AN APPROACH TO DESIGNING CLUSTERS FOR LARGE DATA PROCESSING

RONI SANDEL

A THESIS SUBMITTED TO

THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS AND TECHNOLOGY

YORK UNIVERSITY

TORONTO, ONTARIO

NOVEMBER 2014

Abstract

Cloud computing is increasingly being adopted due to its cost savings and abilities to scale. As data continues to grow rapidly, an increasing amount of institutions are adopting non standard SQL clusters to address the storage and processing demands of large data. However, evaluating and modelling non SQL clusters presents many challenges. In order to address some of these challenges, this thesis proposes a methodology for designing and modelling large scale processing configurations that respond to the end user requirements. Firstly, goals are established for the big data cluster. In this thesis, we use performance and cost as our goals. Secondly, the data is transformed from relational data schema to an appropriate HBase schema. In the third step, we iteratively deploy different clusters. We then model the clusters and evaluate different topologies (size of instances, number of instances, number of clusters, etc.). We use HBase as the large data processing cluster and we evaluate our methodology on traffic data from a large city and on a distributed community cloud infrastructure.

Acknowledgements

I would like to first and foremost express the deepest appreciation to my supervisor, Professor Marin Litoiu for not only being an incredible supervisor, but also giving me the opportunity to learn new technologies and giving me valuable experience. I would also like to thank him for his guidance and persistent help with this thesis. It was truly an honour.

Furthermore I want to also thank Professor Radu Campeanu (who is also the Chair of the committee) for his mentorship throughout my undergraduate studies at York University and for introducing me to the MAIST program at York University.

My gratitude goes to Professor Sotirios Liaskos who is part of my supervisory committee for all the help he has given me throughout my Graduate Studies, as well as giving me a better understanding of Software Architecture and Requirements Engineering.

Moreover, I would like to thank Professor Henry Kim for his insightful comments and interest in my topic, as well as taking on the role as an external examiner for my defense.

I would also like to express my gratitude to Dr. Mark Shtern for being an outstanding mentor and being there whenever I had a problem. He has been my mentor throughout my thesis adventure. During my time as a Masters student, I worked closely with Dr. Mark Shtern and Dr. Marin Litoiu to publish a position paper on mitigating Low

and Slow Distributed Denial of Service Attacks aimed at the application layer and had the privilege to present it at a conference in Boston [1].

I give my appreciation to Dr. Rizwan Mian, who was also my thesis mentor for all the excellent help he has given me and for teaching me strategies for approaching complex problems. He was my thesis mentor for a shorter period of time but he has helped me tremendously.

I am so thankful for the support of Professor Younes Benslimane for being a great supervisor and a mentor during my teaching assistantship.

I want to express my gratitude to Dr. Mike Smit for introducing me to Amazon EC2, for advising me to use Cloudera, and for his expertise. I want to thank Brad for introducing me to Hadoop Distributed File System and Map/Reduce, as well as helping me with some of my rookie technical issues when I first started.

Moreover, I want to thank Dr. Hadi Bannazadeh for providing me with the Smart Application Virtual Infrastructure Test Bed to perform my experiments for free and helping me whenever I had any problems with the machines. This thesis would have been too expensive to produce had it not been for the availability of the SAVI cloud. I also want to thank Thomas Lin for also helping me with SAVI-related issues.

I want to thank the Connected Vehicles and Smart Transportation (CVST) team at University of Toronto, especially Dr. Ali Tizghadam, Ali Shariat, and Mohamed Elshenawy for introducing me to the traffic domain and providing me with the data that I needed to carry out my experiments, as well as giving me a walkthrough of this data.

I would also like to thank my other fellow lab mates: Hamoun Ghanbari, Cornel Barna, Przemyslaw Pawluk, Parisa Zoghi, Vasileios Theodorou, Hongbin Lu, and Saeed Zareian.

I want to send my gratitude to York University Graduate Scholarship, CVST, and Ontario Graduate Scholarship Program for their generous financial support during my research.

Last but certainly not least, I would like to thank my family, friends, and girlfriend for all the support they have given me throughout this process. I want to especially thank both my mother and father, Arie and Hadasa Sandel, who helped get me where I am.

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1 Problem and Motivation

Enterprises are increasingly adopting cloud computing because of its economic advantage and its ability to scale [2],[3],[4]. By eliminating up-front costs, the cloud allows companies to scale hardware and software resources on a demands-need basis [2], [5]. These benefits have also allowed for improved management of Big Data.

Today, Big Data is a popular term to describe the exponential growth and availability of data, both structured and unstructured [6]. The characteristics of Big Data are commonly described as variety, volume, and velocity [7]. As systems are becoming more and more complex, data is increasing in size and thus effective data management of large data sets has been a major research problem. According to Agrawal et al. [8], researchers have been seeking to manage Big Data through both distribution and scaling for more than three decades.

This need has led to the birth of a new class of systems referred to as NoSQL which are being widely adopted by various organizations [9]–[12]. These types of databases are different than traditional relational databases. NoSQL removes support that is found in traditional relational databases, such as SQL language, transactions, and other additional features found in traditional relational databases in exchange for faster reading, faster writing, larger storage, ease of expansion, and low cost [13]. It is also important to note that open source relational database management systems have a shortage of cloud

features and organizations have to opt for commercial solutions, which can get very costly making NoSQL databases more attractive [8]. In the domain of NoSQL, the MapReduce application [14] and the open source implementation known as Hadoop [15] has also seen widespread adoption in industry and academia alike. Hadoop is an open-source framework that was designed for distributed processing of large data sets across clusters of machines. MapReduce is a library developed by Google research lab to process large amounts of data. These tools will be explained further in the background sections of the thesis.

Due to early stages in development of these applications, organizations have been increasingly facing challenges pertaining to Big Data environments. The first challenge is coming up with an objective way to evaluate the HBase clusters with faster performance. There is a high number of possible ways to configure HBase clusters which leaves open the question of what approach should be taken to address this challenge and how can the complexities of this challenge be controlled in a reasonable way.

This leads to the second challenge of finding which factors would have an impact on the performance of the HBase cluster. For instance, will having larger number of machines verses smaller number of machines affect performance? Will having a particular HBase data schema influence performance over choosing a different data schema?

Furthermore, measuring response times over a larger space can take an extensive amount of time. As an outcome of this, these experiments can limit resource availability

on a resource-limited cloud service and can be very costly in terms of dollars on a paid cloud service due to the amount of instances running over a longer period of time. This leads us to the third and final challenge, which is extrapolating a model that can help us predict response times in a larger space. Having a predictive model would enable researchers to have the option of approximating response times with only a smaller space, allowing for shorter periods of experimentation.

This thesis addresses these questions by illustrating the process in a transportation traffic domain scenario. Furthermore, this thesis provides a framework for optimizing Big Data topologies by comparing different metrics and extrapolating a model from these metrics.

1.2 Research Objectives

The main research objective is to quantify and model the performance of HBase clusters.

To reach the research objective we are going to focus on the following research questions:

- **Research Question 1:** How do we objectively compare the performance of different HBase clusters?
- **Research Question 2:** Which factors have an impact on performance for HBase?
- **Research Question 3:** How can we model the response time of an HBase cluster?

To answer the questions above we start with the following hypotheses:

Hypothesis 1: To answer the first question, we have to consider that there are a large number of possibilities for constructing the clusters. If we consider all clusters have the same cost, then we can limit their numbers; therefore starting with a cost we hypothesize that we can build a set of configurations that can be compared.

Hypothesis 2: To answer the second question, we hypothesise that the schema and the nature of the configuration of the database is going to have an impact on the response time.

Hypothesis 3: For the third question, our hypothesis is that we can build an experimental model (linear or non-linear) and use it for prediction.

1.3 Methodology and Research Contributions

Our research methodology is based on experimentation and the use of public traffic data and public cloud infrastructure. Based on the experiments, we made the we made the following contributions:

We demonstrated that we can model the response time of an HBase cluster as a linear model. The parameters of the model depend on the cluster type and schema. We show that the model can be constructed with few experiments and then can be used across a large space to predict the response times. We found that we can abstract all our experiments by providing a linear regression formula.

A process methodology was introduced for evaluating clusters with faster performance and modelling the clusters. We also utilized this methodology in real-time. This methodology consists of three main steps and two iterative processes. First,

data files are imported into a MySQL database in bulk. Secondly, the data from MySQL is then migrated to HBase. This part is repeatable in order for comparing different data schemas after the best cluster has been found. Lastly, workloads are executed on the clusters, in which response times are compared and modelled. This process is also repeatable so that response times are compared for different topologies and a regression model can be extrapolated.

Adding MD5 to the Row-Key of a 2-Dimensional schema resulted in significant improvement in response time. Due to HBase ordering row-keys in lexicographical order and the way HBase groups keys per region, a row-key without an MD5 is known to cause what is called “region hotspotting”. Region hotspotting is the phenomenon where one or only a few machines (or RegionServers) get large amounts of client traffic therefore causing performance degradation and potentially leading to region unavailability [16]. We found that overall performance was dramatically affected and that adding an MD5 resulted in significantly faster results.

Larger clusters were found to perform faster with out-of-the-box Cloudera settings. When executing workloads on different clusters, we found that clusters with the most instances performed the fastest in terms of response time while clusters with the lowest amount of instances performed the slowest. Clusters were configured to have the same maximum capacities but different amounts and types of instances. This means that the out-of-the-box settings do not fully utilize clusters and more research needs to be done in the future to better configure these clusters for maximum utilization.

1.4 Thesis Organization

The thesis is structured as follows. Chapter 2 provides a background on important concepts related to this research. Chapter 3 presents related research in the Big Data field. Chapter 4 presents the details about the methodology for comparing clusters and presents our original contributions. Chapter 5 describes the experiments and results that validate the methodology. Lastly, we summarize the thesis and present possible work for the future in Chapter 6.

2. Background

This chapter describes the background work, specifically the concepts and tools that are used in this thesis. We provide a brief overview of the main areas, namely cloud computing, SAVI, Hadoop, HBase, and Cloudera as well as how these concepts and tools are used in our thesis.

2.1 Cloud Computing

According to the National Institute of Standards and Technology (NIST), cloud computing is a “model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (eg. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.” [17]

Developers with innovative ideas for new Internet services no longer require large capital in order to purchase hardware or the human expense to operate it nor do they need to be concerned about buying more network capacity than they need to in order to meet user expectations. Rather, they pay for resources as they need them or in other words, “pay-as-you-go” [18]. The NIST calls this “On-demand self-service” and extends the definition by saying that consumers can do this automatically without requiring human interaction with each service provider [17].

Another characteristic of cloud, according to NIST, is that capabilities can be accessed through standard mechanisms that promote use by heterogeneous thin or thick

client platforms (eg., mobile phones, tablets, laptops, and workstations) as they are available across the network. This is a feature known as “Broad network access” [17].

In Cloud Computing, the provider’s computing resources are pooled to serve multiple consumers, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Customers generally have no control or knowledge over the exact location of the provided resources but may be able to specify the location of these resources at a higher level (eg. country, state, or datacenter). This is known as “resource pooling” [17]. Examples of resources include storage, processing, memory, and network bandwidth.

Cloud computing has also allowed for rapid elasticity of capabilities, meaning that their systems are able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic fashion, such that at each point in time the available resources are comparable to the current demand as closely as possible [17],[18].

Cloud systems can also automatically control and optimize resource use (eg. storage, processing, bandwidth, and active user accounts) by a measurement (such as pay-per-use). This is known as a “measured service” [17].

The three most popular cloud service models are [8],[17]:

1) Infrastructure as a Service (IaaS)
2) Platform as a Service (PaaS)
3) Software as a Service (SaaS)

IaaS is a capability to provision processing, storage, networks, and other computing resources where consumer is able to deploy and run software, which can include operating systems and applications [8],[17]. The consumer does not manage or control the underlying infrastructure but has control over operating systems, storage, and deployed applications, as well as possible limited control over other select networking components such as host firewalls.

Moreover, PaaS is where a provider gives the consumer the capability to deploy onto the cloud infrastructure applications created using programming languages, libraries, services, and tools supported by the provider [8],[17]. The consumer however does not manage or control the underlying cloud infrastructure which includes network, servers, operating systems, or storage, but has control over the deployed applications and possibly settings for the application-hosting environment.

Lastly, SaaS is the ability for the consumer to use the provider's running applications on the cloud infrastructure [8],[17]. The applications can be accessed from various client devices through either a thin client interface such as web browser or a program interface. The consumer does not manage or control the underlying infrastructure, including network, servers, operating systems, storage, or individual application capabilities, unless the application includes user specific application configuration settings.

Cloud Computing also includes several different deployment models [17]:

1) Private Cloud

2) Community cloud

3) Public cloud

4) Hybrid cloud

Private cloud is a cloud infrastructure for exclusive use by a single organization comprising multiple consumers (eg. business units) [17],[18]. It may be owned, managed, and operated by the organization, a third party, or a combination of them. It may also exist on or off the organization's premises. Examples of private cloud vendors include Rackspace Private Cloud[1] and HP Helion[2].

Furthermore, community cloud is a cloud infrastructure used exclusively by a specific community of consumers from organizations that have shared goals [17]. It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some of them.

Thirdly, a public cloud is a cloud infrastructure open for use by the general public [17],[18]. It may be owned, managed, and operated by a business, academic, or government organization, as well as a combination of them. It exists on the premises of the cloud provider. Examples of public cloud include Amazon EC2[3] and Rackspace1.

A combination of the above models (two or more) is known as a hybrid cloud [17]. These models are bound together by standardized technology that enables data and application portability (eg. load balancing between two clouds).

[1] http://www.rackspace.com
[2] http://www8.hp.com/ca/en/cloud/helion-overview.html
[3] http://aws.amazon.com/ec2

In this thesis, we use a PaaS known as Smart Application Virtual Infrastructure (SAVI)[4] testbed (described in the next section) to provision and decommission instances (which are also known as virtual machines) with varying flavors (medium, large, and extra large). The flavors of these instances will be explained in the Chapter 6 (experiments and results) portion of this thesis. We describe SAVI in the next section.

2.1.1 Smart Application Virtual Infrastructure (SAVI)

According to Pan et. al., numerous nations are investing into national-scale research programs focused on the Future Internet and applications [19]. These research programs have been addressing content oriented paradigms, mobility, and ubiquitous access to networks, cloud-computing-centric architectures, security, and experimental testbeds. In Canada, Smart Application Virtual Infrastructure (SAVI) project (involving several universities and industrial partners) was established to address the design of future application platforms built on flexible, versatile, and evolvable infrastructure that can be readily deployed, maintained, and decommissioned. These applications can be large in scale, short-lived, and distributed [20].

A platform known as Smart Application Virtual Infrastructure (SAVI) testbed (TB) is used for deploying the virtual machines. The SAVI TB platform architecture includes components and interfaces. The interfaces are for both internal and external communications.

The SAVI TB is comprised of the following physical entities:

[4] www.savinetwork.ca

1) Core Nodes
2) Edge Nodes
3) SAVI Network
4) SAVI TB Control Center

Resources on Core nodes and Edge nodes are used to create applications. These resources include computations, storages, networks, optical access, wireless access, and reconfigurable hardware resources. The Core nodes are contained by conventional cloud computing resources (compute, storage, and basic networking). On the other hand, the Edge nodes include more advanced resources such as reconfigurable hardware resources. The SAVI network is also considered a resource in the SAVI TB.

Core nodes, Edge nodes, and SAVI TB control center are all unified by the SAVI network which is a dedicated research network. Core and Edge Nodes together are referred to as the extended cloud in SAVI.

Edge Nodes are deployed on sites located at participating universities (including York University). The Core Nodes are deployed in fewer universities compared to the Edge Nodes. For instance there can be one or two Core nodes across SAVI TB platform. For this thesis, the Core node is used, which is hosted in University of Toronto.

SAVI testbed uses Open Stack[5], which is open source software for building clouds. This software also includes a "portal" user interface, accessible by any browser, allowing for easy provisioning and decommissioning of machines for building

[5] http://www.openstack.org/

applications and experiments. These applications and experiments can be deployed to different components of the SAVI TB [20]. In SAVI, applications and experiments are different. Applications are aimed at delivering features to end users and need to guarantee a service level where as experiments are shorter-lived, used by researchers, and aimed at gathering measurement data or user feedback. However, both applications and experiments are treated equally by SAVI TB.

An application or experiment is deployed on SAVI TB by allocating slices of resources to that application or experiment. All SAVI resources are virtualized in SAVI TB and allocation to each application or experiment is performed by the SAVI TB Control and Management plane.

2.2 Big Data

Big data is a term used for massive data sets having large, varied and complex structure that pose difficulties in storing, analyzing, and visualizing for further processes or results [6][21]. It is also a popular term to describe the exponential growth and availability of data, both structured and unstructured [6].

The characteristics of Big Data were first described in 2001 by Laney as variety, volume, and velocity [7]. Variety is the different varieties of data (such as photos, audio, video, etc). Volume is the amount of data storage needed for the data (terabytes, petabytes, etc). The velocity is the speed of data coming in and going out (real time, periodic, batch, etc). To date, these attributes have become the defining attributes of Big Data. However, authors and business specialists extended these defining attributes with

further aspects such as dedicated storage, management, and analysis techniques [22],[23],[24]. IBM further added a fourth V known as veracity, emphasizing the aspect of data quality [25]. Ebner et al. [26] has taken these extensions into account and has defined Big Data as “as a phenomenon characterized by an ongoing increase in volume, variety, velocity, and veracity of data that requires advanced techniques and technologies to capture, store, distribute, manage, and analyze these data”.

The quest for conquering challenges posed by management of big data has led to a wide range of systems [8] such as Hadoop and HBase. In this thesis we use Hadoop, HBase and Cloudera, which are talked about in the next sections.

2.2.1 Hadoop

The Apache Hadoop software library[6] is a framework that allows for distributed processing of large data sets across many instances and consists of several modules (including HDFS and MapReduce). It is designed to scale from single to thousands of nodes, each offering local computation and storage. Rather than relying on hardware to deliver high-availability, HDFS itself is designed to detect and handle failures at the application layer [15].

Moreover, HDFS consists of the Master/Slave architecture [27] in which a master server controls the overall distributed file system spanning many servers. The HDFS architecture is divided into nodes called Name nodes and Data nodes. The architecture is illustrated in Figure 1 [27].

[6] http://hadoop.apache.org/

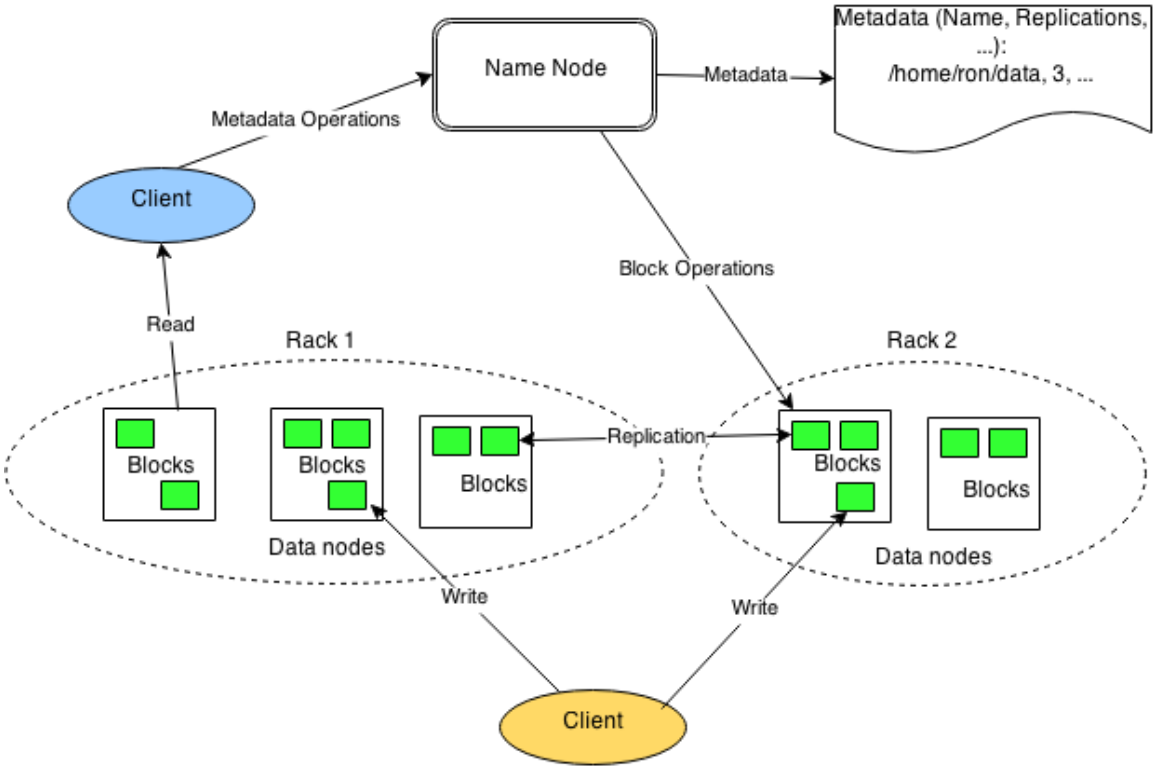


Figure 1. HDFS Architecture

Furthermore, the Name node contains all information of HDFS metadata, including where these data nodes are and controlling the replication of the data blocks. Blocks of data are replicated across data nodes so that if any block fails, data is not lost. This includes data nodes that are on a different rack (physical location of machine). Each Data node runs on a separate machine and stores HDFS data in files in its local file system [27].

The Data node has no knowledge about HDFS files and stores each block of HDFS data in a separate file in its local file system. The Data node does not create all files in the same directory but rather uses an algorithm to determine the optimal number

of files per directory and creates subdirectories appropriately. Creating all local files in the same directory may not be optimal because the local file system might not be able to efficiently support a huge number of files in a single directory. When a Data node starts up, it scans through its local file system and generates a list of all HDFS data blocks that correspond to each of these local files. It then sends this report to the Name node [15]. Next we are going to look at MapReduce.

MapReduce is a library developed by the Google research lab to process large amounts of data [14]. Hadoop has a variation of the MapReduce known as the Hadoop MapReduce framework which works on HDFS [27]. When using MapReduce, the user of the library expresses two functions: map and reduce. Map, written by the programmer, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library then groups together all intermediate values associated with the same intermediate key and passes them to the Reduce function. The Reduce function, also written by programmer, accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values. This allows users to handle lists of values that are too large to fit in memory [14]. In the next section, we look at HBase and how it improves upon HDFS.

2.2.2 HBase

According to HBase documentation, HDFS is well suited for storage of large files but HDFS documentation states that it is not a general purpose file system and does not provide fast individual record lookups in files [28]. HBase, on the other hand, provides fast record loops and updates for large tables.

HBase[7] is an open-source database modeled after Google's BigTable [29]. HBase is currently being used in large data centric applications such as Facebook and Twitter because of its portability and massive scalability [11], [12]. It is part of Apache Hadoop project and runs on top of HDFS, providing capabilities found in Google BigTable, including fault tolerance when storing large quantities of sparse data. It also adds to HDFS functionality by allowing for random, real time, read and write access to large data. HBase applications are written in Java utilizing HBase API.

Moreover, HBase has what are called RegionServers, which are built on top of the data nodes of HDFS and a Master which is built on top of the Name node of HDFS. This is illustrated in Figure 2. The master is in charge of coordinating and monitoring the RegionServers in the cluster. RegionServers in turn, are responsible for serving and managing regions. Regions are chunks of rows of a table.

[7] http://hbase.apache.org/

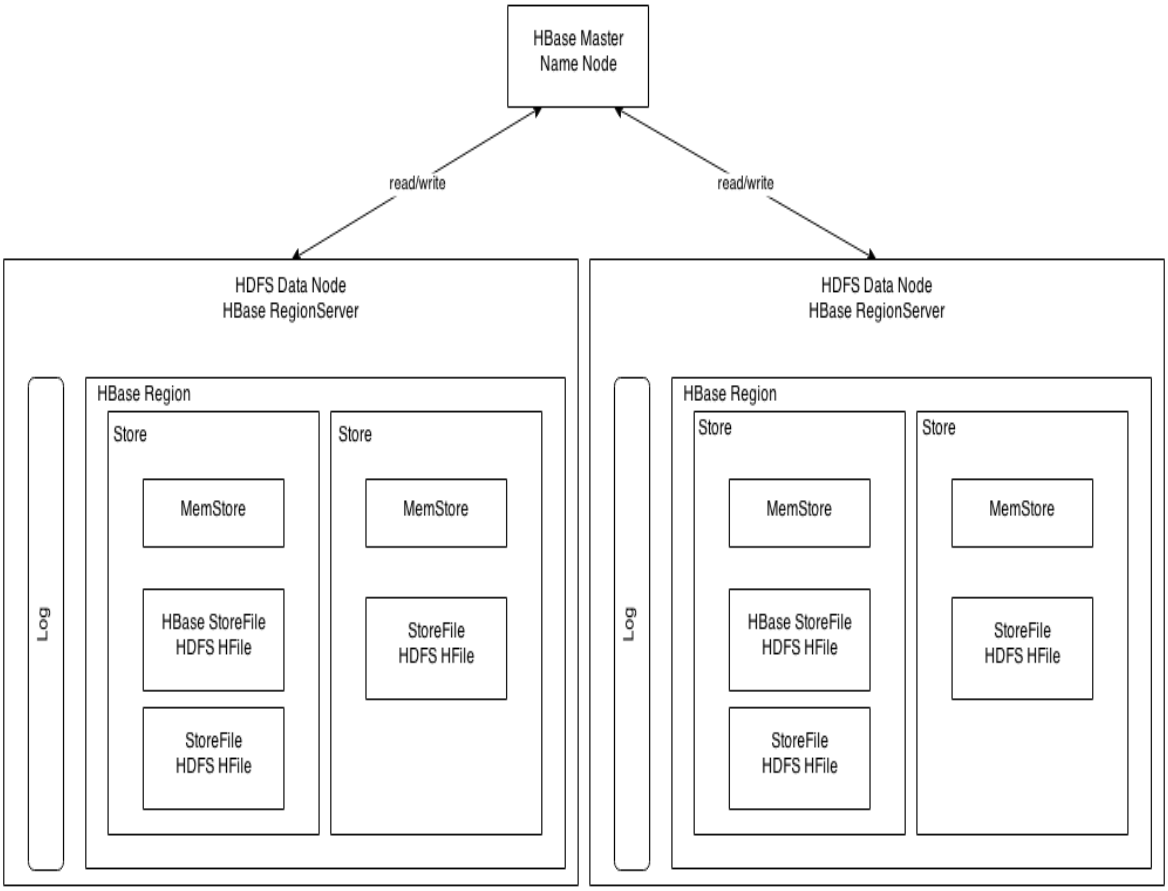


Figure 2. HBase Architecture

Furthermore, HBase is made up of a table, which is made up of multiple rows. Each row contains a row key. Rows are sorted alphabetically by the row key as they are stored, therefore the row key design is important as the goal is to store data in such a way that related rows are grouped together by row keys. Rows also contain one or more columns. Columns include a column family and a column qualifier, delimited by a colon character. Column families group a set of columns and their values. Each column family has a set of properties relating to storage, such as how data should be compressed,

whether values should be memory cached, etc. Each row in the table has the same column families, but a given row might not store anything in a given column family if there is no data. Column families are specified when the HBase table is created. Column qualifiers are added to column families to provide the index for a given piece of data. As column qualifiers can change greatly between rows, they are considered mutable. The combination of a row, column family, and column qualifier is a cell. A cell contains a value and a timestamp, which represents the value's version. Timestamps by default are represented by the time on the region server when the data was written and is written alongside the value [30].

The hierarchy of the region is as follows [28]:

- Table (HBase Table)
    - Region (Regions for the table)
        - Store (Store per ColumnFamily for each Region for the table)
            - MemStore (One MemStore for each Store for each Region for the table)
            - StoreFile (0 or more StoreFiles for each Store for each Region for the table)
                - Block (Blocks within a StoreFile within a Store for each region for each table)

When data is imported, it writes to the region's MemStore (in-memory space) and when the MemStore gets full, it is flushed to StoreFiles on HDFS [31]. A StoreFiles is a

façade of HFile (in HDFS).

As data increases, there may be many StoreFiles in HDFS which can degrade read/write performance. Thus HBase will automatically pick two smaller StoreFiles and rewrite them into a bigger one in a process known as “minor compaction” [31]. For some situations, or when triggered by a configured interval (once a day by default), a major compaction runs automatically. Major compactions will drop the deleted or expired cells and rewrite all the StoreFiles, which will usually improve performance. However, during this process, a major compaction rewrites all of the Stores’ data and therefore a heavy volume of disk I/O utilization and network traffic might occur during the process. This would not be acceptable on a heavy load system with many users. Along with each RegionServer, there is a log file known as a “HLog”. A “HLog” records all edits to the StoreFiles. It is also called the HBase “write-ahead-log” [32].

2.2.3 Cloudera

“Cloudera Distribution Including Apache Hadoop” (CDH) or “Cloudera” [8] is a distribution of open-source Apache Hadoop-based tools. It comes in both free version and paid version. The paid version is known as Cloudera Enterprise and the free version is known as Cloudera Express. Cloudera Express is used in this thesis. Cloudera Express uses Hadoop Distributed File System (HDFS) and Hadoop MapReduce as the main core elements.

The remaining tools of Cloudera Express allow for easy integration between Big Data Tools, as well as support for data management (including monitoring tools), data

[8] http://www.cloudera.com/

accessibility, data migration, and querying. In this thesis, we use Cloudera Express Manager to set up, monitor, and manage the Hadoop clusters.

These tools include:

1) Hadoop Distributed File System
2) Hadoop MapReduce
3) HBase
4) ZooKeeper[9]
5) Sqoop[10]

The first three tools are described in previous sections under Hadoop and HBase respectively. ZooKeeper is an open-source centralized service used to enable highly reliable distributed coordination. It acts as a centralized manager for the entire cluster in terms of electing a master server, managing group membership, and managing metadata [33]. It was designed for developers to focus mainly on their application logic rather than coordination.

Sqoop is short for “SQL to Hadoop” [34]. It is service used to transfer bulk data from relational databases such as MySQL to Apache Hadoop data stores (such as HDFS and HBase) and vice versa. Taking advantage of MapReduce, Hadoop’s execution engine, Sqoop performs the transfers in a parallel manner. Sqoop is executed using command-line statements in shell.

[9] http://zookeeper.apache.org
[10] http://sqoop.apache.org

2.3 Summary

In this chapter, we gave a background about relevant concepts regarding this research. We described how cloud computing allows for on-demand resources over the internet and saves on cost. We also talked about SAVI testbed, a cloud computing platform, which is used in this thesis. Lastly, we went over the concept of Big Data and described the software that facilitate in its management (Hadoop, HBase, and Cloudera).

3. Related Work

In the last chapter we provided an in-depth background of Big Data. However, there are many challenges facing the Big Data research field. In this chapter, we are going to analyze these challenges and the existing contributions made by the research community.

The chapter is organized as follows:

Firstly, in section 3.1 we review different approaches to modelling data schemas and how this relates to our research. Secondly, in Section 3.2, we present literature relevant to Big Data configurations for large data processing. Lastly, in Section 3.3, we look at how existing literature shows that certain performances in Big Data clusters can be modelled and used to predict values in a larger space.

3.1 Data Schemas for Big Data

Researchers have looked at which data schemas would be optimal for querying data in a Big Data context. Hadoop allows for relatively more data structure flexibility as it does not have the traditional column and rows structures, which can cause confusion as to which data schema would be suitable for different data domains. It is important to also note that an unsuitable data structure may cause poor performance. For instance, HBase currently does not perform well with anything above two or three column families [35]. This calls for a structural systematic method for NoSQL database design as it is an important problem for researchers and practitioners.

Han and Stroulia [36] have studied performance of data schemas by running workloads on two different datasets. The first dataset was a cosmology dataset and consisted of 321,065,547 particles from 9 snapshots with a total size of approximately 14 GB binary format. Another dataset they used was Bixi, a public dataset collected by a bicycle renting service in Montreal, Quebec, Canada which totaled 12 GB and contained 96,842 data-points for all the stations.

Three schemas were used to test performance of queries on the data sets where the second two schemas would be three dimensional. The version dimension would act as the third dimension. A version dimension specifies a cell and by default, HBase has 3 versions maximum per cell. If data with the same row-key and column as another data is imported, that older data will not be replaced, rather it will be "versioned". In the case of Bixi data, if they wanted to store values by day, they would use the date and station id as their row-key (no time/hours/minutes). All the 1440 records for one day would be stored on the same cell through "versions" (hence there would 1440 versions for each cell). Han and Stroulia found that using the third dimension of HBase improves performance and that the distribution of data across cluster nodes highly impacts performance [36].

However, Han and Stroulia also mention that in HBase "many functions are not very stable, including functionalities around versioning". According to HBase's official website book regarding schema design, it is not recommended setting number of max versions to a level exceeding hundreds of versions or more as this will greatly increase the store file size [37].

In their section entitled “Schema Smackdown”, HBase authors specify that rows should generally be used over versions if the versions would be significantly over the maximum versions (being 3). They also give preference to rows over columns in extreme cases when deciding between wide tables such as having 1 row with 1 million attributes or having tall tables such as 1 million rows with 1 column apiece [38].

In addition to this, transforming complex relational databases into HBase is another problem that is increasingly faced among organizations as not only does the schema impact performance, but the data representation may have to be consistent with the database it is migrating from. Chongxin Li presented an approach for this problem and demonstrated how to follow this approach in a case study [39]. This approach comprised of two phases.

The first phase would have the relational schema transformed into an HBase schema utilizing a set of guidelines. The first rule in these guidelines is to group correlated data in a column family. Li refers to user information, access patterns, and write patterns in a blog domain as examples of grouping correlated data.

However, relationships between tables need to be taken into account which leads to the second rule, which is for each side of a relationship one must add the foreign key references of the other side if it needs to access the other side’s objects. In relational tables, foreign keys are used to maintain a relation (one-to-one, one-to-many/many-to-one, and many-to-many). They are also used to reference parent and child objects. For One-To-One relationship we do not worry about such a relationship as the foreign key is

treated as an ordinary column in HBase and can be grouped with other columns based on the first rule of these guidelines. In One-to-Many relationships, foreign keys are only put on the “many” side of a one-to-many relationship since multiple values are not allowed in RDBMS because of Normal Form 1 however HBase allows multiple values to be grouped together in a column family. To reference objects on the “many” side, Li suggests a new column family to be created on the “one” side to contain a set of foreign keys of the “many” side. For a Many-To-Many relationship, Li suggests using a third table to manage this relationship where foreign keys for both tables are kept or to create new column families to capture row keys of both sides.

Although these references are still referred to as foreign keys by Li, they are different from those of a relational database as in a RDBMS these relationships are guaranteed by the database itself that data is always in a consistent state and the user data cannot violate the foreign key constraint however in HBase, applications have to ensure these references instead.

The third rule is to merge attached data tables to reduce foreign keys. This can be done by using a table that contains the most important data as the “main table” if it can be used independently in the application. If a table has only one foreign key and this must be used, then a reference table is created known as an “attached table”. Data with the same foreign key in the “attached table” can be combined into a single row of the “main table” based on the foreign key.

In the second phase, relationships between the source and destination schemas are expressed as a set of nested schema mappings which would be employed to create a set of queries or programs to transform the source data into the target representation. Li gives a practical example of this by using Tableau to represent mapping algorithms for a basic blog. Tableau is a way of describing all the basic concepts and relationships that exist in a schema. He then shows these nested mapping representations in query-like notations as a way for the expressions to be employed in a query.

3.2 Configuring Topologies for Processing Big Data

Configuring cloud clusters for large data has also been a growing issue. It is important to understand what the trade offs are for deploying fewer machines with higher resources per machine versus deploying more machines with fewer resources per machine as this decision can have an impact on both performance and cost.

To begin with, the cloud environment allows for heterogeneous hardware and resource demands. Lee et al. have found that it is important to exploit these features to make data analytics in cloud efficient [40]. They present a system architecture to allocate resources to a Hadoop data cluster in a cost effective manner. In this architecture, nodes are grouped into one of two pools: (1) long-living core nodes to host both data and computations and (2) accelerator nodes that are added temporarily to the cluster when more computing power is needed for workloads. A cloud driver is used to manage these nodes and makes decisions on adding/removing nodes based on the hints provided by the

users when they submit the job. Hints include memory requirements, ability to use special features like GPUs, and the deadline.

They experimented with two queries and found that using certain configurations had higher performance per cost compared to other configurations because some machines had faster CPUs at lower prices than “larger” machines [40]. However, the machines with the lower price point had less memory, which might be of no use for jobs requiring a large amount of memory per machine. They also found that using more accelerators can cost less while having faster performance due to the fact that the instances are not being used for so long. The number of users who would use the data was not addressed, which can make a significant difference in how the topology should be created.

Furthermore, in another contribution, Zaharia et al. [41] found that MapReduce does not perform well in heterogeneous Hadoop clusters. Hadoop assumes that any detectably slow node is faulty. However, nodes can be slow for other reasons. According to Zaharia et al. in a non-virtualized data center, there may be multiple generations of hardware. In a virtualized data center where multiple virtual machines run on each physical host, such as Amazon EC2, co-location of VMs may cause heterogeneity. Although virtualization isolates CPU and memory performance, VMs compete for disk and network bandwidth.

Zaharia et al. state that heterogeneity of machines (mixed instances with various sizes) seriously impacts Hadoop’s scheduler [41]. The scheduler uses a fixed threshold

for selecting tasks to speculate (that is, if a node happens to be slow, the tasks are copied to a faster node to finish the computation sooner) and therefore, too many speculative tasks may be launched, taking away resources from useful tasks. Also, the wrong tasks may be chosen for speculation first because the scheduler ranks candidates by locality. For example, if the average progress was 70% and there was a 2x slower task at 35% progress and a 10x slower task at 7% progress, then the 2x slower task might be speculated before the 10x slower task if its input data was available on an idle node.

Zaharia et al. designed a Longest Approximate Time to End (LATE) scheduler which is a new speculative task scheduler to try to compete with this issue, which adds features to the Hadoop task scheduler [41]. The primary feature behind this algorithm is that it always speculatively executes the task that the system thinks will finish farthest into the future, because this task provides the greatest opportunity for a speculative copy to overtake the original and reduce the job's response time. This is contrast to the original heuristic that was used which was comparing each task's progress to the average progress which would have worked well for homogeneous environments where poorly performing nodes (stragglers) were obvious. In this case, LATE is robust to node heterogeneity as it only relaunches slowest tasks and only small number of tasks. It also takes into account node heterogeneity when deciding where to run speculative tasks. Lastly, LAST focuses on estimated time left rather than the progress rate. LATE speculatively executes tasks that will improve job response time rather than individual slow tasks' response time. According to Zaheria et al. LATE can improve Hadoop response times by a factor of 2 in clusters with 200 virtual machines on Amazon EC2.

3.3 Modeling

Researchers have also focused on modeling performance in Hadoop clusters. Song et al. looked at proposing a simple framework to predict performance of Hadoop jobs [42]. They found that the execution time for map and reduce had a linear relationship with the amount of data (64M to 8G for 4 different kinds of jobs). They did this through modeling the relationship through linear regression. They also compared the prediction from smaller samples for both map and reduce tasks to actual values from the larger samples in order to see what the error rate is. The error rate was minimal, meaning that they can approximately predict the execution time for both map and reduce tasks.

In another research contribution, Bortnikov et al. explores performance bottlenecks in MapReduce tasks. According to Bortnikov et al., extremely slow tasks are a major performance bottleneck in MapReduce systems [43]. These researchers came up with a way to predict execution bottlenecks in MapReduce clusters. They came up with the slowdown predictor model, which is a “machine-learned oracle for MapReduce systems forecasting execution bottlenecks”. The predictor takes profiles of the tasks and the hardware, and then estimates the task’s slow down. The predictor can be applied during the assignment of the task or during the execution. The predictor employs a popular gradient-boosted decision tree algorithm [44], which is an “additive regression model comprised of an ensemble of binary decision trees.” [43] In the case of the slowdown predictor model, each binary tree is split on some feature at a specific value,

with a branch for each of the possible outcomes. Each leaf node contains a score, which corresponds to the decision path. The resulting prediction is the sum of the scores returned by individual decision trees. They evaluate their model on real-time data sets on a production Hadoop cluster at Yahoo![11]. They found that the prediction for mappers was more accurate than for reducers.

3.4 Summary

In this section we talked about current research that relates to this thesis. We firstly spoke about data schemas and how they influence performance of an HBase cluster. We then talked about how researchers have developed approaches for improving data processing for Big Data through cluster configurations in addressing its challenges. Lastly, we illustrated how existing literature allows for modeling performances of Hadoop clusters, which can be used to objectively evaluate performances of existing clusters.

[11] www.yahoo.com

4. A Process for Designing Big Data Configurations

One of the main challenges in designing big data solutions is the design of the physical configurations. By configuration we mean the definition of the schema and the runtime components to access the data. In case of HBase, a configuration is made of the HBase schema and the physical topology of Hadoop. When designing the configurations, cost and performance are two main and conflicting goals. Design decisions include the number and the type of VM instances that Hadoop uses. For example, is it better in terms of cost and performance, to have a larger instance, a large amount of small instances, or a combination of both?

To characterize the performance of configurations, many experiments are needed. Experiments are costly, in terms of time and in terms of infrastructure since they are performed in cloud as well. Therefore, a natural question is: can we deduce a performance model for a given configuration? Also, how can someone extrapolate a model from a limited number of users to predict the response time for a larger number of users? This is also important as how you decide to deploy your topology will not only impact performance, but also cost.

This chapter addresses the above challenges by focusing on the following research questions:

- How do we compare different HBase clusters to objectively evaluate topologies with fastest performance?

- Which factors have higher influence on performance for HBase?
- How can we model the response time of an HBase cluster?

The remainder of this chapter is organized as follows: Section 4.1 presents a general high-level repeatable methodology that can be used to generate a configuration and characterize performance. Section 4.2 focuses on a method to generate configurations. Section 4.3 illustrates performance characterization and modelling. Section 4.4 presents a summary and the conclusions.

4.1 An Iterative Process

In this section, we describe the iterative process, which is the methodology we use to compare performance of big data configurations. This methodology consists of steps for transforming the data from non-relational databases to relational database.

The process is shown in Figure 3 on the next page. It has two iterative sub-processes:

Topology design. For a given schema, we iterate experimentally between different HBase topologies until we obtain the desired results, that is, the performance and cost specified by the requirements.

Data schema design. Based on the SQL schema, a set of possible HBase schemas are generated as being possible solutions to design requirements.

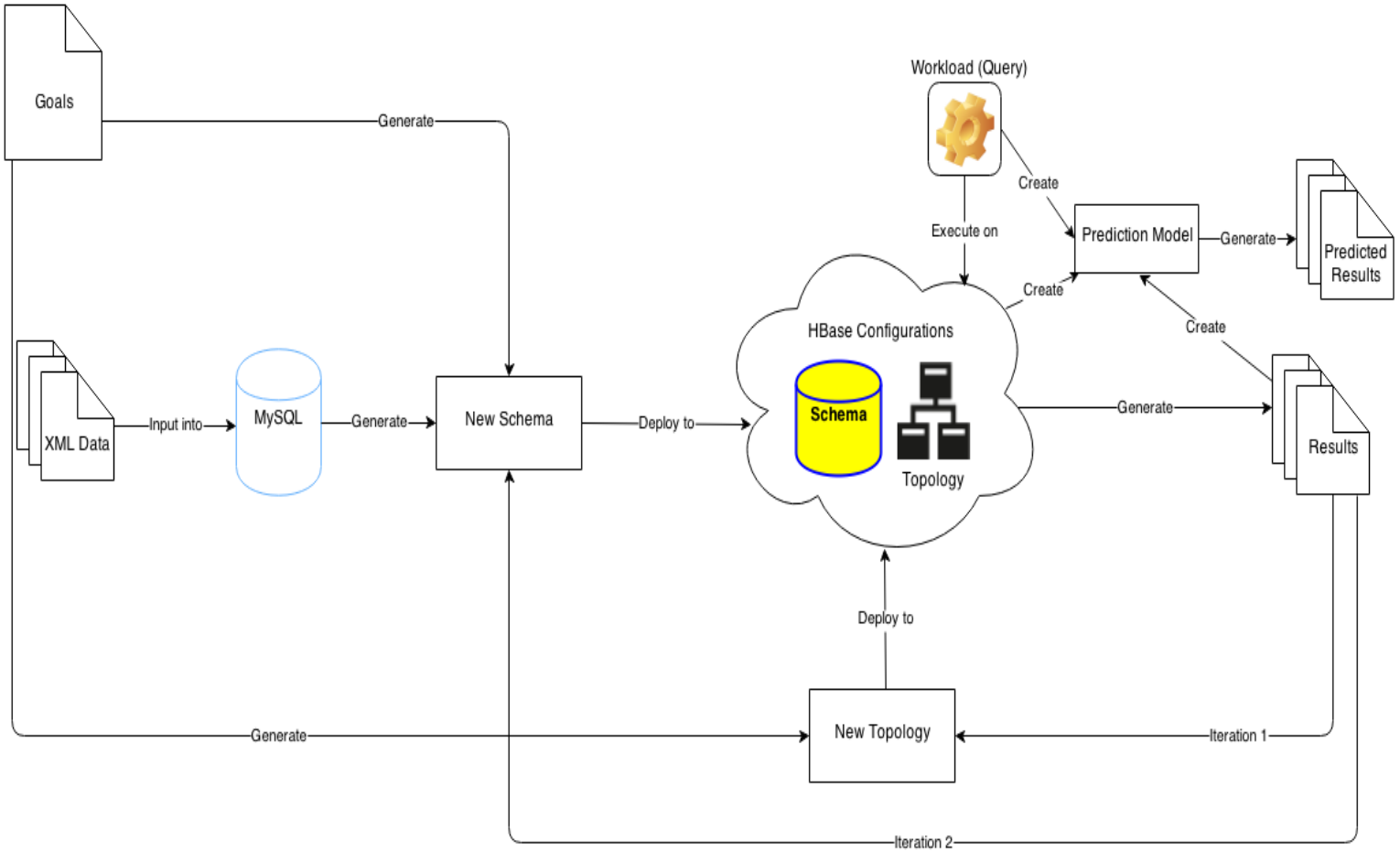


Figure 3. Iterative Process

Firstly, we assume we have the big data configurations goals already in place, which are the results of the requirements engineering phase. These goals are either retrieved from an end-user, an internal source, or are mutually agreed upon by many different stakeholders. Examples of these goals can include fast performance and low cost. These goals can also be very specific (that is, they can include a budget and specific service level agreements). During this first step, we also retrieve large bulk data files from a data source in order to use existing real data. The source for this data could be either internal or external to the organization or company. We expect a large amount of real data to give a more accurate picture when assessing the schemas and clusters. The non-relational database (HBase) used in this thesis distributes rows across machines, which also means that different data can have different distributions. We assume in our method that we have real data. Having data that is not real (synthetic data that is generated randomly) is an option as long as it is generated to be similar to real data. In order to have an accurate reading of how the non relational database will perform in a production environment, it is imperative to have a large set of real data.

Furthermore, the bulk data files are then converted and imported into a relational database. This relational database acts as a "back up" for the data and for verifying query results on the non relational database. We also use a relational database as it only requires one machine (whereas a non relational database like HBase may require many machines) and therefore is a less costly way to have a backup of the data. The data is then copied

from the relational database to the non-relational database. In this step we can either adopt a commercial off-the-shelf product or create an application in house for loading the data. In our thesis we use a tool known as Sqoop that has a built-in MapReduce for faster transfer of data. By having a standardized tool to transfer data from one database to another in a quick fashion allows for researchers to save time in this regard. This is another benefit to having the data stored into the “back up” database, as the alternative would be to repeatedly import large data files individually or merge these files together before importing which would be otherwise heavily time consuming. Instead, we do the bulk loading process once for the relational database side, as this relational database includes an easy to use built-in tool for importing individual large data files and a simple bash script can import all these files at once into the relational database. This step also involves transforming the data and storing it into the database with a proper schema.

When speaking about transformation, we look at how the data should be represented. It is important to choose a proper schema for the relational database to quickly verify query results from the non relational database. An appropriate data schema for the non relational database is also needed as this can influence the performance. Even though schemas are compared on the fastest cluster after the performance comparisons (as will be illustrated later), there needs to be a proper distribution of keys across the cluster in order to utilize all machines. This is important for when coming up with the initial schema design as the baseline. The transfer of data happens twice for both compressed and non-compressed data. Compression allows for the data to be reduced in size allowing for the clusters to store more data without having to commission more

instances. In this thesis we use compression on data as the defining differentiating characteristic between two workloads but for other cases, researchers can use different types of workloads and can use more than two workloads.

After the data is transferred to the non relational database, the next step is to create different topologies by utilizing different machines from a set budget. These topologies can be deployed one at a time to save cost (if deployed on a public cloud) and physical resources (if deployed on a private cloud). Two workloads are executed on each of the individual clusters by utilizing what's known as a "scan" query in HBase on both "compression" and "non-compression" data. Scan queries retrieve records sequentially [45]. These workloads are executed by an application (written in Java) that allows for inputting a maximum number of users and an increment number of users. For example, we might want to test 1,500 users and increment by 500 therefore the application will execute 500 users first, then 1000 users, and then 1,500 users. We assume that there will be a linear relationship between number of users and response time due to the sequential nature of the scan queries (we explain more in section 4.3) thus we use the maximum number of users we would have wanted to execute, reduce that number by a large percentage (enough to save enough time and resources but also enough to create a relatively close approximation). We keep the same iterations that we would have done before for this maximum number of users. We do this because we can predict future results after constructing a performance model thus saving time. For example, let's say we would like to have a graph of the response time for 5000 users with iterations of 500 users executing workloads. Instead, we can reduce the number of 5000 to 2500 and

approximate the rest of the results using a prediction model in order to save time and resources. When we talk about resources we talk about both physical resource and cost of having the instances running for a certain amount of time. If the workloads were to be executed over the entire 5000 users, all these workloads can take days depending on the queries and the data size. This translates to a higher cost if a public cloud is used or higher resource consumption if a private cloud is used which would restrict other people's usage of the same infrastructure. Therefore reducing the workloads and approximating the response times saves both time and resource.

To create the prediction model, we take the results of a fewer numbers of users and use a linear regression algorithm to create the formula in order to find response times given a certain number of users as will be discussed in Section 4.3.

4.2 Cost and Configurations

In this section, we describe how we define a set of topologies based on a given set budget. We also describe how we compare the topologies for the first iterative process and for the second iterative process we describe how we compare the different schemas.

As stated in the last section, we define our goals as our baseline. In our thesis, we focus on performance and cost as being our goals. From this cost, we then construct different variations of topologies with different machine capacities and number of machines. We assume that the cost will equal the same across different clusters, which

have a set maximum capacity. We can see this from looking at the Amazon [46] and Rackspace [47] prices displayed in Table 1 and Table 2 respectively.

Instance Type	**VCPU**	**Random Access Memory**	**Solid State Drive Size**	**Price**
c3.large	2	3.75 GB	32 GB	$0.105 per hour
c3.xlarge	4	7.5 GB	80 GB	$0.210 per hour
c3.2xlarge	8	15 GB	160 GB	$0.420 per hour
C3.4xlarge	16	30 GB	320 GB	$0.840 per hour
C3.8xlarge	32	60 GB	640 GB	$1.680 per hour

Table 1. Amazon EC2 prices for each instance

From Table 1, we see that the prices for Amazon EC2 instances are the same for cost per capacity (with the exception of solid state drive space). The instances with names c3.xlarge, c3.2xlarge, c3.4xlarge, and c3.8xlarge are double, quadruple, and octuple the capacity size and price of c3.large respectively. For example, c3.xlarge has 4 CPUs and 7.5 GB of RAM which is double that of c3.large which only contains 2 CPUs and 3.75 GB of RAM. The storage difference is negligible as storage space for each cluster should have more than enough space for holding existing and future data. The VCPU and RAM are the most important parameters when determining performance.

Instance Type	**VCPU**	**Random Access Memory**	**Solid State Drive Size**	**Price**
Performance1-1	1	1 GB	20 GB	$0.037 per hour
Performance1-2	2	2 GB	60 GB	$0.074 per hour
Performance1-3	4	4 GB	80 GB	$0.148 per hour
Performance1-4	8	8 GB	120 GB	$0.296 per hour

Table 2. Rackspace prices for each instance

Table 2 shows that besides the drive size, the capacities per price (that is, the parameters that influence response times) are the same once again but for a different cloud service provider. Performance1-2, Performance1-3, and Performance1-4 are double, quadruple, and octuple the size of Performance 1-1 respectively. Here we see more proof that maximum capacity will have the same cost across clusters.

While constructing our topologies, we ensure that the machines running the Master nodes and Name nodes are the same capacity per machine across experiments but the RegionServer's and Data node's capacity can change from one experiment to the next, as well as the number of machines. This is done to define a scope that ensures that the comparisons are objective.

After these topologies are designed, a *new topology* is deployed on the cloud by using a platform such as Open Stack that allows for deployment of instances. As mentioned before, these topologies are created one at a time to save on cost and resources. Once a *new topology* is created, the appropriate tools are then installed on the

cluster of machines. These tools include Cloudera (HDFS, Hadoop MapReduce, HBase, ZooKeeper, and Sqoop). The data is transferred to the newly created HBase cluster with an appropriate schema.

An application is then deployed on an extra large instance external to the cluster and used to execute representative workloads (workloads are combinations of number of users and query types). After the workloads have finished, the application generates a data file, which shows the response times for each workload. The experiment is executed several times to reduce cloud variability in which performance can change time to time depending on the amount of traffic on the cloud, how many users are using the same physical machine, or any other factors that may influence performance.

In the next step, the results of the experiments are processed and used to create a *Prediction Model* that will be used to predict future results (*Predicted Results* in Fig 3). After this is done, the results are plotted onto a line graph and this process is iterated until desirable results are achieved. The process that is iterated includes: Generating new configurations, deploying them, running the same workloads, rebuilding the performance model, and graphing the new results with both the observed and predicted values.

After the desired results have been achieved, we further try to improve performance by comparing different schemas on the fastest cluster. Data is repeatedly transferred from MySQL to HBase with different transformations. We then use the same application to execute the same maximum number of workloads along with the same number of iterations as we did before for the topologies. The results are then inputted into

the *performance model* and a formula is outputted that allows seeing what the response time will be for a larger space of users. All the schemas are graphed onto a line graph and whichever one performs the fastest is chosen.

4.3 Modeling the Cluster

This section proposes a model for performance of the cluster, namely a quantitative relationship between the response time, number of users, and the type of configurations.

We propose the following model:

$$R(C) = A_C * x + \beta_C \quad (1)$$

where x is the number of users, C is the configuration, A_C is the slope of the configuration, β_C is the intercept of the configuration, and R(C) is the predicted response time for the particular configuration. We assume the model is linear because we use scan queries, which we found returned results on a first in first out (FIFO) basis due to the query's sequential nature [45]. This means, that there is a notion of queuing happening at each ServerRegion of the HBase cluster.

In order to quantify the configuration, we assume that A_C and B_C depend on the configuration and that the coefficients have to be determined experimentally in order to get these values.

To find the predictor equation for a set of data, we assume that we have a sample of *n* data points consisting of pairs of values of *x* and *y*, say $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$.

For example, if *n*=3 data points they could be (500, 23.1988), (1000, 40.3606), (1500, 63.84427) where 500, 1000, and 1500 are the users (x values) and the y values are the response times.

In order to use these sample values to estimate the model parameters, we want to find estimators A_c and B_c that minimize the sum of squared errors. By minimizing sum of squared errors, we mean that we want to produce a line closest to all n observations [48]. This means that we find the line that minimizes the distances of each observation to the line.

The method that produces these estimators is called the method of least squares. For a given data point, say the point(x_i, y_i), the observed value of R(C) is y_i and the predicted value of R(C) would be obtained by substituting x_i into the prediction equation:

$$R(C) = A_C * x_i + \beta_C \quad (2)$$

The deviation of the *i*th value from y from its predicted response time value is [48]:

$$y_i - R(C) = [y_i - (A_C * x_i + \beta_C) \quad (3)$$

Therefore the sum of squares of errors (SSE) of the y-values about their predicted values for all the *n* points is defined as:

$$SSE = \sum [y_i - (A_C * x_i + \beta_C)]^2 \quad (4)$$

The quantities of A_C and β_C that make the sum of squared errors (SSE) minimum are called the least squared estimates of the parameters.

Before finding the values of A_C and β_C we must first find the values of the sum of cross-deviations for *x* and *y* and the squared deviations of *x*. The line over the *x* ($\overline{x}$) and the *y* ($\overline{y}$) represent the averages of all *x*'s and all *y*'s respectively. These are calculated by using the following formulas:

$$SS_{xy} = \sum (x_i - \bar{x})(y_i - \bar{y}) \quad (5)$$

$$SS_{xx} = \sum (x_i - \bar{x})^2 \quad (6)$$

Alternatively, the following "shortcut" formulas can be used (n is the number of observations or sample size) [48]:

$$SS_{xy} = \sum x_i y_i - \frac{(\sum x_i)(\sum y_i)}{n} \quad (7)$$

$$SS_{xx} = \sum x_i^2 - \frac{(\sum x_i)^2}{n} \quad (8)$$

The values of A_C and β_C that minimize the SSE are given by the following formulas [48]:

$$Slope: A_C = \frac{SS_{xy}}{SS_{xx}} \quad (9)$$

$$y - intercept: \beta_C = \bar{y} - A_C \bar{x} \quad (10)$$

We can illustrate the use of these formulas using the example data points that were mentioned earlier and construct the following table where *x* is the number of users and *y* is the response time:

	x	y	$x_i - \bar{x}$	$y_i - \bar{y}$	$(x_i - \bar{x})(y_i - \bar{y})$	$(x_i - \bar{x})^2$
	500	23.1988	-500	-19.26908889	9634.544445	250000
	1000	40.3606	0	-2.107288889	0	0
	1500	63.84427	500	21.37637778	10688.18889	250000
Average	**1000**	**42.46789**		**Sum**	**20322.73333**	**500000**

Table 3. Calculation Table for Prediction Modeller

The mean of x ($\bar{x}$) is 1000 and the mean of y ($\bar{y}$) is 42.46789 and are used to calculate the values for the four different columns. In the next step we calculate slope by using the slope formula. We take the sums of the last two columns which represent SS_{xy} and SS_{xx} respectively. These sums are divided as such:

$$A_C = \frac{20322.73333}{500000} \quad (11)$$

We then find that A_C = 0.040645467. From this value, we can then easily find the y intercept given the mean of y and the mean of x:

$$\beta_C = 42.46789 - .040645467 * 1000 \quad (12)$$

We find β_C to be 1.822423. From this we can then construct our prediction model as:

$$R(C) = 0.040645467 * x + 1.822423 \quad (13)$$

This means that if we would like to know what the response time would be for a larger number of users (like 3000) we can substitute the value for the number of users we want to predict for *x*. This is illustrated in Table 4.

x	y
500	23.1988
1000	40.3606
1500	63.84427
2000	83.11336
2500	103.4361
3000	123.7588
3500	144.0816
4000	164.4043
4500	184.727
5000	205.0498

Table 4.Table with Predicted Values

The gray portion represents the values that are observed while the white portion represents the predicted response times for the given cluster configurations. We can then plot these values onto a line graph as shown in Figure 4.

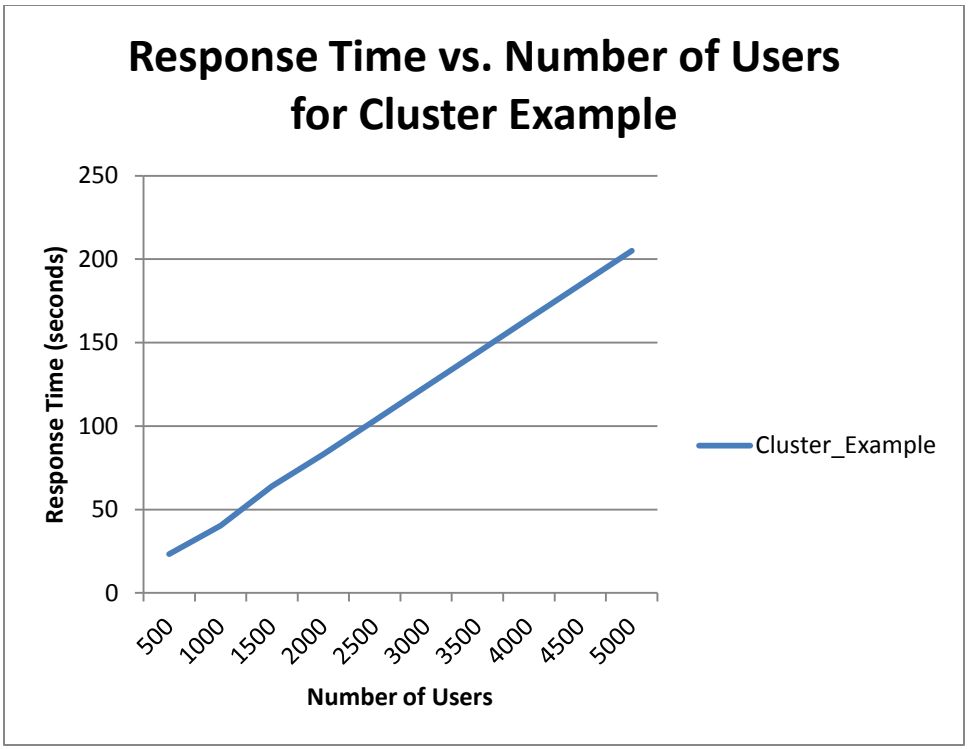


Figure 4. Cluster example line graph

As shown above, the line represents a particular cluster. The *x* axis represents the number of users and the *y* axis represents response time in seconds. As more and more clusters are modelled for performance predictions, more lines will appear on this line graph, allowing for researchers to objectively evaluate the performances of different clusters.

4.4 Summary

In the first part of this section we discussed the iterative process methodology, in which we described the methodology and the rationale for the components of the methodology. The second part described the cost and configurations for the experiments,

which looked at how to design the different topologies given a maximum capacity and set cost. The last section proposes a model for performance of the cluster. In the next chapter, we validate this approach.

5. Experiments and Results

The goal of this chapter is to

- Illustrate the importing and migration phase of the proposed process
- Validate the hypotheses that each configuration is characterized by a linear model
- Validate the accuracy of predictions

The chapter is organized as follows: Section 5.1 illustrates through an example how the import and transfer of data is implemented; Section 5.2 presents the validation of the models. Section 5.3 presents a summary and conclusions.

5.1 Importing and migrating data

This section describes how we import data into MySQL from a source data file and how data is transferred from MySQL to HBase, the first steps of the iterative process illustrated in Fig. 3. We evaluate this process on a real case scenario. Besides validating part of our process, the real case scenario gathers quantitative and qualitative guidelines using the process as well as evaluates tools for supporting the process.

The primary focus is on spatial-temporal data, using a set of traffic data as the data set. We have gathered 3 months (256 GB) worth of real data from the Ministry of Transportation for Ontario (through the Toronto Intelligent Transportation Systems Society of Canada) which are stored in XML format. This project is part of a bigger project known as the Connected Vehicles and Smart Transportation (CVST) project [49]

and aims to allow for data miners to quickly and easily analyze Big Data. The Big Data will be stored on Smart Applications on Virtual Infrastructure (SAVI) testbed [20] acting as the cloud.

Following the iterative process described in Chapter 4, firstly, we have to import XML data into a relational database. In this case, we convert the XML files into CSV files and store it into a MySQL database. The first step is to choose as data schema in order to understand how to transform this data. The data contains large amounts of "simple data" such as date, sensor IDs, average speeds, occupancy, and vehicle lengths.

In order to choose a proper data schema, there has to be a decision about whether or not the schema would be data warehouse or online transaction processing system (OLTP). The decision is that this schema will be a data warehouse due to several requirements [50] : Firstly, the database had to accommodate ad hoc queries as workloads may not be known in advance when dealing with traffic patterns. Secondly, the database is updated on a regular basis by Extract-Transform-Load process using bulk data modification technique which would not be directly updated by end-users. Thirdly, the schema needs to be denormalized to optimize query performance as opposed to optimizing update/insert/delete performance and guaranteeing data consistency. Lastly, the queries would involve scans of hundreds of thousands of records as opposed to a handful records. Therefore, we use a data warehouse.

As for the specific data warehouse schema, a star schema is chosen. A star schema consists of one fact table with one or more dimension tables [51]. The reason for

choosing a star schema is because the data contains a large amount of "simple data" such as date, sensor IDs, average speeds, occupancy, and vehicle length. Only one dimension (description of data) is needed, which is where the sensors were located. The overall schema is kept on a smaller scale for faster querying of data and overall performance. Ideally, we want to put a primary key on the date, contract ID, and periodNum but the files sent had duplicates in the invalid values (that is the date, contract IDs and periodNums were all the same). However, this comprised less than 1% of all data. Also, having an id as the primary key allows Sqoop, the tool we use to migrate the data, to easily split the table in preparation for MapReduce (as will be explained later in this section). We use indexing on the date, contractId, validThisPeriod, and periodNum in order to make the queries run quickly for validating HBase queries (are the results from MySQL and HBase the same given a particular query?). The schema used for the relational database is illustrated in Figure 5 on the next page.

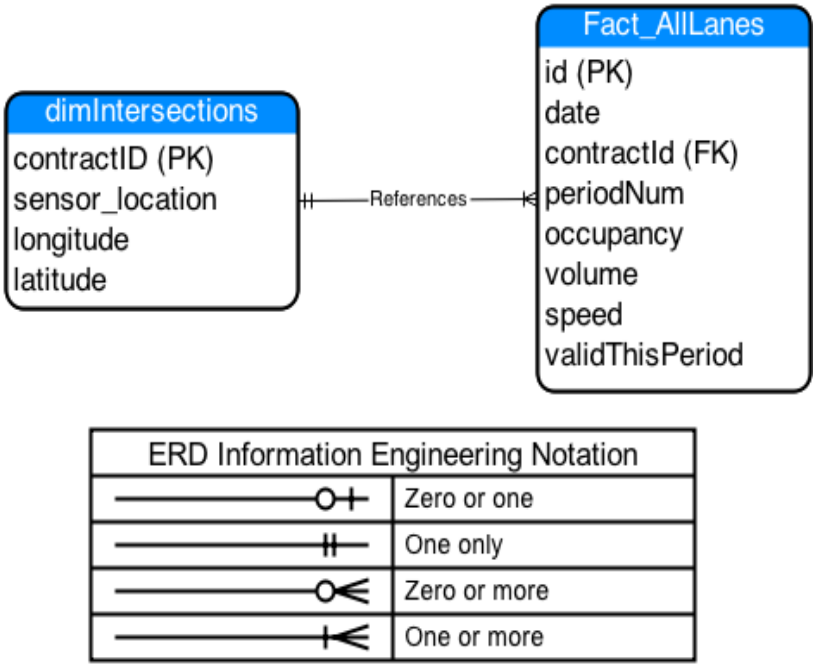


Figure 5. Warehouse Schema

The files are parsed using a Java application which is developed in-house with Ximpleware parser (more specifically VTD-XML)[12] to convert from XML to CSV files in order to import into MySQL. After all files are converted, the CSV files are bulk loaded using mysqlimport utility in the MySQL database system.

For the first stage, some cross-checking between the XML file and generated CSV file are performed by visual inspection. This includes verification of boundary cases (start and end of XML file) and random checking of an XML file in a date file against the csv file. We also validate the first few and last few files by counting the number of vdsData elements (a representation for each "all lanes" record) and match them to the total number of records that the output CSV file has.

[12] http://www.ximpleware.com/

For the second stage, we assume that mysqlimport ingests a CSV file into the highway table without any data corruption. This is reasonable since MySQL and its core utilities belong to a production strength database system. We run a bash script to import all files for each month. To ensure each file finished importing into the MySQL database, we make the system display a message that it is finished along with the timestamp. For the first file imported, the number of records are counted in the database and compared to the final number of records in the CSV file, as well as the XML file (by counting vdsData elements with Java application).

In the next step, we move the valid data from MySQL to HBase with an appropriate schema. Before the migration step, a set schema is to be chosen based on common practice for HBase.

Choosing a proper schema in HBase is imperative as this can highly impact performance and it is important to note that HBase is not a relational database therefore a relational schema will not work in this context. As such, a data schema is chosen based on past works and guidelines from other researchers [36], [38], [39]. We decide to choose a 2-Dimensional schema due to its simplicity and support from built-in tools for validating that all the data is there in a quick manner of time. Currently, HBase is not optimized for using versioning, which is why we do not to use this for our application. HBase's website currently suggests not using 100s of versions or more [37]. If it was decided to version records by period number, there would be thousands of versions as there are thousands of periods in a day. Also, only valid values are imported to HBase,

which means it will be hard to distinguish which value belongs to which period as the concept of period would be replaced with timestamps. Timestamps can also be customized to be replaced with period numbers but this would further add to the complexities because we would then have to validate for all records that the correct values are in the correct periods, which is difficult to do when one has millions, billions, or even trillions of records. This also adds to the complexities as we have repeatable processes in our methodology.

For the first iteration of *Iteration 2*, an MD5 will be added to the key in order to avoid region hot spotting [45]. Hot spotting occurs when there are too many keys on one region server and if users are continually querying keys on the same region server. This phenomenon happens because HBase stores everything in lexicographical order and when the key is not randomized, you could have an unequal distribution of keys across region servers. This also results in RegionServers being underutilized. We use MD5 to randomize the keys so that there is an equal distribution of data across RegionServers in order to avoid this situation. MD5 is a cryptographic hash function designed by Ron Rivest in 1991 for producing a 16-byte hash value, expressed in text format as a 32 digit hexadecimal number [52]. For instance, the MD5 for "Hello" is "8b1a9953c4611296a827abf8c47804d7"[13] and the MD5 for "Hello a" is "fc1a88fc1e6ad7ba6d6814e9d11e6fa0". We can see how both these strings, though similar, are completely randomized when utilizing the MD5 hash to convert the text.

[13] http://www.miraclesalad.com/webtools/md5.php

The key will consists of an MD5 on the sensor and date, the sensor itself, date itself, and period number. The reason we put the MD5 on the sensor and date, but not period number is because we want to have sensors and dates grouped together. Adding an md5 to the period number would put it in a random order. We also put padding to the period numbers (0001 instead of just 1) to have these values in order as HBase stores everything in lexicographical order otherwise, these values will not be properly ordered. For instance, period 10 will come before 2, which is not the order we would like to have in our values. The values in the row-keys are separated by underscores. The reason we decide to put the period numbers as rows rather than columns is for validation purposes (it is easier to count the rows than the columns), HBase has the ability to skip rows and StoreFiles (if you want to find specific periods during the day, this will be faster) [45], and existing migration tools do not support transposition. This would also require more than one column family to represent the speed and volume, which would be expensive in terms of memory.

Column names are renamed to single characters in order to conserve hard disk memory. HBase requires that all columns have at least one column family and in this case, we only need one generic column family. Figure 6 illustrates the transformation of the MySQL schema into the HBase schema. The quotation marks represent the actual name for either the column family or the column qualifiers (“a” is average speed and “v” is volume) while the values within the RowKey are the actual values.

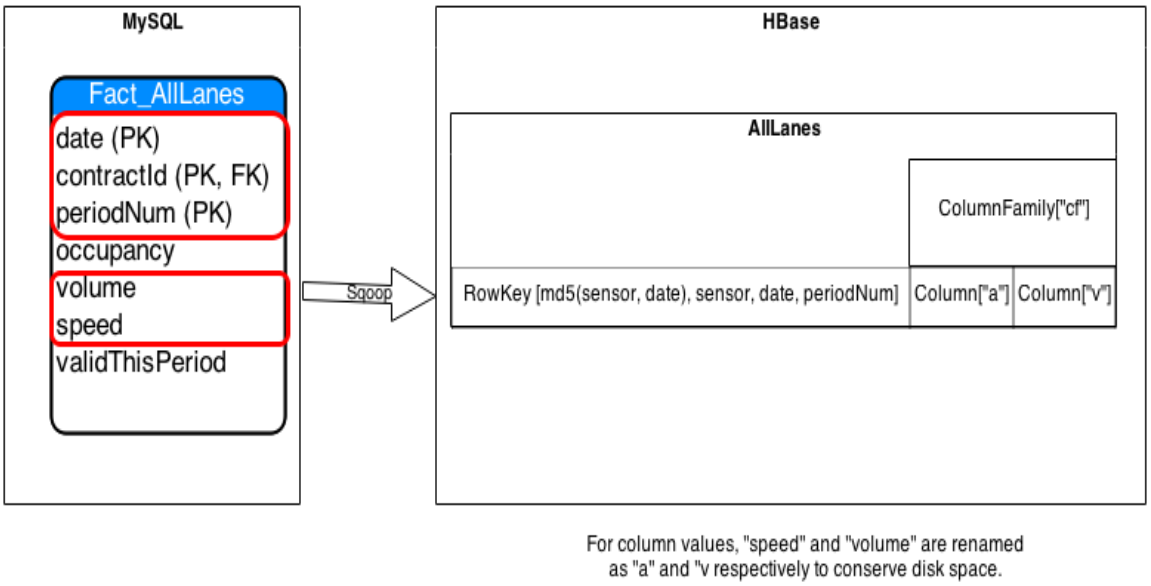


Figure 6. Schema Transformation

Open Source tools (which are in their early stages) are being developed for facilitating migration of raw data from MySQL to HBase such as Sqoop[14] and MySQL Applier (Beta)[15]. We use Sqoop (version 1.4.4) due to its ease of use, ease of installation (it comes prepackaged with Cloudera toolset), and it has an operational release as opposed to being in the Beta phase. Sqoop uses MapReduce framework to transfer data from MySQL to HBase, allowing for a faster process than serially transferring the data. Sqoop automates most of this process. The only problem encountered is that it requires a MySQL connector, which is not included in the package due to licensing reasons therefore the user has to download MySQL connector and put it in the proper directory in order to use Sqoop for transferring from MySQL databases. In order to speed up the process of experimentation, we decide to only import the data that will be queried. In this

[14] http://sqoop.apache.com
[15] http://dev.mysql.com/tech-resources/articles/mysql-hadoop-applier.html

case, only the volume and the average speed are imported. We also choose to import one month worth of data for the purpose of testing the behaviour of all the different topologies because the length of time to import all this data is too long for a repeatable experimentation process (which is over two hours).

In order to validate that all records were imported, all records are counted using the built-in counter function of HBase shell and setting caching to 10000 in order to make the process faster. The total count of all the records is then compared to the records in MySQL. Basic queries such as average speeds given a sensor or number of people speeding given particular sensors are executed and compared to results from MySQL in order to ensure accuracy of results.

5.1.1 Discussion

The main challenges of importing and migrating data is designing a data model and transferring the data in a short amount of time. To begin with, we learned that it is imperative to have an optimal schema for MySQL in order to understand what data schema will be needed for HBase, as well as for comparing both databases' query outputs in a short amount of time. Furthermore, it is important for the column qualifiers and column families in HBase to be as short as possible to conserve disk space, as well to make the migration between databases faster. Lastly, we also found that Sqoop is the ideal tool to use for transferring data between the two databases due to its use of MapReduce. In the next sections, we look at how to optimize Hadoop configurations.

5.2 Validating the performance models

The purpose of this section is to evaluate different topologies with the same cost and to validate the performance models.

We assume we have two goals:

- Cost goal, expressed as a fixed budget
- Performance goal: most scalable configuration for the budget

In section 5.2.1, we construct variations of topologies when given a cost goal (as price per hour). After these topologies are created, a *new topology* is deployed on the cloud and a workload generator is used to execute the *workloads*. After the *results* are generated with measured response times for each cluster, we compare the performances across topologies and schemas in Section 5.2.2 and Section 5.2.3 respectively. A *performance model* is then computed for each cluster in Section 5.2.4. We identify the type of model as linear across all clusters by looking at the graphs with the different response times over number of users and using regression analysis to illustrate this. We then take the first three values of selected clusters and construct models. The predicted response times are then calculated and compared against the real values that were measured to see how far the predicted values deviate from the observed values.

5.2.1 Experimental Setup

This section will talk about the experimental set up and how to construct different topologies for deployment. To approach this problem, one workload (or query) on different variations of topologies are executed for the third process.

On each topology, we compare the performance of utilizing compression for the data versus not using any compression. The compression that is used for all experiments will be GZIP as it compresses the data to the smallest size and uses higher CPU utilization when unzipping the files than other compressions, which may affect behaviour [45].

The workload tests different performances of the different topologies and find which topology is optimal, as well as investigate any patterns that may be found. It is important to note that the cloud may exhibit different behaviours at times (a term known as "cloud variability") depending on the number of real users on the same physical machine and whether or not they are running an I/O intensive task. In order to eliminate this, the experiments are executed five times for each table (compression and non-compression table) and an average is taken for all five experiments. This makes a total of ten experiments for each cluster. The queries are executed in alternate order between compression and non-compression.

Firstly, we start with a budget from our goals which is $0.84 cents per hour. We consider the prices from Amazon in order to decide how to form our clusters. Different costs are evaluated based on current market price values as some topologies may be more

expensive than others. Approximate pricing for each instance is taken from Amazon EC2 that is comparable to the SAVI instances [46].

Instance Type	**VCPU**	**Random Access Memory**	**Solid State Drive Size**	**Price**
c3.large	2	3.75 GB	32 GB	$0.105 per hour
c3.xlarge	4	7.5 GB	80 GB	$0.210 per hour
c3.2xlarge	8	15 GB	160 GB	$0.420 per hour

Table 5. Amazon EC2 Pricing for different instances comparable to SAVI

Secondly, from these price points and our set budget, we then extrapolate the topologies that will be deployed. Each cluster must total $0.84 an hour. We construct these topologies so that the clusters fit the out-of-the-box requirements for Cloudera such as having a minimum of three data nodes/RegionServers.

The topologies for each cluster are shown in Figure 7 on the next page.

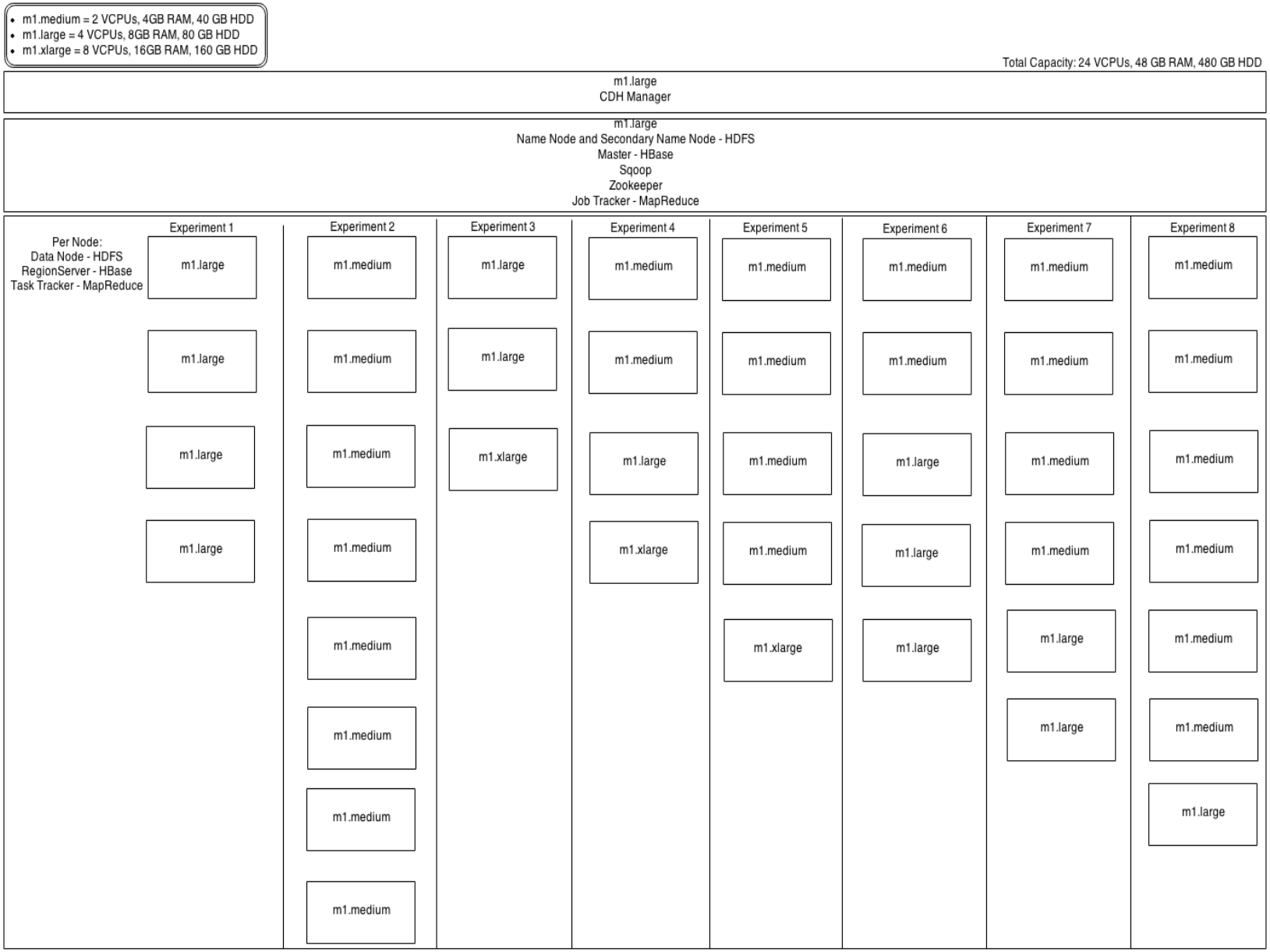


Figure 7. Experiments

Furthermore, all the machines in each cluster will have Linux installed as the operating system. One large node has Cloudera Hadoop (CDH) Manager running separately from other nodes so that results are not influenced. Another large node contains the Name Node (HDFS), the Master Node (HBase), Sqoop, and ZooKeeper. This setup is for all clusters in order to isolate the experiments to focus on the data nodes (HDFS)/region servers (HBase), which is what will change throughout the experiments. All region servers are mapped to the data nodes. All HBase clusters are configured with 1 GB in Java heapsize for the Name Node and Secondary Name Node as Cloudera recommends. Only one Zookeeper is used but it is important to note that the Hadoop documentation recommends three Zookeepers when working in production environments in order to better handle failure [53]. Using multiple ZooKeepers would require running extra instances (instance availability is limited for experiments) and because the thesis is focussed mainly on optimizing performance rather than handling mean time between failures, only one Zookeeper is used. Also, a noticeable characteristic is the different Java Heap Size for the HBase Region Servers given the size of the instance. For Medium instances, the Java Heap Size is default set at 531,685,481 bytes, while for large instances, it is default set at 863,075,931 bytes and for Extra Large instances the default is set at 715,791,403 bytes. Currently, there is a lack of research regarding configuration of this metric with a given size instance so the default out of the box Cloudera metrics are kept. Major compactions are disabled in order for performance not to be influenced and in order to test in an HBase scenario where records for all sensors are coming in every 20 seconds.

Moreover, an in-house application is created in Java with Hadoop API to execute multiple user queries concurrently from a single m1.xlarge instance to the Hadoop cluster (the Java Archive or JAR file is placed in the instance away from the Hadoop Cluster). This is built for convenience purposes to test specific queries onto HBase. This instance simulates an application server which receives user requests and executes the queries from these user requests to the Hadoop cluster. The type of query that will be executed is called a "Scan" query and the application will calculate the traffic volume average for a particular day at a particular sensor. The day is randomized for each user and the sensor is fixed. We execute increments of 500 users (500, 1000, 1500, etc.) making queries to the database until reaching the maximum number of 5000 users (which is around the maximum that could be handled by the clusters before crashing).

5.2.2 Comparing configurations

After executing all the queries, an average is taken for response times for each experiment in each cluster and the total average for each cluster is calculated. Figure 8, on 65, shows the resulting graph from the queries executed on the table without compression. Each "e" in the graph such as "e1" and "e2", represents the word "experiment" and the numbers in which order the experiments were executed in. We can see in this figure that the larger amount of instances seem to have the fastest response time where as the lowest amount of instances have the slowest response time in most cases with the exception of e6 (5 instances) versus e1 and e4 (which both have 4

instances). This is explained by Cloudera's out of the box configuration as the Java Heap Sizes end up being more on the largest amount of instances than the smallest amount in total. An example of this is demonstrated with the cluster with the largest amount of instances totalling 4.25 Gigabytes versus 2.44 Gigabytes in total Java Heap Size for the cluster with the smallest number of instances. This means that clusters are underutilized when default Cloudera configurations are used. As mentioned previously, there are currently no guidelines for setting these heap size configurations.

Each cluster name represents the order of the experiments executed so e1 represents "experiment 1" which represents a cluster with 4 m1.large instances for the region servers and so on. This means that experiment 1 is the first cluster to be executed.

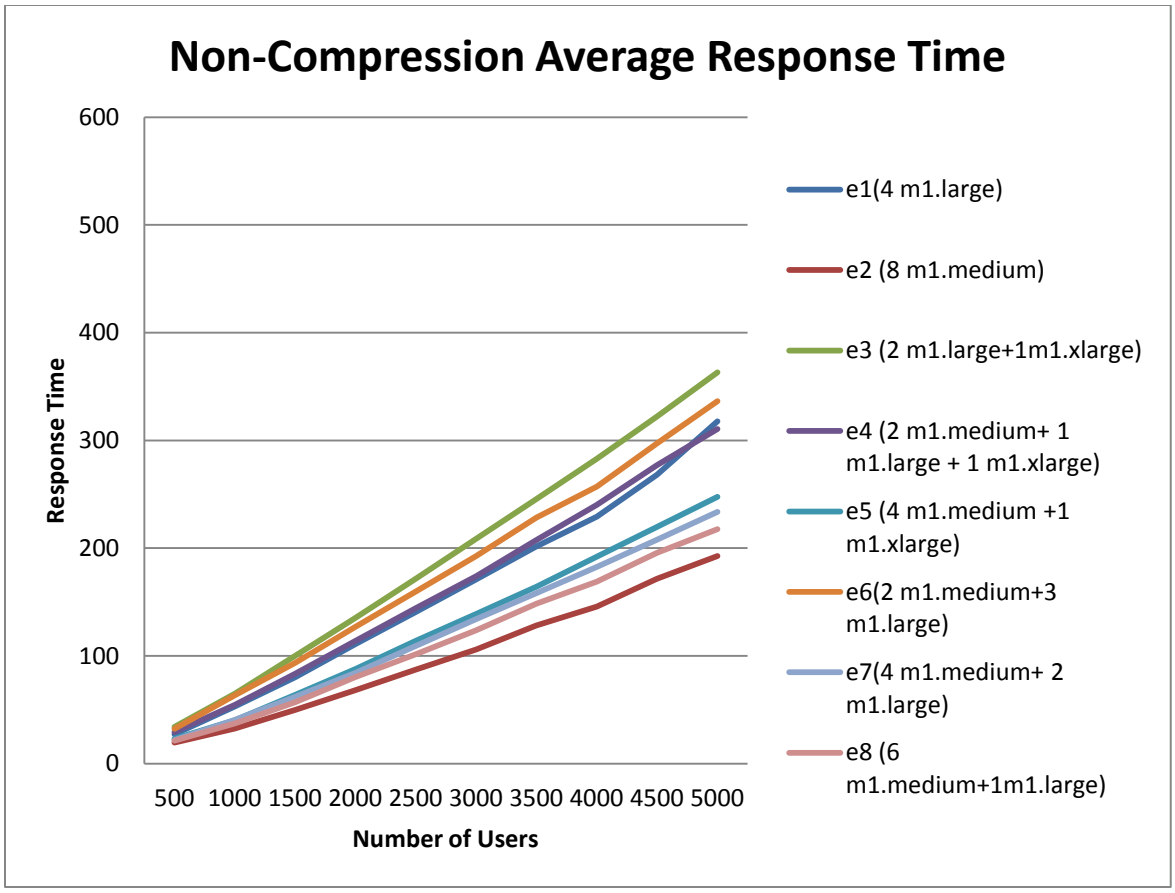


Figure 8. Response Times for Queries on non-Compress Tables for all topologies

We see that the cluster with the largest number of instances, e2, has the fastest response time. Figure 9, on the next page, shows the graph representing response time for workloads executed against tables that are compressed. In Figure 9, there is behaviour change in all clusters but there does not seem to be a trend in which cluster has a faster response time. The overall response time values of e2 and e7 are closer than e2 and e8 in the last experiment. Overall, e2 seems to be the cluster with the fastest all around response time as shown in both graphs when comparing response times for each table. Even though e7 does perform better in Figure 8, it performs even slower on average than

e2 in Figure 9. We can see this by calculating the mean differences. The mean difference for the table without compression is 23.30735 verses 8.478 seconds for compression table. Therefore e2 is the ideal cluster when assessing the tradeoffs.

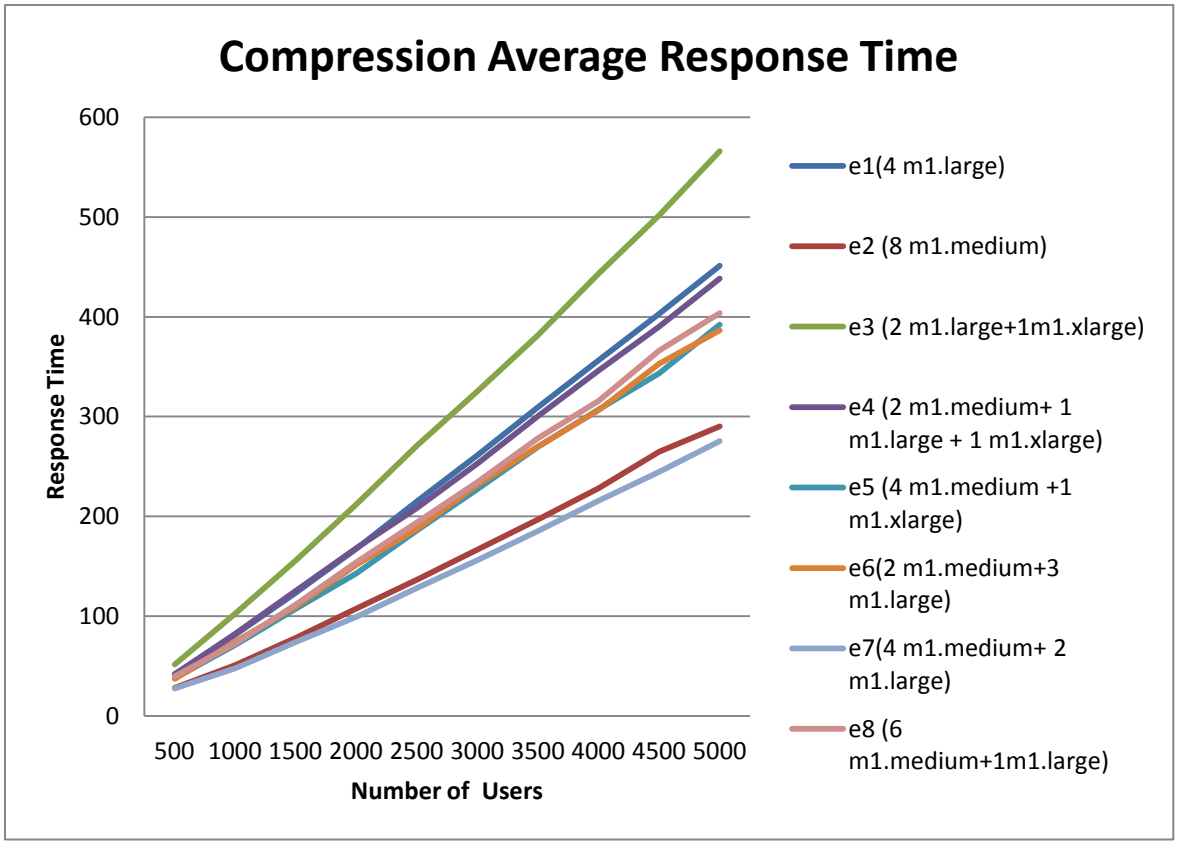


Figure 9. Response Times for Queries on Compression Tables for all topologies

5.2.3 Comparing Schemas

In the next step, we take the overall fastest cluster (e2) and execute queries on different schemas. The results are illustrated in Figure 10 for non-compression and Figure

11 for compression. In one schema, we switch the date and sensor ID (“switchid” on the figure), while in another schema, we remove the MD5 (“noMD5”). The schema chosen earlier is known as “default” , which includes the MD5, the sensor, the date, and the period number.

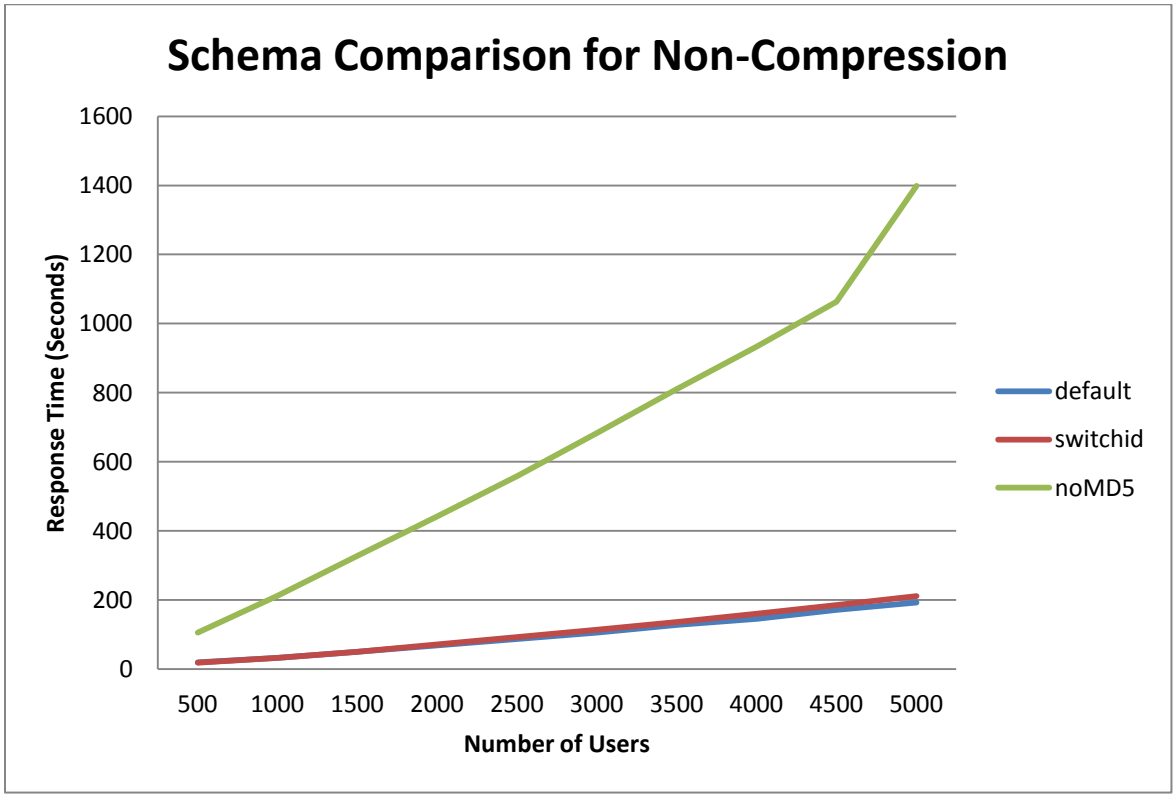


Figure 10. Non-Compression graph illustrating response times for different schemas

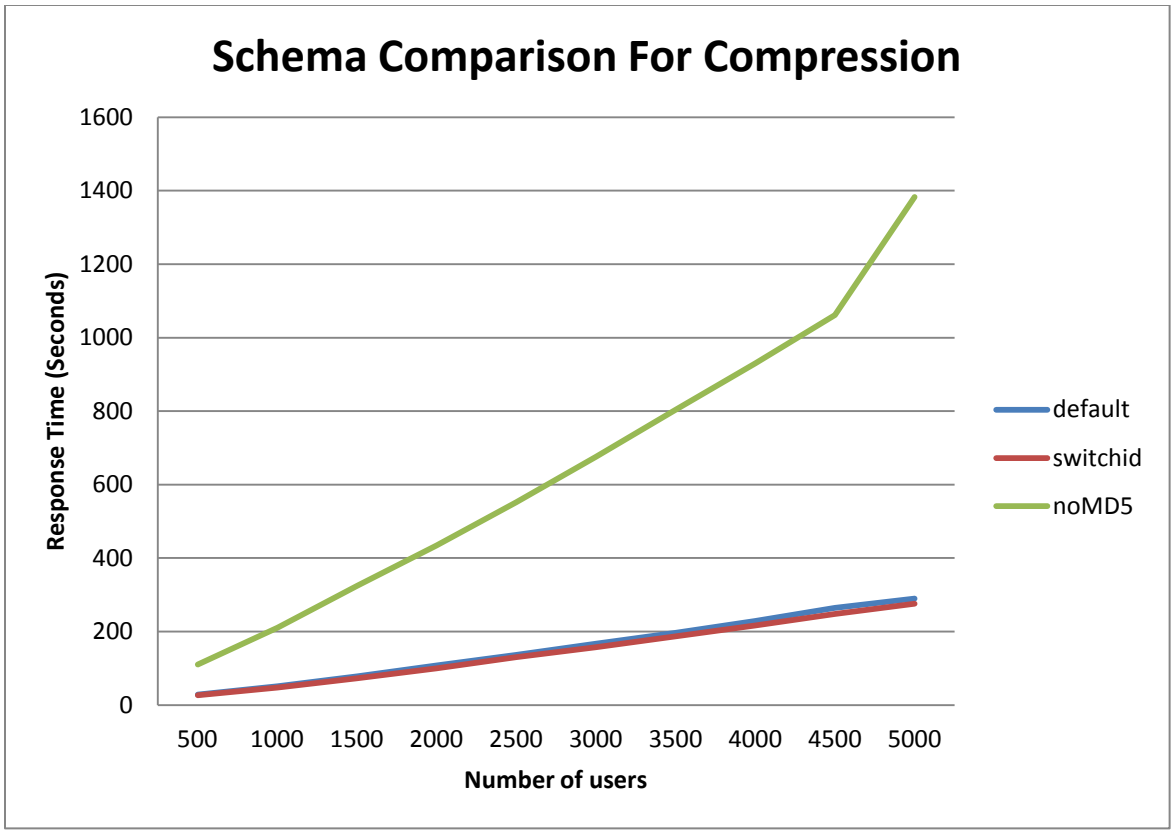


Figure 11. Compression graph illustrating response time for different schemas

Removing the MD5 confirms that there is region hot spotting as in both cases, the line begins to spike around 5000 users due to one of the region servers receiving too many requests. When the MD5 is left in, there is an equal key distribution across the region servers which allows for elimination of region hot spotting and for region servers to receive the same load as was shown in the literature section. Also, the graphs demonstrate that MD5 is much faster in response time than non-MD5. We can also check this by using a two-sample t-test assuming unequal variances to see if there is a significant difference in both cases. We find that our values have a normal distribution,

which is shown in Appendix A (Normal Distribution for t-Test). Our Normal Distribution validation is for the 10 observations for schema with MD5 and for the 10 observations for schema without MD5. This is done for both compression and non-compression workloads. As stated earlier, we have 10 observations for a schema with MD5 and a schema without MD5, which represent the average of the five average response times for *x* iteration of users. For example, 500 users would have 500 response times generated from the application. These 500 values are averaged out. We run the experiment four more times and follow the same process. We should then have five values which represent five average response times. We then take the average of these five response times to get the final average response times which would represent one point for 500 users. We do this for 10 iterations of users (500-5000 users). We compare the final 10 averages of a schema with MD5 to a schema without MD5 for both compression and non-compression workloads.

According to the t-statistic analysis, p is less than α at 5% for non-compression and compression. The analysis can be found in Table 5 and Table 6. Therefore we can conclude that there is a significant difference between having MD5 and not having MD5 on the schema.

No Compression
t-Test: Two-Sample Assuming Unequal Variances

	noMD5	*default*
Mean	653.5575435	100.1812
Variance	164739.5757	3481.974
Observations	10	10
Hypothesized Mean Difference	0	
df	9	
t Stat	4.266578594	
P(T<=t) one-tail	0.001045404	
t Critical one-tail	1.833112923	
P(T<=t) two-tail	0.002090808	
t Critical two-tail	2.262157158	

Table 6. t-test for noMD5 vs default for No Compression

Compression
t-Test: Two-Sample Assuming Unequal Variances

	noMD5	*default*
Mean	648.504285	154.9088
Variance	161794.2556	8095.16
Observations	10	10
Hypothesized Mean Difference	0	
df	10	
t Stat	3.786936774	
P(T<=t) one-tail	0.001780611	
t Critical one-tail	1.812461102	
P(T<=t) two-tail	0.003561221	
t Critical two-tail	2.228138842	

Table 7. t-test for noMD5 vs default with Compression

5.2.4 Models

Furthermore, all the results from cluster and schema experiments can be modelled using a linear regression. As can be shown across schemas and across clusters, we can see that there is a positive linear relationship between the number of users and the response time. We will explain how the model demonstrates the linear dependency of response time on number of users for the fastest cluster, which is e2 (8 m1.medium). The results are generated using the IBM SPSS tool[16].

e2 (8 m1.medium) Regression

Correlations

		e2 (8 m1. medium)	users
Pearson Correlation	e2 (8 m1.medium)	1.000	.998
	users	.998	1.000
Sig. (1-tailed)	e2 (8 m1.medium)	.	.000
	users	.000	.
N	e2 (8 m1.medium)	10	10
	users	10	10

Table 8. Correlations for e2

[16] http://www-01.ibm.com/software/analytics/spss/

The correlations table shows Pearson correlation coefficients and the number of cases with non-missing values. We see that we have a strong positive correlation (0.998) between the two variables. From the significance test p-value we see that there is very strong evidence ($p<0.001$) to suggest that there is a linear correlation between the two variables [48].

Model Summary[b]

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.998[a]	.996	.995	4.080756205

a. Predictors: (Constant), users

b. Dependent Variable: e2 (8 m1.medium)

Table 9. Model Summary for e2

The R from the model summary table is the correlation coefficient which is a measure of the strength of linear relationship between the response time variable and the user variables. For simple linear regression, this is the same as Pearson's correlation coefficient we have already seen [48].

R Square or coefficient of determination is the proportion of variation in the response variable explained by the regression model. The values of R square range from 0 to 1; small values indicate that the model does not fit the data well. 99.7% of the variation in response time values can be explained by a fitted line [48].

The standard error of the estimate is the estimate of the standard deviation of the error term of the model, σ. This gives an idea of the expected variability of predictions [48].

ANOVA[a]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	31204.544	1	31204.544	1873.857	.000[b]
	Residual	133.221	8	16.653		
	Total	31337.765	9			

a. Dependent Variable: e2 (8 m1.medium)

b. Predictors: (Constant), users

Table 10. ANOVA for e2

The ANOVA table indicates that the regression model predicts the dependent variable significantly well as the statistical significance (under "Sig." Column for "Regression" row) is p=0, which is less than α at 5%. This indicates that the regression model significantly predicts the response time variable *y* and that it is a good fit for the data [48].

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	95.0% Confidence Interval for B	
		B	Std. Error	Beta			Lower Bound	Upper Bound
1	(Constant)	-6.784	2.788		-2.434	.041	-13.213	-.356
	users	.039	.001	.998	43.288	.000	.037	.041

a. Dependent Variable: e2 (8 m1.medium)

Table 11. Coefficients for e2

The unstandardized coefficients are the coefficients of the estimated regression model. Thus the expected response time value is given by:

$$R(C) = 0.039 * x - 6.784 \quad (14)$$

We include both the constant and slope as they are significant to the model due to the fact that the p value is below α at 0.05. As for the values in the equation, 0.039 is the

slope, x is the number of users, -6.784 is the intercept, and *y* is the response time. The standardized coefficient is the same value as shown before in the first table. We can also be 95% confident that the slope is within the range of between 0.037 and 0.041. We are also 95% confident that the intercept will be in the range between -13.213 and -0.356 [48]. We validate the model assumptions in Appendix B (Model Assumptions).

In the next part of the analysis, we can also see that the linear model can be extrapolated by using just three of the values. In order to show this, we use three of the clusters as shown in the next page. Two of the clusters are non-compression (e1 and e8) and the other cluster is compression (e6). We first extrapolate a model by calculating the slope and intercept for the first three values. After we have these two values, we create an approximate linear model (eg. $R(C) = 0.052631067 * x + 0.920622223$ as shown in the next page) [48]. We then calculate all the predicted values for each given number of users. The difference between the measured *y* values and predicted *y* values are then calculated. We then calculate the percentage of the difference and calculate the mean for the last seven values to see how accurate the model predicts the values. As can be seen in the tables and graphs on the next pages, the model relatively approximates the predicted values. The mean percentage of the seven values is below 12.8% which means that the average of rate of errors is relatively low. The minimum percentage difference is below 5% and the maximum is approximately 16% for all three clusters, which adds evidence to the fact that the error rates are relatively low. The next page shows this information in detail and is graphed to visually show how the model can approximate the values.

$$R(C) = 0.052631067 * x + 0.920622223 \quad (15)$$

e1			
Measured y Values	Predicted y Values	Difference	%Difference
27.3896	27.23615573	0.15344427	0.56022823
53.2448	53.55168923	-0.30688923	-0.5763741
80.02066667	79.86722273	0.153443936	0.19175538
111.0799	106.1827562	4.89714377	4.40866779
140.81392	132.4982897	8.31563027	5.90540358
170.9486667	158.8138232	12.13484343	7.09853061
201.6801714	185.1293567	16.55081469	8.206466
229.3647425	211.4448902	17.91985225	7.81281903
268.0817581	237.7604237	30.32133437	11.3104803
317.79912	264.0759572	53.72316277	16.904755
		Mean % for last 7 values	8.80673176

Table 12. Measured vs Predicted Values - e1

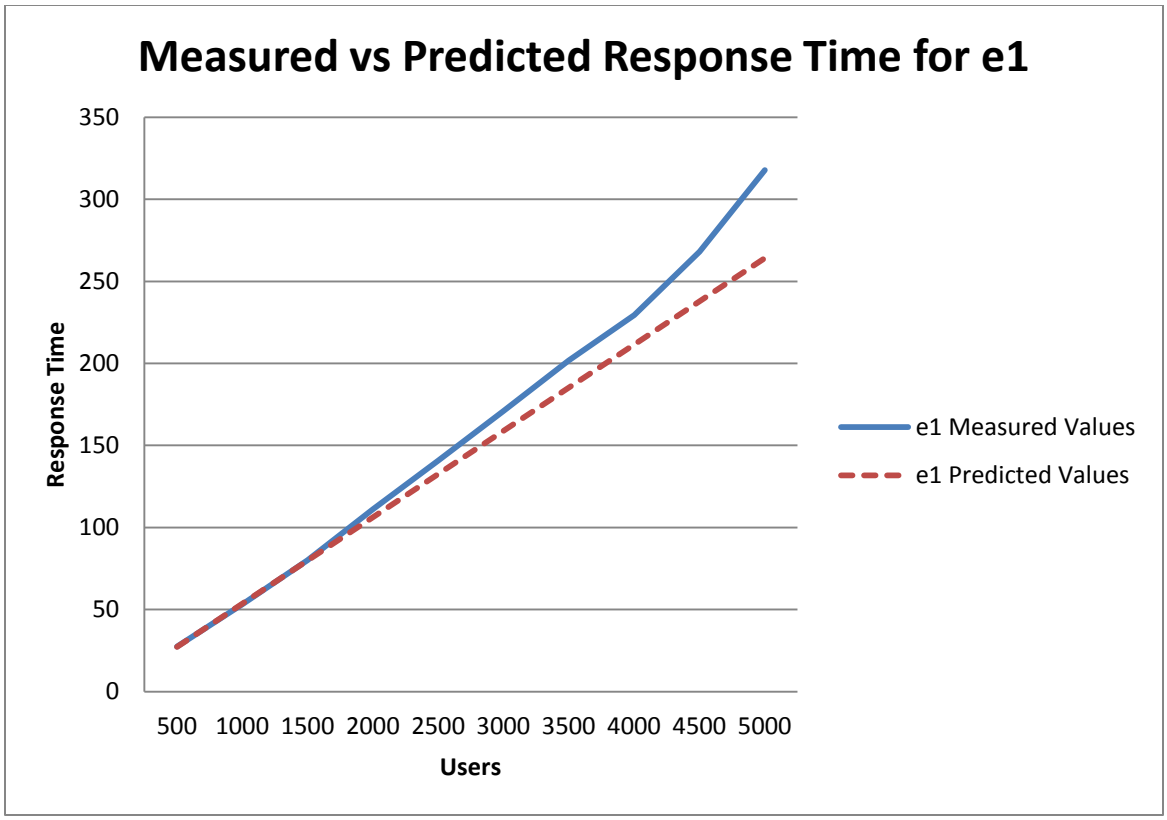


Figure 12. Measured vs Predicted Response Time for e1 Graph

$$R(C) = 0.035596533 * x + 2.908311111 \quad (16)$$

e8			
Measured Values	Predicted Values	Difference	%Difference
21.2872	20.70657761	0.580622389	2.72756581
37.3436	38.50484411	-1.161244111	-3.1096202
56.88373333	56.30311061	0.580622723	1.02071838
80.3078	74.10137711	6.206422889	7.728294
101.37096	91.89964361	9.471316389	9.34322452
123.7573333	109.6979101	14.05942321	11.3604769
148.4385714	127.4961766	20.94239483	14.1084589
168.8869	145.2944431	23.59245689	13.9693824
195.8008445	163.0927096	32.70813485	16.7047976
217.72192	180.8909761	36.83094389	16.916507
		Mean for last 7 values	12.8758773

Table 13. Measured vs. Predicted Values - e8

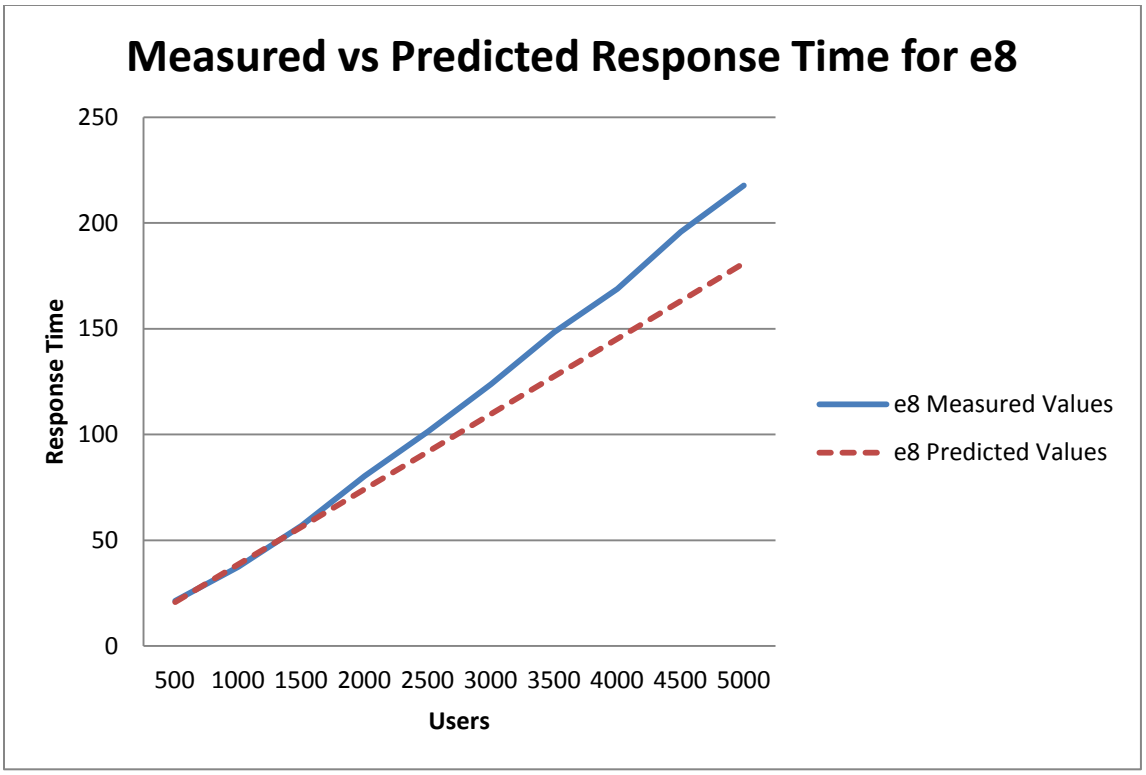


Figure 13. Measured vs Predicted Response Time for e8 Graph

$$R(C) = 0.0722408 * x - 4.346733333 \quad (17)$$

e6 Compressed			
Measured Values	Predicted Values	Difference	%Difference
37.4056	31.77366667	5.631933333	17.7251602
74.3146	67.89406667	6.420533333	9.45669283
109.6464	104.0144667	5.631933333	5.41456733
151.8334	140.1348667	11.69853333	8.34805328
187.5128	176.2552667	11.25753333	6.3870621
231.7607333	212.3756667	19.38506667	9.12772493
270.0800572	248.4960667	21.58399049	8.68584794
306.63435	284.6164667	22.01788333	7.73598365
353.1127556	320.7368667	32.37588889	10.0942212
386.53388	356.8572667	29.67661333	8.31610173
		Mean for last 7 values	8.38499927

Table 14. Measured vs Predicted Values - e6

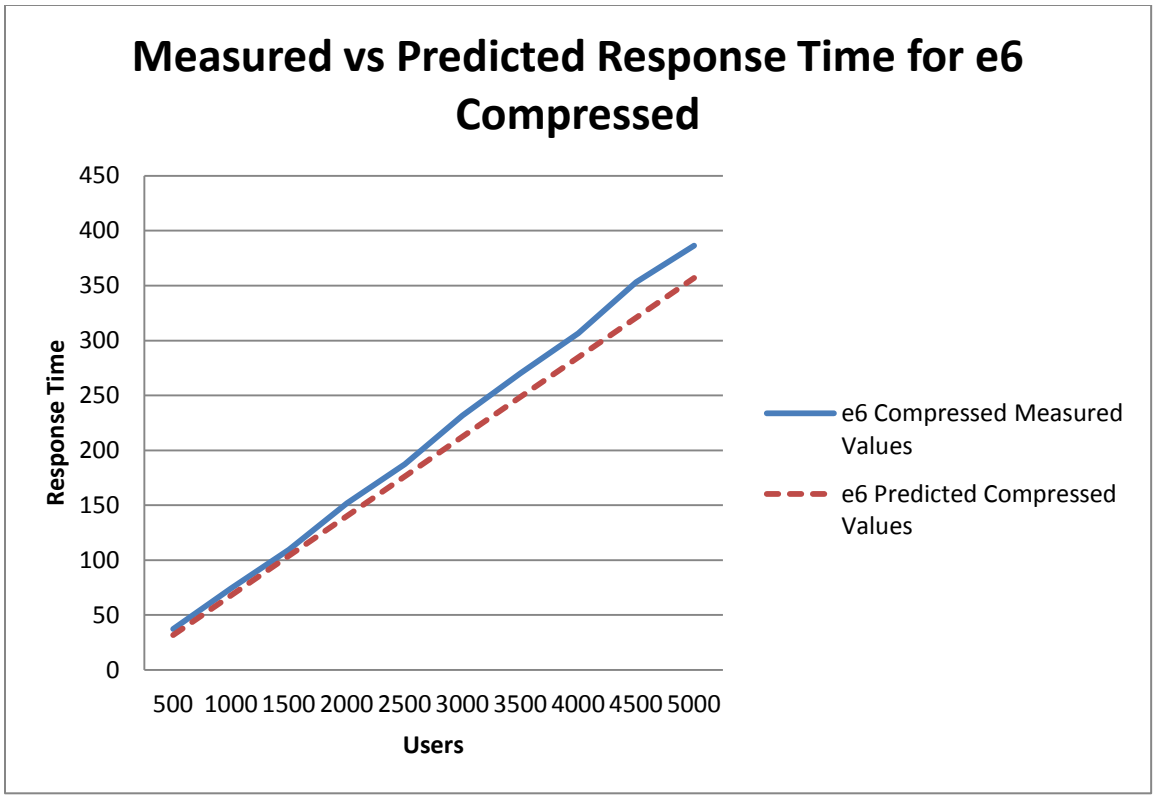


Figure 14. Measured vs Predicted Response Time for e6 Graph

This analysis is valuables for researchers who follow this methodology as it can save them time and cost. In order to eliminate any potential outliers (as values are very few), it is very important to run the experiment for each cluster several times (in this case, we ran the experiments five times) and take the average of these values. It is also important to note that this shows that we can predict the values for up to 5000 users given these clusters, but this does not show that we can go above 5000 users.

5.3 Summary

This chapter first showed how we implemented our iterative process methodology by looking at the import and migration process of an XML big data source. We then investigated the performance of topologies generated for the same cost and validated our performance model through the last steps of the methodology. We first illustrated the experimental setup for executing our workloads. We then showed the response time comparisons and found that the cluster with the most machines had the quickest response time. We also showed that there is a significant difference in response time when adding an MD5 to the row key. We found that there is a linear relationship for all clusters by performing a linear regression analysis on all clusters. We also found that you can approximately predict future values by modelling only the first three data points of each experiment.

6. Conclusion

This thesis presented an approach for quantifying and modeling the performance of HBase clusters. This is demonstrated within the CVST project on SAVI platform acting as the cloud. All experiments were executed in real time on real data. We summarize the contributions as follows:

To begin with, we introduced and applied an iterative methodology for evaluating and characterizing HBase clusters. This iterative methodology consists of several steps:

First, large data documents are stored into the relational database. In this thesis, our data documents are XML standard and we use MySQL as our relational database. A set of goals is also established for the non-relational database cluster. In our thesis, we chose performance and cost as our goals.

Second, the data is transferred from relational database to non-relational database (HBase), through an iterative process in order to find the best schema after the best cluster is found. The relational database acts as a “back up” store to save time from re-importing all individual files and for verification purposes, as well as being supported by existing migration tools. Also, the relational database only requires one machine where as HBase would require many machines therefore saving on cost and physical resources.

Thirdly, the last step in the methodology evaluates different topologies based on performance in real time against different types of workloads and also acts as an iterative process for finding the best topology of machines. In order to speed this process up, a

small sample of workload response times are measured and from these observations, a prediction model that is linear is formed. Once this prediction model is created, then the future values can be approximated. This saves time and resource (in terms of cost and physical resources) for the researcher as they do not have to execute a larger number of concurrent queries.

Moreover, we demonstrated and validated this linear predictive model behaviour across clusters and schemas. We constructed a prediction model from a few response time values and then calculated predicted response times. We then compared the predicted values from the prediction model with the observed values from our experiments and found that there was a small percentage in difference between these values. In addition to this, statistical analysis was used to prove that each cluster configuration indeed had a linear regression for all observed values. This confirmed our assumptions that there would be a linear relationship between number of users and response times due to the sequential nature of the “Scan” workloads that were used [45].

Lastly, we showed that row keys with MD5 were found to be significantly faster than row keys without MD5. This was due to the RegionServer hotspotting and keys not having a proper distribution across machines [16]. Furthermore, clusters with higher number of instances performed consistently faster due to cluster underutilization with out-of-the-box configuration.

It is important to note however, that the results are valid only for the “scan” type of requests. Also, different big data sources may yield different results. This can be due to

the schema type required for the data or how much data is being stored in each cell (in our thesis, we only store numbers). Further experiments and statistical analysis may be required in order to generalize these results. Moreover, how to configure clusters for maximum cluster utilization remains to be an open question. In the future, we would like to extend our methodology to facilitate finding the best practice configurations/settings for different clusters in HBase in order to enable maximum utilization of clusters. After this is successful, we plan to do similar experiments with the clusters being fully utilized and see how it would behave. In addition to this, we would like to also see if there are other HBase functionalities that can be modelled in order to add to our existing methodology.

References

[1] M. Shtern, R. Sandel, M. Litoiu, C. Bachalo, and V. Theodorou, “Towards Mitigation of Low and Slow Application DDoS Attacks,” in *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, Washington, DC, USA, 2014, pp. 604–609.

[2] M. Litoiu, M. Woodside, J. Wong, J. Ng, and G. Iszlai, “A Business Driven Cloud Optimization Architecture,” in *Proceedings of the 2010 ACM Symposium on Applied Computing*, New York, NY, USA, 2010, pp. 380–385.

[3] M. Shtern, B. Simmons, M. Smit, and M. Litoiu, “An Architecture for Overlaying Private Clouds on Public Providers,” in *Proceedings of the 8th International Conference on Network and Service Management*, Laxenburg, Austria, Austria, 2013, pp. 371–377.

[4] D. J. Abadi, “Data Management in the Cloud: Limitations and Opportunities,” *IEEE Data Eng Bull*, vol. 32, no. 1, pp. 3–12, 2009.

[5] M. A. Babar and M. A. Chauhan, “A Tale of Migration to Cloud Computing for Sharing Experiences and Observations,” in *Proceedings of the 2Nd International Workshop on Software Engineering for Cloud Computing*, New York, NY, USA, 2011, pp. 50–56.

[6] “What is Big Data? | SAS.” [Online]. Available: http://www.sas.com/en_us/insights/big-data/what-is-big-data.html. [Accessed: 18-Mar-2014].

[7] D. Laney, “3D data management: Controlling data volume, velocity and variety,” *META Group Res. Note*, vol. 6, 2001.

[8] D. Agrawal, S. Das, and A. El Abbadi, “Big Data and Cloud Computing: Current State and Future Opportunities,” in *Proceedings of the 14th International Conference on Extending Database Technology*, New York, NY, USA, 2011, pp. 530–533.

[9] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, “PNUTS: Yahoo!’s Hosted Data Serving Platform,” *Proc VLDB Endow*, vol. 1, no. 2, pp. 1277–1288, Aug. 2008.

[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,” *SIGOPS Oper Syst Rev*, vol. 41, no. 6, pp. 205–220, Oct. 2007.

[11] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer, “Apache Hadoop Goes Realtime at Facebook,” in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2011, pp. 1071–1080.

[12] J. Huang, X. Ouyang, J. Jose, M. Wasi-ur-Rahman, H. Wang, M. Luo, H. Subramoni, C. Murthy, and D. K. Panda, “High-Performance Design of HBase with RDMA over InfiniBand,” in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, 2012, pp. 774–785.

[13] J. Han, E. Haihong, G. Le, and J. Du, “Survey on NoSQL database,” in *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, 2011, pp. 363–366.

[14] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” in *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, Berkeley, CA, USA, 2004, pp. 10–10.

[15] “Welcome to Apache™ Hadoop®!” [Online]. Available: http://hadoop.apache.org/. [Accessed: 18-Mar-2014].

[16] “6.3. Rowkey Design.” [Online]. Available: http://hbase.apache.org/book/rowkey.design.html. [Accessed: 07-Oct-2014].

[17] P. M. Mell and T. Grance, “SP 800-145. The NIST Definition of Cloud Computing,” National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011.

[18] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and others, *Above the clouds: a Berkeley view of cloud computing. UC Berkeley Reliable Adaptive Distributed Systems Laboratory*. Technical Report No. UCB/EECS-2009-28, 2009.

[19] J. Pan, S. Paul, and R. Jain, “A survey of the research on future internet architectures,” *Commun. Mag. IEEE*, vol. 49, no. 7, pp. 26–36, Jul. 2011.

[20] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, “SAVI testbed: Control and management of converged virtual ICT resources,” in *Integrated Network*

Management (IM 2013), 2013 IFIP/IEEE International Symposium on, 2013, pp. 664–667.

[21] S. Sagiroglu and D. Sinanc, “Big data: A review,” in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, 2013, pp. 42–47.

[22] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. Byers, “Big data: The next frontier for innovation, competition, and productivity. 2011,” *McKinsey*.

[23] H. Chen, R. H. Chiang, and V. C. Storey, “Business Intelligence and Analytics: From Big Data to Big Impact.,” *MIS Q.*, vol. 36, no. 4, pp. 1165–1188, 2012.

[24] E. Dumbill, “What is big data,” *Introd. Big Data Landscapeonline Httpstrata Oreilly Com201201what--Big-Data Html*, 2012.

[25] T. Morgan, “IBM Global Technology Outlook 2012,” *Technol. Innov. Exch. IBM Warwick*, 2012.

[26] K. Ebner, T. Buhnen, and N. Urbach, “Think Big with Big Data: Identifying Suitable Big Data Strategies in Corporate Environments,” in *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, 2014, pp. 3748–3757.

[27] “HDFS Architecture Guide.” [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. [Accessed: 01-Apr-2014].

[28] “Chapter 9. Architecture.” [Online]. Available: http://hbase.apache.org/book/architecture.html. [Accessed: 09-Sep-2014].

[29] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A Distributed Storage System for

Structured Data," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, Berkeley, CA, USA, 2006, pp. 205–218.

[30] "Chapter 5. Data Model." [Online]. Available: http://hbase.apache.org/book/datamodel.html. [Accessed: 09-Sep-2014].

[31] Y. Jiang, *HBase Administration Cookbook*. Packt Publishing, 2012.

[32] "HLog (HBase 2.0.0-SNAPSHOT API)." [Online]. Available: https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/regionserver/wal/HLog.html. [Accessed: 29-Sep-2014].

[33] F. Junquira and B. Reed, *ZooKeeper: Distributed Process Coordination*, 1st ed. United States of America: O'Reilly Media, 2013.

[34] K. Ting and J. Cecho, *Apache Sqoop Cookbook*, 1st ed. United States of America: O'Reilly Media, 2013.

[35] "6.2. On the number of column families." [Online]. Available: http://hbase.apache.org/book/number.of.cfs.html. [Accessed: 24-Mar-2014].

[36] D. Han and E. Stroulia, "A three-dimensional data model in HBase for large time-series dataset analysis," in *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012 IEEE 6th International Workshop on the*, 2012, pp. 47–56.

[37] "6.4. Number of Versions." [Online]. Available: http://hbase.apache.org/book/schema.versions.html. [Accessed: 22-May-2014].

[38] “6.10. Schema Design Smackdown.” [Online]. Available: http://hbase.apache.org/0.94/book/schema.smackdown.html. [Accessed: 15-Sep-2014].

[39] C. Li, “Transforming relational database into HBase: A case study,” in *Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on*, 2010, pp. 683–687.

[40] G. Lee, B.-G. Chun, and H. Katz, “Heterogeneity-aware Resource Allocation and Scheduling in the Cloud,” in *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, Berkeley, CA, USA, 2011, pp. 4–4.

[41] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, “Improving MapReduce Performance in Heterogeneous Environments,” in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, Berkeley, CA, USA, 2008, pp. 29–42.

[42] G. Song, Z. Meng, F. Huet, F. Magoules, L. Yu, and X. Lin, “A Hadoop MapReduce Performance Prediction Method,” in *High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, 2013, pp. 820–825.

[43] E. Bortnikov, A. Frank, E. Hillel, and S. Rao, “Predicting Execution Bottlenecks in Map-reduce Clusters,” in *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, Berkeley, CA, USA, 2012, pp. 18–18.

[44] J. H. Friedman, “Greedy function approximation: A gradient boosting machine.,” *Ann. Stat.*, vol. 29, no. 5, pp. 1189–1232, 2001.

[45] L. George, *HBase: The Definitive Guide*, 1st ed. O’Reilly Media, 2011.

[46] “AWS | Amazon EC2 | Pricing.” [Online]. Available: http://aws.amazon.com/ec2/pricing/. [Accessed: 24-Jun-2014].

[47] “Rackspace Public Cloud Pricing.” [Online]. Available: http://www.rackspace.com/cloud/public-pricing/#cloud-servers. [Accessed: 22-Sep-2014].

[48] J. McClave, P. G. Benson, and T. Sincich, *Statistics for Business and Economics*, 11th ed. United States of America: Pearson, 2010.

[49] “Connected Vehicles and Smart Transportation.” [Online]. Available: http://cvstproject.com/. [Accessed: 25-Mar-2014].

[50] “Data Warehousing Concepts.” [Online]. Available: http://docs.oracle.com/cd/B10500_01/server.920/a96520/concept.htm. [Accessed: 29-Mar-2014].

[51] J. M. Perez, R. Berlanga, M. J. Aramburu, and T. B. Pedersen, “Integrating Data Warehouses with Web Data: A Survey,” *Knowl. Data Eng. IEEE Trans. On*, vol. 20, no. 7, pp. 940–955, Jul. 2008.

[52] R. L. Rivest, *The MD5 Message Digest Algorithm*. 1992.

[53] “ZooKeeper Administrator’s Guide.” [Online]. Available: http://zookeeper.apache.org/doc/r3.1.2/zookeeperAdmin.html. [Accessed: 23-Jun-2014].

[54] S. Chatterjee and J. Simonoff, *Handbook of Regression Analysis*. Hoboken, New Jersey: John Wiley and Sons, 2013.

Appendix A (Normal Distribution for t-Test)

First we check if both of the dependent variables of compression and non-compression have normal distribution. On the next pages (Table 6 and Table 7), we can see that the skewness and kurtosis are between -1.96 and +1.96. It is hard to tell from the histograms whether or not the data is normally distributed as there is a small amount of data points so we look at both the Shapiro-Wilk test and Q and Q plots (Table 8). In the Shapiro-Wilk test, if the p value falls below 0.05, then the data is not normally distributed. However, if it is above 0.05 it may or may not be normally distributed. This test is complemented by the Normal Q-Q plot test. We look at the Normal Q-Q plot to see if the expected values and the normal values match up, which approximately do as illustrated in the relevant graphs (Figure 12-Figure 15). Also the de-trended normal Q-Q plot shows that standard deviation is close to 0 and the box and whisker plot is symmetrical therefore we can conclude that the data for all four scenarios are approximately normally distributed [48].

Normal Distribution Analysis

Descriptives

			Statistic	Std. Error
default	Mean		100.1812	18.66005
	95% Confidence Interval for Mean	Lower Bound	57.9693	
		Upper Bound	142.3932	
	5% Trimmed Mean		99.5080	
	Median		96.5406	
	Variance		3481.974	
	Std. Deviation		59.00825	
	Minimum		19.67	
	Maximum		192.81	
	Range		173.13	
	Interquartile Range		106.83	
	Skewness		.191	.687
	Kurtosis		-1.194	1.334
default_compression	Mean		154.9088	28.45200
	95% Confidence Interval for Mean	Lower Bound	90.5459	
		Upper Bound	219.2716	
	5% Trimmed Mean		154.4302	
	Median		151.7083	
	Variance		8095.160	
	Std. Deviation		89.97311	
	Minimum		28.34	
	Maximum		290.09	
	Range		261.75	
	Interquartile Range		165.99	
	Skewness		.107	.687
	Kurtosis		-1.261	1.334
noMD5_compression	Mean		648.5043	127.19837
	95% Confidence Interval for Mean	Lower Bound	360.7616	
		Upper Bound	936.2470	
	5% Trimmed Mean		637.6060	
	Median		613.8113	
	Variance		161794.256	
	Std. Deviation		402.23657	
	Minimum		109.95	
	Maximum		1383.22	
	Range		1273.27	
	Interquartile Range		666.65	
	Skewness		.445	.687
	Kurtosis		-.492	1.334

Table 15. Descriptives

Descriptives

			Statistic	Std. Error
noMD5	Mean		653.5575	128.35092
	95% Confidence Interval for Mean	Lower Bound	363.2076	
		Upper Bound	943.9075	
	5% Trimmed Mean		642.5566	
	Median		620.7724	
	Variance		164739.576	
	Std. Deviation		405.88123	
	Minimum		106.02	
	Maximum		1399.12	
	Range		1293.10	
	Interquartile Range		667.54	
	Skewness		.444	.687
	Kurtosis		-.437	1.334

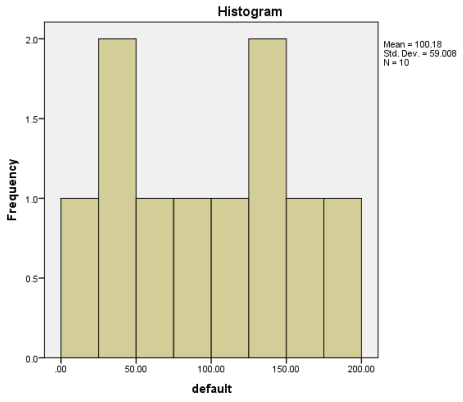
Table 16. Descriptives (second part)

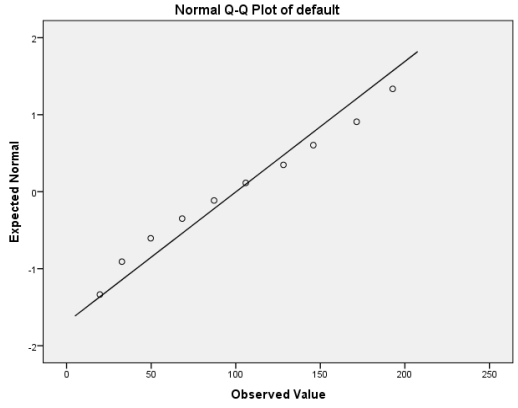
Tests of Normality

	Shapiro-Wilk		
	Statistic	df	Sig.
default	.963	10	.818
default_compression	.962	10	.811
noMD5_compression	.973	10	.919
noMD5	.975	10	.930

Table 17. Test of Normality

Default_Compression graphs





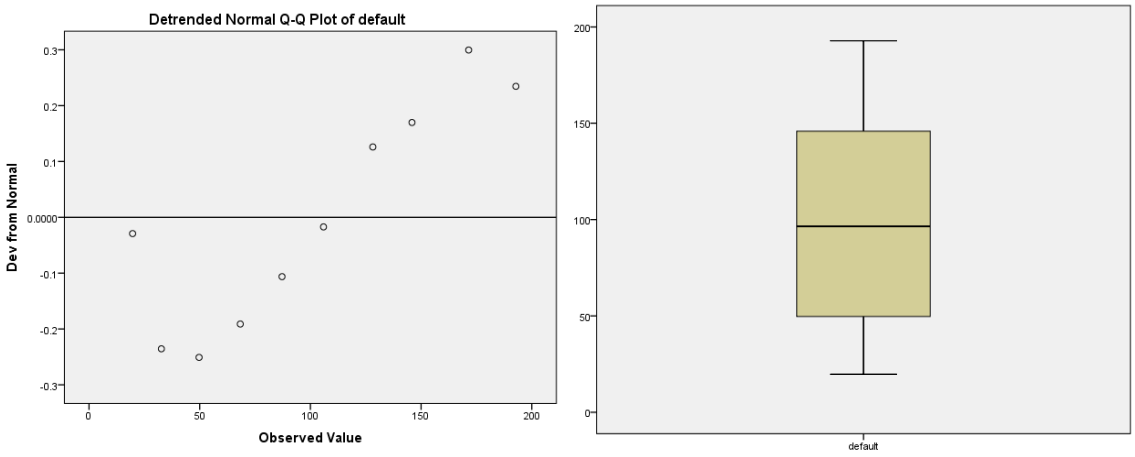


Figure 15. Default Normal Distribution Graphs without Compression

Default_compression graphs

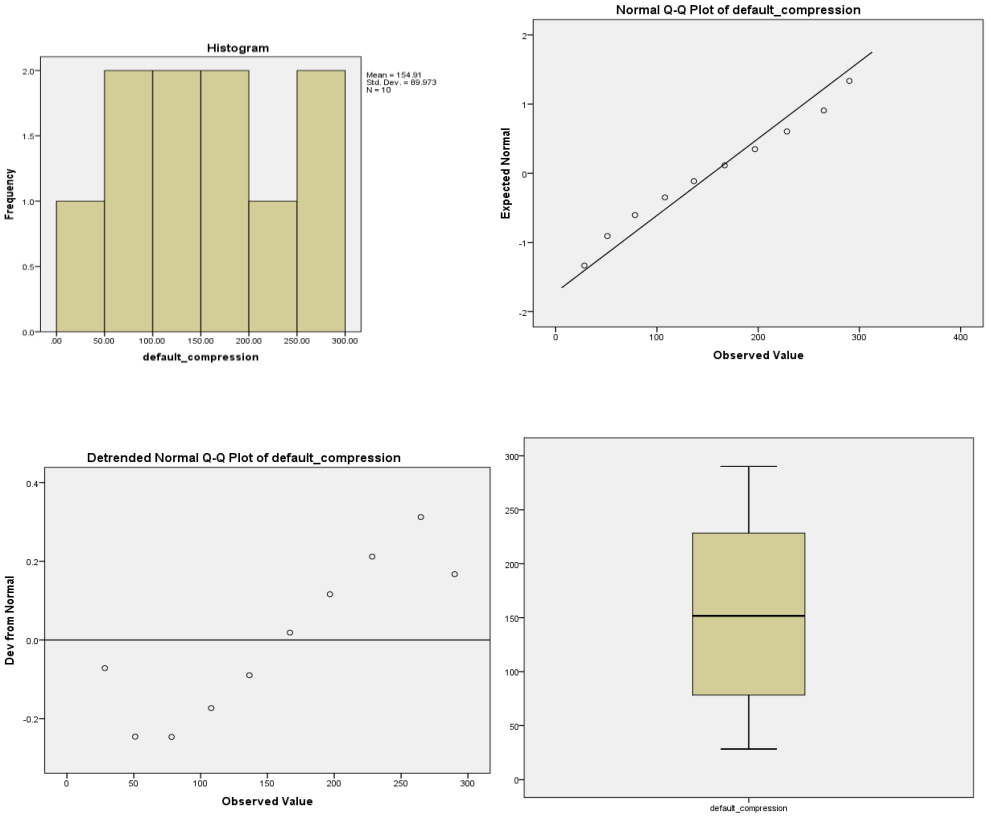


Figure 16. Default schema with Compression Normal Distribution Graphs

noMD5_compression graphs

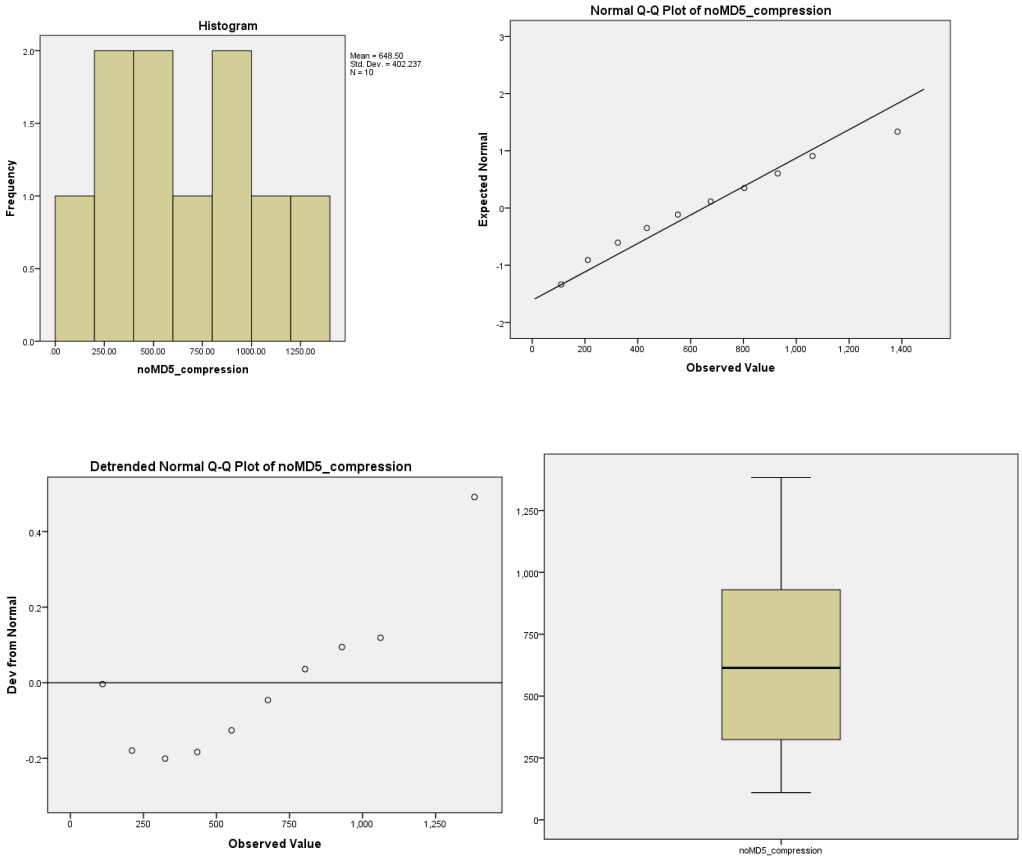


Figure 17. noMD5 with Compression Normal Distribution Graphs

noMD5 without Compression graphs

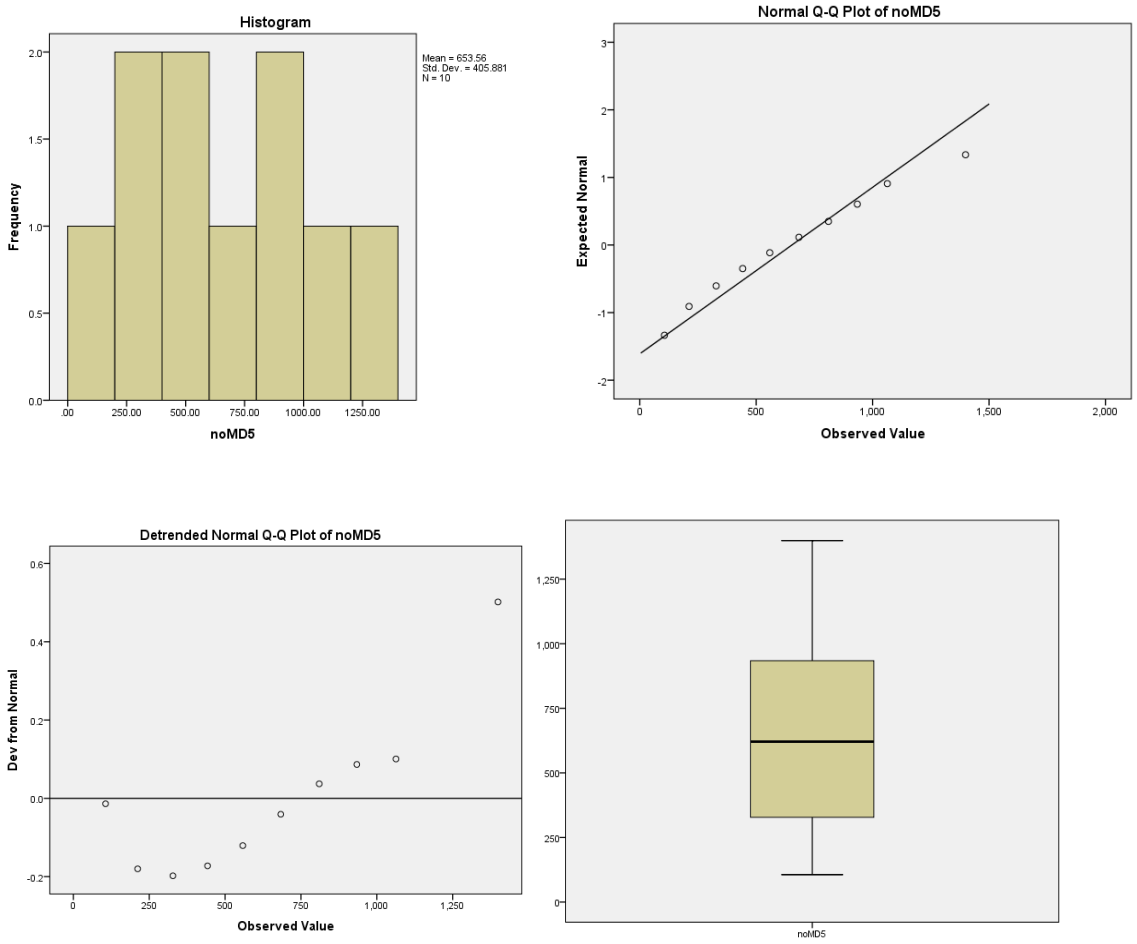


Figure 18. NoMD5 without Compression Normal Distribution Graphs

Appendix B (Model Assumptions)

We check the following model assumptions about residuals, ε, which are differences between observed and predicted responses [54]:

Normality: The probability distribution of ε is normal.

Linearity: Residuals should have a straight line relationship with predicted responses. The mean for probability distribution of ε is 0 over an infinitely long series of experiments for each setting of independent variable *x*.

Homoscedasticity: The variance of the probability distribution of ε is constant for all settings of the independent variable *x*.

Residuals Statistics[a]

	Minimum	Maximum	Mean	Std. Deviation	N
Predicted Value	12.66382027	187.6986389	100.1812315	58.88269338	10
Residual	-3.998921633	7.010179520	0E-15	3.847373847	10
Std. Predicted Value	-1.486	1.486	.000	1.000	10
Std. Residual	-.980	1.718	.000	.943	10

a. Dependent Variable: e2 (8 m1.medium)

Table 18. Residual Statistics for e2

The Residuals Statistics table summarises standardized, as well as unstandardized predicted values and residuals [48]. As shown, the mean of the probability distribution of ε is 0. Given the standardized values, we can also see that there are no outliers as the standardized values are around 1.5.

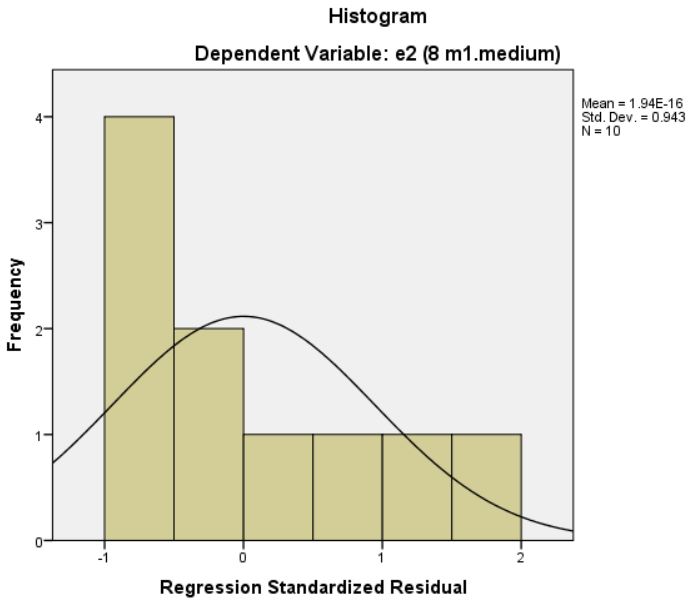


Figure 19. e2 Standardized Regression Histogram

It is difficult to tell whether or not the histogram is normally distributed due to the small number of values. If we look at the Normal P-P Plot of Regression Standardized Residual graph though we can see that the plotted points approximately follow a normal straight line [48].

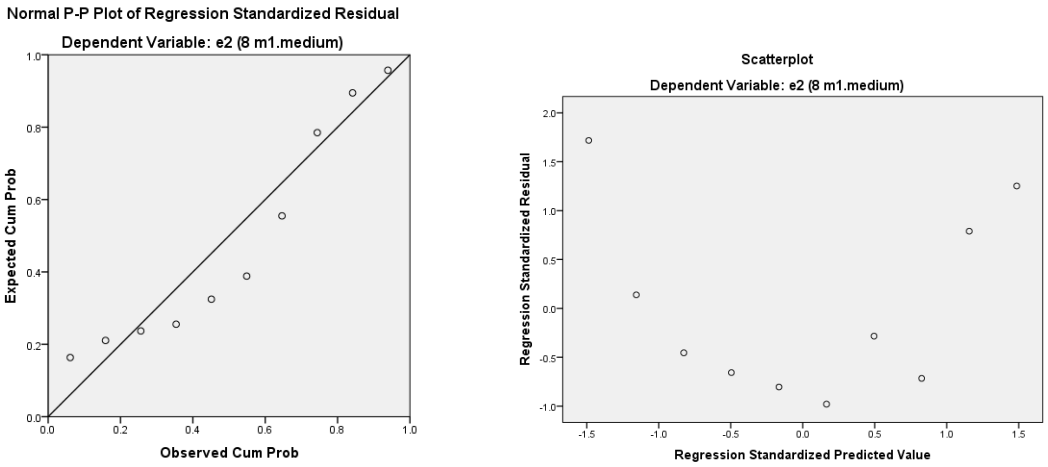


Figure 20. Normal P-P Plot and Scatterplot

The scatter plot of standardized residuals against predicted values graph shows a random pattern centred around 0. We can see no clear relationship between the residuals and predicted values which is consistent with assumption of linearity.