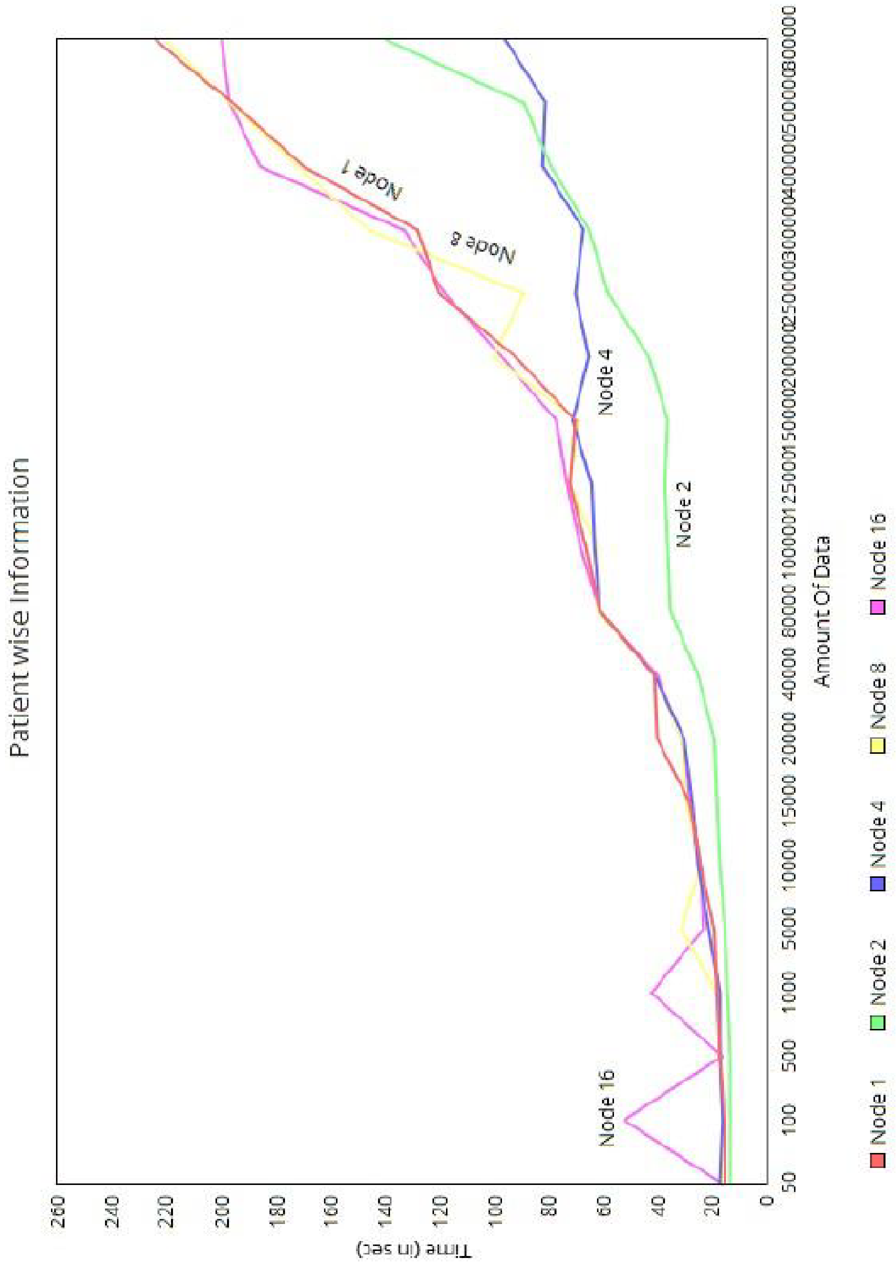
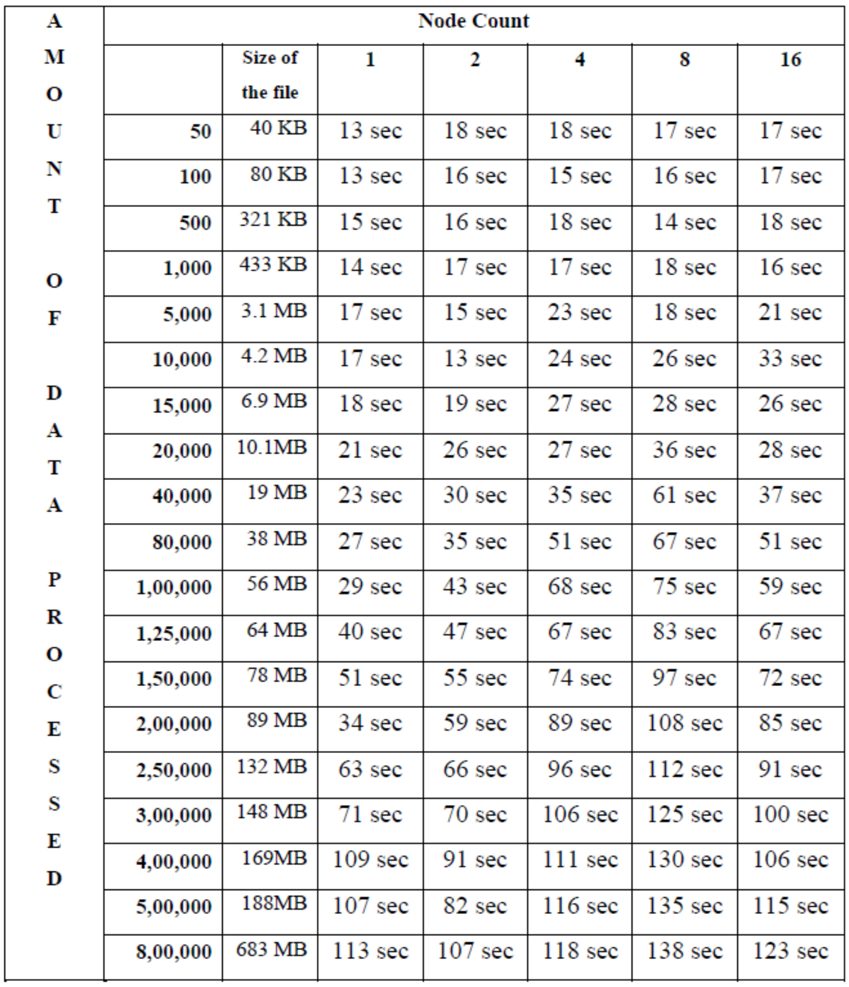


FIGURE 4.4: node search for time taken to search patient details

Query 3: Find all ECG report of a particular patient with name ′x′.



AMOUNT OF DATA PROCESSED	Node Count					
	Size of the file	1	2	4	8	16
50	40 KB	13 sec	18 sec	18 sec	17 sec	17 sec
100	80 KB	13 sec	16 sec	15 sec	16 sec	17 sec
500	321 KB	15 sec	16 sec	18 sec	14 sec	18 sec
1,000	433 KB	14 sec	17 sec	17 sec	18 sec	16 sec
5,000	3.1 MB	17 sec	15 sec	23 sec	18 sec	21 sec
10,000	4.2 MB	17 sec	13 sec	24 sec	26 sec	33 sec
15,000	6.9 MB	18 sec	19 sec	27 sec	28 sec	26 sec
20,000	10.1MB	21 sec	26 sec	27 sec	36 sec	28 sec
40,000	19 MB	23 sec	30 sec	35 sec	61 sec	37 sec
80,000	38 MB	27 sec	35 sec	51 sec	67 sec	51 sec
1,00,000	56 MB	29 sec	43 sec	68 sec	75 sec	59 sec
1,25,000	64 MB	40 sec	47 sec	67 sec	83 sec	67 sec
1,50,000	78 MB	51 sec	55 sec	74 sec	97 sec	72 sec
2,00,000	89 MB	34 sec	59 sec	89 sec	108 sec	85 sec
2,50,000	132 MB	63 sec	66 sec	96 sec	112 sec	91 sec
3,00,000	148 MB	71 sec	70 sec	106 sec	125 sec	100 sec
4,00,000	169MB	109 sec	91 sec	111 sec	130 sec	106 sec
5,00,000	188MB	107 sec	82 sec	116 sec	135 sec	115 sec
8,00,000	683 MB	113 sec	107 sec	118 sec	138 sec	123 sec

FIGURE 4.5: ECG report search

The above query extracts all the information regarding ECG report for a particular patient

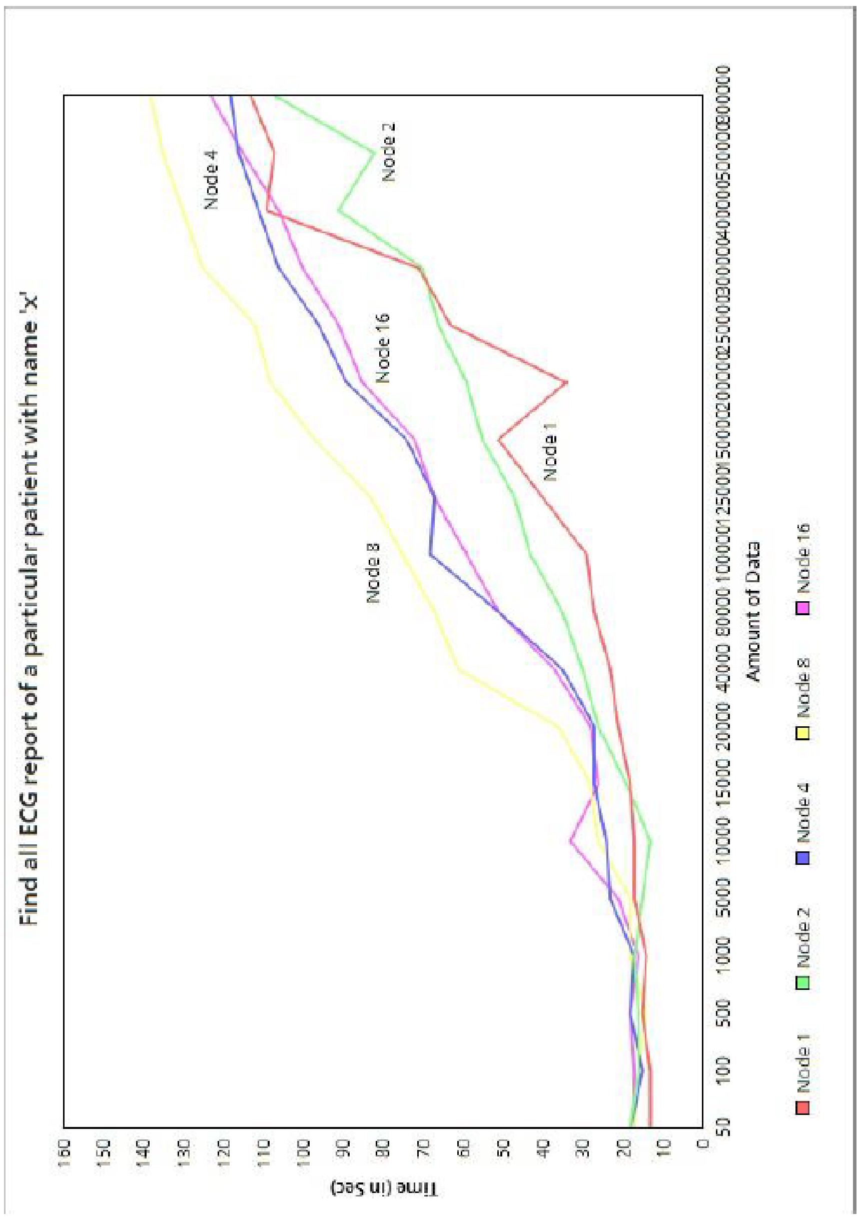
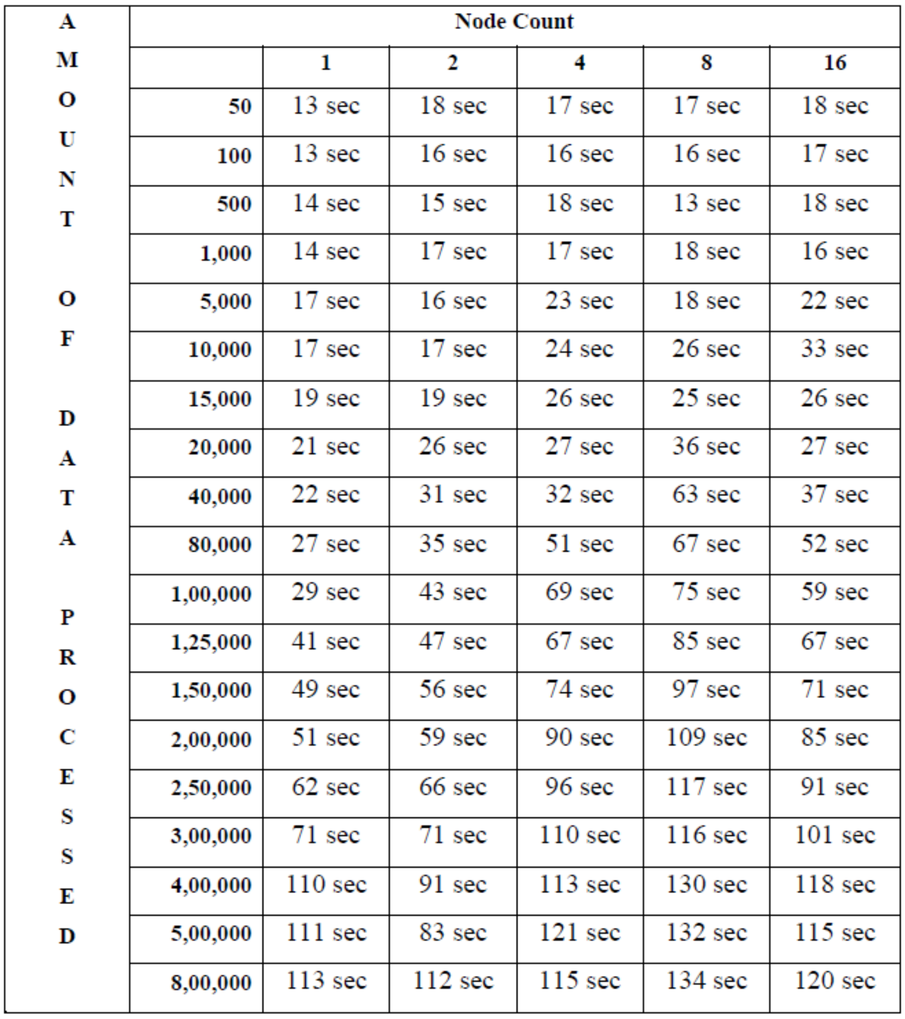


FIGURE 4.6: ECG report node search

Query 4: Find history of 'Diabetes' of a particular patient with name 'x'.



AMOUNT OF DATA PROCESSED	Node Count				
	1	2	4	8	16
50	13 sec	18 sec	17 sec	17 sec	18 sec
100	13 sec	16 sec	16 sec	16 sec	17 sec
500	14 sec	15 sec	18 sec	13 sec	18 sec
1,000	14 sec	17 sec	17 sec	18 sec	16 sec
5,000	17 sec	16 sec	23 sec	18 sec	22 sec
10,000	17 sec	17 sec	24 sec	26 sec	33 sec
15,000	19 sec	19 sec	26 sec	25 sec	26 sec
20,000	21 sec	26 sec	27 sec	36 sec	27 sec
40,000	22 sec	31 sec	32 sec	63 sec	37 sec
80,000	27 sec	35 sec	51 sec	67 sec	52 sec
1,00,000	29 sec	43 sec	69 sec	75 sec	59 sec
1,25,000	41 sec	47 sec	67 sec	85 sec	67 sec
1,50,000	49 sec	56 sec	74 sec	97 sec	71 sec
2,00,000	51 sec	59 sec	90 sec	109 sec	85 sec
2,50,000	62 sec	66 sec	96 sec	117 sec	91 sec
3,00,000	71 sec	71 sec	110 sec	116 sec	101 sec
4,00,000	110 sec	91 sec	113 sec	130 sec	118 sec
5,00,000	111 sec	83 sec	121 sec	132 sec	115 sec
8,00,000	113 sec	112 sec	115 sec	134 sec	120 sec

FIGURE 4.7: Patient history search

The node search performances for hadoop is shown in below graph

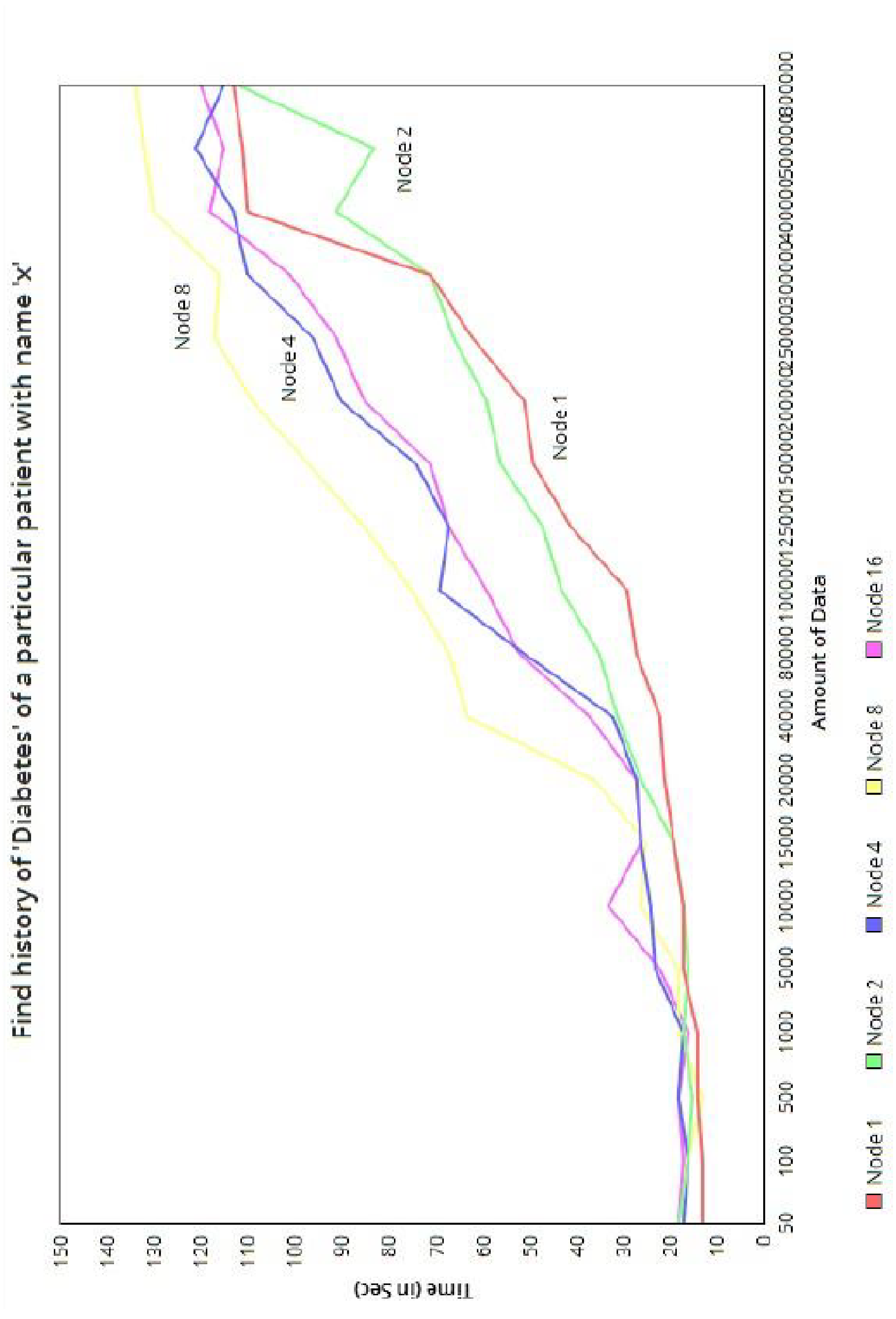
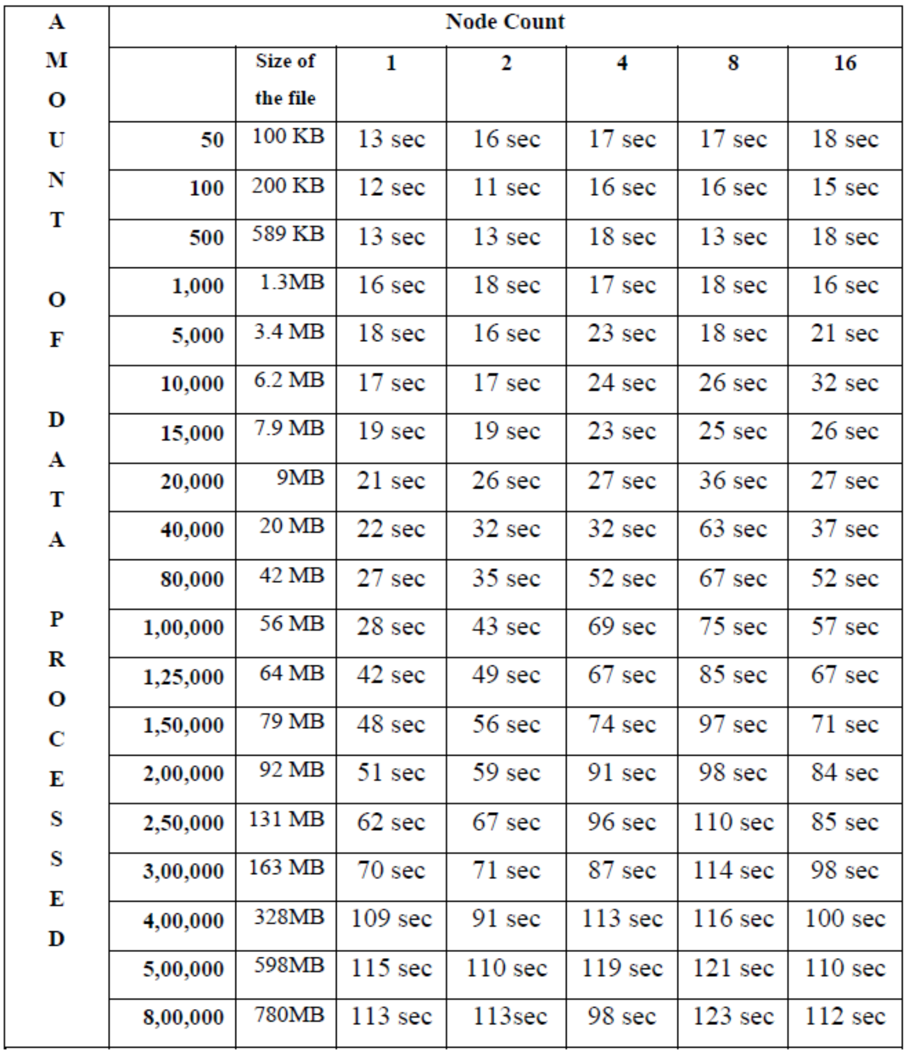


FIGURE 4.8: Patient history search in nodes

Query 5: Find sensor observed data of a patient for one hour.



AMOUNT OF DATA PROCESSED	Size of the file	Node Count 1	2	4	8	16
50	100 KB	13 sec	16 sec	17 sec	17 sec	18 sec
100	200 KB	12 sec	11 sec	16 sec	16 sec	15 sec
500	589 KB	13 sec	13 sec	18 sec	13 sec	18 sec
1,000	1.3MB	16 sec	18 sec	17 sec	18 sec	16 sec
5,000	3.4 MB	18 sec	16 sec	23 sec	18 sec	21 sec
10,000	6.2 MB	17 sec	17 sec	24 sec	26 sec	32 sec
15,000	7.9 MB	19 sec	19 sec	23 sec	25 sec	26 sec
20,000	9MB	21 sec	26 sec	27 sec	36 sec	27 sec
40,000	20 MB	22 sec	32 sec	32 sec	63 sec	37 sec
80,000	42 MB	27 sec	35 sec	52 sec	67 sec	52 sec
1,00,000	56 MB	28 sec	43 sec	69 sec	75 sec	57 sec
1,25,000	64 MB	42 sec	49 sec	67 sec	85 sec	67 sec
1,50,000	79 MB	48 sec	56 sec	74 sec	97 sec	71 sec
2,00,000	92 MB	51 sec	59 sec	91 sec	98 sec	84 sec
2,50,000	131 MB	62 sec	67 sec	96 sec	110 sec	85 sec
3,00,000	163 MB	70 sec	71 sec	87 sec	114 sec	98 sec
4,00,000	328MB	109 sec	91 sec	113 sec	116 sec	100 sec
5,00,000	598MB	115 sec	110 sec	119 sec	121 sec	110 sec
8,00,000	780MB	113 sec	113sec	98 sec	123 sec	112 sec

FIGURE 4.9: sensor observed data

the query results in observing data for patient. below graph shows the node search performance.

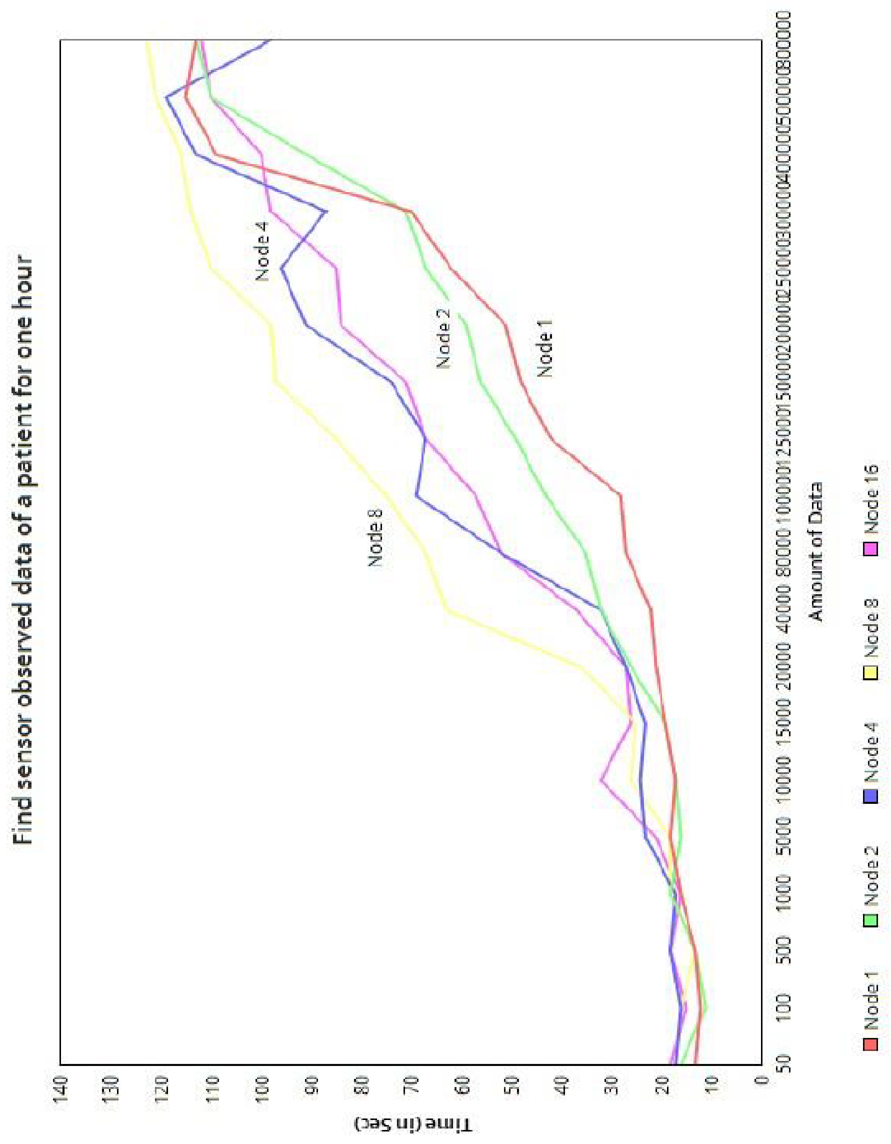
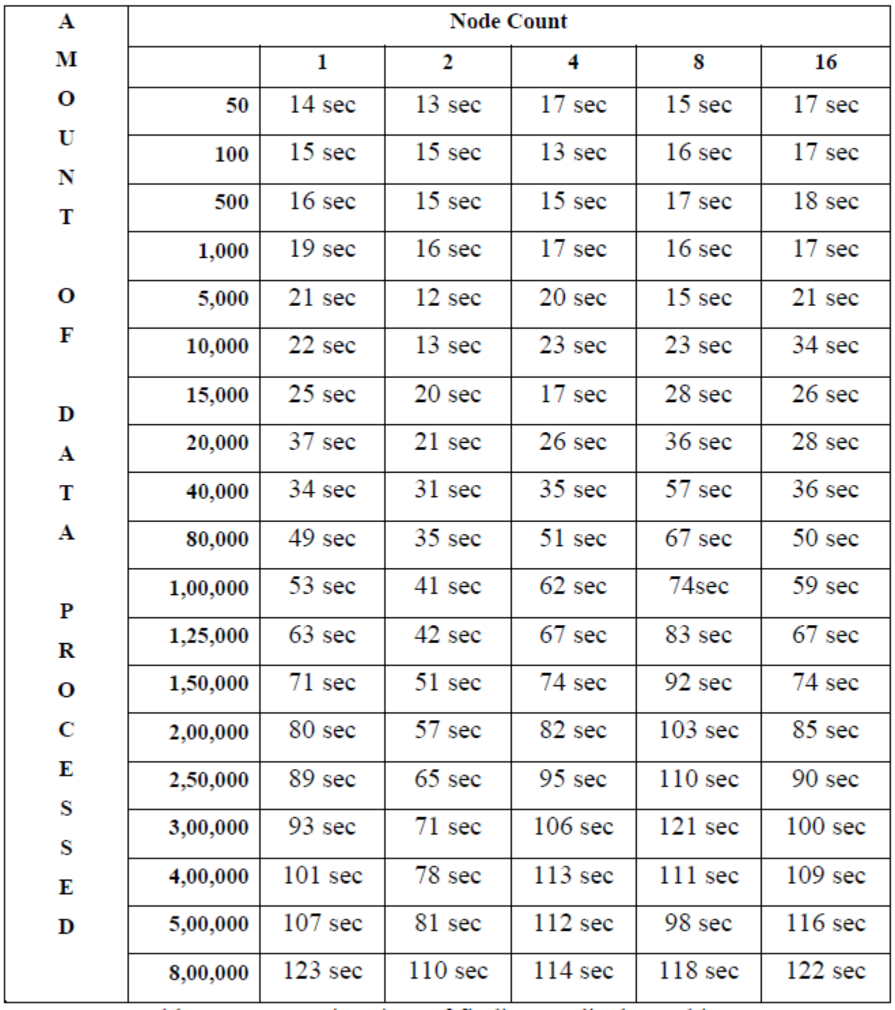


FIGURE 4.10: sensor observed data

Query 6: Treatment case history of a patient with name 'B'.



AMOUNT OF DATA PROCESSED	Node Count				
	1	**2**	**4**	**8**	**16**
50	14 sec	13 sec	17 sec	15 sec	17 sec
100	15 sec	15 sec	13 sec	16 sec	17 sec
500	16 sec	15 sec	15 sec	17 sec	18 sec
1,000	19 sec	16 sec	17 sec	16 sec	17 sec
5,000	21 sec	12 sec	20 sec	15 sec	21 sec
10,000	22 sec	13 sec	23 sec	23 sec	34 sec
15,000	25 sec	20 sec	17 sec	28 sec	26 sec
20,000	37 sec	21 sec	26 sec	36 sec	28 sec
40,000	34 sec	31 sec	35 sec	57 sec	36 sec
80,000	49 sec	35 sec	51 sec	67 sec	50 sec
1,00,000	53 sec	41 sec	62 sec	74sec	59 sec
1,25,000	63 sec	42 sec	67 sec	83 sec	67 sec
1,50,000	71 sec	51 sec	74 sec	92 sec	74 sec
2,00,000	80 sec	57 sec	82 sec	103 sec	85 sec
2,50,000	89 sec	65 sec	95 sec	110 sec	90 sec
3,00,000	93 sec	71 sec	106 sec	121 sec	100 sec
4,00,000	101 sec	78 sec	113 sec	111 sec	109 sec
5,00,000	107 sec	81 sec	112 sec	98 sec	116 sec
8,00,000	123 sec	110 sec	114 sec	118 sec	122 sec

FIGURE 4.11: Treatment case history search

the output is extracted and below graph shows the node search performances

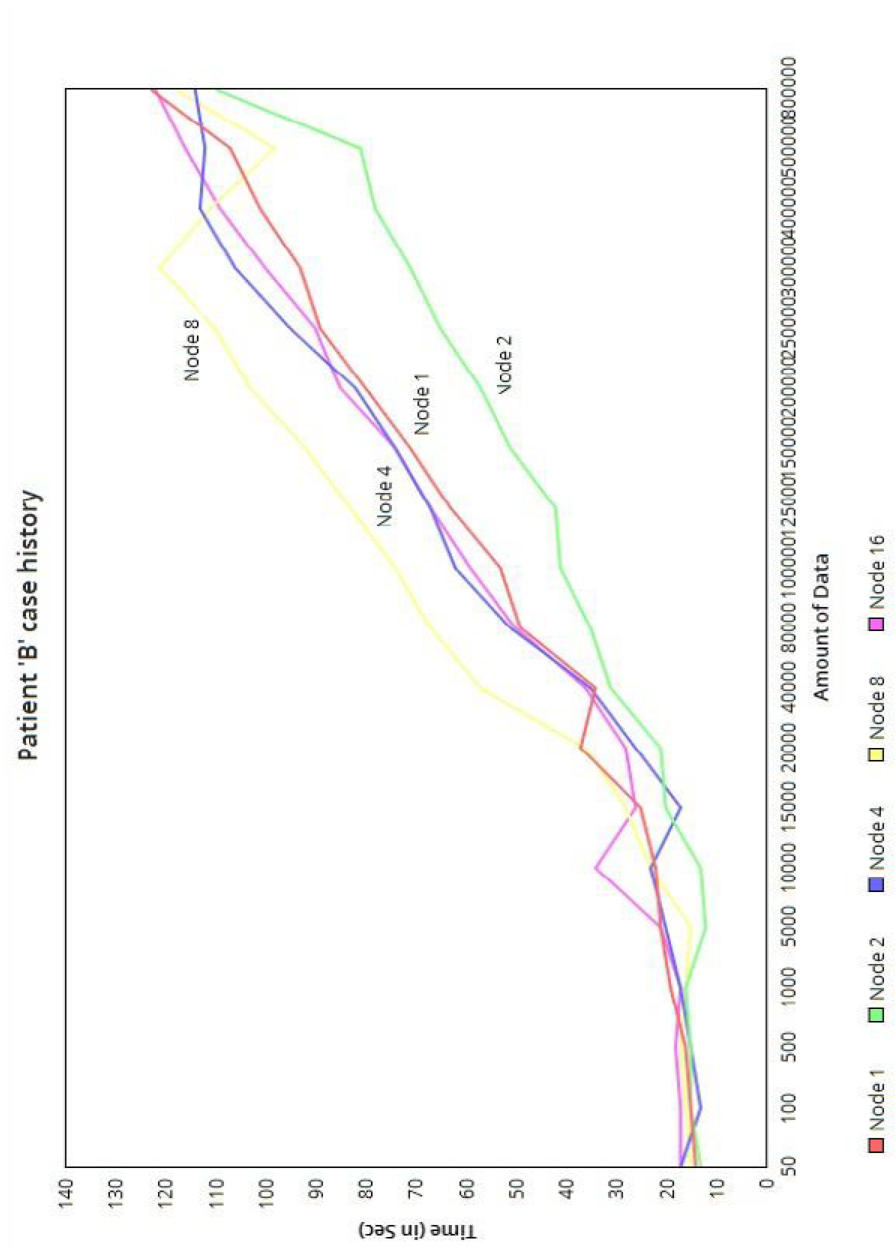


FIGURE 4.12: Treatment case history node search

The above figures the performance bottlenecks observed in Hadoop to process the simple queries with the proposed data model.It can be easily noticed from the plotted graphs, that the computation time increases when the data volume grows gradually. But in case node count increases, for the small volume of data, the computation time is increasing as it need more time to map the data and reduce the data after computation. However when the data volume grows in a rapid scale, the computation time increases gradually with data volume.

4.3 A Study On Performance improvement in Hadoop

The growing need of big data technologies in healthcare and other sector drags lot of attention of researchers for carrying out the exploration for the issues related to big data. The HadoopDB [15] is one of the most relevant work where the advantages of hadoop is allied with that of the relational databases by placing the analogous data in similar nodes with fault tolerance and dynamic scheduling.

Various work has been done like in [16] [17] where a benchmark has been tried to be set to overcome the performance gap between Parallel Databases [18] and Hadoop where as Hadoop++ [19] has been proposed where "trojan" file has been created. Hadoop++ doesn't require any change in the core hadoop architecture as it is not a dynamic approach. This is static approach to rearrange the data which is provided by the user as input file.

Jiang et al. [23] led an exquisite benchmark of different parts of Hadoop's processing pipeline. It was noticed that (among others) indexing and map-side "partition joins" can highly enhance performance of Hadoop. On the contrary, in their extended work, they do not co-place partitioned data fragments. [20] and [21] change the physical layout and replaces HDFS by full fledged relational databases HadoopDB, whereas Hadoop++ invokes indexes and co-partitiones data into raw data files directly.

Cheetah and Hive are two data warehousing solution alternatives for Hadoop, and extend many ideas from parallel databases. But, they neither supports co-placement and exploitation. GridBatch [22] is also an extended work of Hadoop with various new operators and a new file type, which is fragmented by a user-defined partitioning function. GridBatch enables applications to mark files that needs to be co-placed. The findings amalgamet the partitioning and the colocation at the file system level.

In more advanced fragmenting features of parallel database systems [23], such as TeraData [24], IBM DB2 [25], Aster Data [26], tables are co-partitioned, and the query optimizer utilizes the fact to spawn optimal query execution plans. This approach accommodate these ideas to the MapReduce paradigm, while maintaining Hadoop's flexibility as well as dynamicity.

However all the research either extends the work of Hadoop or proposes a system that works statically. This thesis intends to propose a framework that provides a platform for healthcare data analytics and storage platform.

Chapter 5

Proposed Architecture

5.1 Problem Definition

Hadoop has become one of the most advanced and preferable choice among state of the art frameworks for processing and analyzing big data. However, it obviously has some bottlenecks.

The main performance bottleneck [24] is due to unwavering lack of high disk I/O and network bandwidth. The effect of bottlenecks caused by disk environments is more evident than that caused by CPU or memory performance for applications which is data intensive. There are multiple causes for this I/O performance problem bottleneck. The performance gap between processors and I/O systems in a clusters is rapidly amplifying. As an example, processor performance has seen an annual increase of approximately 60% for the last two decades, while the overall performance improvement of disks has been hovering around an annual growth rate of 7% during the same period of time. Second, the heterogeneity of various resources in clusters makes the I/O bottleneck problem [28] even more pronounced.

HDFS always places the first block replica onto the writer node if the node is in the cluster, and on a node with highest proximity otherwise. This scheme makes the cluster very unbalanced in terms of data placement in case the write node does not leverage MapReduce. All the first block replicas would be placed on the writer node. Over a long course of run, this makes the writer node a hot spot. Also, analyzing the MapReduce layer it can be seen that, it tries to execute application copies on cluster nodes that have required data locally available. As it is generally the case that disk I/O is faster than network I/O, moving computation is cheaper than moving data. Since Hadoop has been into its use and prominence for quite a long time, hardware components have evolved since the early stages of Hadoop. The general implementation of HDFS does not take the gap in generation of hardware components into account. Considering disk I/O, an application copy running on a recent disk generation, like Solid State Disks or faster SATA, with twice as much I/O speeds shall be able to transfer, approximately, twice as much as data blocks on an older generation. If the gaps in hardware technologies are given significant weightage, it would surely play an important role in the overall performance of the cluster. This thesis is intended to categorize the shortcomings of Hadoop as of below.

5.1.1 Architectural Drawback

HDFS is not utilized to its full potential due to scheduling delays in the Hadoop architecture that result in cluster nodes waiting for new tasks. Instead of using the disk in a streaming manner, the access pattern is periodic. Further, even when tasks are available for computation, the HDFS client code, particularly for file reads, serializes computation and I/O instead of decoupling and pipelining those operations. Data prefetching is not employed to improve performance, even though the typical MapReduce streaming access pattern is highly predictable.

5.1.2 Restricted Portability

Some performance-enhancing features in the native filesystem are not available in Java in a platform-independent manner. This includes options such as bypassing the filesystem page cache and transferring data directly from disk into user buffers. As such, the HDFS implementation runs less efficiently and has higher processor usage than would otherwise be necessary.

5.1.3 Portability Assumption

HDFS is strictly portable, but its performance is highly dependent on the behavior of underlying software layers, specifically the OS I/O scheduler and native file system allocation algorithm.

These problems basically rely on the architectural point of view from machine but there are also certain issues with hadoop that prematurely happens with the design of workflow and algorithmic implementation. The less number of nodes used for the distribution of fragments over the cluster nodes signifies the lesser degree of concurrency while greater number of nodes increases the degree of concurrency but creates more JobTracker-to-TaskTracker [29] messages creating extra overhead in MRV1.

These shortcoming became a great bottleneck for scalability and performance. IBM highlighted [30] that with this architecture can be scalable to 5000 node and 40000 jobs. However, YARN has been introduced to overcome this problem which can scale to 32000 to 40000 nodes and 26 millions job.

In spite of having this problem there are other problems too that should be resolved to improve Hadoop performance a lot. The big data file which is fragmented into many parts follow a blind partitioning resulting in the random distribution of blocks. The blind partitioning of data is only helpful for faster fragmentation but causes to deploy the MapReduce functions in an unguided manner. This further leads to communication and computation overhead leading towards the poor performance of Hadoop MapReduce model. The fragmentation of big data does not allow the user choice based on key. Besides, the replication is also not used intelligently. The replication factor is only used for recovery and fault tolerance.

5.2 Proposed Scheme Assumptions

In this thesis an architectural framework is proposed to overcome the problem faced in the existing system as discussed in the earlier section. The proposed architecture aims to store big data efficiently and make the analytical query faster. The proposed architecture tries to solve the problem of performance bottleneck and architectural drawbacks thoroughly examined in the existing Hadoop framework. However, the proposed architectural framework is based on some assumptions which is discussed below.

5.2.1 Node Failure

The proposed framework is a collection of cheap commodity cluster nodes which is scalable to hundreds and thousands of nodes. Every node in the cluster is a server machine that serves the requests of clients or end user. All the node are connected via LAN cables and network switches. So it is obvious that any node of the cluster can be failed at any instance of time. Either the network switch or hardware component of the node can face problems. Node failure can be very dangerous in terms of reliability and data availability. It has been investigated that the failure can be of two types. One type is Network failure and the other is individual node failure. It has been also observed that the probability of network failure is very less than that of the node failure. Network failure may lead to multiple node failure which is fatal for the system however not resulting in the data loss.The main aim of using the cheap commodity cluster node is to reduce the cost. High performance in reduced cost with higher scalability makes the application more preferable and effective one. As the failure in the system is evident, the proposed architecture must identify and cope up with these type of failures to make the system more efficient and reliable one.

5.2.2 Read Many Intensive Application

The proposed architecture is very much suitable for the data storage space for healthcare data. The healthcare data are generally historical data which is stored for predictive analytics and disease similarity. Thus the proposed architecture is very much appropriate for more read operation and less write operation. The proposed architectures also does not fit into the scenario where real time data is processed with in-memory computation is done than historical batch data analysis.

5.2.3 Large Dataset

The exponential growth of wearable devices, advanced patient monitoring systems and sensory technologies led to generate gigantic amount of health data which needs to be stored, managed and processed. The proposed architecture assumes to have larger size of data and very capable to handle that amount of data. However smaller dataset is not suitable for the proposed architecture as it causes unnecessary communication and data fragmentation leading towards the poor performance.

5.2.3.1 Portability and Performance

Portability refers to the independence of the platform and support of the heterogeneity. The proposed architecture is easily portable irrespective of the platform. Simple consistency model of the proposed architecture is another feature making the read-write operation consistent enough without affecting the degree of concurrency. The proposed architecture also concentrates on moving the computation to data for cost effectiveness and high performance.

5.3 Architectural Overview

The above mentioned assumptions are general cases for high scale high performance distributed processing. The total view, components and restrictions are discussed in the following sections.

5.3.1 Client-Server Architecture

The proposed framework maintains a server/slave architecture as it gives the flexibility of high scalability of adding or removing nodes. The cluster being the combination of many server nodes, one node is assigned with special responsibility being the server Node and all other nodes are client nodes which follow the instruction comes from the server node. No direct communication is done between the client nodes but every client node can directly communicate to the server node.

5.3.2 Server Node

In general the server node is conceptualized as a high available server with high performance computing power which keeps the client number of nodes in the cluster, node health information, data block information, node status information, file access allowance of a client. This server node is the single point of entry and manipulation point for the cluster and deploying the MapReduce.

- The server nodes keep all the information about client node like the IP addresses of each node, number of client nodes in the cluster, available spaces in the cluster secondary disks, blocks present in the disk.
- It also maintains the metadata of all the blocks stored in the cluster along with the client identity, block number and filename for that block.
- The server node also maintains the information regarding the size of the files and blocks, location of the blocks stored in a particular cluster, block sizes, block numbers, file hierarchy and file permissions for the client.
- There are two files associated with the metadata. These are FState which keeps the complete state of the file and blocks with the name, size and directory structure since the start of the server node and UpdateLog that keeps tracks the modified blocks and timing of the modification.

- The server node keeps track of every change in the file system from renaming to deleting. The server node UpdateLog is updated instantly after every modification operation such as delete, write or rename of the file along with the timestamp.
- The server node also takes care of the replication factor which will be later elaborated in the replication section.
- The server node always keeps all the information in the RAM for instant access and high availability.
- The server node also receives the heartbeat and blockstate report to keep track of the client node healths and state of the blocks

As the server node is the single entry point with lots of responsibilities associated with it, there exists single point of failure leading towards the whole system breakdown.To overcome this single point of failure one secondary node is kept additionally which saves the states of all the information gathered by the server node periodically. The secondary server node keeps itself in sleep mode except the periodic state save operations of the server node. When the server node fails, the secondary node takes the charges of the server node with the manual startup.

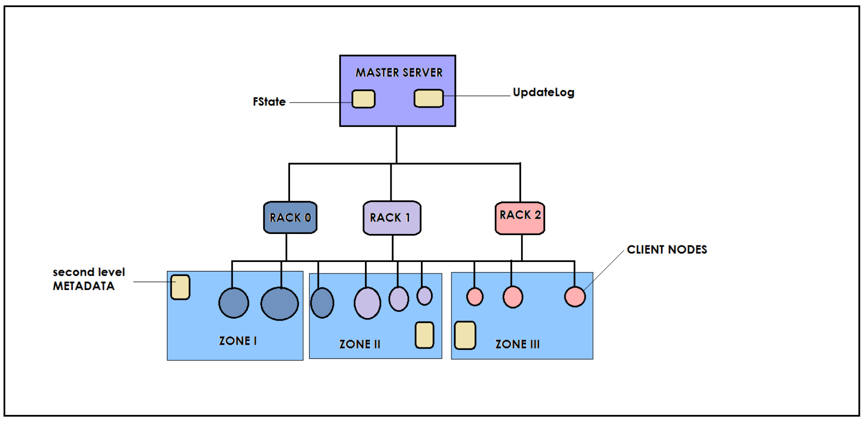


FIGURE 5.1: Server-client in zone

The concept of Zone has been introduced which refers to the set of client nodes that stores all the fragments based on one key whereas the rack refers to the set of nodes connected under one switch which is depicted in the above figure.

5.3.3 Client Node

The client nodes are the main source of storage and computation pool that carry out the storage of all the fragments and their respective replications and the computation also migrates to the client nodes for analytical queries. Client nodes are also the server nodes but their hardware configuration is similar to cheap commodity machines. One client node cannot communicate with other client node directly. The client nodes also have some responsibilities as of

- A client node sends heartbeat messages to the server node to inform that it is alive.
- A client node also performs low level file read and write operations.
- It also receives request of MapReduce function via RPC and pushes back the output of the analytics to the data pipeline.

5.3.4 Horizontal Scalability

The client node is collection of cheap commodity hardware nodes. Vertical scalability refers to the integration of hardware parts for a single node. Vertical scalability is not really a optimal solution in terms of cost effectiveness. In fact vertical scalability lessens the degree of concurrency where as horizontal scalability [31] refers to the addition of nodes in the cluster. Horizontal scalability is very cost effective and can fully scale to hundreds or thousands of nodes with transparency. The proposed architecture prefers to scale horizontally i.e. adding any number of node to the cluster forming a distributed storage pool.

5.3.5 File System

The file system of the proposed architecture is hierarchical and read write operations are supported like other file systems. Directory inside directory can be created by the user in hierarchical manner. Moving, renaming and removing the files is also supported.

The updation or any change in any file stored in the client node are instantly reported to the server node with the timestamp which is in turn recorded in the UpdateLog and FState json file by the server node.

The file system of the proposed architecture never replaces the underlying architecture provided by the OS. The core file system remains intact with its own set of operations. The file system of the proposed architecture is a layer which filters information and passes to the underlying OS thus increasing the ease of maintenance and portability to the other platforms.

5.3.6 Big Data File Source

The enormous chunk of big data files are generally imported from cloud storage and also it can be gathered by web scraping and from any other sources. However, the proposed architecture seeks for URL of the big data to fetch the data and distribute among nodes.

5.3.7 File Fragmentation

Fragmentation of big data refers to partitioning of big data in subsets to distribute them among the Datanodes of cluster. It provides a great advantage of no single point of failure and higher degree of parallel programming. Distributed processing enhances the analytical performance as the degree of parallel processing increases. Horizontal scalability is also effective as reason that storage pool can be increased at any point of time to accommodate any size of the data. However, efficient scheme of fragmentation and optimization in fragment placing techniques are required to improve the system efficiency drastically.

The huge amount of dataset can't be stored in one Datanode. Even if it is possible to store that much amount of data in just one Datanode parallel processing becomes quite impossible resulting in the poor performance for query and analytics. So the importance of fragmenting big data in smaller subsets is explained in the following section.

5.3.7.1 Parallel Processing

The more number of nodes signify the more number of fragments. Distribution of fragments can easily avoid single point of failure and maintaining replication can be easily utilized for parallel processing. The degree of concurrency generates high throughput of the system.

5.3.7.2 Reliability

Distribution of fragments ensures that the whole dataset is spread over the multiple nodes of the cluster so that failure of one node can't result in the loss of whole data. So fragmenting data removes the high probability of single point of failure.

5.3.7.3 Bottleneck Avoidance

Application views are usually subsets of the relations. Therefore, the locality of accesses of applications is not defined on entire relations but on their subsets. Hence it is better to consider subsets of relations as distribution units.

5.3.7.4 Efficiency

Since in hadoop computation migrates to data fragmentation increases the degree of concurrency of processing, query and optimizes the data at a rate of high throughput.

These importance makes the process of fragmentation a much needed step towards efficient big data storage and analytics platform. However, fragmentation should follow certain criteria as of follows.

5.3.7.5 Disjointedness

The fragmentation technique should be efficient enough to partition the data into non overlapping subsets. Non-overlapping subsets are ideal to maintain the consistency for the write operation.

5.3.7.6 Completeness

The fragmentation algorithm must cover the whole dataset. Fragmentation of partial dataset is misleading and may lead to poor incorrect performance.

5.3.7.7 Reconstruction

The fragmentation techniques must follow this criteria of reconstructing the fragments into the whole dataset with intact form. This approach follows accuracy and increases consistency of the system.

The fragmentation techniques must follow the above mentioned approaches to avoid incorrectness of the system due to inefficient fragmentation.

The main problem of the existing system like Hadoop is that it performs a blind partitioning of the big data file without leaving any option for the end user resulting in blind MapReduce deployment and more communication overhead. The proposed framework supports a new approach of key based vertical and horizontal fragmentation and key-less horizontal and vertical fragmentation. Key based fragmentation is referred to the partitioning of big data file based on the attributes in chunk sizes, decided by the end user. By default the buffer size is of 128 MB but the user can decide the size of the buffer based on the business need.

The end user gives input like buffer size, Big data file source (generally URL), array of keys and client node ids in which the user intends to distribute the file fragments. The user fully interact with server node. The end user and the client nodes don't have the direct interaction. The server node then start reading the record based on the key and create buffers for every distinct occurrences of key in the record. The keys are chosen wisely by the end user according to the business need and analytical need. The server node creates buffer for every distinct occurrence of key and push the records into the buffer till it reaches the size of buffer decided by the user otherwise the buffer size remains 128 MB. when the buffer gets filled, it is pushed to that client nodes which is selected by the end user.

The user can choose the big data file to be fragmented by the vertical way or horizontal way. The user can choose as many number of attributes as keys, based on which the file fragments should be clustered upon. If the user can chose to fragment the file with blind partitioning, it is also supported by the proposed architecture. The end user gets the full flexibility of choosing the client nodes with their ids to decide which client node should store which key based fragments. In this manner every fragment holds all of its records based on the same key or attribute and further mapped to the desired client node.

The known mapping of already sorted and clustered fragments are very much helpful for guided deployment of analytical queries with less communication overhead.

Suppose for scenario of a health dataset of 5 city of a country with 5 key attributes like patient_id, patient_name, city, disease and age is chosen to be fragmented based on 'city' among 10 client nodes selected by the end user then assuming an even occurrence of records for each city may lead to accommodating 2 client nodes per city with the awareness of end user. So for city based analytics or query the user need to deploy MapReduce function in only two client nodes for a specific city. Besides the city, other attributes can also be chosen but the sorting clusterization is depicted in Replication section.

5.3.8 Locality of reference

The locality of reference is referred to the phenomenon in which same set of memory is tend to be accessed by a program for a particular period of time. In general the loops and subroutine programs tends to access the same memory location or the nearby address space to increase the cache efficiency. The key based sorted clustering results in the same set of records with respect to key for the selected client nodes. This further implies that MapReduce functions deployed for the analytics based on the key tends to access the same set of client nodes for a particular time period which is termed as 'Locality of Reference' with respect to cluster node access.

5.3.9 Replication

The replication scheme for the proposed architecture is very different with respect to other replication scheme of existing big data storage and processing platform. In general the main purpose of the replication scheme is to recover automatically and quickly from failure and data availability. Replication also helps to enhance the performance based on the selection algorithm of a replica for a particular scenario. Replica placement scheme is thus very important for performance enhancement and data failure recovery.

The number of replicas is termed as Replication Factor. The replication factor of Hadoop and the proposed architecture is by default 3. The replication factor is configurable in Hadoop cluster. Exact copies of the block in hadoop is created to make it fault tolerant.

The proposed architecture uses the replication in very different way. If the number of keys chosen by the end user id less than 3 then the default replication factor is used by the proposed architecture. However, if the number of keys chosen by the user is greater than or equal to 3 then the replication factor of the proposed system is set to be the number of keys. Every replication of the block is not the exact image of the parent block but every replication of block is a key based sorted clusterization fragment.

Suppose for a big data file that is fragmented in 'N' partitions based on the Key K_1, then again that file will be fragmented based on the key K_2 in 'M' partitions where N $\geq$ M or N $\leq$ M. This will continue till K_n where n is the number of key chosen based on which the big data file should be fragmented.

5.3.10 Replica Placement

Replica placement is very important for a distributed system to make it fault tolerant. The client nodes connected in one network switch is defined to be in the same 'Rack'. That means every connection by a network switch is termed as 'Rack'. Various Rack awareness replica placement strategies are proposed for the optimal performance of the analytical queries. However, all the replica placement strategies are based on the general scenario of one replica being the exact image of the other replica. As the replica in the proposed architecture are very different from the existing strategies, replica placement is very important as well as challenging. The client nodes under one network switch form a rack. Similarly a concept of 'Zone' is introduced in the proposed scheme. A Zone is referred to the collection of client nodes storing all the fragments of a big data file based on one key. So for a big data file which is chosen to be fragmented based on the n number of keys will have n Zones. No client node will hold the fragments based on two or more keys. So no two Zones overlap in terms of storing key based fragments. Suppose W is the set of all the client nodes in a cluster under one server node. W_1 is a the set of ids of all the client nodes containing all the fragments based on some key K_1. Similarly W_2 is the set of ids all the client nodes containing all the fragments based on some key K_2 of this continues till the W_n being the set of ids of all the client nodes containing all the fragments based on K_n. According to the proposed scheme, the replicas should be placed in the zone such that

$$W_1 \cap W_2 \cap \cdots\cdots W_n = \phi \tag{5.1}$$

The replica placement according to equation 5.1 helps in failure recovery which is discussed in the failure and recovery section.

This scheme helps the user to choose multiple key based sorted clusterization of the records for big data file much needed for guided deployment of MapReduce function resulting in the decrease of communication overhead and performance improvement.

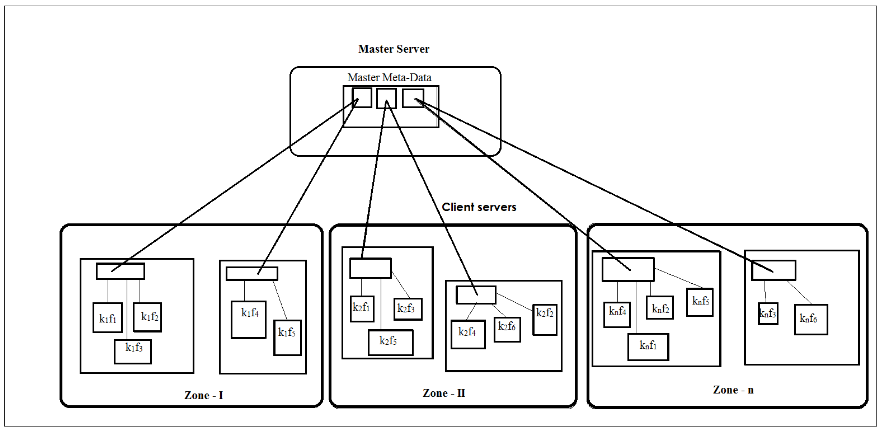


FIGURE 5.2: The concept of ZONE is depicted

5.3.11 Metadata and Indexing

The file size in the big data is expected to be greater than 1TB for which multiple key based fragments will be generated and distributed among the client nodes. It is mandatory to maintain the metadata. In the proposed architecture two level hierarchical metadata is proposed to maintain the mapping information of fragments to node which is stored in server node efficiently. One level of metadata is maintained in the server node which is kept in the RAM for quick access of information. The metadata is a json file. Another level of metadata is kept in the client nodes. The server node indexing only keeps the information of document name, fragment ids and their corresponding client node ids. The raw structure of the server metadata is

- DocumentName = < FileName.*>
- Key = < Distribution AttributeKey>
- Distribution = < KeyID>
- DistributionAttributeKey = < DocumentContent >
- ClusterID = < ClusterIdentifierIDs >
- ClusterIndexRef = < ClusterID + ClusterF ileName >

The tree like structure of metadata helps to identify the fragmentation key and corresponding node ids. Individual client nodes also preserve the metadata which stores the root file name, number block and sizes of the fragments. The raw structure of the client node metadata is

- DocumentName = < ClusterIndexRef >
- Distribution = < blockID + ClusterID + KeyID + FragmentID + >

The conservation of the metadata helps an easy access to the file fragments and their structure to guide the MapReduce function.

5.3.12 RESTful Communication

The proposed schemes follows the RESTful architecture [32]. REST stands for Representational State Transfer protocol, an architecture for designing loosely coupled application over HTTP. The requesting system is allowed to access and manipulate the resources of web by using a set of predefined and uniform rules. Our system being a client-server architecture strictly follows the six constraints defined by the REST architecture explained in following sections.

5.3.12.1 Uniform Interface

The proposed schemes follows the RESTful architecture. The requesting system is allowed to access and manipulate the resources of web by using a set of predefined and uniform rules. Our system being a client-server architecture strictly follows the six constraints defined by the REST architecture explained below-

5.3.12.1.1 Resource-Based Individual resources are identified in requests. For example API/users requested to represent the pages associated with this which further implies from which resource the page should be viewed.

5.3.12.1.2 Manipulation of Resources Through Representations Client has representation of resource which contains enough information to modify or delete the resource on the server, provided, it has permission to do so.Usually user get a user id when user request for a list of users and then use that id to delete or modify that particular user.

5.3.12.1.3 Self-descriptive Messages Each message includes enough information to describe how to process the message so that server can easily analyses the request.

5.3.12.1.4 Hypermedia as the Engine of Application State (HATEOAS)

It needs to include links for each response so that client can discover other resources easily.

5.3.12.2 Statelessness

It means that the necessary state to handle the request is contained within the request itself and server would not store anything related to the session. In REST, the client must include all information for the server to fulfill the request whether as a part of query params, headers or URI. Statelessness enables greater availability since the server does not have to maintain, update or communicate that session state. There is a drawback when the client need to send too much data to the server so it reduces the scope of network optimization and requires more bandwidth.

5.3.12.3 Cacheable

Every response should include whether the response is cacheable or not and for how much duration responses can be cached at the client side. Client will return the data from its cache for any subsequent request and there would be no need to send the request again to the server. A well-managed caching partially or completely eliminates some client–server interactions, further improving availability and performance. But sometime there are chances that user may receive stale data.

5.3.12.4 Client-Server

REST application should have a client-server architecture. A Client is someone who is requesting resources and are not concerned with data storage, which remains internal to each server, and server is someone who holds the resources and are not concerned with the user interface or user state. They can evolve independently. Client doesnot need to know anything about business logic and server doesnot need to know anything about the frontend UI.

5.3.12.5 Layered System

An application architecture needs to be composed of multiple layers. Each layer doesnot know anything about any layer other than that of immediate layer and there can be lot of intermediate servers between client and the end server. Intermediary servers may improve system availability by enabling load-balancing and by providing shared caches.

5.3.12.6 Code On Demand

It is an optional feature. According to this, servers can also provide executable code to the client. The examples of code on demand may include the MapReduce code invoking in the client nodes.
The proposed architecture strictly follows the six constraints specified by the RESTful architecture.

5.3.13 Metadata Disk Failure

The metadata in both the server and client node is very important for the guided MapReduce deployment and the data availability or data perseverance. If the metadata disk failure cannot be recovered then that will cause a system break down and all the information regarding fragments and files will be lost. Hence metadata perseverance is one of the most important factors to consider for making the system fault tolerant.

In the proposed system two level metadata indexing is proposed. For every updation of the metadata it sends one copy to the secondary node which keeps it safe and sends an receiving acknowledgement to the master node. If the master nodes fails then secondary node can be manually started with the last received metadata to take charge of the master node.

5.3.14 Node Failure Detection

The node failure in a distributed cluster of hundred and thousands of cheap commodity machines is very common. The client nodes send heartbeat messages to the master node in every 3 seconds interval to provide the evidence of their aliveness. The heartbeat message sent by the client nodes are nothing but the HTTP request for which the master node sends HTTP response. When the client gets the HTTP response from the server node, it again waits for 3 seconds to repeat the process.

The HTTP request/response message simply implies that the communication channel between client to server is still alive. The server node maintains a statusLog that keeps track of the alive nodes in the system. Whenever the server node gets the HTTP request message from the client node it updates the status of that node as alive.

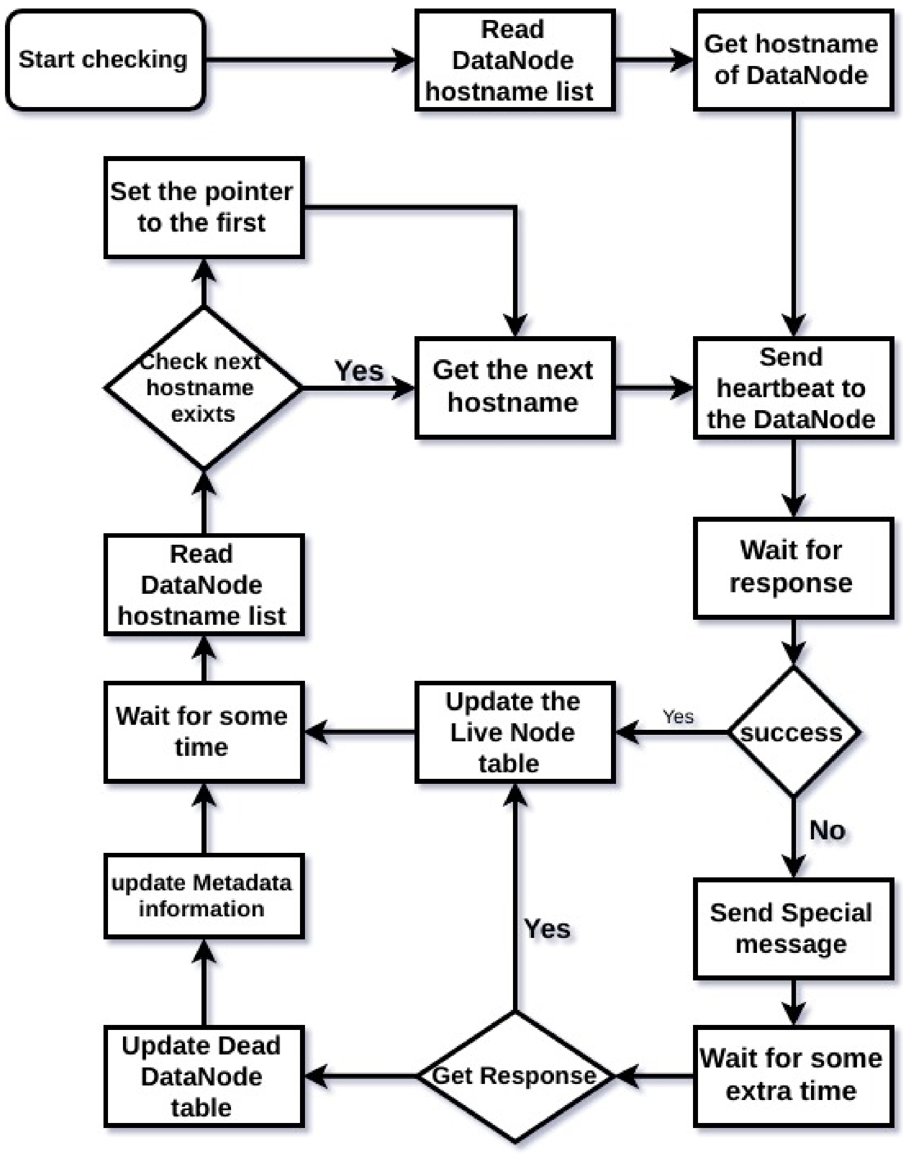

Start checking
Read DataNode hostname list
Get hostname of DataNode
Set the pointer to the first
Check next hostname exixts
Yes
Get the next hostname
Send heartbeat to the DataNode
Read DataNode hostname list
Wait for response
Wait for some time
Update the Live Node table
Yes
success
No
update Metadata information
Yes
Send Special message
Update Dead DataNode table
Get Response
Wait for some extra time

FIGURE 5.3: Flowchart shows the node failure detection

However, failure can occur any time due to network congestion or node failure. The client node expects an immediate response from the server node after sending the HTTP request in every 3 second interval. If the response doesn't come immediate then the client node assumes network congestion and sends another HTTP request packet after waiting 5 seconds. If the response still don't come then the connection between server to that client is lost. Server meanwhile not getting any response for nearly 9 seconds declares the client as dead node and updates the status of the client as dead.

This helps to achieve to keep track of the status of the communication channel's between client and server. The message request from the client doesn't make the server too busy. In general HTTP request uses TCP connection. The client sends the request packet with just one TCP connection and as it is stateless architecture the response packet which comes from the client node to serve node do not use the TCP connection.

5.3.15 Node Failure Recovery

As soon as the node failure detection is discovered, the fragments and metadata information should be recovered immediately for performance improvement. Although the replication of the fragments are maintained in the other nodes, but they are not the exact images of the last fragments. All the fragments of the big data file based on the key forms a Zone and every Zone consists different key based fragments. Let's assume, S is the set of all records stored in the big data file which is to be fragmented in fragments such as $K_1 f_1, K_1 f_2, K_1 f_3, \ldots, K_1 f_n$ partitions. It is obvious that

$$\{K_1 f_1 \cup K_1 f_2 \cup K_1 f_3 \cup \cdots\cdots K_1 f_n\} = S \tag{5.2}$$

So whenever a node failure of a particular Zone occurs, all the fragments of the same big data file from another Zone is accumulated and fragmented according to the key of the failed Zone to be stored in other alive available node immediately. This way the system is capable to recover from failure automatically and quickly. Thus the replication factor is also used very differently not only to make the system fault tolerant but to make the system generating high throughput with guided MapReduce deployment.

5.3.16 Data Loss Probability

The node failure is so evident in the distributed cluster that sometime fatal error or inefficient replica placement or management may lead to data loss. The proposed architecture uses replication factor very differently for performance enhancement. Still there exists a slight amount of data loss probability. The replicas in the proposed scheme are fragmented based on the different key. If one node failure occurs then it is detected immediately and all the information regarding the file fragments are recovered with the zone information. With all the recovered information and zone information, the first approach is to detect the zone and node id which is containing the other replicas of the failed node files. Then all the replicas are accumulated from that retrieved zone to again get back the key based fragments that was lost due to the failure of node. But there is a probability of failing at least one node from every zone containing at least one record occurring in all the failed node. Then that record is impossible to be recovered and data will be permanently lost. But it is evident that the probability of failure of at least one node from every zone containing same set of records that too at the same time is very low and it decreases with the increase of replication factor. Let's assume that probability of failure of a node in a particular time is P and there exists n number of zone from Z_1 to Z_n each containing K number of client nodes.For simplicity K is assumed to be same for every zone but in real scenario it may vary. The data can only be impossible to recover if at least 1 node from every zone fails containing at least same set of records. So the failure of the systems can occur when (at least 1 node failure from zone Z_1) and (at least 1 node failure from zone Z_2) and ..... And (At least 1 node fail from zone Z_n) which further signifies (1 - no of fails) and (1 - no of fails) and (1 - no of fails) ... so on . It can be deduced from binomial distribution that the maximum probability of data loss can be deduced from the above statement is

$$1 - \binom{K}{0} P^0 (1-P)^K \times 1 - \binom{K}{0} P^0 (1-P)^K \times \cdots\cdots n^{th} term \tag{5.3}$$

which further implies the data loss probability as below.

$$\{1 - (1-P)^K\} \times \{1 - (1-P)^K\} \times \cdots\cdots \tag{5.4}$$

which finally can be simplified as

$$\{1 - (1-P)^K\}^n \tag{5.5}$$

From the equation 5.5 it is very much clear that the data loss probability of the proposed system is very low and it decreases drastically with the increase of replication factor or the number of Zones.There exists a trade off between the number of Zones and number of cluster nodes in one Zone.It has been observed that if number of cluster nodes decreases then the data loss probability increases slightly than the scenario when the number of Zone increases and Data loss probability decreases.But both of the scenario increase in the number of zone and number of cluster nodes defines a bare minimum of data loss.

Chapter 6

Implementation

6.1 Implementation of the framework

The proposed system is experimented using 5 node cluster implying a client server architecture from which one node is used for the master server and the rest of the nodes are used as client nodes. Another node is used to act as the secondary node. RESTful architecture is used for communication towards client to server and vice versa. Node js Express version 4.17.0 is used for making the server application. Express is the minimalist web framework of node js that has myriad of HTTP utility methods and middleware. It provides a thin layer of fundamental web application feature.

Node js 10.15.3 LTS version is used. The main reason behind choosing node js is that it is an event-driven JavaScript runtime which is very easy to integrate and designed specially to build scalable network applications.

Fragmentation of file, splitting and key-based fragmentation, metadata indexing is done through python version 3.7.1 as it is very efficient in terms of code readability and reusability.

The frontend of node status and file fragmenting User Interface is built using Bootstrap 3.4.1, HTML5, CSS3 and Electron js Version 4.2.0.

6.2 System Requirement

The master node used in the proposed architecture has Intel Core i7-8700 3.2 GHz 6-Core LGA 1151 Processor with 1TB HDD AND 16GB DDR4 RAM with the client nodes having the configuration of AMD A8-7600 Kaveri Quad-Core 3.8GHz Socket FM2+ Desktop Processor with 4GB DDR4 RAM. The secondary node used in the proposed architecture has same configuration as the client nodes. All the machines are in a same hosting facility attached with D-Link 8 port 10-100 MBPS bandwidth switch. Out of the 1TB of hard disk space, 20% is reserved for OS and application program and 80% for storage and processing.

6.3 Dashboard

The application when starts opens at the default browser automatically on the port number 5000 of the hist machine. The port number can be configurable by the administrator. The dashboard seeks for the big data file source URL for which it locate the file and shows the attribute of the file as a checkbox.

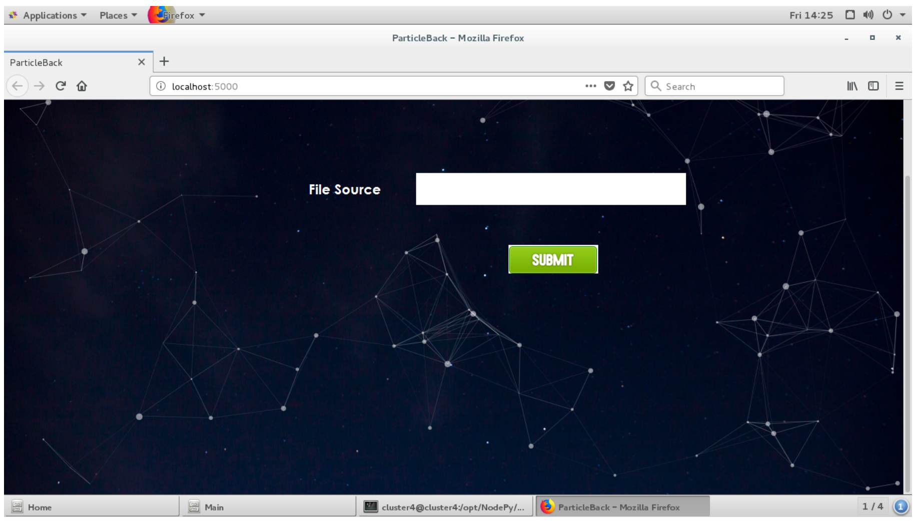
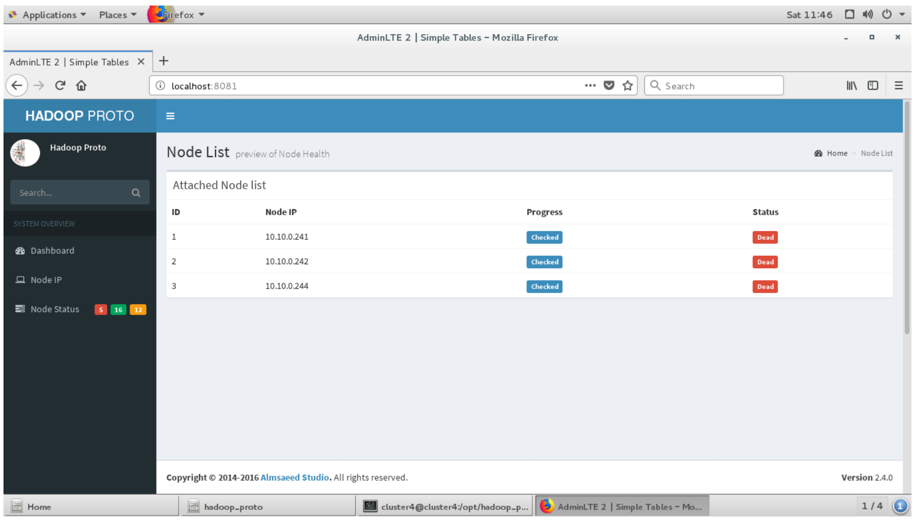


FIGURE 6.1: The dashboard of the proposed architecture

The file source and head of the file is shown in the browser of master server. The user chooses the attributes as the check box to fragment the big data file based on the chosen attribute row wise or column wise. If the number of chosen attribute is less than 3 then by default 3 replicas of each fragment are maintained. If no attributes are checked in the provided checkbox then random partitioning takes place with default replication factor 3 with row wise fragmentation. The front end of the dashboard is made using particle.js V2.0.0. and the server is made with nodejs express framework. The application also supports an automatic opening in the default browser of the host machine.

6.4 Node Health Administration

Node health monitoring dashboards has also been made with bootstrap 4 which shows the status of the nodes as active or failed. The back-end node js scripts helps to monitor the node failure following the procedure stated earlier.



Node List preview of Node Health

Attached Node list

ID	Node IP	Progress	Status
1	10.10.0.241	Checked	Dead
2	10.10.0.242	Checked	Dead
3	10.10.0.244	Checked	Dead

Copyright © 2014-2016 Almsaeed Studio. All rights reserved.   Version 2.4.0

FIGURE 6.2: The dashboard for Node health monitoring

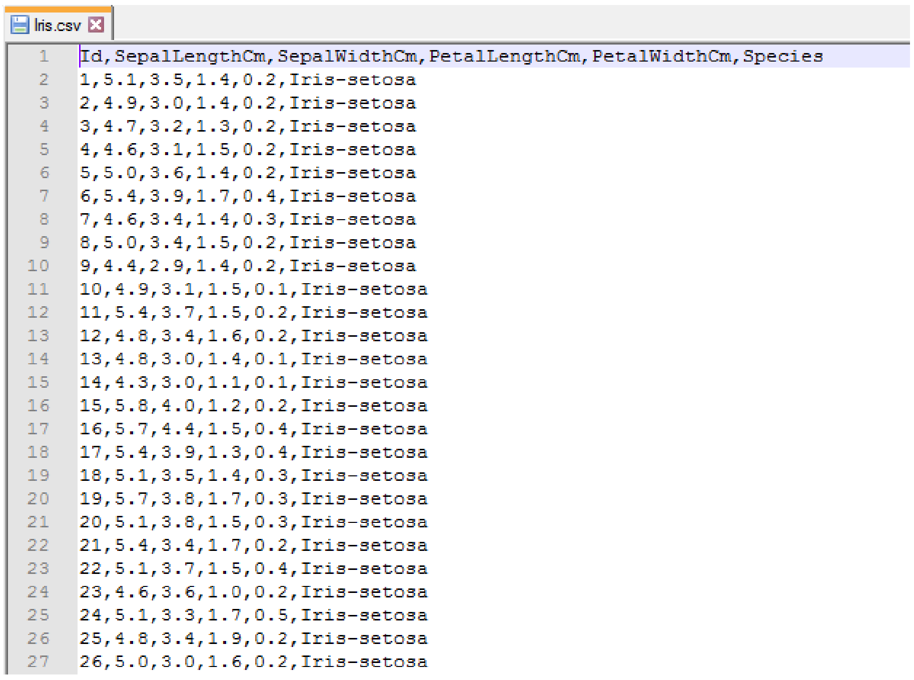
The node health monitoring dashboard shows the total number of nodes, total number of active and dead nodes and task progress.

6.5 File Type

The proposed architecture intends to manage all the file types related to Healthcare data.However, the proposed architecture has been experimented with three types of file discussed below.

6.5.1 CSV File

CSV file are the the tabular structured comma separated plain text data where each row is a new line and each comma is a column. the first line is the attributes and the rest of the lines are it's values. The Iris.csv dataset has been used for row and column wise fragmentation.



```
Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
3,4.7,3.2,1.3,0.2,Iris-setosa
4,4.6,3.1,1.5,0.2,Iris-setosa
5,5.0,3.6,1.4,0.2,Iris-setosa
6,5.4,3.9,1.7,0.4,Iris-setosa
7,4.6,3.4,1.4,0.3,Iris-setosa
8,5.0,3.4,1.5,0.2,Iris-setosa
9,4.4,2.9,1.4,0.2,Iris-setosa
10,4.9,3.1,1.5,0.1,Iris-setosa
11,5.4,3.7,1.5,0.2,Iris-setosa
12,4.8,3.4,1.6,0.2,Iris-setosa
13,4.8,3.0,1.4,0.1,Iris-setosa
14,4.3,3.0,1.1,0.1,Iris-setosa
15,5.8,4.0,1.2,0.2,Iris-setosa
16,5.7,4.4,1.5,0.4,Iris-setosa
17,5.4,3.9,1.3,0.4,Iris-setosa
18,5.1,3.5,1.4,0.3,Iris-setosa
19,5.7,3.8,1.7,0.3,Iris-setosa
20,5.1,3.8,1.5,0.3,Iris-setosa
21,5.4,3.4,1.7,0.2,Iris-setosa
22,5.1,3.7,1.5,0.4,Iris-setosa
23,4.6,3.6,1.0,0.2,Iris-setosa
24,5.1,3.3,1.7,0.5,Iris-setosa
25,4.8,3.4,1.9,0.2,Iris-setosa
26,5.0,3.0,1.6,0.2,Iris-setosa
```

FIGURE 6.3: The parital view of Iries.csv file

6.5.2 TSV File

Tab Separated File is also a tabular structure plain text data format where the values are separated by the tab. each line of the text is a row and every tab separated value is column. Iris.tsv file has been used for the experimentation purpose.

6.5.3 TXT File

Text file of iris dataset used to experiment row wise and column wise fragmentation and storing them on the available cluster nodes.

6.6 File Fragmentation

The experimentation of file fragmentation is implemented using Python language as file handling and management in python is very flexible for its rich amount of library files and functionalities. Binary and text are the basic two file type supported in python languages. a python script is written which is triggered by the node js server to read and load the data in chunks and read it line wise to insert it into a list separated by the separators. At first, the first line of contents are collected to create a Key-Value mapping. the data structure used for this implementation is python dictionary. as soon as the attributes are stuffed in the dictionary some buffers are created and the rest of the contents are buffered in the same. Then the buffer is traversed to collect it's items to sort them based on the user decision of key based, or row or column wise fragmentation. Then they are passed to the active and available client nodes chosen by the administrator. The fragments are handed over to the node js script that is running on the client node server to capture and write as a file with the information of fragment number,fragment key, time stamp. All the fragments of the files gets stored in the "/opt/node/fragments" directory.

Bibliography

[1] Menachemi, N., Collum, T. H. (2011). Benefits and drawbacks of electronic health record systems. Risk management and healthcare policy, 4, 47.

[2] Gartner, Inc. is an American information technology research and advisory firm [online] available : http://en.wikipedia.org/wiki/Gartner

[3] Ray, H. S., Naguri, K., Sil Sen, P., Mukherjee, N. (2016, February). Comparative Study of Query Performance in a Remote Health Framework using Cassandra and Hadoop. In Proceedings of the International Joint Conference on Biomedical Engineering Systems and Technologies (pp. 330-337). SCITEPRESS-Science and Technology Publications, Lda.

[4] Han, J., Haihong, E., Le, G., Du, J. (2011, October). Survey on NoSQL database. In 2011 6th international conference on pervasive computing and applications (pp. 363-366). IEEE.

[5] Lohr, S. (2012). The age of big data. New York Times, 11(2012).

[6] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., ... Warfield, A. (2003, October). Xen and the art of virtualization. In ACM SIGOPS operating systems review (Vol. 37, No. 5, pp. 164-177). ACM.

[7] Driesen, V., Eberlein, P. (2014). U.S. Patent No. 8,924,384. Washington, DC: U.S. Patent and Trademark Office.

[8] Shvachko, K., Kuang, H., Radia, S., Chansler, R. (2010, May). The hadoop distributed file system. In MSST (Vol. 10, pp. 1-10).

[9] Ghemawat, S., Gobioff, H., Leung, S. T. (2003). The Google file system.

[10] Dean, J., Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

[11] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., ... Saha, B. (2013, October). Apache hadoop yarn: Yet another

resource negotiator. In Proceedings of the 4th annual Symposium on Cloud Computing (p. 5). ACM.

[12] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., ... Ghodsi, A. (2016). Apache spark: a unified engine for big data processing. Communications of the ACM, 59(11), 56-65.

[13] Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., Stoica, I. (2014). Graphx: Graph processing in a distributed dataflow framework. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14) (pp. 599-613).

[14] Shvachko, K., Kuang, H., Radia, S., Chansler, R. (2010, May). The hadoop distributed file system. In MSST (Vol. 10, pp. 1-10).

[15] Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., Rasin, A. (2009). HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. Proceedings of the VLDB Endowment, 2(1), 922-933.

[16] Dean, J., Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

[17] Afrati, F. N., Ullman, J. D. (2010, March). Optimizing joins in a mapreduce environment. In Proceedings of the 13th International Conference on Extending Database Technology (pp. 99-110). ACM.

[18] Zhou, J., Bruno, N., Wu, M. C., Larson, P. A., Chaiken, R., Shakib, D. (2012). SCOPE: parallel databases meet MapReduce. The VLDB Journal—The International Journal on Very Large Data Bases, 21(5), 611-636

[19] Dittrich, J., Quiané-Ruiz, J. A., Jindal, A., Kargin, Y., Setty, V., Schad, J. (2010). Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing). Proceedings of the VLDB Endowment, 3(1-2), 515-529.

[20] Abouzeid, Azza, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, and Alexander Rasin. "HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads." Proceedings of the VLDB Endowment 2, no. 1 (2009): 922-933.

[21] Dittrich, Jens, Jorge-Arnulfo Quiané-Ruiz, Alekh Jindal, Yagiz Kargin, Vinay Setty, and Jörg Schad. "Hadoop++: making a yellow elephant run

like a cheetah (without it even noticing)." Proceedings of the VLDB Endowment 3, no. 1-2 (2010): 515-529.

[22] Liu, Huan, and Dan Orban. "Gridbatch: Cloud computing for large-scale data intensive-batch applications." In Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on, pp. 295-305. IEEE, 2008.

[23] Tom White. Hadoop: The Definitive Guide. O'Reilly Media, Inc., 3rd edition, 2009

[24] Al-Kateb, M., Ghazal, A., Crolotte, A., Bhashyam, R., Chimanchode, J., Pakala, S. P. (2013, March). Temporal query processing in Teradata. In Proceedings of the 16th International Conference on Extending Database Technology (pp. 573-578). ACM.

[25] Karlsson, J. S., Lal, A., Leung, C., Pham, T. (2001). IBM DB2 everyplace: A small footprint relational database system. In Proceedings 17th International Conference on Data Engineering (pp. 230-232). IEEE.

[26] Rowan, L. C., Mars, J. C. (2003). Lithologic mapping in the Mountain Pass, California area using advanced spaceborne thermal emission and reflection radiometer (ASTER) data. Remote sensing of Environment, 84(3), 350-366.

[27] GARG, P. (2014). Performance Enhancement of Big Data Processing in Hadoop Map/Reduce (Doctoral dissertation, INDIAN INSTITUTE OF TECHNOLOGY BOMBAY MUMBAI).

[28] Shafer, J., Rixner, S., Cox, A. L. (2010, March). The hadoop distributed filesystem: Balancing portability and performance. In 2010 IEEE International Symposium on Performance Analysis of Systems Software (ISPASS) (pp. 122-133). IEEE.

[29] Manjaly, J. S., Chooralil, V. S. (2013, August). Tasktracker aware scheduling for hadoop mapreduce. In 2013 Third International Conference on Advances in Computing and Communications (pp. 278-281). IEEE.

[30] Lu, L., Jin, H., Shi, X., Fedak, G. (2012, September). Assessing MapReduce for internet computing: a comparison of Hadoop and BitDew-MapReduce. In Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing (pp. 76-84). IEEE Computer Society.

[31] Garcia, D. F., Rodrigo, G., Entrialgo, J., Garcia, J., Garcia, M. (2008, August). Experimental evaluation of horizontal and vertical scalability of cluster-based application servers for transactional workloads. In 8th International Conference on Applied Informatics and Communications (AIC'08) (pp. 29-34).

[32] Feng, X., Shen, J., Fan, Y. (2009, October). REST: An alternative to RPC for Web services architecture. In 2009 First International Conference on Future Information Networks (pp. 7-10). IEEE.