A Novel Approach for Transformation of Relational Database into Column-Oriented Database (HBase)

Thesis submitted in partial fulfillment of the requirements for the award

of degree of

Master of Engineering

in

Information Security

Submitted By

Baljit Kaur

(801233003)

Under the supervision of:

Dr. (Mrs.) Rinkle Rani

Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

July 2014

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, ***"A Novel Approach for Transformation of Relational Database into Column-Oriented Database (HBase)"***, in partial fulfillment of the requirements for the award of degree of Master of Engineering in **Information Security** submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Dr. Rinkle Rani** and refers other researcher's work which are duly listed in the reference section.

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

Baljit Kaur

(Baljit Kaur)
Roll No. 801233003

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

Rinkle 07/07/14

(Dr. Rinkle Rani)
Assistant Professor
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

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

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

i

Acknowledgement

The successful completion of any task would be incomplete without acknowledging the people who made it possible and whose constant guidance and encouragement secured the success.

First of all I wish to acknowledge the benevolence of omnipotent God who gave me strength and courage to overcome all obstacles and showed me the silver lining in the dark clouds. With the profound sense of gratitude and heartiest regard, I express my sincere feelings of indebtedness to my guide **Dr. Rinkle Rani**, Assistant Professor, Computer Science and Engineering Department, Thapar University for her positive attitude, excellent guidance, constant encouragement, keen interest, invaluable co-operation, generous attitude and above all her blessings. She has been a source of inspiration for me.

I am grateful to **Dr. Deepak Garg**, Head of Department and **Ms. Jhilik Bhattacharya**, P.G. Coordinator, Computer Science and Engineering Department, Thapar University for the motivation and inspiration for the completion of this thesis.

I will be failing in my duty if I do not express my gratitude to **Dr. S. K. Mohapatra**, Senior Professor and Dean of Academics Affairs in the University, for making provisions of infrastructure such as library facilities, computer labs equipped with internet facility, immensely useful for the learners to equip themselves with latest in the field.

Last but not the least I would like to express my heartfelt thanks to my Parents and my Friends who with their thought provoking views, veracity and whole hearted co-operation helped me in doing this thesis.

(**Baljit Kaur**)

Roll No. 801233003

Abstract

Relational databases have been serving as a storage platform from several decades. However, with the increase in Science and Technology, we are awashed with data. In order to increase scalability and computational power, applications running on existing databases needs to be migrated. Many enterprises are shifting their data from Relational databases towards the databases that provide high availability, flexibility and scalability. Such databases come under category of NoSQL databases. NoSQL databases are open source, non relational Internet age databases. NoSQL data stores are schema-less in nature. In this report, an algorithm has been proposed that approaches towards migration of schema as well as data from Relational database to HBase. HBase is one of the NoSQL database which uses Column Oriented approach. HBase deals with structured as well as unstructured data, providing faster access to random read/write. The proposed algorithm performs schema transformation from MySQL database to HBase and subsequently migrates the data. The algorithm has been well demonstrated with sample database of BIRT.

Table of Contents

List of Figures

List of Tables

`

1. Introduction

1.1 Introduction to Big Data

Big data is defined as massive amount of data that is difficult to process, capture, analyse and manage by traditional software techniques in reasonable time frame [1]. These data can be complex, vast, diverse and heterogeneous in nature. Sources of these data can be online transactions, video, emails, audio, images, logs, posts, tweets, search queries, science data, health records, sensor data or social media interactions [2]. Figure 1.1 describes the famous 5V's of Big Data. These 5V's are described as [1][3].

I. **Volume:** Nowadays, Enterprises are awashed with data. Volume of data is increasing in such a pace that rather than in tera or peta bytes, Data is now available in exabytes or zettabytes.

II. **Velocity:** Velocity in Big data describes the speed at which data is created, retrieved from various sources, stored and analyzed. Data is streaming in a very unprecedented motion and should be dealt in timely manner.

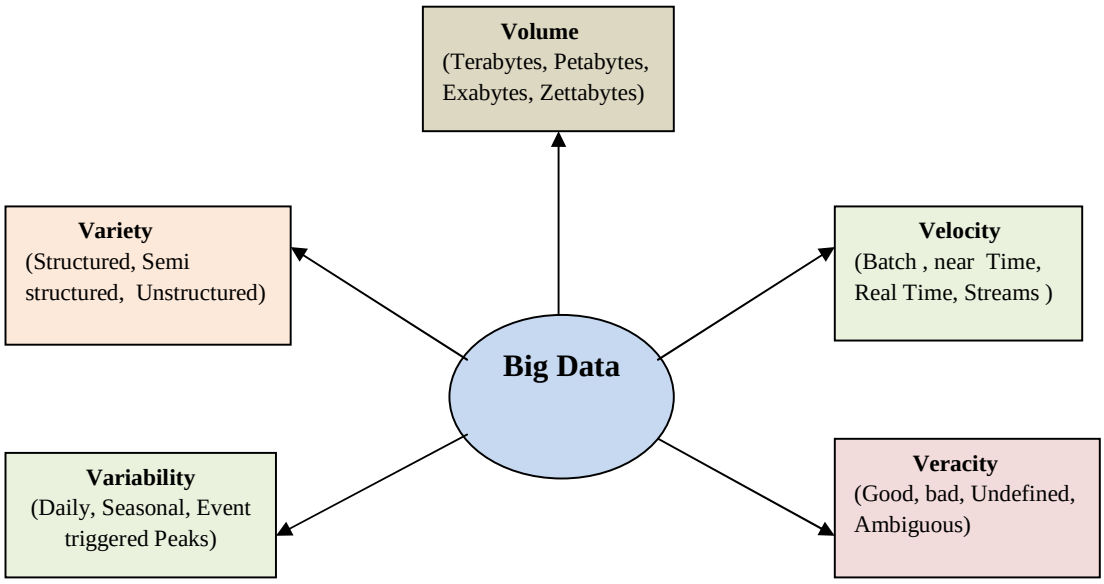


Figure 1.1: 5 V's of Big Data[1]

III. **Variety:** As Big Data comes from various sources, therefore data is not available in single format. Basically, data is divided as: Structured, Unstructured and Semi structured. Structured data is the data which resides within fixed field. Semi structured data is tabular, relational, categorical or meta-data. Unstructured data is text, messages, tweets or posts.

IV. **Variability:** This basically deals with inconsistencies of data flow during periodic peaks. Data loads are challengeable job to maintain during increase in usage ,that causes peak loads due to specific event triggered.

V. **Veracity:** This basically deals with quality and provenance of received data. Data to be processed can be characterized as good, bad, ambiguous or inconsistent [2].

1.1.1 Challenges of Big Data

As the name suggests, Big Data is used to store vast amount of different varieties of data. However, there come many challenges to Big data. Some of the challenges of Big Data are summarized as follows:

I. **Growth of Data:** With the advent in Science and Technology, We are inundated with data. Unstructured data in enterprises tends to grow rapidly, leading to threat of being swamp all.

II. **Processing Power:** With the rapid increase in data, it is not possible for a single powerful computer to process Big Data. Divide and Conquer technique using commoditized hardware is a favorable approach [4].

III. **Storage Issues:** The available storage is not enough to store enormous amount of data coming with high velocity, variety and volume. Retrieving and storing massive volume of data needs huge storage space and resources [4].

IV. **Skill Requirement:** Since Big Data is new emerging technology that entices youth as well as enterprises to develop new skills. Proper training in terms of technical, research and analytical approach is needed for this field [3] .

V. **Privacy and Security Issues:** It is considered as most important issue as it leads to security breaches of data stored by insiders or outsiders. Many times, information regarding people are collected without any permission or consent, just to add value to organization [5].

1.1.2 Importance of Big Data

Big data provides great opportunities to enterprises that are dealing with storage and analysis of large pool of industry data. Few of the importance of Big Data is as follows:

I. **Digital Marketing Optimization:** It basically includes collection and analysis of digital data in order to measure optimized web usage. It also analyses web click streams of users [3].

II. **Log Storage in IT Industries:** IT industries needs to keep record of large amount of log data, even if they need that data very rarely. The log data of any enterprise keep on increasing by time. Changes may also occur due to hardware and software updates. Big data not only process, manage this raw, semi structured, huge data but also helpful in storing it for a long time. Thus Big Data is important for long term data retention [3].

III. **Sensor Data:** Managing and Mining of sensor data is a challenging job. Due to lack of physical storage and techniques, Organizations are making use of only small portion of data for analysis. Sensor data is basically data at rest and data in motion. Therefore, huge massive data needs to be analyzed for safety and efficiency purpose [3].

IV. **Social Media:** Big data is most in use for social media and maintaining customer relationship [4]. Analyzing customer's reviews about product help business organizations to understand their market reputation and competitors. Analyzing a large record of users accessing sites, calculating number of sites accessed by different users gives an idea of which sites gets maximum hits, helpful to upload advertisement having maximum hits by users.

V. **Risk Analysis:** Financials Organizations needs to process large amount of data in order to calculate risks. There is large amount of data which is still underutilized, needs adequate amount of processing and integration to be done to analyze Risk patterns [3].

1.2 No SQL Databases

As Big data is difficult to handle, store and manage via traditional Relational data stores. Therefore, in order to process vast amount of real time heterogeneous data, a new

solution called NoSQL is developed. NoSQL is one of the promising technology that focus on providing highly concurrent real time requests. NoSQL databases are generally described as open source non relational databases. Sometimes, they are also referred to as Not only SQL, i.e. the databases that doesn't offer SQL interface [6]. Unlike relational databases, NoSQL databases are flexible in nature, i.e. they don't require any fixed or static schema [7]. NoSQL solutions have been emerged as a technology for harnessing unstructured real time data and providing high availability [8].

1.2.1 Classifications of NoSQL databases

NoSQL databases are broadly classified into four classes: Key-Value databases, Column-Oriented databases, Document databases and Graph databases [9]. Each of these data stores deals with different types of data and have different applications. Table 1.1 describes the taxonomy of NoSQL databases.

I. Key-Value Databases

The simplest data store, in which data is represented as collection of 'Key-Value' pairs. Basically, the fundamental data model comprises of associative array of entries [6]. This data store allows to store flexible and schema-less data, by allowing storage of arbitrary data which is being indexed by unique key for retrieval of values associated with it [8]. Visualizing it with relational databases, it can be considered as multiple rows having two columns- key and value. The key can be synthetic or auto generated whereas the values can be String, JSON (JavaScript Object Notation) and BLOB (Binary Large Object). Generally, the key length is limited to a certain number of bytes while there is fewer limits on length of values to be stored. Figure 1.2 depicts basic representation of Key-Value Data stores.

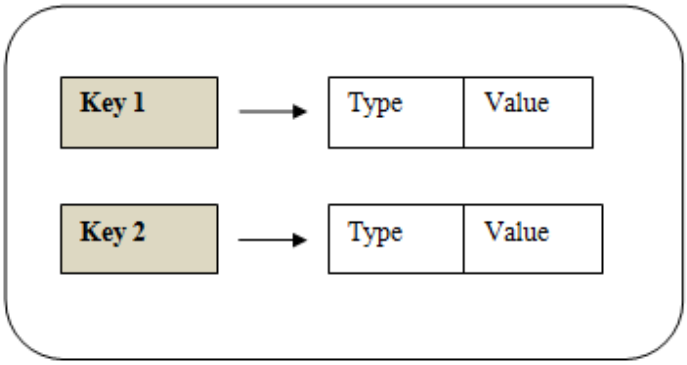


Figure 1.2: Representation of Key-Value Databases

Key-Value stores offer high scalability, strong consistency, high query Execution speed, lesser query execution time. The Key-Value model can be further extended to an ordered key-value model in which keys are unique and ordered. In ordered key-value data stores, keys are sorted by each byte in order maintaining lexicographic order. This add-on yields an effective abstraction, allows efficient processing of key ranges and enables efficient storage and retrieval. Few popular Key-Value databases are explained as [9]

- **Amazon's Dynamo**

Like Amazon Simple Storage Service or Amazon S3, Amazon Dynamo is also one of the data store used at Amazon. It is highly available distributed database with a simple key-value interface storing values as BLOBs. In order to achieve scalability and availability, data partitioning and replication is done using consistent hashing. Versioning of objects stored in partitions among nodes are done. Amazon Dynamo maintains consistency among replicas during updates using quorum-like mechanism and a decentralized replica synchronization protocol. Due to its high availability for writes, Dynamo is said as "always writeable". Many emerging key value stores are greatly inspired from Dynamo and are considered as descendants of Dynamo [10].

- **Project Voldemort**

Project Voldemort is an open source, distributed Key-Value data store. It is written in java and was initially developed for LinkedIn. It is still used at LinkedIn for its high scalability storage feature. Project Voldemort is available under the Apache 2.0 license, providing Thrift API to access the database. Data in Voldemort gets automatically replicated over a cluster of nodes. Also automatic partitioning is done to have subset of data for each server. In addition to storing only simple scalar values, both keys and values in Voldemort could be of lists and maps. Project Voldemort uses derivative of MVCC (Multi-Version Concurrency Control). If comparing it to relational databases, it doesn't attempt to satisfy arbitrary relations, i.e. constraints on Foreign keys are not possible. Also Voldemort doesn't possess complex query capabilities [11].

- **Redis**

Redis is a relatively new open source BSD licensed Key-Value data store. It is written in ANSI C. In Redis, data is organized in the form of key-value pairs, where values doesn't store only bytes rather it could be lists and sets. The whole document is loaded into main

memory, where all the operations are performed. So performance factor is high due to memory operations. As the data is loaded to main memory, so the main downside of using Redis is that amount of data to store depends on size of main memory [8]. Redis adopts master slave architecture. Redis shell is used to have command line interaction with Redis [12].

- **Memcached and MemcacheDB**

Memcached is a popular, powerful, distributed, high performance and fast memory-caching system. It is often used to stimulate large dynamic web applications by reducing their database load. Memcached is a Key-Value store that makes use of map/dictionary API. A hash table is distributed across multiple machines for faster storage and retrieval. In order to have persistent storage, it is conformed by MemcacheDB. As Memcached doesn't support replication, so MemcacheDB make use of BerkeleyDB as backend storage [13].

- **Tokyo Cabinet and Tokyo Tyrant**

Tokyo Cabinet is a key-value data store, developed in Japan. It was developed for famous social networking site 'mixi' by Mikio Hirabayashi [8]. Tokyo Cabinet is a library of routines in which key and values can be of variable length. Key and value can be any binary or character string data. Data can be stored in three formats - hash tables, fixed length array and B+ tree indexes. Tokyo Tyrant is a used to provide remote access to the data stored in Tokyo Cabinet via HTTP and Memcached protocol. This results in improved usability, space and time efficiency. Both Tokyo Cabinet and Tokyo Tyrant written in C language. In order to achieve good compression ratios, LZW Compression algorithm is used by Tokyo Cabinet to compress the pages [14].

II. Column-Oriented Databases

Column-Oriented data stores are designed for processing and storing of vast huge amount of heterogeneous and sparse nature data, when there is a need to have faster access to random read/writes. The data model comprises of rowkey and Columns [15]. Corresponding to unique rowkey, data in Column-Oriented databases are stored by columns [8]. Values belonging to single column are stored contiguously. Columns of

similar data are stored together in Column Family [6]. Figure 1.3 illustrates basic representation of Column-Oriented data stores. Column-Oriented databases also provides versioning feature. It allows storage of multiple versions corresponding to each column. Column-Oriented databases are well suited for OLAP (Online Analytical Processing) layouts. Some popular open source Column-Oriented databases are:

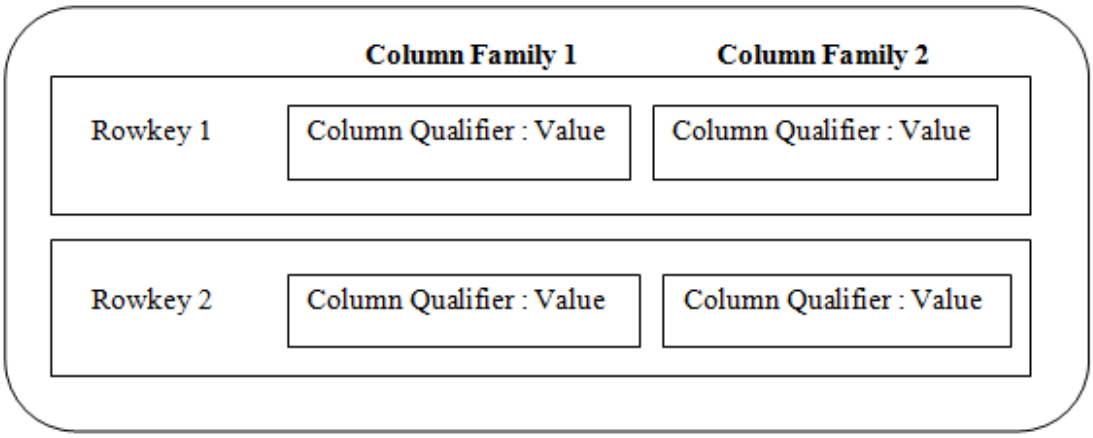


Figure 1.3: Representation of Column-Oriented Databases

- **Google's Big Table**

The development of Big Table was started in 2004, making use of Google File System in order to have persistence of its data and log files. Big table can be described as sparse, distributed storage system, designed to scale vast amount of structured data which is otherwise difficult to handle by traditional Softwares. Big Table was designed to achieve wide availability, distributed storage, high availability and high performance. The data structure of Big Table is collection of key-value pairs. Big Table consists of column families. Column families consist of rows. Rows consist of column with key-value pairs. Timestamp are used to version the value of each column key [16]. Big Table is used in number of Google projects such as web indexing, Google Maps, Google Earth, Google Finance, Orkut, Youtube, Gmail and Google Docs. Many emerging column-oriented databases have been patterned after Big Table.

- **HBase**

HBase is an open source top level project of Apache Software Foundation. Also, Facebook's messaging platform implements HBase as their storage infrastructure.

Basically, HBase is data repository for big data. HBase is described as a distributed, scalable, sparse, multidimensional, and Column-Oriented No-SQL database, which acts as a platform to store and manage massive structured and un-structured data. Thus HBase is key-value database possessing fault tolerance capability. HBase is preferred when there is a need to have random read/write access to real time data [17]. Data in HBase is logically stored in the form of rows and columns. Rows are sorted lexicographically based on Rowkey. Column is described as 'Column Family: Column Qualifier'. Column Family groups the data in a row. Data belonging to same Column Family are physically stored together. HBase table's cells are versioned. HBase auto assigns timestamp at the time of insertion [18].

- **Hypertable**

Originally developed by Zvents Inc., Hypertable is described as open source, scalable and efficient Column-Oriented database [8]. It is sponsored by Baidu since 2009, which is china's leading web services company. Hypertable requires an underlying distributed file system such as Hadoop HDFS or cloud KFS. It is written in C++ language supporting SQL like query language called HQL [7]. Similar to HBase, Hypertable uses concept of arbitrary number of column qualifiers in a column Family. Also timestamp is done to provide versioning of data. Replication and Partitioning of tables are done by ranges of row keys [19].

- **Cassandra**

Cassandra is scalable, flexible, distributed storage system and an open source by Facebook [8]. It is designed by Facebook engineer Prashant Malik and Dynamo's engineer Avinash Lakshman. Thus, it is basically a confluence of Big Table and Dynamo, implementing structured data model of Big Table and consistent and decentralized storage model of Dynamo [7]. The data model of Cassandra table consists of rowkey, Column Family, Supercolumns and Columns. In addition to Column Families, concepts of Super columns have been added to provide another level of grouping. A Column Family either contains Supercolumns or Columns. A supercolumn in turn, contains columns. Like Columns, Super columns can also be added dynamically to corresponding Column Family. Cassandra is written in JAVA Language and Thrift API framework has been used to create Facebook client interfaces. Besides Facebook,

Cassandra has been embraced by various companies like Rackspace, Twitter and Dig [20].

III. Document Databases

A Document-Oriented database is designed to process, store and retrieve document oriented data. All the data is stored in the form of documents. The format of stored document can be XML, YAML, JSON and BSON [8]. Different Documents are grouped together forming Collections. The basic representation of Document based data stores is illustrated in Figure 1.4. Document in the database is addressed and represented via a unique key. The main feature of Document data stores is that a document need not to adhere static schema, means documents can have different number of fields. Comparing it with Relational database, documents inside the collection is same as records whereas collection corresponds to relational table. Also, in relational databases records of same table contains same number of fields, whereas in document databases, documents of same collection can have different number of fields [9]. The popular Document databases are explained as:

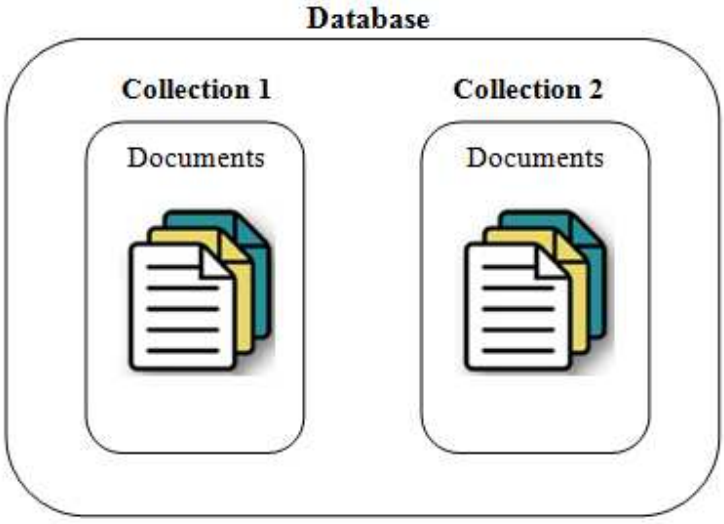


Figure 1.4: Representation of Document-Oriented Databases

- **Amazon SimpleSB**

Amazon SimpleDB is a Document database which provides high availability, scalability, and flexibility to store and query information using web services requests. It is used as part of Amazon's proprietary cloud providing cloud offering, in concert with Amazon Elastic Compute Cloud (EC2) and Amazon S3. Thus, Amazon SimpleDB is used as a

complement EC2 and S3. It was announced in 2007. As the name suggests, it is the simplest document data store. SimpleDB provides eventual consistency, rather than transactional consistency. SimpleDB doesn't provide automatic partitioning, but still supports partition tolerance feature [21].

- **CouchDB**

CouchDB is a document database, generally referred to as “Cluster of unreliable commodity hardware” database. It has been an Apache project since early 2008 and is written in Erlang [7]. It stores the data with JSON (Java Script Object Notation) format in order to have concurrent access, documents are versioned using SequenceID, thus implementing MVCC (Multi-Version Concurrency Control) [22]. The structure of stored data is illustrated using 'Views'. Views are broadly of two types- Temporary and Permanent views. Temporary views are loaded into memory whereas Permanent views are stored on disk. Each view in CouchDB is constructed in JavaScript; implementing Map/Reduce functions to process stored data and returning the results. CouchDB documents are accessible via a RESTful HTTP interface which also allows read and update of documents [8].

- **MongoDB**

MongoDB is a schema-free, open source document database initiated by 10gen company providing professional services around MongoDB. Documents in MongoDB are grouped together in the form of collections. MongoDB stores documents in the format of BSON which is binary format of JSON [8]. BSON supports String, double, Boolean, Date, Array and Object data types. As MongoDB is schema-less database, so it is easier to insert new fields and made updations to existing one. MongoDB supports automatic sharding in order to distribute collection over multiple nodes [9]. MongoDB supports master-slave architecture, where master nodes are replicated over slave nodes for backups and read purposes [23].

- **Terrastore**

Terrastore is a modern document store which is built on the Terracotta. Terracotta is an industry-proven and clustering technology. It is written in Java and is available under Apache 2.0 license. Terracotta is designed to support advanced scalability and elasticity features, but not at the cost of consistency [24]. It is accessed universally via HTTP

protocol. As Terrastore follows schema less model, therefore documents are added in JSON format without predefining the schema. It supports automatic partitioning of data over cluster of machines. Terrastore basically involves MongoDB's consistency feature and CouchDB's Map/Reduce functionality.

IV. Graph Databases

Graph Databases is a network structure model making use of nodes and edges to store and manage data. Edges in graph database are used to connect two nodes. However, representing relationship between nodes. Both nodes and edges have their own properties representing real data associated with them. Also relationship representation is direction-oriented, associating direction to the edges add meaning to the relationships [6]. The basic representation of Graph Databases is described in Figure 1.5. Graph Databases provide index-free adjacency storage system, which results in faster processing rates for finding relationships. This is because Graph based storage doesn't follow index look-ups terminology, in fact each node points directly to the nodes adjacent to it [9]. Graph Databases are best suited for the applications that need interconnection between data. Social networks and Web-Site link structures are best candidates for graph data stores.

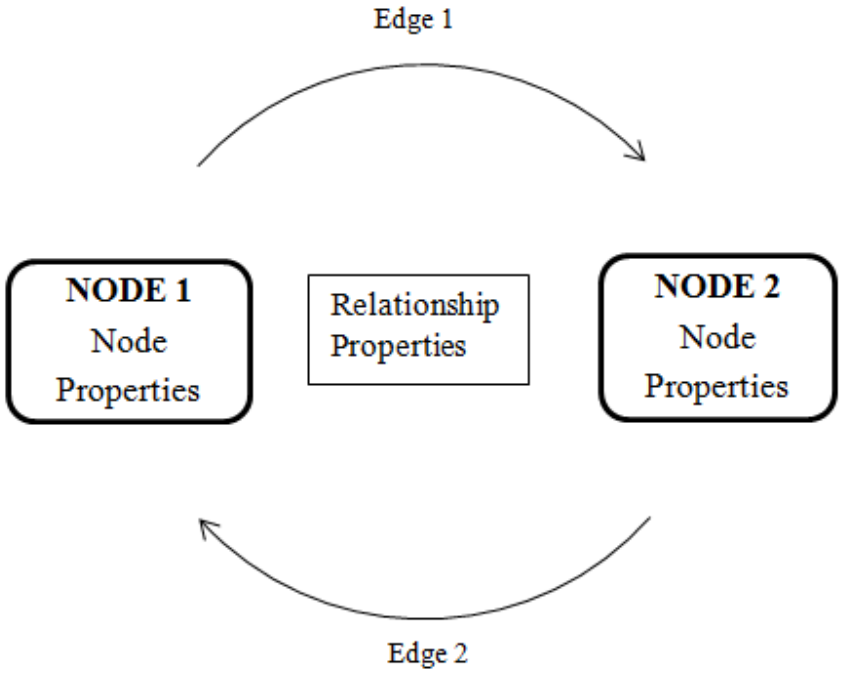


Figure 1.5: Representation of Graph Databases

The fascinating feature about Graph Databases is that with the increase in size of data in Graph data stores, the performance doesn't degrade. Some popular Graph Databases are explained as follows:

- **Neo4j**

Neo4j is one of the popular, highly scalable and robust open source graph data store which is ACID complaint [25]. Supported by Neo technology, it is written in Java language. It is suited for deployment for enterprises, where high availability clustering is of more concern. It specifies feature of business responsiveness providing visual node-link structure and meaning. Neo4j is best suited for representation of hierarchical relations. Cypher query language is often used to have interaction with Neo4j. Also, Neo4j make use of graphical web interfaces. Thus, the status of Neo4j data store can be monitored by

http://localhost:7474

- **OrientDB**

OrientDB is a open source database comprising features of both document and Graph databases. Data in OrientDB is stored in the form of documents and relationships are managed as in Graph based data stores. It is written in Java language. A new indexing algorithm MVRB-Tree is used by OrientDB, which is a blend of Red-Black tree and B+ tree [26]. It is a NoSQL database, but supports a subset of SQL as query Language to handle relationships without using joins.

- **Titan**

Titan is one of a highly scalable graph database, open source with Apache 2 license. It is designed to store and query graphs having billions of nodes and edges distributed over cluster of machines. Titan is capable of supporting large number of concurrent users [27]. It also supports data distribution and replication over the nodes.

- **AllegroGraph**

AllegroGraph is a recent, open source RDF and Graph based Database supporting SPARQL and RDFS++. Data and Meta data in AllegroGraph are stored in the form of Triples. It is basically designed to accelerate load and query speed. It provides dynamic and automatic indexing, also favoring triple level security [28].

Table 1.1: NoSQL Taxonomy

Category	Example Data Stores
Key-Value Databases	▪ Voldemort ▪ Redis ▪ Memcached ▪ Tokyo Cabinet
Column-Oriented Databases	▪ Google's Big Table ▪ Hypertable ▪ HBase ▪ Cassandra
Document Databases	▪ Amazon SimpleDB ▪ CouchDB ▪ MongoDB ▪ Terrastore
Graph Databases	▪ Neo4j ▪ OrientDB ▪ Titan ▪ AllegroGraph

1.3 HBase

HBase is a Column-Oriented No-SQL database of Hadoop, an Apache open-source project. HBase is built on the top of HDFS, i.e. HBase files are stored internally in HDFS. HBase provides a way of storing large quantity of data, providing the feature of fault tolerance. It is used to store and process massive amount of real time data.

1.3.1 Data Model of HBase

HBase Tables are considered as top level structure for storing data [29]. Data is logically stored in Rows and Columns in HBase Tables. In HBase, Table names and Column Families are composed of Strings and characters. Whereas Rowkey and Column

Qualifiers don't possess any data types and are treated as byte[]. A cell in HBase table is uniquely identified by the combination of Row key, Column key and Timestamp. i.e.

$$\{\text{Row, Column, Timestamp}\} \rightarrow \text{Cell}$$

Each of these data model elements are explained as:

1) Row

- Data in tables is stored according to its row.
- Each row is uniquely identified by its Rowkey, i.e. Rows of table are sorted lexicographically based on row key.
- Row Key can be any arbitrary array of bits.
- Row key is compared on binary level, byte to byte from left to right.
- Two types of operations can be applied on Row Key-
    I. **Single Row Operations:-** In order to perform Single Row operations on HBase table, Put, Get and Scan operations are used. Get and Scan operations are used to read values from table. Get operation is used to retrieve back single row whereas Scan operation can retrieve multiple rows. However, Put operation is used to insert data values in HBase Table. Put operation is considered as compliment of Get operation.
    II. **MultiRow Operations:-** For MultiRow operations, Scan and Multiput operations are used.

2) Column

- In HBase, Columns are grouped together as Column Family.
- Basically, Column can be addressed as-

Column = Column Family: Column Qualifier

I. **Column Family**

- Column Family groups the data in a row.
- Data belonging to same Column Family are physically stored together. It is considered as separation between HBase tables.
- Each row of HBase Table belongs to same Column family.

- It is mandatory to have at least one Column Family for each HBase table [29]. Also it is always taken care that number of Column Families should be reasonable, whereas Column Qualifiers can be as much .
- Column Families are defined when table is created, making any alteration to it afterwards, is a tedious job.

II. **Column Qualifier**

- Column Qualifier is used to address the data within a Column Family. They are added to existing Column Family of table [30].
- There can be arbitrary number of Column Qualifiers in a Column Family.
- Column Qualifiers are dynamic, so they need not to be predefined.
- Column Qualifiers can vary between rows.
- Also HBase is schema-less database, so there is no need to assign any data type to Column Qualifier. In fact, it is always treated as a byte[].

3) Timestamp

- HBase table's cells are versioned.
- HBase auto assigns timestamp at the time of insertion.
- It is possible to have several versions of column for same row key.
- The default value of versioning is 3.
- Data type for versioning is long.

4) Cell

- Corresponding to Column Family and Column Qualifier, a value is tagged.
- Each stored data in cell is referred to as cell value.

Therefore, in order to access to data value from table [18]:

{Table name, Rowkey, Column family, Column qualifier, Timestamp} → *Value*

Figure 1.6 illustrates the basic data model of HBase representing Rowkey, Column Families and Column Qualifiers.

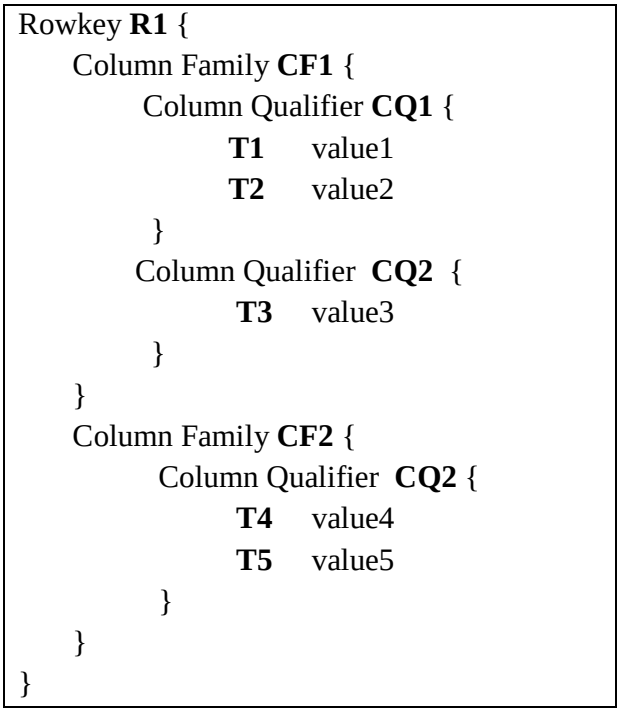


Figure1.6: HBase Logical Data Model

2. Literature Review

2.1 Big Data

Big data is described as huge massive amount of data which is complex, diverse and heterogeneous in nature [1]. These data is difficult to process, analyze and store by traditional systems [3].

According to review conducted on Big data [1], It is studied that nowadays we are awashed with digital data. Till 2003, 5 exabytes of data was created, but today this figure can be achieved in 2 days. Volume of data is increasing in such a pace that it would reach to 8 zettabytes in the coming Year. Currently, In order to store to all world's data, it would requires billion of powerful computers. A vast variety of semi structured or unstructured data is created by social media. 10 billion text messages are sent by mobile subscribers daily. Velocity with which videos, audios, tweets, posts, emails, social interactions are created is really unprecedented. Due to increase in massive amount of data, it is estimated that information would increase by 50 times in the coming 10 years. Big data have its importance in many fields like storing logs in IT industries, storing and analyzing disease pattern in Healthcare, Digital data optimization, Social media interactions, Demand forecasting and Risk reduction in financial institutions. However security and privacy is considered as main issue for big data. So, it needs to be tackled by implementing framework which includes authentication or cryptographic secure system [5].

2.2 Hadoop

Hadoop is considered as core platform for structuring Big Data. It is a heterogeneous open source framework hosted by Apache Organization [1]. It is inspired from Google File System, includes programming model to perform parallel computation and data storage providing feature of scalability and fault tolerance [31]. Hadoop follows scale-up policy rather than scale out. i.e. rather than having large powerful system for processing huge amount of data, Hadoop uses reasonable price commodity hardware where data is distributed across the daemons for processing [32].

2.2.1 History of Hadoop

Exponential growth of data was the main reason for growth of Hadoop Framework. The evolution of Hadoop is explained as follows [31][32]:

- It is a java based platform, designed by Doug Cutting [4].
- Hadoop had its origins from Apache Nutch, which was a subproject of text search library named Apache Lucene. Doug Cutting was creator of Apache's Nutch project too.
- Google published their papers on Google File System and MapReduce framework in 2003 and 2004 respectively. Nutch team started implementing these frameworks in Nutch leading to development of an independent and open source project named Hadoop.
- The origin of name 'Hadoop' was not an acronym. It was named after stuffed yellow elephant toy of Doug Cutting's Kid.
- In 2006, Yahoo! Company hired Doug Cutting to work on improving Hadoop, making it run at web scale.
- In 2008, Hadoop was Apache's top-level project, that provide robustness, accessibility and scalability yet a simple tool.

2.2.2 Building Blocks of Hadoop

A Hadoop ecosystem follows Master-Slave Architecture. Generally, Hadoop's architecture comprises of few master nodes and several worker nodes [31]. In a nutshell, NameNode, DataNode, Secondary NameNode, JobTracker and TaskTracker comprises basic building blocks of Hadoop [32]. Each component with its role within Hadoop is discussed as follows:

1) Master Node

Hadoop environment consists of few Master Nodes. More than one master node instance is created, to eliminate risk of single point of failure. Figure 2.1 shows components of Master Node. Master Node comprises of JobTracker and NameNode.

- **JobTracker:** This process distributes the MapReduce tasks to worker nodes in the cluster. Job tracker also deals with client applications.
- **Namenode:** It manages directory tree and meta data of all files of HDFS. It also keep information of data is located on which worker node.

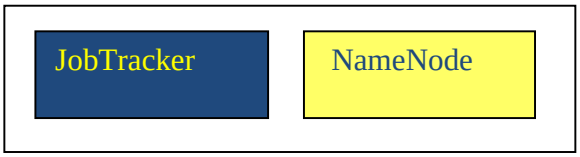


Figure 2.1: Components of Master Node [31]

2) Worker Nodes

There are many worker nodes (from dozens to hundreds) in a cluster. These nodes provide computational power as well as storage space for large volume of data. Figure 2.2 shows components of Worker Node. Worker Node comprises of DataNodes and TaskTracker.

- **DataNodes:** These are the systems which stores data. All the replication of chunks of data is done there. Namenode stores all the information about these nodes.
- **TaskTracker:** All the tasks which includes map, sort, and shuffle and reduce is performed by task tracker. It receives tasks from Job tracker.

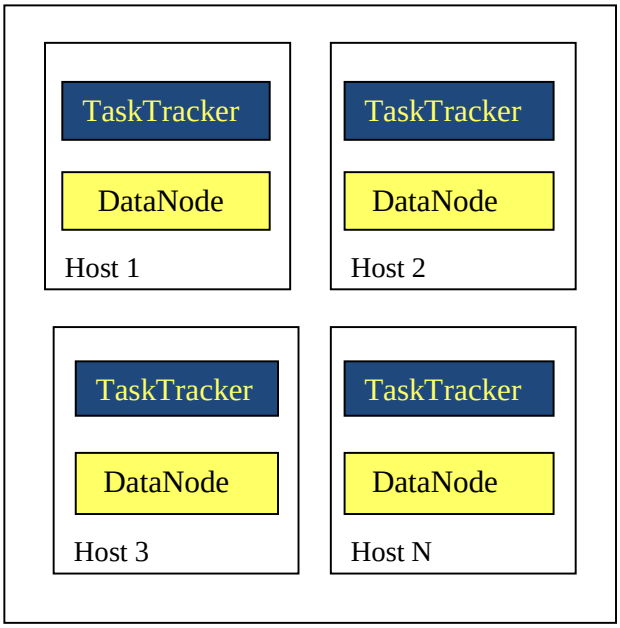


Figure 2.2: Components of Worker Node [31]

2.2.3 Hadoop Core Components

There are many open source components that are under Apache's Hadoop Project. MapReduce and HDFS are considered as core components of Hadoop [33]. HBase is a

Column-Oriented data store of Hadoop that supports random reads and writes. Zookeeper is a Coordination Service developed by Yahoo! from distributed systems. Pig is a high level data processing programming framework originated at Yahoo! [29]. Hive is a SQL interface providing Data warehousing. Oozie is a scheduler that manages flow of Hadoop processes. The fundamental components of Hadoop comprise of Programming paradigm (MapReduce) as well as a distributed file system (HDFS). The components are briefly explained as:

I. HDFS

HDFS **(**Hadoop Distributed File System) [11] is used to store large files, providing streaming access to them. HDFS works on the concept of blocks. Files are broken down into fixed size chunks called blocks. These blocks are then distributed over cluster of machines and also replicas are created. In HDFS, the block size is much larger than normal file system, thus minimizing disk seeks. Also, as the requested data is accessed from multiple blocks at once, leading to increase in performance of application and boosting up the response time [33].

HDFS adopts master-slave architecture, where NameNode and DataNodes are constituents of HDFS. Master in HDFS is NameNode which is responsible for managing directory tree and meta data of all files of HDFS. All the metadata information about size and location of blocks are maintained by NameNode. It also provides authorization and authentication. Slaves in HDFS are DataNodes which are responsible for storage of data blocks according to the instructions from Namenode [33]. They create, store and retrieve blocks of actual data or replicas when told by NameNode. The complete list of blocks that DataNodes are storing is periodically reported back to NameNode [31].

- **Read Operation in HDFS**

NameNode, as a repository of HDFS metadata, is responsible for maintaining file system namespace. Basically, Namespace comprises of files and directories hierarchy. When reading data from HDFS files, the client contacts NameNode in order to have blocks. NameNode returns back with information such as block ID, number of replicas of desired block and location of data blocks. The client then contacts DataNode to read the

blocks. Eventually, client reads block contents from the DataNode which is closest to the client. Figure 2.3 depicts read process in HDFS.

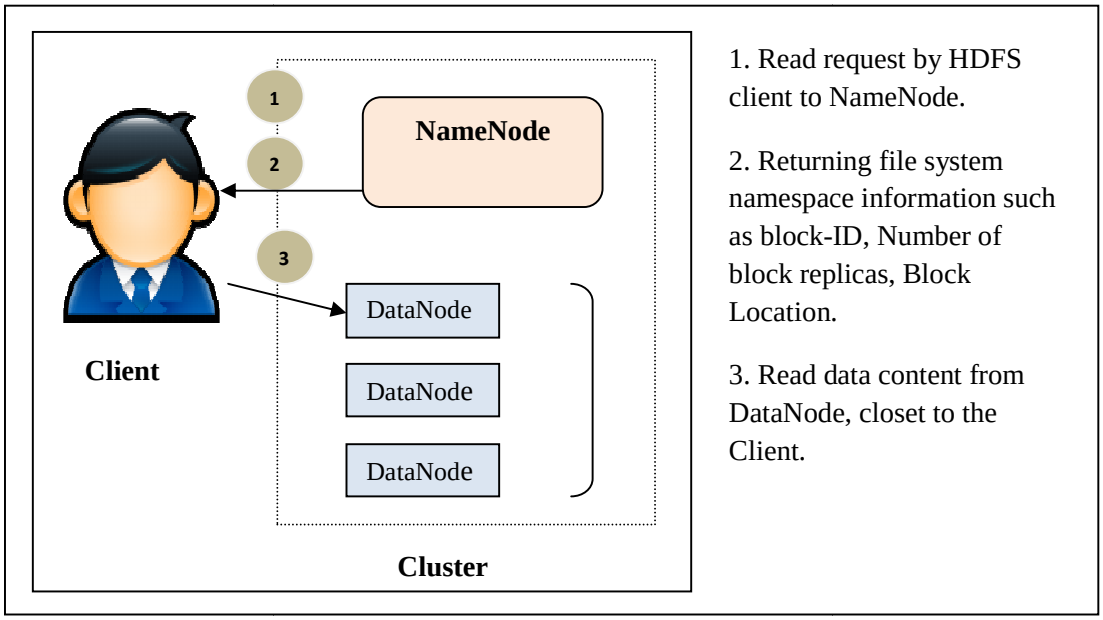


Figure 2.3: Read Operation in HDFS

- **Write Operation in HDFS**

A Client wanting to write file to HDFS, first contacts NameNode. As NameNode is responsible for the replication of data blocks in the cluster.

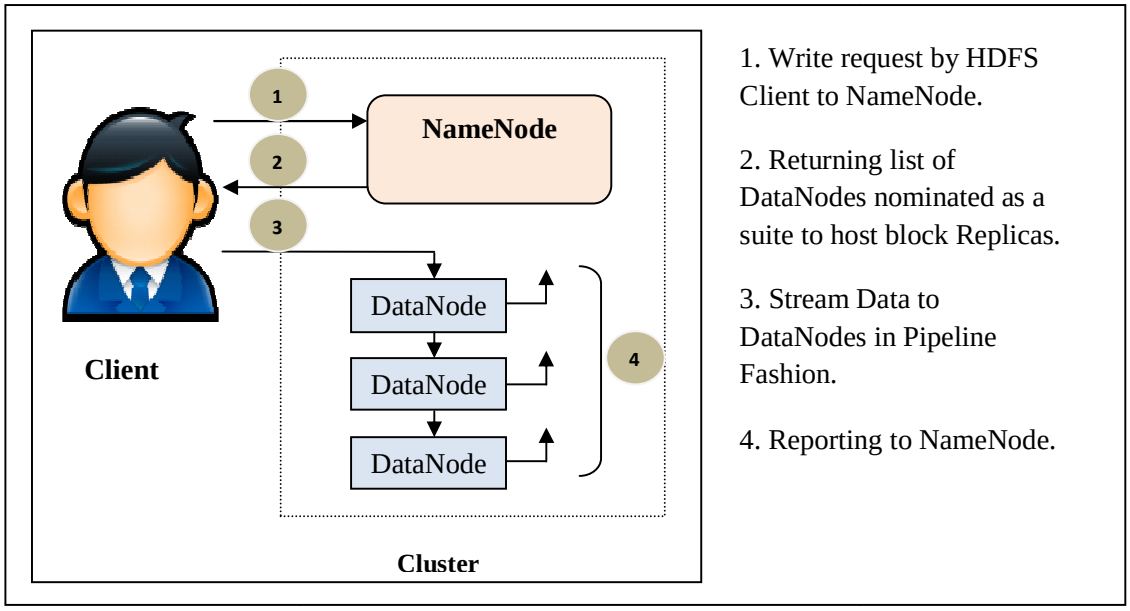


Figure 2.4: Write Operation in HDFS

Therefore, The NameNode returns a list of DataNodes, in order to host Block replicas. Basically, NameNode specifies namespace information such as block ID, number of blocks for replication. For new blocks to write, a unique block ID is allocated by NameNode. The data is written in chosen DataNodes in a pipeline fashion. A Client is acknowledged about the success or failure to write data. The DataNodes send a report to DataNode about the creation of block replicas. Figure 2.4 depicts write operation process in HDFS.

II. MapReduce

MapReduce is a Programming paradigm which acts as substantial base for processing big data problems into smaller units of work, facilitating the feature of Parallelization [34][35]. MapReduce Framework consists of two functional programming primitives.

- Map
- Reduce

Map() Function: The Map() function processes input records as input <*key1, value1*> pairs and produces a list of intermediate <*key2, value2*> pairs. Based on the intermediate key, MapReduce runtime system groups all intermediate <*key2, value2*> pairs and passes them to Reduce() function [36].

Reduce () Function: The Reduce() function reduces set of values to smaller set of values by performing its default sort-merging operation and thus produces the final result. Typically, Reduce () function reduces the intermediate set of values that shares same key and produces zero or more output pairs [36].

i.e. Map Function [37]

$$\text{Map (key1, value1)} \rightarrow \text{List (key2, value2)}$$

Reduce Function [37]

$$\text{Reduce (key2, List (value2))} \rightarrow \text{List (value2)}$$

Figure 2.5 illustrates MapReduce Process stages. The Workflow for MapReduce process moves as :

Input → Map → Shuffle & Sort → Reduce → Output

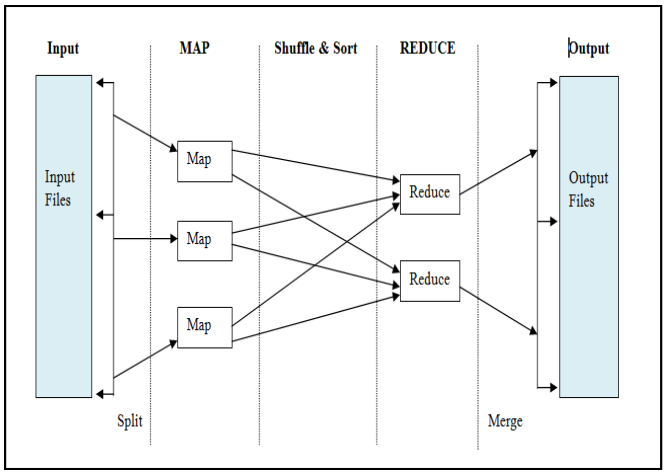


Figure 2.5: MapReduce Process [34]

2.3 CAP Theorem

Professor Eric Brewer is known as Father of CAP theorem. In 2000, He proposes this theorem under the title “Towards Robust Distributed Systems” at ACM's symposium. This theorem is widely adopted for implementing design of NoSQL data stores. The term CAP refers to Consistency, Availability and Partition tolerance. These basic requirements are explained as:

- Consistency means the data is same across all the nodes of the cluster, regardless of any delete or update. i.e. the client would read or write to the same data from any node of cluster.
- Availability means regardless of node failures in cluster, the system is providing response and is executing queries. i.e. despite of failure of some nodes, data is still available from remaining nodes of cluster.
- Partition-tolerance means the system continues to function despite of communication break between two nodes of cluster.

CAP theorem basically emphasizes on the fact that a shared data system can't meet these three distinct characteristics simultaneously, rather the system need to compromise on any of the one characteristic. Therefore, core idea behind CAP theorem is, any distributed system can at most choose two from these three parameters. Figure 2.6 depicts CAP Theorem.

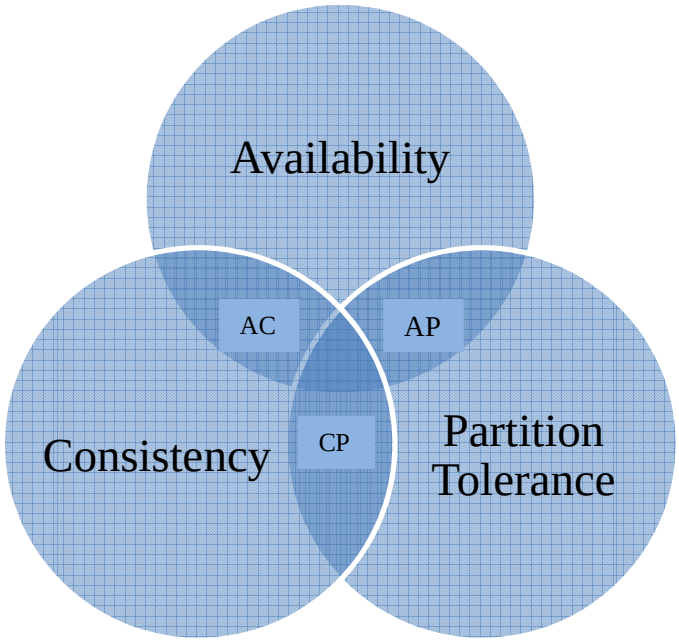


Figure 2.6: The CAP Theorem

I. Concerned about Consistency and Availability (CA)

As long as all the nodes of the cluster are up, read and write from any node would provide same and consistent data. These types of databases doesn't concerned about tolerance of network partition. The popular data stores that follow the CA concern are [8]:

- MySQL (Relational)
- PostgreSQL (Relational)
- Vertica (Column-Oriented)
- GreenPlum (Relational)

II. Concerned about Consistency and Partition Tolerance (CP)

Data is consistent as well as Partition tolerance is maintained between all the nodes of the distributed system. These systems don't ensure availability when the nodes in the cluster go down. The main data stores that follows CP are [8]:

- MongoDB (Document)
- HBase (Column-Oriented)
- Big Table (Column-Oriented)
- Redis (Document)
- MemcacheDB (Key-Value)

III. Concerned about Availability and Partition Tolerance (AP)

Nodes provide response, even if there exists communication failure between nodes. Resyncing of data is done, when the communication breakage is resolved. These systems don't guarantee consistency i.e. not ensuring that the data is same to all the nodes of cluster. The popular data stores that follow AP are [8]:

- Cassandra (Column-Oriented)
- Riak (Key-Value)
- Dynamo (Key-Value)
- SimpleDB (Document)
- CouchDB (Document)
- Voldemort (Document)

2.4 HBase

Today, We are overwhelmed with rapid expansion of digital data. These data comes in many different formats such as HTML, XML, graphs, tables, BLOB or spreadsheets. In order to access this wide real time enterprise data from one application, it needs to be integrated. Integration of heterogeneous data costs approximately 35% of telecommunicating Organization's budget [38].

As, it becomes difficult to retrieve, store and analyze data efficiently and economically by traditional database management softwares. Thus, in order to meet growing need of organizations, HBase is one such database that is used to process heterogeneous data. It runs on top of HDFS. It is distributed NoSQL Column-Oriented database, inhibits Flexibility, Scalability and Fault Tolerance. HBase is considered to have faster response to read and write access. From a logical point of view, HBase data is labeled in tables consisting of rows and columns [31]. Each cell in table can be retrieved as :

{Table Name + Rowkey + Column Family + Column Qualifier +Value + Timestamp}

Table's rows are sorted by Rowkey, which acts as a Primary key [23]. Column Family is necessary to define while creating table.

As HBase works on the top of HDFS, so it stores its data in a block-structured file system. By default HDFS replicates three copies of actual data, in order to provide availability, fault-tolerance and reliability. HBase mainly consists of HBase Master, an HDFS NameNode and multiple slave nodes in the cluster [26].

Studies have shown Performance Evaluation of HBase by having comparison of HBase-HDFS with SQL database MYSQL. Performance was based on random read and write access response time and throughput of both databases. In the initial stage, when the number of concurrent users was less, Performance of both HBase-HDFS and MYSQL-HDFS were at par. But later on, as the load increases, i.e. increasing the number of concurrent users, margin of Performance between HBase-HDFS and MYSQL-HDFS increases. Results show that performance of HBase-HDFS was better. So it highlights the fact that HBase enables faster read and write access for large sets of data [24].

2.4.1 Physical Structure of HBase

HBase adopts Master Slave architecture. Each cluster of HBase has one Master server and multiple Region Servers. HMaster is responsible for assigning regions, coordinating with slaves and executing administrative Operations. Slaves are responsible for managing HRegions. In HBase, the tables are broken into smaller chunks called Regions. HBase automatically splits the regions, when they become large. Regions are responsible for load balancing and scalability. HRegionServer have the burden of reading and writing data to the Regions. Therefore, HMaster and Region Servers are responsible for determining the strategy of data to be distributed among regions. Region Server serves and manages regions. The data is eventually stored in distributed file system of HBase. There are three types of files in HBase- MemStore, StoreFile and HLog. The Data is first written to MemStore. When the MemStore fills up, the data is flushed to StoreFile. When the size of StoreFile reaches to certain specified threshold, it triggers split operation that leads to creation of two Regions from one and are distributed to corresponding Region Servers [33]. -ROOT- and .META. are catalog tables, which maintains the current list,

state, recent history and location of all regions. The -ROOT- table holds the list of .META. Table regions. The .META. table holds the list of all user-space regions [39].

2.4.2 Write Operation in HBase

When a write is made by client, it goes to MemStore as well as to WAL. A write is made permanent only after the changes are made to both places and confirmed. This behavior is done in order to maintain data durability. Before making permanent write, HBase accumulates data in Memstore.

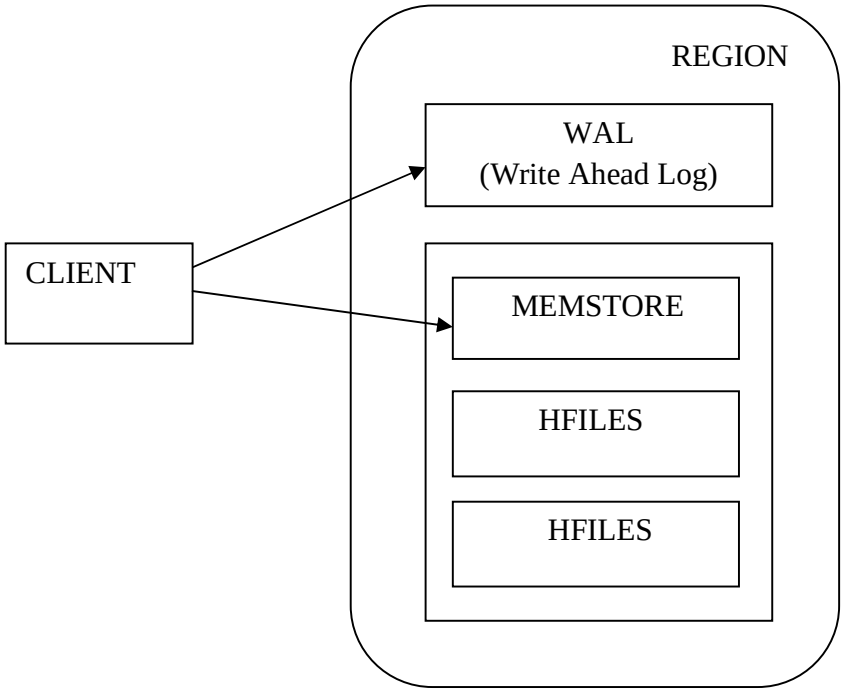


Figure 2.7: Write Operation in HBase Region [29]

When the MemStore fills up, its contents are flushed to disk and an HFile is formed. Rather writing to existing HFile, a new HFile is formed for every flush. HFiles are underlying storage format of HBase data [29]. There can be multiple HFiles in a Column Family [18]. Figure 2.7 illustrates Write Operation in HBase. In case there is a crash of data in Memstore that has not been flushed yet. That data could be considered lost from memory, but it would still be in WAL. Thus HBase provides no point of failure exception when writing data.

2.4.3 Read Operation in HBase

When reading a row from HBase table, first pending modifications are checked in Memstore. Then Block Cache is accessed in order to check whether the block that contains row is accessed recently. If that data block is not accessed recently, then relevant HFile is read [29]. Figure 2.8 illustrates basic Read Operation in HBase.

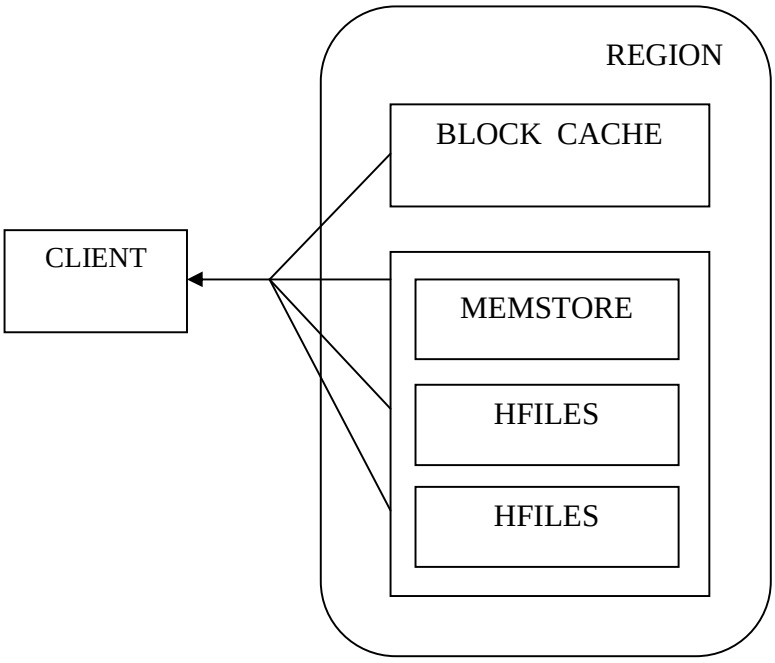


Figure 2.8: Read Operation in HBase Region [29]

2.4.4 Modes of HBase

HBase can be installed in three run modes [17].

1) Standalone Mode: This is the default mode that make use of single machine running only one process. In this mode, HBase doesn't make use of distributed file system, instead local file system is used. A single java VM runs zookeeper, mappers, reducers and all HBase daemons. Standalone mode in HBase is basically used for testing and experimentation purposes [18].

2) Pseudo-Distributed Mode: This mode is simply a distributed mode running over a single machine, i.e. all the component processes are distributed over all the cores on a

single host. As over a single machine, there run many java processes, files are written on to HDFS. This mode is best suited for configuration, testing and prototyping purposes.

3) Fully-Distributed Mode: In this mode daemons runs over cluster of machines. Fully-Distributed Mode needs an instance of HDFS (Hadoop Distributed System) to run [18].

2.4.5 HBase Table Conceptual and Physical View

The Conceptual and Physical views of HBase table are illustrated with an example table 'college_data'. Table 2.1 shows information of B.tech and M.tech of two colleges.

Table 2.1: Table displaying 'college_data'

Row Key	Data
Thapar College	B.tech {'Student': '2000', 'Teacher':'250'}
	M.tech {'Stream' : 'CS', 'Stream':'IS', 'Gate_stud' : '90'}
Rayat College	B.tech { 'Student' : '850' , 'Teacher' : '78'}
	M.tech {'Stream' : 'CS' , 'Teacher' :'10' }

According to concept of HBase, there would be two Column Families in the table. Data is stored according to Column Family 'B.tech' and 'M.tech'.

Table 2.2: Conceptual view of Data Storage in HBase table

Row Key	Timestamp	Column Family : B.tech	Column Family : M.tech
Thapar College	T9		M.tech:Stream="CS"
Thapar College	T8		M.tech:Stream ="IS"
Thapar College	T7		M.tech:Gate_stud ="90"
Thapar College	T6	B.tech:Student = "2000"	
Thapar College	T5	B.tech:Teacher = "250"	
Rayat College	T4		M.tech:Stream = "CS"
:	:	:	:

Table 2.2 gives the conceptual view for storing data in HBase tables. In actual, each record in table is divided into Column Families. Column Families are considered as separation between tables. Table 2.3 and Table 2.4 represent storage of data with respect to particular Column Families. Every row in HBase table should consist of same Column Family.

Table 2.3: Storage of data in HBase table corresponding to Column Family 'B.tech'

Row Key	Column Key	Timestamp	Cell Value
Thapar College	B.tech: Student	1276863104567	2000
Thapar College	B.tech: Teacher	1123467896531	250
Rayat College	B.tech: Student	1276893456781	850
Rayat College	B.tech: Teacher	1287345671239	78

Unlike Relational Databases, in HBase table null values are not stored, i.e. cells and columns containing null values are not stored.

Table 2.4: Storage of data in HBase table corresponding to Column Family 'M.tech'

Row Key	Column Key	Timestamp	Cell Value
Thapar College	M.tech:Stream	1276863125674	CS
Thapar College	M.tech:Stream	1287345678231	IS
Thapar College	M.tech:Gate_stud	1235985313578	90
Rayat College	M.tech : Stream	1234567543128	CS
Rayat College	M.tech :Teacher	1285633333789	10

2.5 Difference between HBase and RDBMS

The main Comparisons of HBase with traditional Relational databases are described in Table 2.5 [18][29][30].

Table 2.5: Table showing Comparisons of HBase with Relational Database

Parameters	HBase	Traditional RDBMS
Data Layout	Distributed, Column-Oriented	Generally Row-Oriented
Schema Type	Flexible Schema or Schema-less	Fixed Schema
Type(s) of data supported	Structured, Semi as well as Unstructured	Structured
Read/write throughput limits	Millions of Queries/second	Thousands Queries/second
Integration with MapReduce	Tight integration	No
Indexes	Rowkey only	Any arbitrary column
Joins	Individually doesn't support Joins	Optimized with Joins
Query Language	No (put,get,scan ,etc..)	SQL language
Data size	~1PBs	TBs
Random Lookup	Faster access and retrieval even in case of massive data	Comparatively slow
Security	Security works are in progress	Authentication/Authorization
Main Emphasis	Storing and Retrieving large Volume of data in seconds of time	Consistency, Referential integrity, Abstraction from Physical Layer
Null Values	Cells and Columns are not stored	Cells need to be set and occupy space

3. Problem Statement

3.1 Existing Relational Databases

Relational Databases have been serving as a storage platform for many enterprises since early 1980s. These databases strictly abide Relational model rules as described by E.F.Codd. All the available data is expressed in tabular formats. Primary and Foreign key structures are followed in order to provide reference to columns of tables. Concept of Joins and Normalization is used in defining relationships between tables. However, the performance of Relational Databases is good for small datasets only [39].

3.2 Research Gaps in Existing Relational Databases

- Due to enormous popularity of HTTP, interactive web applications are flooded with huge volume of data. Rather than terabytes or petabytes, the data is now in exabytes or zettabytes. However relative to incoming uninterrupted traffic, relational databases become unsuccessful to scale-up the applications. Thus proves inefficient for large size data workloads.
- A large percentage of data comes from online transactions, video, emails, audio, images, logs, posts, tweets, search queries, science data, health records, sensor data or social media interactions [2]. This human readable data can be in any format such as XML, HTML, Graphs, Tables, Audio, Video or BLOB. These formats of data can't be stored in the forms of entities and relationships so easily.
- Also, Relational databases need the data schema to be defined prior, which is not possible in case of electronic data of IT organization.

3.3 Problem Formulation

NoSQL is one of the emerging technologies that focus on providing highly concurrent real time requests. NoSQL databases are generally described as open source non relational Internet age databases. Unlike relational databases, NoSQL databases are flexible in nature, i.e. they don't require any fixed or static schema. With the constant change in domains model, NoSQL provides solutions for harnessing unstructured real

time data, in addition providing high availability. NoSQL databases are broadly classified into four classes: Key-Value databases, Column-Oriented databases, Document databases and Graph databases. In Column-oriented databases, data is stored by columns. Values of a single column are stored contiguously. Columns of similar data are stored together in Column Family. The factors leading to formulation of a problem, transforming Relational database into Column-Oriented database are:

- The Primary concern of migrating data from Relational data store to Column-Oriented data store is to have a database that provides flexibility when the data is semi structured as well as unstructured. Grabbing the advantages of NoSQL databases, these data stores also stores dynamic and ad-hoc data. Thus approaching towards a database that could fit all types of data easily
- Another concern for transforming Relational database to Column-Oriented database is to solve the problem of Scalability, which is the need of hour. Automatic partition and distribution of large data is done across nodes of clusters.
- Unlike Relational databases, a Column-Oriented database also allows multiple version storage of each column. Each version is differentiated from other version by Timestamp value. Also, Column-Oriented databases provides better efficiency of hard disk accesses over Row Oriented databases [6].

3.4 Objectives of Proposed Research Work

- To study existing Relational and Non-Relational Databases.
- To install and configure Hadoop and HBase.
- To propose an algorithm for schema and data transformation from Relational to Column-Oriented NoSQL database.
- To implement the proposed algorithm using MySQL BIRT database and JAVA APIs.
- To test and validate the proposed algorithm through Query Execution in MySQL and HBase.

4. Implementation

4.1. Experimental Environment

In order to implement the proposed Algorithm, our experimental platform mainly consists of configuring and installing of Hadoop and HBase.

4.1.1 Hardware and Software Specifications

Some specifications of System parameters that are used in implementation are given in Table 4.1.

Table 4.1: Hardware and Software Specifications

System Parameters	Specifications
Operating System	Ubuntu-13.04, 64bit
Hadoop	Hadoop-1.2.2
HBase	HBase-0.90.0
Programming Platform	Eclipse-jee-Kepler-SR1
Relational Database	MySQL-5.5.34.0

4.1.2 Configuration and Installation of Hadoop

For installation of Hadoop, stable version of Hadoop is downloaded [41]. Both MapReduce and HDFS are configured on the same node. Replication factor is set to be 3. For the creation of RSA key pair, *ssh-keygen* is used. JDK version of Oracle java 1.6 is installed for working of Hadoop. Hadoop make use of graphical web interfaces in order to specify working of JobTracker and NameNode.

The status of JobTracker and NameNode is monitored by:

http://localhost:50030 for JobTracker

http://localhost:50070 for NameNode

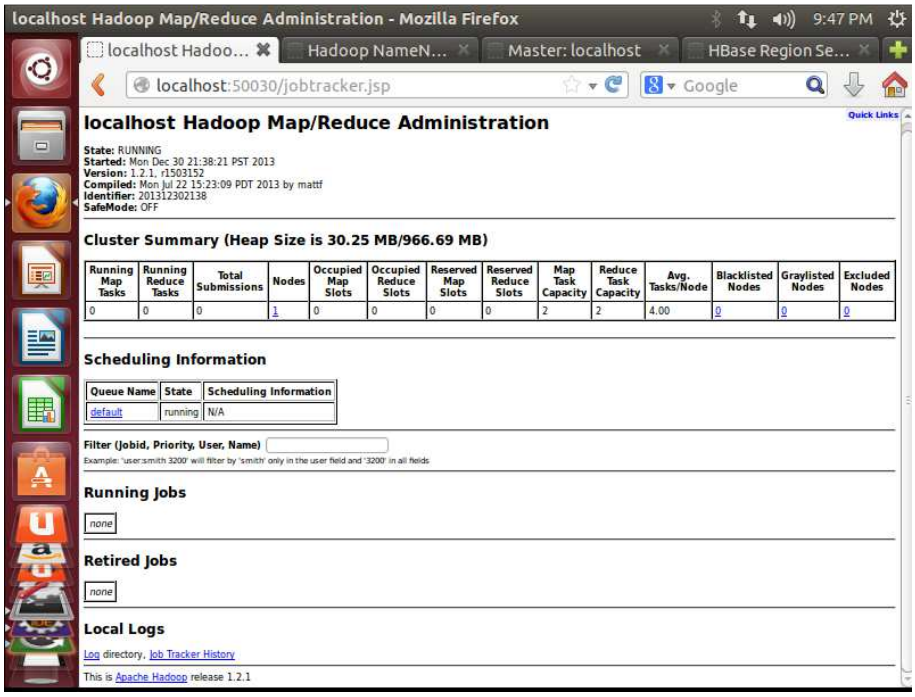


Figure 4.1: Snapshot Displaying GUI of Hadoop's MapReduce Administration

4.1.3 Configuration and Installation of HBase

HBase works in three modes: Standalone, Distributed and Pseudo-distributed mode[19]. As all daemons run on a single machine, therefore Pseudo-distributed mode is configured for the system. To have interaction of HBase database via command line, *HBase shell* is made to run. HMaster, DataNode, Task Tracker, Secondary Namenode, HBase Region server, Namenode and HQuorumpeer (Zookeeper) are configured in *hbase-site.xml* file [42]. Like Hadoop, HBase also make use of web interfaces to specify status of Master and Region Server.

http://localhost:60010 for Master Status

http://localhost:60030 for Region Server Status

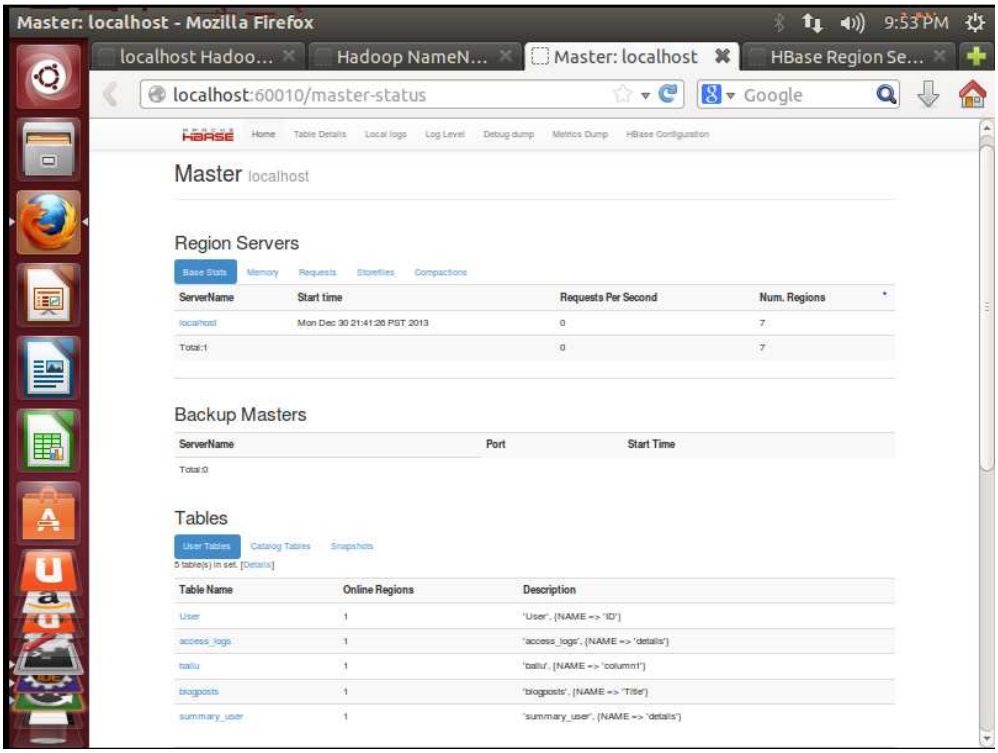


Figure 4.2: Snapshot displaying GUI of HBase Master Status

4.2 Database Used

The algorithm used for transformation of SQL database to HBase data store is well illustrated using BIRT sample database [43]. BIRT is an open source software platform that acts as a reporting tool providing business analytics. IBM, Actuate and Innovent Solutions are project sponsors of BIRT. This database is provided under the terms Eclipse.org Software User Agreement. The BIRT sample database provides a set of Tables and data associated with it.

The database named 'ClassicModels' basically contains information of a retailer, trading models of finest cars. The schema carries ideal trading statistics such as Customers, Orders, Offices, Employees and Products. Figure 4.3 illustrates Entity Relationship diagram of 'ClassicModels' database representing all the relationships existing between tables.

4.2.1 Schema Description

The database name 'ClassicModels' consists of eight tables. Table 4.2 provides overview of 'ClassicModels' database. Schema of each table and its corresponding elements are briefly explained as:

- **Offices:** Table 'Offices' stores information about all the sales offices of the Organization. This table comprises of fields such as office code, city, phone, address, state, country, postal code and territory. *officeCode* acts as primary key field. There exists One-to-Many relationships between table 'Offices' and 'Employees' describing employees working in a particular office.
- **Employees:** This table stores information about all the employees of the organization. *employeeNumber* is the primary key field whereas other fields contains information such as first name, last name, email, office code and job title. *officeCode* acts as a foreign key referencing primary key of table 'Offices'. Also, there exists relationship between 'Employees' and 'Customers' table, giving information of sales representative who works with customers.
- **Customers:** This table stores required personal information about customers. Also it refers to sales representative from 'Employees' table associated with particular customer. *customerNumber* acts as Primary key field. Other field contains information regarding name, phone, address, city, state, country, credit limit and sales representative dealing with that customer. *customerNumber* is used for referencing field of 'Payments' and 'Orders' table.
- **Orders:** This table stores information about order placed by customers. *orderNumber* field which is unique and non null integer value, acts as primary key. Other field provides description about data on which order is placed, shipped date, status of order and customer number. *customerNumber* acts as a Foreign key referencing Primary key of table 'Customers'.
- **OrderDetails:** The table 'OrderDetails' gives description about items within an order placed. Basically it refers to details of order placed by customer i.e. quantity of order, price of each product ordered. This table contains composite Primary

key *(orderNumber, productCode)*. Composite primary key comprises of two or more columns, collectively making a Primary key.

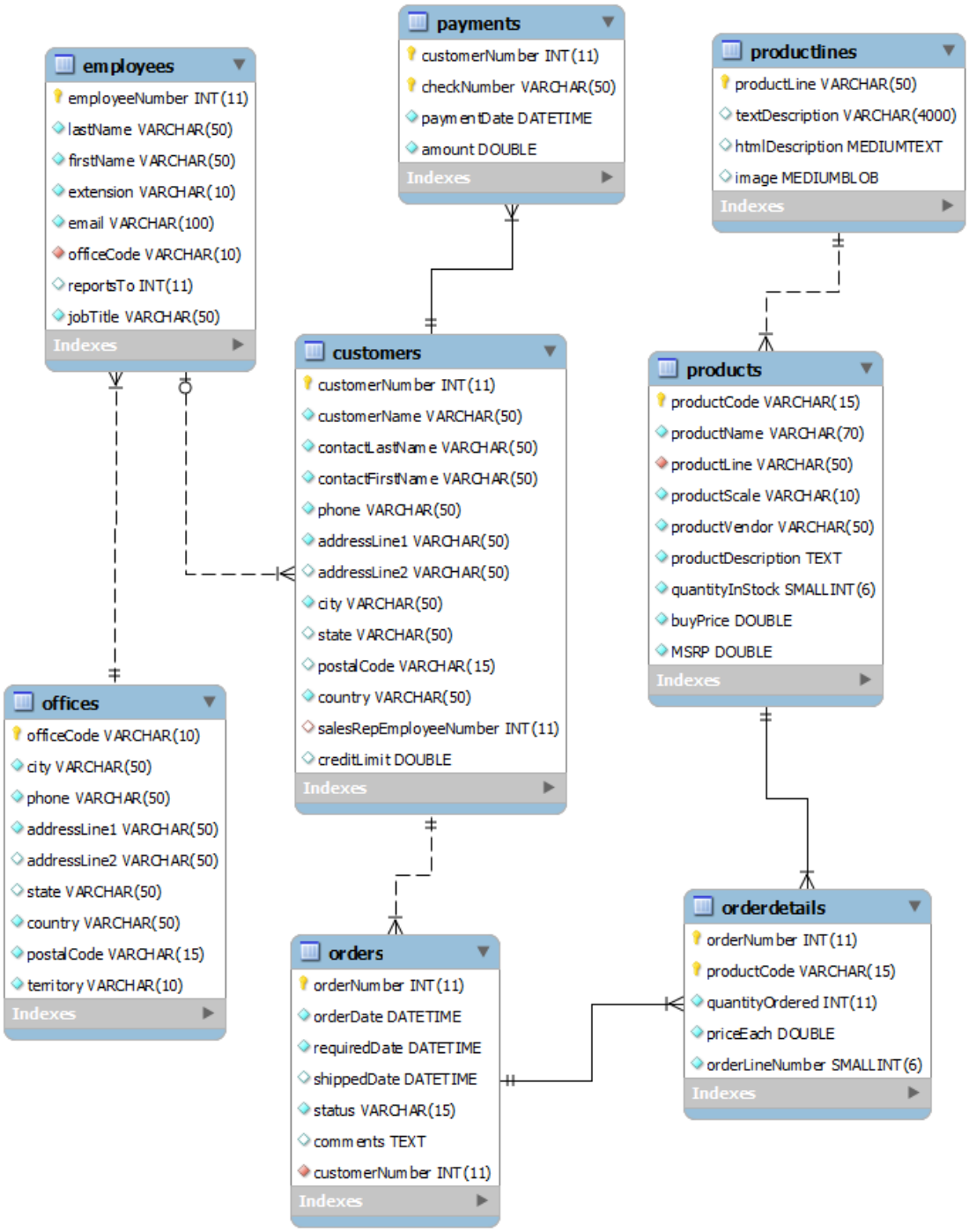


Figure 4.3: Entity Relationship Diagram of 'ClassicModels' Database

- **Payments:** This table contains information about the payment made by customer. This table also contains composite primary key *(customerNumber, checkNumber).* Other field contains information about the amount and date on which payment is made.
- **ProductLines:** Apart from product description, there is a table 'ProductLines' consisting of a product line associated with the product. The primary key field in table is *productLine*. Other fields contain image, text description and html description. Also there exits One-to-Many relationship with table 'Products'.

Table 4.2: 'ClassicModels' Database Schema Description

S.No.	Table Name	Record Size	Primary Key Column	Description
1	Customers	122	customerNumber	Personal details of Customers
2	Employees	12	employeeNumber	Employees Personal details
3	Offices	7	officeCode	Sales Offices of Organization
4	OrderDetails	2996	orderNumber, productCode	Details of items within placed Order
5	Orders	326	orderNumber	Placed Order Description
6	Payments	273	customerNumber, checkNumber	Payment Details of Customers
7	ProductLines	7	productLine	Product detailed Description
8	Products	110	productCode	List of Products of Organization

- **Products:** The table 'Products' stores a list of products of the organization. This table gives comprises of information such as product name, product vendor, product scale, product description, quantity, buy price and manufacturer's suggested retail price. *productCode* acts as a Primary key field.

4.3 Architectural Overview

The proposed architecture for transforming Relational Database to Column-Oriented Database comprises of four main modules. The flow of this proposed approach is represented in Figure 4.4.

4.3.1 Extract_Table_Attributes Module()

This is the first and foremost level that basically deals with programming operations of MySQL database. Each table of Relational database corresponds to HBase Tables. Also, Primary key of each Relational table would become Rowkey of Corresponding HBase Table. In case of composite Primary key in Relational table, Rowkey of HBase would also be combination of both keys. The function *gettable()* will extract all the tables of database. Corresponding to each extracted table, primary key is retrieved using *getPrimaryKeys()* function. In order to extract all the remaining columns, *getColumns()* function is used.

4.3.2 Relationship_Information Module()

Unlike Relational Database, HBase doesn't support Joins, Foreign keys and indexes. Only Rowkey is used to retrieve data. In Relational databases, there exists three main kinds of relationships- One-to-One, One-to-Many/Many-to-One and Many-to-Many. These relationships are generally managed using foreign key and Joins concept. However, HBase doesn't make use of Foreign keys and Joins, so some rules have been proposed so as to deal with these relationships [44].

I. **One-to-One Relationship:** The term one-to-one deals as a pair of two items in which one can only belong to the other. In Relational databases, these relationship are viewed as each row of that entity is associated with single row of other entity. Example of one-to-one relationship are:

- An employee working in a organization
- A person having one passport
- A research paper publishes by one author

Rule for One-to-One Relationship: In relational databases, these type of relationships are maintained by putting foreign keys on both sides of the relationship. This Foreign key column is treated as normal column in HBase, which is grouped in some Column Family.

II. **One-to-Many/Many-to-One Relationship:** The term one-to-many refers to a scenario where one entity contains values that refer to another entity. Entity here refers to a column or set of Columns. In Relational Databases, these relationships are viewed when Primary key of parent table is associated with Foreign Key of child table. As Foreign key is not unique, so can have multiple values associated with Primary key of parent table. Therefore, corresponding to one row of primary key there may be multiple rows referring to that key. Example of One-to-Many relationship are:

- A department can have several department
- A state comprises of many cities

Rule for One-to-Many Relationship: In Relational databases, for One-to-Many relationships, Foreign key is put on *many* side. As in HBase, multiple values are allowed. Therefore, a new Column Family is created on *one* side, which contains Rowkey of *many* side. Take an instance from Figure 4.8, there exists several one-to-many relationships among tables. Taking the case of 'Offices' and 'Employees' Table, where there exists One-to-Many relationship. Rowkey of 'Employees' table *employeeNumber* would be placed in new Column Family created in 'Offices' Table.

III. **Many-to-Many Relationship:** The term Many-to-Many refers to a scenario where entities of both tables can have multiple occurrences to entity of other table. Example of Many-to-Many Relationship are:

- A School subject has many students and each student may opt several subjects.

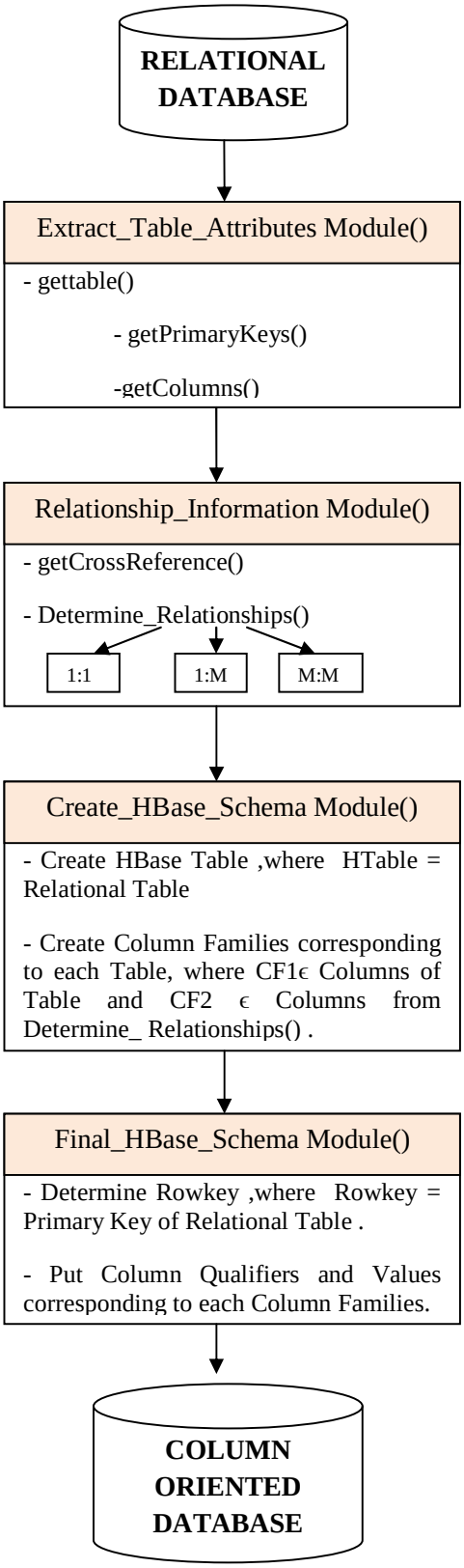


Figure 4.4: Architectural Overview of Proposed Algorithm

- A same hobby can be of many persons and a person can have more than one hobby.

Rule for Many-to-Many Relationship: In Relational databases, junction table is created, where Foreign keys of both tables are stored. In HBase, Many-to-Many relationships are maintained, creating Both new Column Families on both sides, which contains Rowkey of each other's Table. At the same time, if there exists junction table in table schema, it would be removed.

4.3.3. Create_HBase_Schema Module()

This module basically deals with HBase Connectivity, Instantiation of HBase Admin and HTable. In this module, corresponding to each table extracted in module 4.3.1., HBase table is created. Therefore, Column Families corresponding to each table are added.

4.3.4. Final_HBase_Schema Module()

This is the last and complex module that includes migration of schema as well as data to HBase tables. Extracted Primary keys from module 4.3.1 will acts as Rowkey for HBase table. All the related columns are grouped together in a Column Family. These columns are called as Column Qualifiers in HBase schema model.

4.4 Algorithm for Transforming SQL Database to HBase Database

The Algorithm for migrating SQL database to HBase database comprises of four Sections. These sections include step-by-step transformation of SQL schema as well as data to HBase database. These sections are:

I. Table Attributes Extraction Section
II. Relationship Information Extraction Section
III. HBase Column Family Creation Section
IV. HBase Schema Creation Section

These Sections are explained as:

4.4.1 Table Attributes Extraction Section

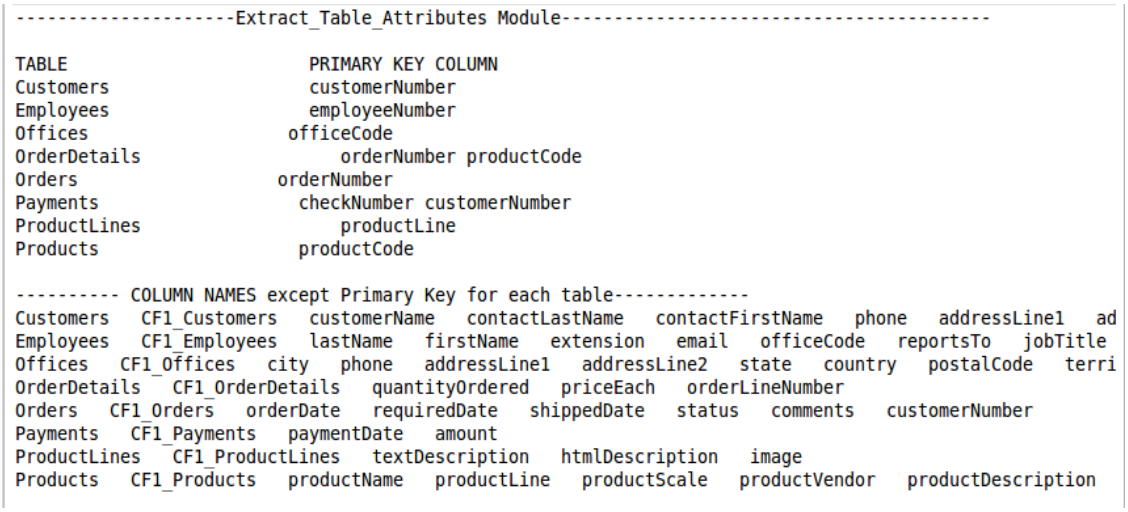
Description: Extract_Table_Attributes() function is first and foremost module in transforming Relational database to Column-Oriented database. The array Table_name[] contains table name of all the tables in the database. For each table, corresponding Primary key and remaining columns are retrieved. Primary_name[][] is a 2D array, which stores table name at its index 0 and corresponding Primary key column name at its next index. Similarly Column_name[][] contains all the column names of the table excluding Primary key column. Column_name[][] stores table name at its index 0. At index 1 of array 'CF1' concatenated table name is stored, as to give Column Family some relevant name and corresponding remaining column names are stored at its next indexes. This is done because in HBase, Corresponding to each table of relational database, a HBase table would be created. Primary Key of Relational database corresponds to Rowkey of HTable. Also, all the remaining columns of relational table would be included in one column Family. Thus, column data in one Column Family would be stored together. In order to describe this section, the pseudo code for extracting table attributes from MySQL database is explained in table 4.3.

Table 4.3 Pseudo code for Table Attributes Extraction Section

Pseudo code for Extracting Table Attributes information of Relational Database.
Extract_Table_Attributes() *Function* **Input:** jdbc connectivity with MySQL Database. **Output:** Arrays containing list of Table names, Primary Key name, Column names of Database respectively. **procedure** Extract_Table_Attributes(database db) ResultSet rs = getTables() **while**(rs.next()) **do** Retrieve Table Names and store in Array Table_name[] ResultSet rs1 = getPrimaryKeys()

```
        while(rs1.next()) do
                Retrieve Primary key column name and store in Primary_name[]
        end while
        ResultSet rs2 = getColumns()
        while(rs2.next()) do
                Retrieve Column names
                if (Columnname ϵ Primary_name[0][]) then
                        store it in array Column_name[]
                else
                        pass
                end if
        end while
    end while
end procedure
```

Figure 4.5 displays the output of 'Table Attributes Extraction Section', retrieving Primary key column name corresponding to each table name of MySQL database.



```
--------------------Extract_Table_Attributes Module---------------------------------------

TABLE                       PRIMARY KEY COLUMN
Customers                   customerNumber
Employees                   employeeNumber
Offices                    officeCode
OrderDetails                    orderNumber productCode
Orders                     orderNumber
Payments                    checkNumber customerNumber
ProductLines                    productLine
Products                   productCode

---------- COLUMN NAMES except Primary Key for each table-------------
Customers   CF1_Customers   customerName   contactLastName   contactFirstName   phone   addressLine1   ad
Employees   CF1_Employees   lastName   firstName   extension   email   officeCode   reportsTo   jobTitle
Offices   CF1_Offices   city   phone   addressLine1   addressLine2   state   country   postalCode   terri
OrderDetails   CF1_OrderDetails   quantityOrdered   priceEach   orderLineNumber
Orders   CF1_Orders   orderDate   requiredDate   shippedDate   status   comments   customerNumber
Payments   CF1_Payments   paymentDate   amount
ProductLines   CF1_ProductLines   textDescription   htmlDescription   image
Products   CF1_Products   productName   productLine   productScale   productVendor   productDescription
```

Figure 4.5: Snapshot displaying output of Table Attributes Extraction Section

All the remaining Column names of table are stored in an array, where table name is stored at index 0 and a unique name representing as Column Family is stored at index1.

4.4.2 Relationship Information Extraction Section

Description: The function *getCrossReference()* is used to determine reference relationship if any exists between the tables. The array Table_name[][] consists of table names of database tables. At first, *getCrossReference()* takes table name from index 0 and table from index1 to determine if there exists some reference relationship between them. Then this function checks relationship between table name at index 0 and table at index 2 and so on. This function provides output as a list containing Primary key table name, Primary key column name, Foreign key table name and Foreign key column name. These values are stored in a multidimensional array named CrossRefer[][]. In order to retrieve Primary key of Foreign key table, Foreign key table is matched with table names of Primary-name[][]. From the matched row corresponding primary key is extracted. As in HBase, this primary key would serve as Rowkey of the table. According to the rules defined earlier, a new Column Family would be created in Primary key table, which contains Rowkey of Foreign key table, maintaining One- to-Many relationship between tables. This information is stored in Column_name[][] array. In a nutshell, Column_name[][] contains columns that would be grouped together in Column Families. Table 4.4 describes the pseudo code for extracting relationship information between tables of Relational database.

Table 4.4 Pseudo code for Relationship Information Extraction Section

Pseudo code for Extracting Relationship information between Relational Database tables and determining Column Families for HBase Tables.
Relationship_Information() *Function* **Input:** Three Arrays containing list of Table names, Primary Key column name, remaining Column names respectively. **Output:** Arrays containing Cross Reference Relations and Column Families respectively. **procedure** Relationship_Information(Table_name[][], Primary_name[][], Column_name[][]) **for** each integer i ϵ Table_name. Length **do**

```
        for each integer j = i+1 to Table_name. Length do
                ResultSet  rs3 = getCrossReference()
                while(rs3.next()) do
                        Retrieve Primary Key Column name and Table name
                        Retrieve Foreign Key Column name and Table name
                        Store in array CrossRefer[][]
                        if(foreign key table ϵ Primary_name[][0]) then
                                Extract Primary key of Foreign Key Table
                        end if
                end while
        end for
    end for
    add it to array Column_name[][]
end procedure
```

The Cross Reference Relationship between tables is displayed in Figure 4.6. These relations are then used to understand one-to-many relationships between tables.

```
-----------------------Relationship_Information Module----------------------------
PRIMARY KEY Table      PRIMARY KEY Column       FOREIGN KEY Table       FOREIGN KEY COLUMN
Customers               customerNumber           Orders            customerNumber
Customers               customerNumber           Payments            customerNumber
ProductLines               productLine           Products            productLine
Products               productCode          OrderDetails            productCode
Orders               orderNumber           OrderDetails            orderNumber
Offices               officeCode           Employees            officeCode
Employees               employeeNumber           Customers            salesRepEmployeeNumber
```

Figure 4.6: Snapshot displaying output of Relationship Information Extraction Section

4.4.3 HBase Column Family Creation Algorithm

Description: This section describes the creation of HBase tables and corresponding Column Families. In order to make connection with HBase, object of *HBaseConfiguration* needs to be initialized.

HBaseConfiguration config = new HBaseConfiguration();

HBaseAdmin admin = new HBaseAdmin(config);

In order to connect to HMaster, an instance of HBaseAdmin is initialized. Creation, modification and deletion of tables and Column Families is provided by methods of HBaseAdmin. HBaseAdmin, in return requires instance of HBaseConfiguration. So instance of HBaseAdmin and HBaseConfiguration is initialized to have connection with HBase HMaster.

HTableDescriptor descriptor = new HTableDescriptor(Table name);

descriptor.addFamily(new HColumnDescriptor(Column Family name));

HTable instance is used to describe methods to create, insert rows and columns. HColumnDescriptor instance is used for each column family.

After HBase connectivity, HBase Tables are created corresponding to tables of relational database. These table names are stored in Table_name[][]. Further, it is checked that whether the table to be created, already exists. If the table already exists, then existing table is first disabled and then deleted. Corresponding to each table, Column Families are created from array Column_name[][]. As Column Families are static in nature, i.e. they need to be created at runtime. Making alteration to it afterwards is a tedious job. However, number of Column Families should always be reasonable.

Table 4.5 Pseudo code for HBase Column Family Creation Section

Pseudo code for creating HBase Tables and their corresponding Column Families .
Create_HBase_Schema() *Function* **Input:** HBase connectivity and Arrays containing list of Table names, Primary key column name, Column Family names respectively. **Output:** Creation of HBase Tables and their corresponding Column Families. **procedure** Create_HBase_Schema(HBaseConfiguration config, HBaseAdmin admin,

```
Table_name[][], Primary_name[][], Column_name[][] )
    for each integer a ϵ Table_name. Length do
        if(HBase Table[a] already Exists) then
                disable Table_name[a]
                delete Table_name[a]
        end if
        for each integer b ϵ Column_name. Length do
                if (Table_name[a] ϵ Column_name[b][0]) then
                        add Column Family to HBase table
                end if
        end for
    create HBase table
    end for
end procedure
```

Tables of 'ClassicModels' database corresponds to HBase tables, resulting in output of 'HBase Column Family Creation Section'. Figure 4.7 displays HBase tables using command *list*.

```
hbase(main):011:0> list
TABLE
Customers
Employees
Offices
OrderDetails
Orders
Payments
ProductLines
Products
8 row(s) in 0.0700 seconds

=> ["Customers", "Employees", "Offices", "OrderDetails", "Orders", "Payments", "
ProductLines", "Products"]
```

Figure 4.7: Snapshot displaying output of HBase Column Family Creation Section

4.4.4 HBase Schema Creation Algorithm

Description: This module deals with creation of final HBase schema that includes adding Column Qualifier and values corresponding to each Rowkey. Unlike Column Families, Column Qualifiers are dynamic in nature. Thus, Final HBase schema is optimized and is capable of answering all possible queries that can be fired on Column-Oriented database.

Table 4.6 Pseudo code for HBase Schema Creation Section

Pseudo code for Extracting data from Relational Database and inserting it to corresponding Column Families.

Final_HBase_Schema() *Function*
 	
Input: HBase table with its Column Families.
Output: Column Oriented Schema.
 	
procedure Final_HBase_Schema(database db)
for each integer a ϵ Table_name. Length **do**
for integer p = 1 to Primary_name[a]. Length **do**
return Σ p as rowkey
0ϵ Primary_name[a]. Length
end for
extract value corresponding to rowkey
Put (rowkey)
for each integer b ϵ Column_name. Length **do**
if(Table_name[a] == Column_name[b][0]) **then**
for integer c = 2 to Column_name[b].Length **do**
if(Column_name[b][c] == Column_name[a][c]) **then**
Retrieve value of each column
Put (Column_name[b][1],Column_name[b][c],value)

```
                else
                  for each integer  u ϵ Primary_name.Length do
                      for integer v = 1 to Primary_name[].Length do
                          if(Column_name[b][c] == Primary_name[u][v]) then
                                  String Tablecol ← Primary_name[u][0]
                                  for each integer v ϵ Crossrefer.Length  do
                                      Extract Foreign Key Column
                                end if
                            end for
                          end if
                       end for
                    end for
                    retrieve value corresponding to the column
                    Put(Column_name[b][1],Column_name[b][c],value)
                  end if
              end for
            end if
        end for
end procedure
```

The output of 'HBase Schema Creation Section' results in migration of schema as well as data to HBase tables. Figure 4.8 displays number of rows in 'Customers' HBase table. 'count' command in HBase shell is used to return number of rows in table.

```
hbase(main):025:0> count 'Customers'
122 row(s) in 0.0510 seconds

=> 122
```

Figure 4.8: Snapshot displaying number of rows in HBase Table 'Customers'

Similarly, Figure 4.9 displays the number of rows in 'Employees' Table.

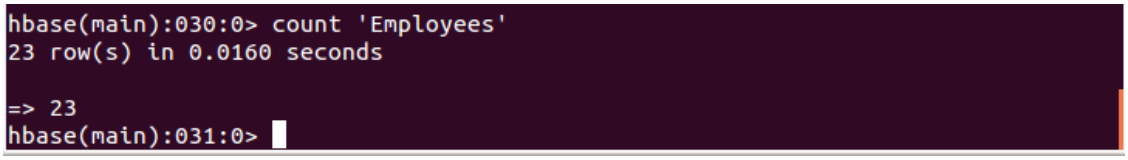


Figure 4.9: Snapshot displaying number of rows in HBase Table 'Employees'

The number of rows in 'Offices' Table is shown in Figure 4.10.

Figure 4.10: Snapshot Displaying number of rows in HBase Table 'Offices'

Record size of 'OrderDetails' is displayed in Figure 4.11. By default, Current count is displayed for 1000 rows. In order to increase Current count, 'interval' can be specified with count.

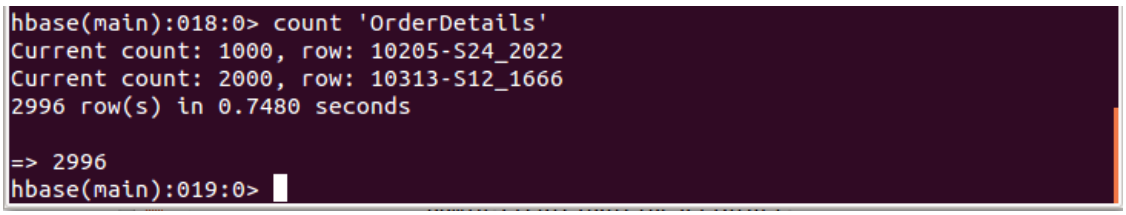


Figure 4.11: Snapshot displaying number of rows in HBase table 'OrderDetails'

Number of rows in 'Orders' table is displayed in Figure 4.12.

Figure 4.12: Snapshot displaying number of rows in HBase table 'Orders'

Figure 4.13 displays the number of rows in 'Payments' Table.

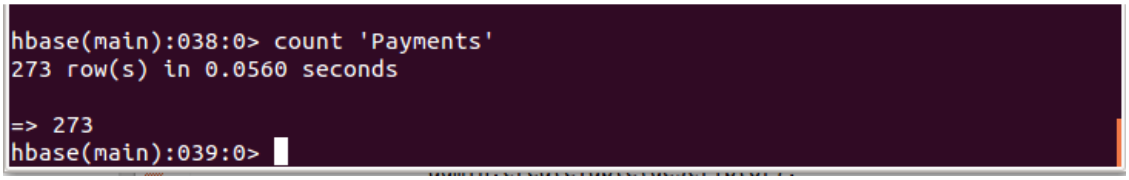


Figure 4.13: Snapshot displaying number of rows in HBase table 'Payments'

Number of rows in 'ProductLines' table is shown in Figure 4.14.

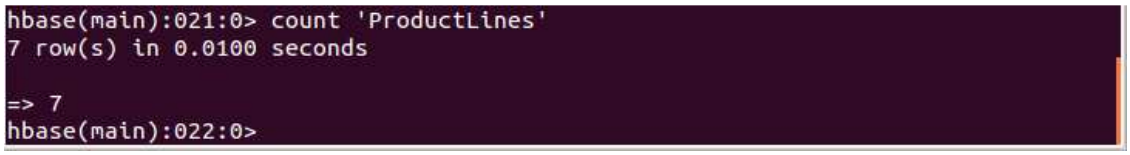


Figure 4.14: Snapshot displaying number of rows in HBase table 'ProductLines'

Similarly, Figure 4.15 displays number of rows of 'Products' table.

Figure 4.15: Snapshot displaying number of rows in HBase table 'Products'

5. Experimental Results

5.1 Modeling and Querying Format

For same database 'ClassicModels', Querying syntaxes of HBase and MySQL are described. Queries of diversified complexities has been taken in order to cover the maximum syntaxes of HBase.

5.1.1 Command Format

1. Command to find version of MySQL database and HBase.

MySQL: Show Variables LIKE '%Version';

```
mysql> SHOW VARIABLES LIKE '%VERSION';
+-----------------+-------------------------+
| Variable_name   | Value                   |
+-----------------+-------------------------+
| innodb_version  | 5.5.34                  |
| protocol_version | 10                     |
| version         | 5.5.34-0ubuntu0.13.04.1 |
+-----------------+-------------------------+
```

Fig 5.1: Snapshot displaying Version information of MySQL database

HBase: version

```
hbase(main):109:0> version
0.96.0-hadoop1, r1531434, Fri Oct 11 15:11:29 PDT 2013

hbase(main):110:0>
```

Figure 5.2: Snapshot displaying Version information of HBase

2. Command to list all the tables of *'ClassicModels'* Database.

MySQL: show tables;

```
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_ClassicModels |
+-------------------------+
| Customers               |
| Employees               |
| Offices                 |
| OrderDetails            |
| Orders                  |
| Payments                |
| ProductLines            |
| Products                |
+-------------------------+
```

Figure 5.3: Snapshot displaying tables of 'ClassicModels' Database in MySQL

HBase: list

```
hbase(main):011:0> list
TABLE
Customers
Employees
Offices
OrderDetails
Orders
Payments
ProductLines
Products
8 row(s) in 0.0700 seconds

=> ["Customers", "Employees", "Offices", "OrderDetails", "Orders", "Payments", "
ProductLines", "Products"]
```

Figure 5.4: Snapshot displaying tables of 'ClassicModels' Database in HBase

3. Command to describe the structure of 'Offices' table.

MySQL: show create Table Offices;

```
----------+
| Offices | CREATE TABLE `Offices` (
  `officeCode` varchar(10) NOT NULL,
  `city` varchar(50) NOT NULL,
  `phone` varchar(50) NOT NULL,
  `addressLine1` varchar(50) NOT NULL,
  `addressLine2` varchar(50) DEFAULT NULL,
  `state` varchar(50) DEFAULT NULL,
  `country` varchar(50) NOT NULL,
  `postalCode` varchar(15) NOT NULL,
  `territory` varchar(10) NOT NULL,
  PRIMARY KEY (`officeCode`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+---------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
```

Figure 5.5: Snapshot displaying Description of 'Offices' Table in MySQL

HBase: describe 'Offices'

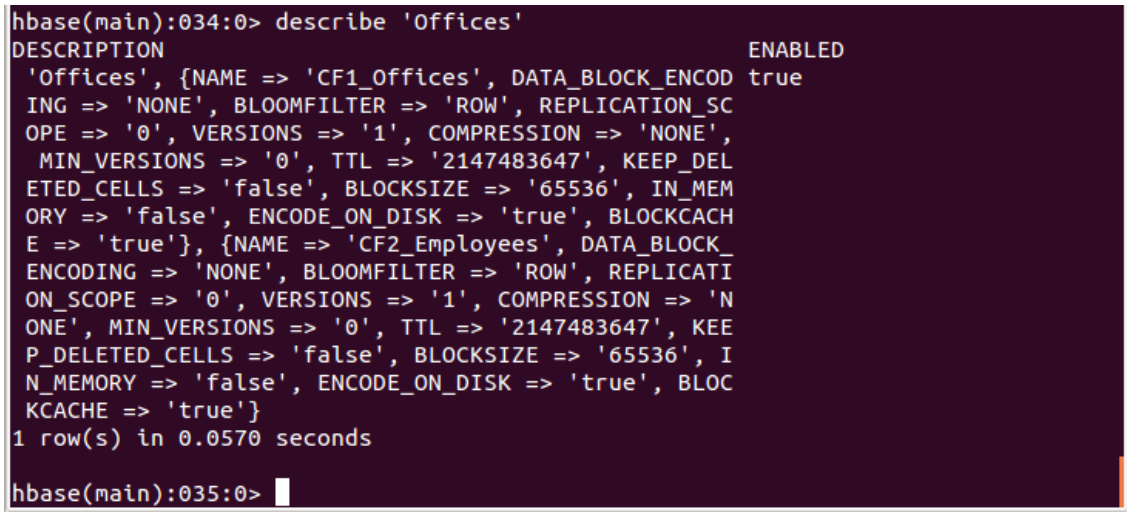


Figure 5.6: Snapshot displaying Description of 'Offices' Table in HBase

4. Finding number of rows in a 'OrdersDetails' table.

MySQL: Select count(*) from Customers;

Figure 5.7: Snapshot displaying number of rows in 'OrderDetails' Table in MySQL

HBase: count 'OrderDetails'

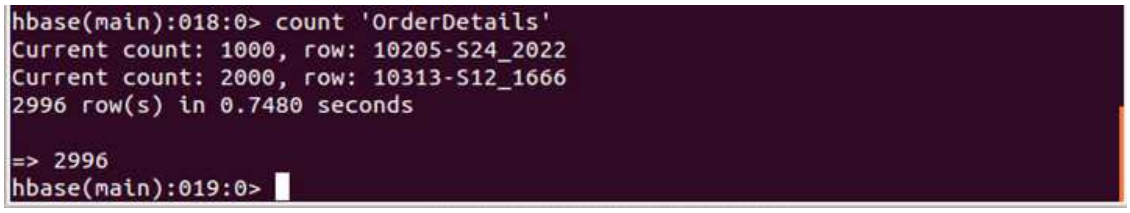


Figure 5.8: Snapshot displaying number of rows in 'OrderDetails' Table in HBase

Table 5.1 shows general format of some commands for MySQL database as well as Column-Oriented database(HBase).

Table 5.1 General Command format for MySQL database and HBase.

Commands	MySQL	HBase
Version of Database	Show Variables LIKE '%Version';	Version
List tables of Database	Show tables;	List
Structure of Table	Show create TABLE Tablename;	Describe 'Tablename'
Number of Rows in table	Select count(*) from Tablename;	Count 'Tablename'

5.1.2 Querying Format

The syntax of writing and executing queries in HBase is depicted by some of the examples shown :

1. Query for retrieving Organization's office details where officeCode is 1.

MySQL: SELECT * from Offices where officeCode ='1';

```
mysql> select * from Offices where officeCode ='1';
+------------+---------------+-----------------+------------------+--------------+-------+---------+----
--------+-----------+
| officeCode | city          | phone           | addressLine1     | addressLine2 | state | country | pos
talCode | territory |
+------------+---------------+-----------------+------------------+--------------+-------+---------+----
--------+-----------+
| 1          | San Francisco | +1 650 219 4782 | 100 Market Street | Suite 300    | CA    | USA     | 940
80      | NA"
"2   |
+------------+---------------+-----------------+------------------+--------------+-------+---------+----
--------+-----------+
```

Figure 5.9: Snapshot displaying office details for officeCode = '1' in MySQL

HBase: get 'Offices' ,'1'

```
hbase(main):008:0> get 'Offices','1'
COLUMN                        CELL
 CF1_Offices:addressLine1     timestamp=1402825125067, value=100 Market Street
 CF1_Offices:addressLine2     timestamp=1402825125067, value=Suite 300
 CF1_Offices:city             timestamp=1402825125067, value=San Francisco
 CF1_Offices:country          timestamp=1402825125067, value=USA
 CF1_Offices:phone            timestamp=1402825125067, value=+1 650 219 4782
 CF1_Offices:postalCode       timestamp=1402825125067, value=94080
 CF1_Offices:state            timestamp=1402825125067, value=CA
 CF1_Offices:territory        timestamp=1402825125067, value=NA"\x0D\x0A"2
 CF2_Employees:employeeNumb   timestamp=1402825125067, value=1166
 er
```

Figure 5.10: Snapshot displaying office details for officeCode = '1' in HBase

2. Query for retrieving name of employee whose employeeNumber is 1370.

MySQL: select firstName from Employees where employeeNumber = '1370';

```
mysql> select firstName from Employees where employeeNumber ='1370';
+-----------+
| firstName |
+-----------+
| Gerard    |
+-----------+
```

Figure 5.11: Snapshot displaying Employee's name whose Number is '1370' in MySQL

HBase: get 'Employees','1370',{COLUMN => 'CF1_Employees:firstName'}

```
hbase(main):018:0> get 'Employees','1370',{COLUMN=> 'CF1_Employees:firstName'}
COLUMN                        CELL
 CF1_Employees:firstName      timestamp=1402825124851, value=Gerard
```

Figure 5.12: Snapshot displaying Employee's name whose Number is '1370' in HBase

3. Query for retrieving name of Customers whose value lies between 300 and 350.

MySQL: select customerNumber, customerName from Customers where customerNumber BETWEEN 300 and 350;

```
mysql> select customerNumber,customerName from Customers where customerNumber BETWEEN 300 AND 350;
+----------------+----------------------------+
| customerNumber | customerName               |
+----------------+----------------------------+
|            303 | Schuyler Imports           |
|            307 | Der Hund Imports           |
|            311 | Oulu Toy Supplies, Inc.    |
|            314 | Petit Auto                 |
|            319 | Mini Classics              |
|            320 | Mini Creations Ltd.        |
|            321 | Corporate Gift Ideas Co.   |
|            323 | Down Under Souveniers, Inc |
|            324 | Stylish Desk Decors, Co.   |
|            328 | Tekni Collectables Inc.    |
|            333 | Australian Gift Network, Co|
|            334 | Suominen Souveniers        |
|            335 | Cramer SpezialitÃ¤ten, Ltd |
|            339 | Classic Gift Ideas, Inc    |
|            344 | CAF Imports                |
|            347 | Men 'R' US Retailers, Ltd. |
|            348 | Asian Treasures, Inc.      |
|            350 | Marseille Mini Autos       |
+----------------+----------------------------+
```

Figure 5.13: Snapshot of Customer names, where number is 300 to 350 in MySQL

HBase: scan 'Customers', {COLUMN=> ['CF1_Customers:customerName'], STARTROW =>'300' , STOPROW=> '351'}

```
ROW                    COLUMN+CELL
 303                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Schuyler Imports
 307                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Der Hund Imports
 311                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Oulu Toy Supplies, Inc.
 314                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Petit Auto
 319                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Mini Classics
 320                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Mini Creations Ltd.
 321                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Corporate Gift Ideas Co.
 323                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Down Under Souveniers, Inc
 324                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Stylish Desk Decors, Co.
 328                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Tekni Collectables Inc.
 333                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Australian Gift Network, Co
 334                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Suominen Souveniers
 335                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Cramer Spezialit\xC3\x83\xC2\xA4ten, Ltd
 339                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Classic Gift Ideas, Inc
 344                   column=CF1_Customers:customerName, timestamp=1402825124146, value=CAF Imports
 347                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Men 'R' US Retailers, Ltd.
 348                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Asian Treasures, Inc.
 350                   column=CF1_Customers:customerName, timestamp=1402825124146, value=Marseille Mini Autos
18 row(s) in 0.2370 seconds
```

Figure 5.14: Snapshot of Customer names, where number is 300 to 350 in HBase

4. Query for retrieving only 5 product names whose code starts with 'S1'.

MySQL: select productCode, productName from Products where productCode like 'S1%' LIMIT 5;

```
mysql> select productCode,productName from Products where productCode like 'S1%' LIMIT 5;
+-------------+--------------------------------------+
| productCode | productName                          |
+-------------+--------------------------------------+
| S10_1678    | 1969 Harley Davidson Ultimate Chopper |
| S10_1949    | 1952 Alpine Renault 1300             |
| S10_2016    | 1996 Moto Guzzi 1100i                |
| S10_4698    | 2003 Harley-Davidson Eagle Drag Bike  |
| S10_4757    | 1972 Alfa Romeo GTA                  |
+-------------+--------------------------------------+
```

Figure 5.15: Snapshot displaying 5 product names starting with 'S1' in MySQL

HBase: scan 'Products', {COLUMN=> ['CF1_Products:productName'], STARTROW => 'S10', LIMIT =>5}

```
hbase(main):049:0> scan 'Products', {COLUMN=> ['CF1_Products:productName'],STARTROW=>'S10', LIMIT=> 5}
ROW                           COLUMN+CELL
 S10_1678                     column=CF1_Products:productName, timestamp=1402825140178, value=1969 Harley D
                              avidson Ultimate Chopper
 S10_1949                     column=CF1_Products:productName, timestamp=1402825140178, value=1952 Alpine R
                              enault 1300
 S10_2016                     column=CF1_Products:productName, timestamp=1402825140178, value=1996 Moto Guz
                              zi 1100i
 S10_4698                     column=CF1_Products:productName, timestamp=1402825140178, value=2003 Harley-D
                              avidson Eagle Drag Bike
 S10_4757                     column=CF1_Products:productName, timestamp=1402825140178, value=1972 Alfa Rom
                              eo GTA
5 row(s) in 0.0640 seconds
```

Figure 5.16: Snapshot displaying 5 product names starting with 'S1' in HBase

5. Query for retrieving Employees working in Office code 3.

MySQL: SELECT e.employeeNumber from Employees e, Offices o where (e.officeCode = o.officeCode)&& (o.officeCode = 3)

```
mysql> select e.employeeNumber from Employees e, Offices o where (e.officeCode =
 o.officeCode)&& (o.officeCode = 3);
+----------------+
| employeeNumber |
+----------------+
|           1286 |
|           1323 |
+----------------+
```

Figure 5.17: Snapshot displaying Employee number who work in Office '3'

HBase: Scan 'Offices', {COLUMN => ['CF2_Employees:employeeNumber'], STARTROW => '3', STOPROW => '4' , VERSIONS => 5}

```
hbase(main):047:0> scan 'Offices',{COLUMN=> ['CF2_Employees:employeeNumber'],STA
RTROW=> '3', STOPROW=> '4', VERSIONS=> 5}
ROW                   COLUMN+CELL
 3                    column=CF2_Employees:employeeNumber, timestamp=14040265986
                      58, value=1286
 3                    column=CF2_Employees:employeeNumber, timestamp=14033512251
                      40, value=1323
1 row(s) in 0.0160 seconds
```

Figure 5.18: Snapshot displaying Employee number who work in Office '3' in HBase

6. Conclusion and Future Scope

Conclusion

As the volume of web data is exploding in this digital world, Organizations need databases that are flexible and scalable enough to handle Semi-structured and Unstructured Data. However, Relational databases show their age, when it comes to the turn of accessing huge amount of real time data. NoSQL databases serves as a solution for providing scalability, availability and flexibility for heterogeneous data. This leads to migration of Organization's Relational databases to NoSQL databases. The proposed algorithm transforms the schema as well as data from Relational Database to Column-Oriented NoSQL Data store. The proposed approach has been implemented in Linux Operating system using MySQL as Relational and HBase as Column-Oriented database. Corresponding to MySQL queries, data modeling as well as querying format of HBase has been explained.

Future Scope

I. The algorithm can be further extended by incorporating MapReduce Functionality in order to have parallel processing of queries.

II. In Future, Concept of nested Column Families can be implemented for advanced Joins operations.

References

[1] S. Sagiroglu and D. Sinanc, "Big data: A review," in *Proceedings of the IEEE International Collaboration Technologies and Systems (CTS) Conference*, San Diego, CA, pp. 42-47, 20-24 May 2013.

[2] D. Zhang, "Inconsistencies in big data," in *Proceedings of the 12th IEE International Cognitive Informatics & Cognitive Computing (ICCI*CC)Conference*, New York, pp. 61-67, 16-18 July 2013.

[3] A. Katal, M. Wazid and R. H. Goudar, "Big data: Issues, challenges, tools and Good practices," in *Proceedings of the 6th IEEE International Contemporary Computing (IC3), Conference*, Noida, pp. 404-409, 8-10 Aug. 2013.

[4] R. D. Schneider, "Hadoop for Dummies," *John Willey & sons Canada Ltd.*, Mississauga, ON L5R Canada 2012.

[5] R. Schell, "Security — A big question for big data," in *Proceedings of the IEEE International Big Data Conference*, Silicon Valley, CA, pp. 5, 6-9 Oct. 2013.

[6] K. Kaur and R. Rani, "Modeling and querying data in NoSQL databases," in *Proceedings of the IEEE International Big Data Conference*, Silicon Valley, CA, pp. 1-7, 6-9 Oct. 2013.

[7] L. Yishan and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in *Proceedings of the IEEE Pacific Rim Communications, Computers and Signal Processing (PACRIM) Conference*, Victoria, BC, pp. 15-19, 27-29 Aug. 2013.

[8] H. Jing; E. Haihong, L. Guan and D. Jian, "Survey on NoSQL database," in *Proceedings of the 6th IEEE International Pervasive Computing and Applications (ICPCA) Conference*, Port Elizabeth, pp. 363-366, 26-28 Oct. 2011.

[9] R. Arora and R. R. Aggarwal, "Modeling and Querying Data in MongoDB," *International Journal of Advanced Studies in Computer Science and Engineering (IJASCSE)*, vol. 2, no. 1, pp. 141-144, July 2013.

[10] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogels, "Dynamo: amazon's highly available key-value store", in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 205-220, 2007.

[11] Project Voldemort: A distributed database. [Online]. Available: http://project-voldemort.com/

[12] Redis.[Online]. Available: http://redis.io/

[13] MemcacheDB. [Online]. Available: http://memcachedb.org/

[14] Tokyo Cabinet. [Online]. Available: http://fallabs.com/tokyocabinet/

[15] P. Raichand, and R. Rani, "Query Execution and Effect of Compression on NoSQL Column Oriented Data-store Using Hadoop and HBase," *International Journal of Scientific and Engineering Research (IJSER)*,vol. 4, no. 9, Sept. 2013.

[16] F. Chang, D. Jeffrey, G. Sanjay, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, “Bigtable: A Distributed Storage System for Structured Data,” *Journal of ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1-26, June 2008.

[17] Apache HBase. [Online]. Available: http://hbase.apache.org/

[18] George, Lars. HBase: the definitive guide. O'Reilly Media, Inc., 2011.

[19] Hypertable. [Online]. Available: http://hypertable.org

[20] Cassandra. [Online]. Available: http://cassandra.apache.org

[21] Amazon SimpleDB. [Online]. Available: http://aws.amazon.com/simpledb/

[22] Couchdb. [Online]. Available: http://couchdb.apache.org/

[23] Mongodb's. [Online]. Available: http://www.mongodb.org/

[24] Terrastore. [Online]. Available: http://code.google.com/p/terrastore/

[25] E. Eifrem., Neo4j. [Online]. Available: http://www.neo4j.org/

[26] Orientdb wiki. [Online]. Available: http://www.orientechnologies.com/orientdb/

[27] Titan distributed graph database. [Online]. Available: http://thinkaurelius.github.com/titan/

[28] Allegrograph. [Online]. Available: http://www.franz.com/agraph/allegrograph/

[29] D. Nick, A. Khurana, and M. H. Ryan, " HBase Fundamentals," in *HBase in Action*, Manning Publications Co., Shelter Island, NY, 2013.

[30] M. N. Vora, "Hadoop-HBase for large-scale data," in *Proceedings of the IEEE International Computer Science and Network Technology (ICCSNT) Conference*, Harbin, vol. 1, pp. 601-605, 24-26 Dec. 2011.

[31] T. White, *Hadoop: the definitive guide*, O'Reilly Media, Inc., Sebastopol, CA, 2012.

[32] C. Lam, *Hadoop in action*, Manning Publications Co., Stamford, CT, 2010.

[33] S. Konstantin, K. Hairong, S. Radia and C. Robert, "The hadoop distributed file system," in *Proceedings of the 26th IEEE International* Sym*posium on Mass Storage Systems and Technologies (MSST)*, Incline Village, NV,pp. 1-10, 3-7 May 2010.

[34] J. Ekanayake, L. Hui, Z. Bingjing, G. Thilina, H. B. Seung, Q. Judy and F. Geoffrey, "Twister: a runtime for iterative MapReduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 810-818, 1-2 Oct.2010.

[35] X. Jiong, Y. Shu, R. Xiaojun, D. Zhiyang, T. Yun J. Majors, A. Manzanares and Q. Xiao, "Improving MapReduce performance through data placement in heterogeneous

Hadoop clusters," in *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, Atlanta, GA, pp. 1-9, 19-23 April 2010.

[36] D. Jiang, C. Beng, S. Lei and W. Sai, "The performance of MapReduce: an in-depth study", *Journal Proceedings of the VLDB Endowment*, vol. 3, no. 1-2 , pp. 472-483,13-17 Sep. 2010.

[37] D. Jeffrey and G. Sanjay, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, 107-113, Jan 2008.

[38] S. Sathya and M. V. Jose, "Application of Hadoop MapReduce technique to Virtual Database system design," *in Proceedings of the IEEE International Emerging Trends in Electrical and Computer Technology (ICETECT) Conference*, Tamil Nadu, pp. 892-896, 23-24 March 2011.

[39] H. Shidong, C. Lizhi, L. Zhenyu and H. Yun, "Non-structure Data Storage Technology: A Discussion," in *Proceedings of the 11th IEEE/ACIS International Computer and Information Science (ICIS)Conference*, Shanghai, pp. 482-487, May 30 2012-June 1 2012.

[40] W. C. Chung, H. P. Lin, S. C. Chen, M. F. Jiang and Y. C. Chung, "JackHare: a framework for SQL to NoSQL translation using MapReduce," *Springer Journal on Automated Software Engineering*, pp. 1-20, Sep 2013.

[41] Hadoop. [Online]. Available: http://hadoop.apache.org/

[42] A. Schangale. (2013, Aug. 31). *installing-pseudo-distributed-hbase-on-ubuntu*. [Online]. Available: http://archanaschangale.wordpress.com/2013/08/31/installing-pseudo-distributed-hbase-on-ubuntu/

[43] BIRT Sample Database. [Online]. Available: http://www.eclipse.org/birt/documentation/sample-database.php

[44] L. Chongxin, "Transforming relational database into HBase: A case study," in *Proceedings of the 26th IEEE International* Softw*are Engineering and Service Sciences (ICSESS) Conference*, Beijing, pp. 683-687, 16-18 July 2010.

List of Publications

I. Baljit Kaur and Rinkle Rani, "Designing URL Access Counter using MapReduce with HBase," in *Proceedings of Elsevier Second International Conference on Emerging Research In Computing Information, Communication and Applications* - (ERCICA-14), Bangalore, 01-02 Aug. 2014.

[Accepted]