2019-12

MACHINE LEARNING OF EXTREMELY LARGE SETS OF SIGNAL COLLECTIONS USING CLUSTER COMPUTING

Ferris, Christopher L.

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MACHINE LEARNING OF EXTREMELY LARGE SETS OF SIGNAL COLLECTIONS USING CLUSTER COMPUTING

by

Christopher L. Ferris

December 2019

Thesis Advisor: Frank E. Kragh
Co-Advisor: James W. Scrofani

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2019	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
MACHINE LEARNING OF EXTREMELY LARGE SETS OF SIGNAL COLLECTIONS USING CLUSTER COMPUTING

5. FUNDING NUMBERS

6. AUTHOR(S) Christopher L. Ferris

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

Multitudes of signals are transmitted over the airwaves at any given moment, creating a large intelligence opportunity and reconnaissance problem. As technology advances, cluster computing methods must be explored to fill the intelligence gap caused by an increasingly large amount of data and a limited number of human analysts. In this thesis, Apache HBase, Phoenix, and Spark are employed on an AWS EMR cluster to store, query, and implement the K-means machine learning algorithm on a large-scale signals database. The signal databases tested consist of up to 100 million randomly generated signals, with nine feature columns of metadata. The signal data set is first bulk-loaded into HBase and a Phoenix layer is implemented. The data is then queried from Spark into a Dataframe for machine learning implementation. Additionally, the K-means implementations are run on multiple different computer-cluster configurations to test performance as a function of the number of computers in the cluster. This thesis demonstrates the capabilities and benefits of utilizing open-source software and cluster computing to implement large-scale machine learning on signal metadata.

14. SUBJECT TERMS machine learning, cluster computing, signal collection, signal analysis	**15. NUMBER OF PAGES** 89
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

MACHINE LEARNING OF EXTREMELY LARGE SETS OF SIGNAL COLLECTIONS USING CLUSTER COMPUTING

Christopher L. Ferris
Lieutenant, United States Navy
BS, Southern Illinois University at Carbondale, 2014

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2019

Approved by: Frank E. Kragh
Advisor

James W. Scrofani
Co-Advisor

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Multitudes of signals are transmitted over the airwaves at any given moment, creating a large intelligence opportunity and reconnaissance problem. As technology advances, cluster computing methods must be explored to fill the intelligence gap caused by an increasingly large amount of data and a limited number of human analysts. In this thesis, Apache HBase, Phoenix, and Spark are employed on an AWS EMR cluster to store, query, and implement the K-means machine learning algorithm on a large-scale signals database. The signal databases tested consist of up to 100 million randomly generated signals, with nine feature columns of metadata. The signal data set is first bulk-loaded into HBase and a Phoenix layer is implemented. The data is then queried from Spark into a Dataframe for machine learning implementation. Additionally, the K-means implementations are run on multiple different computer-cluster configurations to test performance as a function of the number of computers in the cluster. This thesis demonstrates the capabilities and benefits of utilizing open-source software and cluster computing to implement large-scale machine learning on signal metadata.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
AWS	Amazon Web Services
CLI	Command Line Interface
CSV	Comma Separated Value
DB	Decibels
EC2	Amazon Elastic Cloud Compute
EMR	Elastic Map Reduce
HDFS	Hadoop Distributed File System
JSON	JavaScript Object Notation
RDBMS	Relational Database Management System
RF	Radio Frequency
S3	Simple Storage Solution
SNR	Signal to Noise Ratio
SQL	Structured Query Language
SSH	Secure Shell
vCPU	Virtual Processor
YARN	Yet Another Resource Negotiator

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, I would like to thank my wife, Shelli, for her undying love and support. I would also like to thank my advisor, Professor Frank Kragh, for his mentorship and guidance throughout my time at NPS.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

This chapter provides the reader with the motivation, objective, organization and environment for this thesis. Also provided in this chapter is a literature review to orient the reader to related work upon which this research was built.

A. THESIS MOTIVATION

We live in the information age where multitudes of signals are being transmitted over the airwaves at any given moment, and they are simultaneously creating a large intelligence opportunity and reconnaissance problem.

As technology advances, there is a consistent and increasing demand for resources that operate in the radio frequency spectrum [1]. From pagers, cellphones, and tablets, to commercial and military radars, to interference from jammers and cyber-capabilities, as technological capability increases, so does the demand for wireless capabilities [2]. All of this information is being transferred over a wireless medium, thereby increasing the number of radio signals, resulting in a Big Data opportunity and challenge in terms of variety, velocity, and volume. This results in a further congested Radio Frequency (RF) environment challenging our capability to identify and assess the importance of the information contained within the environment [1].

In the age of terrorism, terrorist attacks provide a major threat to communities all over the world. Terrorist organizations and their agents are found in various locations worldwide, and the ability to track and locate them becomes increasingly difficult as the amount of data transmitted continues to increase [3]. Coincidentally, these people and organizations have access to these same technologies that many other countries and organizations are using to transmit across the airwaves. It is difficult to identify signals of interest that we do not know exist. Furthermore, effective reconnaissance requires an increasing ability to sort through excessively large amounts of signal data transmitted across the airwaves to identify what signals warrant further investigation.

While there are plenty of systems available and capable of data processing and analytics, as the data sizes grow beyond the capacity and capability of a single computer

system, there develops a requirement to move to a cluster of computers. Cluster computing, in combination with large data storage and machine learning, can play a vital role in helping to fill the intelligence gap caused by an increasingly large amount of data and a limited number of human analysts.

B. OBJECTIVE

The purpose of this thesis work was to demonstrate the ability to effectively store and apply machine learning to a significantly large dataset, too large to be processed by a single computer, across a cluster of computers. The dataset was designed to specifically resemble both the scale and features of transmitted signal metadata.

First, a dataset was obtained consisting of ten randomly generated signals and scaled up beyond the processing capability of a single computer with 128 GB of RAM, identifying the limit at which a single computer could process and analyze the dataset. The storage solution for the test data was then identified, which was provided by an open-source distributed non-relational database. To ensure upward scalability and reliability across a cluster of computers, this storage solution was implemented within the Amazon Web Services (AWS) cloud. Once the dataset could be uploaded and stored, tools and techniques were developed for the query and transfer the data into a data analytics tool capable of executing large-scale unsupervised machine learning algorithms, while still in the AWS cloud.

The goal of this thesis is to prove that large-scale cluster computing combined with readily available open-source tools provide a significant untapped resource. This resource can be applied for better analysis and identification of signals of interest when a database of signals is too large for any one human, or group of humans to analyze without the help of more than one computer.

C. ORGANIZATION

This thesis is divided into five chapters. Chapter I focuses on the problem introduction and the motivation behind the thesis work. Chapter I also contains the literature review, providing literary resources that were used to provide us with background

education prior to conducting thesis experimentation. Chapter II provides the necessary background information required by the reader to understand the tools and methods performed in this thesis. Chapter III details how each system and tool was designed and implemented for the experiment to include data uploading, storage, transfer, and machine learning analytics. Chapter IV provides the results of the experimentation. Finally, Chapter V details the conclusion, summarizing the work in this thesis and discussing future considerations.

D. ENVIRONMENT

This thesis details the implementation of large-scale data storage and analytics of signal data within a scalable distributed database environment, allowing for database growth in the future. To accomplish this, Apache HBase was chosen as the database for storage of signal data based on its scalability. A data analytics tool was needed which provided the cluster computing and machine learning capability while also being compatible with Apache HBase, therefore Apache Spark was selected for the machine learning software tool. To better streamline the compatibility between Spark and HBase, while also providing user-friendly data querying, Apache Phoenix was implemented as a layer on top of HBase. Finally, AWS was chosen for streamlined resource management and large-scale cluster computing capability. These capabilities combined with the AWS integration with Apache HBase, Phoenix, and Spark through Amazon Elastic Map Reduce (EMR) made AWS the most ideal option for this thesis.

E. LITERATURE REVIEW

There are a broad range of terms, tools, and methods used in this thesis. This literature review is presented to provide the reader an overview of these topics so as to develop a better understanding of the information that are discussed herein.

The term Big Data is widely used in the modern world as data sizes grow and storage and analytical requirements become more difficult to execute. In [4], the authors define the term Big Data, discuss issues and challenges, and provide methods for how to analyze Big Data.

Machine Learning provides many possible methods for analyzing and making sense of Big Data. In [5], the author details the various kinds of machine learning and machine learning models. A more in depth and specific overview of K-means clustering machine learning algorithms and methods is provided in [6], which includes the method chosen for this thesis work.

The Apache Hadoop Ecosystem is comprised of open-source software tools which can be used for various types of data storage and analytics in a computer cluster environment. References [7] and [8] provide a comprehensive description of the Hadoop framework, file system, and many of the tools available.

Apache HBase is an open-source scalable non-relational storage solution. In [8], [9], and [10], the authors describe the design and functionality of HBase, the column-oriented schema, and the integration of HBase with other Hadoop tools. These sources also discuss administrative features and usage concepts.

Structured Query Language (SQL) provides streamlined data querying within large database storage solutions. Apache Phoenix layers SQL capability on top of the HBase framework. In [9], the authors describe how this functionality is achieved. A better understanding of the difference between SQL and no-SQL databases can be found in [11] and [12].

For Data Analytics and Machine Learning, Apache Spark is currently the most capable tool in the Hadoop toolbox. Collectively, [13], [14], [15], and [16] provide an all-encompassing guide to Spark data analytics and machine learning applications. These resources also cover the ability of Spark to integrate with other Hadoop tools.

AWS provides a suite of installed Apache Hadoop cloud-based tools and services through AWS EMR. A detailed guide to all aspects of AWS EMR can be found in [17].

II. BACKGROUND

This chapter includes information pertaining to the open-source tools and concepts required to understand the purpose, experimentation, and results of this thesis work.

A. BIG DATA

Big Data analysis has gained much momentum in recent years. This section defines and outlines Big Data for better understanding of the application to this thesis.

1. Volume, Velocity, and Variety

Garner, a world-leading IT research and consultant company, defines Big Data as "data that contains greater variety arriving in increasing volumes and with ever-higher velocity [7]." This definition contains what are generally referred to as the three V's of Big Data, volume, velocity, and variety [7]. A descriptive diagram of the three V's can be seen in Figure 1.

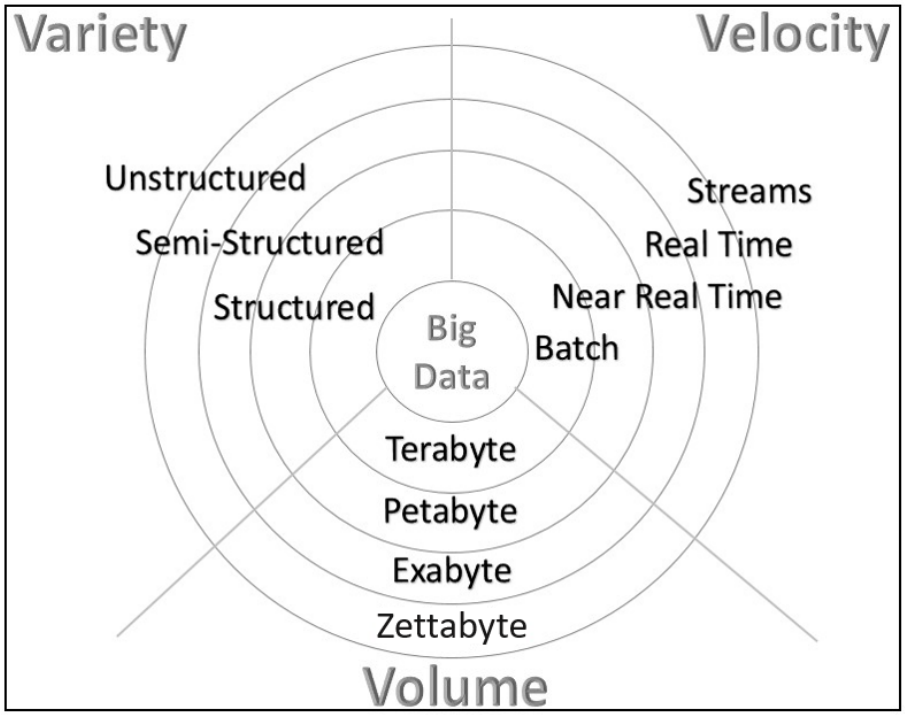

Figure 1. Diagram of the Three V's of Big Data. Adapted from [4].

Volume refers to the quantity of data that has been and is continuing to grow as technology advances. Data volumes are becoming increasingly larger and generally more unstructured [18]. While structured data is relatively organized and easily evaluated, unstructured data is generally random in nature and difficult to analyze [4]. This unstructured data ranges from internet webpage data, to social media click data, to various sensor data. The size of Big Data can range from terabytes (10^{12} bytes) to petabytes (10^{13} bytes) or larger, and these bounds continue to expand [18]. According to an estimate by the International Data Corporation, in 2013, the size of the digital universe was 4.4 zettabytes (10^{21} bytes), forecasted to increase tenfold by 2020 [19].

Velocity, as described by [7], is the rate at which data arrives from the data sources. Velocity also refers to the speed at which the data is expected to be processed or analyzed. In the current technological environment, data is expected to be ingested, analyzed, and stored at a real-time or near real-time pace. As the volume of the data increases, so does the challenge of maintaining capabilities that can address the corresponding increase in velocity [7].

Variety describes the continually growing and changing of data types and formats available. Data comes in from a variety of sources, in varying data types and formats such as audio, video, text, and sensor data as just a few examples. All of this data requires different processing or pre-processing prior to ingestion, causing strain on the velocity of processing, again directly affected by the increasing volume of data [18].

2. Data Management

In addition to the volume, velocity, and variety of Big Data, the overall management of the data also proves a significant challenge. Effective management of Big Data consists of three steps: Integrating, Managing, and Analyzing [18].

As described within [18], integrating the data refers to the pre-processing of the incoming data to ensure that it is properly formatted for ingestion into the chosen storage solution. Managing the data also requires the storage solution to have the scalability necessary for continual growth in storage needs over time. As data storage requirements grow, the use and availability of cloud storage becomes a more desirable option, which is

discussed further later in this chapter. The analysis of the data requires data analytics tools capable of not only querying the data stored, but the ability to read the multiple data formats and provide an analysis of the data. This analysis provides further insight and meaning into the dataset itself. Particularly, Artificial Intelligence and Machine Learning are two cutting edge large-scale data analysis processes in use by industry [18].

B. APACHE HADOOP ECOSYSTEM AND TOOLS

The Apache Hadoop Ecosystem is a widely used, open source library of software tools and applications for storage, management, and analysis of Big Data [19]. This section outlines the Hadoop Ecosystem and describes applicable software tools relevant to the understanding of this thesis.

1. Apache Hadoop

Apache Hadoop was created in 2006 by Doug Cutting, the creator of the widely used text search library Apache Lucene. Hadoop provides Big Data distribution with automatic fault tolerance and redundancy built in, making it a secure, effective, and popular option for Big Data. The built-in fault tolerance and redundancy supports the storage and analysis of extremely large datasets across thousands to tens of thousands of nodes [19].

Hadoop relies upon other built-in components for file storage as well as resource management. For file storage, Hadoop utilizes the Hadoop Distributed File System (HDFS). For resource management, Hadoop relies upon Yet Another Resource Negotiator (YARN). For default data processing, Hadoop uses MapReduce [19].

a. HDFS

Hadoop is built upon a foundational distributed file storage system, HDFS, which provides the capability for both high-speed and high-efficiency read and write operations on large sets of data, all while stored within the file system [19]. HDFS enables the large-scale data file storage and analysis required for Big Data. When a dataset becomes too large for a single machine, it requires partitioning across multiple machines, known as a distributed file system [19]. HDFS is purposefully designed for storing extremely large amounts of data running on multiple computer nodes. Furthermore, HDFS is designed to

be a cost-efficient storage solution even when employed on commodity hardware, thus allowing the user to focus attention on the analytics of the data vice the systems in which the data is stored and analyzed upon [7].

b. YARN

YARN is built within the Hadoop Ecosystem for job scheduling and management of jobs and resources. The resource management capability provided by YARN allows users to simultaneously execute different applications on the same nodes as the data storage, which decreases workload and increases operating efficiency [10]. All YARN resource management actions take place behind the scenes and are done automatically, allowing the user to focus on the software application and data analysis instead of resource management. Figure 2 displays the conceptual framework with the various applications in the application layer. The application layer is built upon YARN in the computation layer, all of which are built upon on the storage layer with use of HDFS and HBase [19].

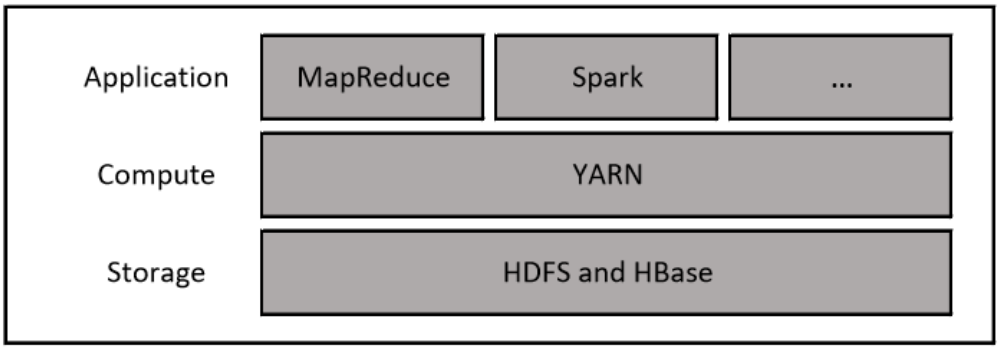

Figure 2. Diagram of Hadoop Ecosystem, Storage, Computation, and Application. Adapted from [19].

This allows for a separation between data processing and resource management [7].

c. MapReduce

MapReduce is the default data processing engine built into the Hadoop Ecosystem for providing data processing functionality. While other more capable data analytics tools have taken over much of this role, MapReduce is still used as the ground-level processing

system for parallel processing of large datasets within Hadoop and Hadoop-related applications. While MapReduce is still effective for processing data, it does not provide as much capability to perform all aspects of data analytics in comparison to other newer Hadoop applications, such as Apache Spark [19]. For the purpose of this thesis, MapReduce will only be used in limited functionality for basic Hadoop-related data loading operations to be discussed in a later chapter.

2. Apache HBase

HBase is an open-source column-oriented database storage solution licensed by Apache, and currently in use by a plethora of companies such as Adobe, Facebook, Twitter, Yahoo!, Ancestry.com, and Mozilla [9]. This section describes the functionality and schema of HBase and what makes it a viable storage solution for large unstructured datasets.

a. Relational Database Management Systems (RDBMS) and SQL

RDBMS, also commonly referred to as SQL databases, have been widely used and readily available for over 40 years and still provide a useful storage solution for various companies and organizations [8]. While RDBMS are a great tool for processing business transactions, one significant problem with RDBMS is the lack of rapid scalability for extremely large sets of data. RDBMS also have very specific guidelines for their structured data architecture as well as the requirement for a predefined schema. This leaves SQL RDBMS systems as a less effective storage solution for many varying datasets and usage applications [11].

b. HBase Structure and Schema

Apache HBase is a NoSQL non-relational database which is built upon the Hadoop Ecosystem and HDFS. HBase was originally created as an Apache subproject for the storage of purely structured data. As the project continued to develop, however, the concept was expanded to store semi-structured and unstructured data as well making it a highly adaptable data storage solution. The allowance of unstructured data in combination with

the non-relational aspect of HBase allows for a schema variability that is not commonly found within SQL or other relational databases [19].

HBase uses a column-oriented approach for data storage, which allows for better data compression and easier query capability than the row-oriented approaches that most RDBMs use. Within an HBase table, the column is the lowest basic unit, and columns together create a row. Each individual row is uniquely identified by a Row Key. The rows are then sorted by their respective Row Key, compared byte to byte, and therefore listed as such from top to bottom. Groups of columns together form column families where columns can be partitioned together as specified by the user during table creation. This particular design allows for separation between groups of data for optimal compression and memory allocation [8]. The schema of an HBase database is displayed in Figure 3.

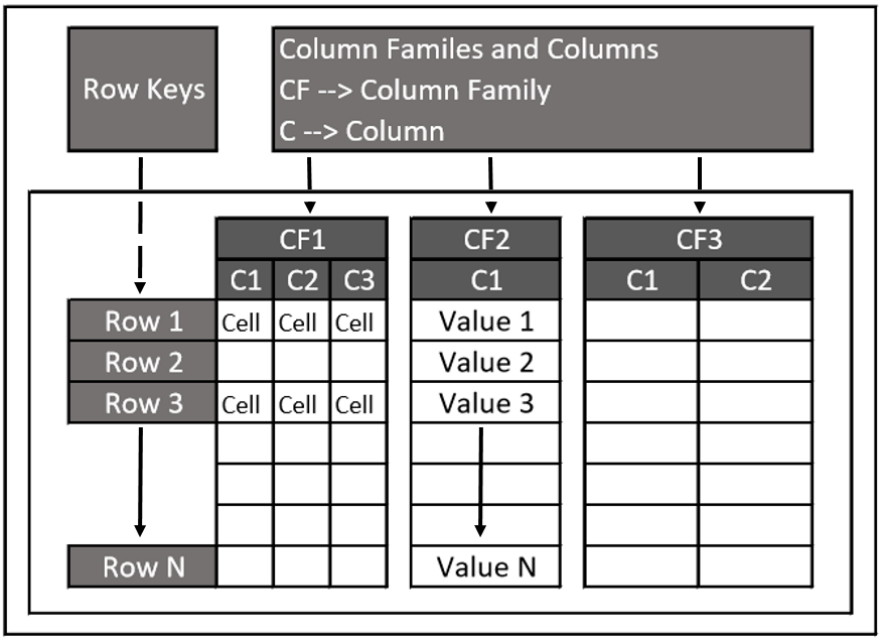

Figure 3. HBase Table Schema. Adapted from [9].

Finally, the columns within a column family are all stored together within an HFile, the standard low-level HBase file format. As a standard, HFiles are stored within the Hadoop Distributed File System (HDFS), upon which HBase is built upon [8].

c. HBase Regions and Autosharding

Aside from the data structure variability that HBase provides, it is also scalable to handle Big Data datasets. HBase accomplishes this by distributing data across multiple partitions, which are defined as Regions. The various Regions are managed by Region Servers and spread across multiple Data Nodes as necessary, dependent on the size of the data stored. As with the resource management capabilities of YARN, HBase will automatically create new Region Servers whenever necessary to balance the amount of data storage, thereby allowing for fast and efficient scaling with the dataset [8].

The rapid scalability and load-balancing capability of HBase is a result of the method by which data is divided when stored, known as autosharding. Each HBase Region consists of multiple rows, grouped and stored together. When groups grow too large, they are then divided up and distributed automatically. When an HBase table is created, it has only one Region. As data is continually loaded into the table and the amount of data storage grows, the system will ensure the maximum allowance is not exceeded by dividing the regions as necessary [8]. The division of rows and regions is shown in Figure 4.

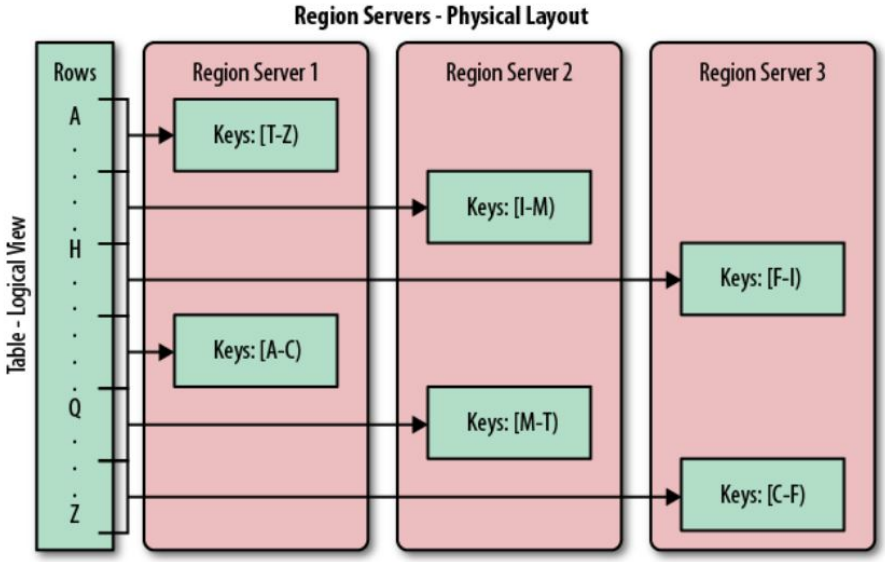

Figure 4. HBase Region Autosharding across Regions and Servers. Source: [8].

d. HBase shell

The HBase shell is the command line tool for interacting with HBase. Within the HBase shell, the user can create or delete tables, insert or remove table data, and perform a few other tasks [8].

e. Apache Zookeeper

HBase relies upon a master server to assign the various Regions to the Region Servers. To accomplish this, HBase uses a highly reliable coordination tool, Apache Zookeeper. Apache Zookeeper is a built-in functionality within HBase that provides coordination services for distributed data applications, specifically for synchronizing the various datasets within HBase. It manages that aspect automatically behind the scenes, relieving the user from these important computer cluster coordination tasks. Zookeeper provides consistency, durability, synchronization, and concurrency within HBase [8].

f. Apache Phoenix

Apache Phoenix is an application that operates on a relational database engine, thereby providing relational SQL query capability on Hadoop, yet utilizes HBase as the data backing storage [20]. Therefore, users can store unstructured non-relational data within the structure and scalability of HBase and add Phoenix as a layer atop of their HBase tables. Layering Phoenix onto the pre-existing HBase tables make use of a more commonly-understood SQL language, for the purpose of making the use of HBase more user friendly [21]. Additionally, Phoenix-based queries are run within the same nodes on which the data is stored, providing for query speed efficiency. While Phoenix runs a dedicated command line interface, no data is actually transferred from the HBase database, therefore the initial database remains intact for storage purposes and there is no cost associated with movement of data from one application to another [20].

3. Apache Spark

Apache Spark is a highly capable open-source data analytics and data processing tool known for the capability to provide data processing at a large scale, currently being

used by entities such as Uber, Netflix, NASA, CERN and the Broad Institute of MIT and Harvard for large scale data analysis and machine learning [15]. This section will provide an overview of some of the capabilities Spark provides, necessary for further understanding of this thesis.

a. Spark Structure

As described in [15], Spark is designed primarily as a computing engine for large-scale parallel analysis to be performed across a cluster of computers. There are three key components intertwined that make Spark an effective data analytics tool. Spark can act as a unified platform, while also solely as a computing engine, all while providing the necessary libraries to perform a wide array of data analysis tasks. Since Spark has a sole purpose as a computing engine, it is therefore limited in other capabilities that may be required by the user. For instance, Spark will pull data from a data source or database for analysis, however, does not provide permanent storage for the pre-processed or resulting data. The storage of the data relies on an integrated storage solution such as HDFS or for large scalable data storage, HBase or Phoenix tables as mentioned earlier in this chapter. Spark can also store data within the Amazon Simple Storage Solution (S3), which will be covered later in this chapter. Spark can read in stored data from multiple different file types, to include Comma Separated Values (CSV), JavaScript Object Notation (JSON) and others. Spark is also capable of connecting and extracting data from other large database storage applications such as HBase and Phoenix [15].

b. Computer Cluster Management

As previously mentioned, the benefits and capabilities of cluster computing greatly outweigh the use of a single computer for large datasets. For this to work efficiently and effectively, proper coordination between the groups of machines is essential. In the same manner as Hadoop, Spark uses YARN as the computer cluster manager. As with Hadoop, YARN operates in the background of the Spark application, allowing for the user to focus solely on data analytics [15].

c. Dataframes and Partitioning

To distribute the data effectively across computer clusters, Spark uses what is called a Dataframe. A Spark Dataframe is a structured Application Programming Interface (API) that is representative of a table of data. Like a standard table, Dataframes are also composed of columns and rows. Instead of holding data within rows and columns on a spreadsheet as on a single computer, Spark spreads the data across up to thousands of computers with the use of the Dataframe. To do this, the data stored in the Dataframes are broken into partitions, allowing the executer processes to work in parallel. These partitions also identify the way in which the data is distributed across the computer cluster. This entire process of Dataframe partitioning is performed automatically by Spark behind the scenes without any user input or ability to manipulate how the data is partitioned [15].

d. Programming Language APIs

As described in [15], Spark was originally written in Scala. However, it does allow for programmers and developers with experience in other programing languages the option to program in Python, Java, and R. For each of the different language options, there is a different command line interface, or shell [15]. For the purpose of this thesis, Scala was the chosen programming language for Spark.

e. Spark for Machine Learning

Also outlined in [15], Spark contains an extensive library of machine learning functions, called MLlib, which are designed to be used with large-scale datasets. MLlib functions are capable of performing all aspects of machine learning from the pre-processing of a dataset, to the training of models, to predictive analysis. One requirement, however, is that the data must be represented as numerical values. Therefore, if any other form of data is present, such as string values, the data must be transformed into numerical values prior to the machine learning application [15]. Further details of machine learning will be discussed in the next section of this chapter.

C. MACHINE LEARNING

Machine learning is one of the most widely-used methods of analyzing Big Data. At the most basic level, machine learning can be defined as the analysis of datasets and data trends for the purpose of teaching and training machines, thereby creating a better understanding of the original and future datasets of similar type [18]. This section provides an overview of the classes of machine learning and the specific method used within this thesis.

1. Supervised and Unsupervised Learning

Machine learning approaches are commonly divided into two classes, supervised and unsupervised learning. Supervised learning requires a set of data to be initially identified as a training dataset. The training data is data which is representative of all the data that will be analyzed in the future. The features and trends of the identified training data are analyzed and identified, thereby providing a standard by which to assess future datasets. The key to supervised learning is that one has the ability to know something about the training data prior to the analysis of the rest of the dataset [5].

Unsupervised learning is machine learning without a training dataset. Therefore, the models and algorithms applied conduct all analysis with no prior knowledge of the data. A common type of unsupervised learning, sometimes used synonymously, is clustering. Clustering is the grouping of data based upon the analysis of data attributes without any prior knowledge or labeling of the dataset. Clustering identifies similarities and differences between the data and then sorts the data points according to those similarities and differences. Similar data is grouped together, whereas less-similar data is grouped separately [5].

2. K-means Clustering

K-means clustering is one of the most popular and widely used clustering machine learning algorithms [15]. K-means clustering can identify patterns or correlations between a set of data points that may not be easily determined otherwise [6]. This section explains

how the K-means algorithm works for a better understanding of the application throughout this thesis.

a. K-means Process

K-means begins with the user selecting a K value. This value represents the number of clusters into which the data will be separated. Initially, the cluster centers are assigned at random to different points within the data. Once each cluster center has been assigned to an arbitrary data point, the rest of the dataset is then assigned to a cluster. The cluster assignment is made based on the smallest Euclidean distance between the data point and the cluster centers. Once all points have been assigned a cluster, the center of each cluster of data points is recalculated and identified as the new centroid. Subsequently, once the new centroid is identified, the process begins again, and all the data points are reassigned to a cluster based on the distance from the previously computed cluster centroids. Once assigned, the centroid for each cluster is updated to the new center of each cluster, and the process repeats again. This process continues to repeat until the position of the centroids converge and no longer change in comparison to the previous position [15]. While this repeating process will perform the most reasonable cluster assignment for the dataset, how well the similarities and differences are represented by the results is reliant upon the number of clusters K that the user selected when running the algorithm [14].

b. K-means Algorithm

As an example, if you have a set of data $X = \{x_i, i = 1,...,n\}$, where the dataset is comprised of n data points which will be grouped into K clusters such that $C = \{c_k, k = 1,..., K\}$ [6], the algorithm will group the data, utilizing the process listed earlier in this chapter, until the resulting set of clusters is reached where the squared error of each of the data points within the clusters are minimized and the centroid position converges. If we let μ_k be the mean of the cluster c_k, the squared error between the points within the cluster is defined as

$$J(c_k) = \sum_{x_i \in c_k} \| x_i - \mu_k \|^2 . \quad (1)$$

The end goal of implementing K-means is to minimize the sum of the squared error over all the K clusters. As the number of clusters increase, there will be less data points in each cluster. Additionally, the data points assigned in each cluster will be closer in proximity to the centroid, decreasing the sum of the squared error [6]. A random set of data generated from [22] can be seen in Figure 5. Figure 6 displays the result of running K-means on the same set of data with $K = 2$. Finally, when K-means is run on the same set of data again, with $K = 3$, the result is displayed in Figure 7.

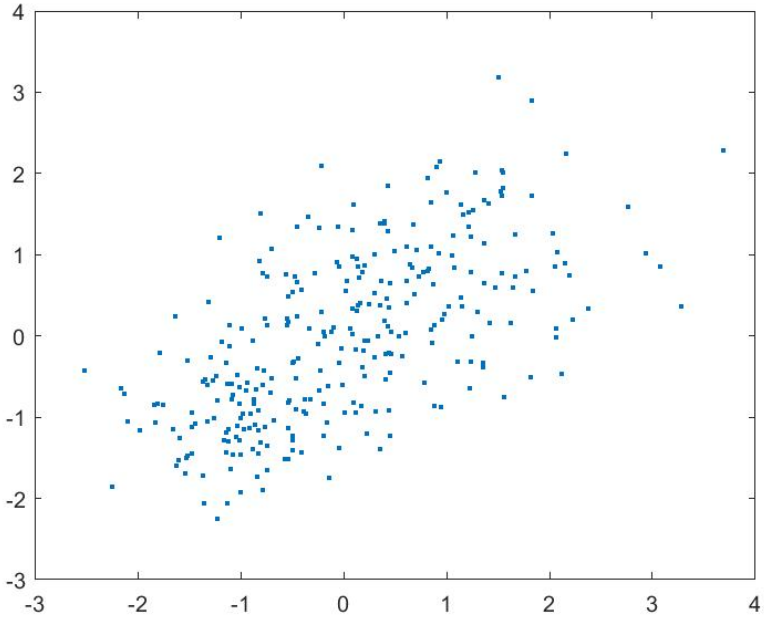

Figure 5. Plot of Randomly Generated Data. Source: [22].

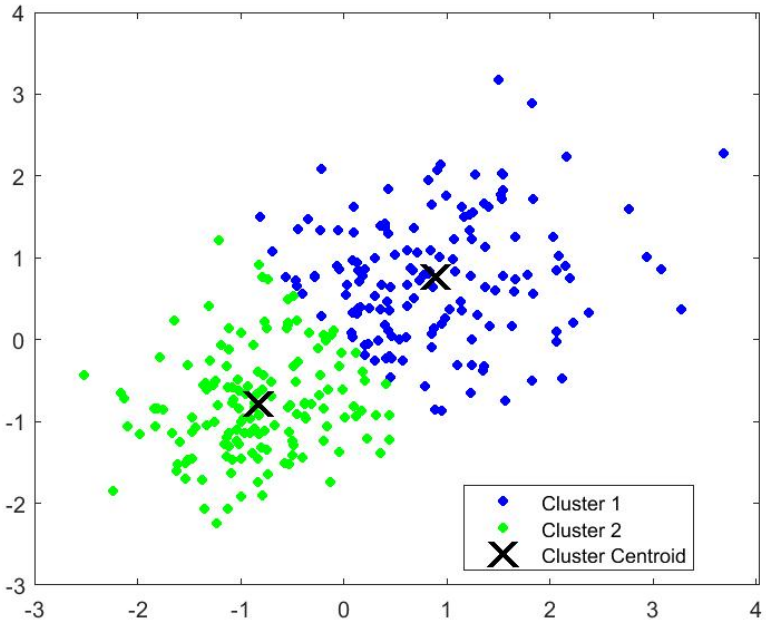

Figure 6. K-means Example Results from Random Data, $K = 2$. Source: [22].

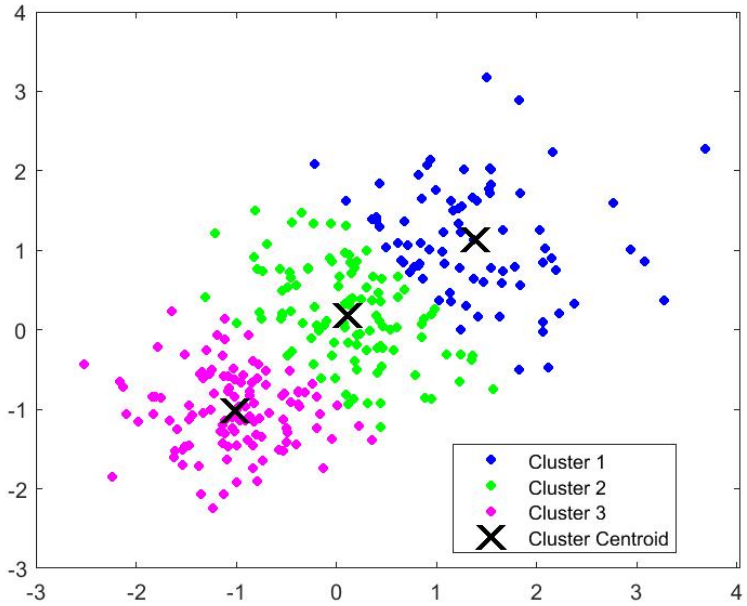

Figure 7. K-means Example Results from Random Data, $K = 3$. Adapted from [22].

c. Evaluating the K-means Model

As mentioned in the previous section, as the number of clusters increases, the total error decreases, resulting in a better cluster-represented result. The goal is to find the best cluster representation of the dataset where the total error is at a minimum. However, as K continues to increase and approaches the number of data points in the dataset, the resulting total error is reduced to zero. Therefore, once K is equal to the number of data points, there is only one point in each cluster and the cluster representation does not actually group any data points together [6].

One of the most common methods of choosing a reasonable number of clusters is by using what is commonly referred to as the elbow method [14]. As described previously, each time you run K-means with a specified K value, the total error, also commonly referred to as the cost, can be calculated. As the number of clusters increases, the cost will decrease. By plotting all the calculated cost values, the resulting plot will often have a noticeable point or set of points at which the slope abruptly decreases. This is the point, or set of points, at which the change in cost is converging to zero and there will no longer be a significant change to cost by increasing the number of clusters. An example of this can be seen in Figure 8, where two slight elbows can be seen at the K value of about 15 and at 36. However, it is worth noticing that as the number of clusters is increased from 15 to 36, the computed cost only changes by approximately $0.10 \text{x} 10^8$. Therefore, there is only an average cost decrease of approximately two percent per each increase in K value from 15 to 36. There continues to be a lot of debate on the best particular method to select the most optimal K values, however this method proves a reasonable option [14].

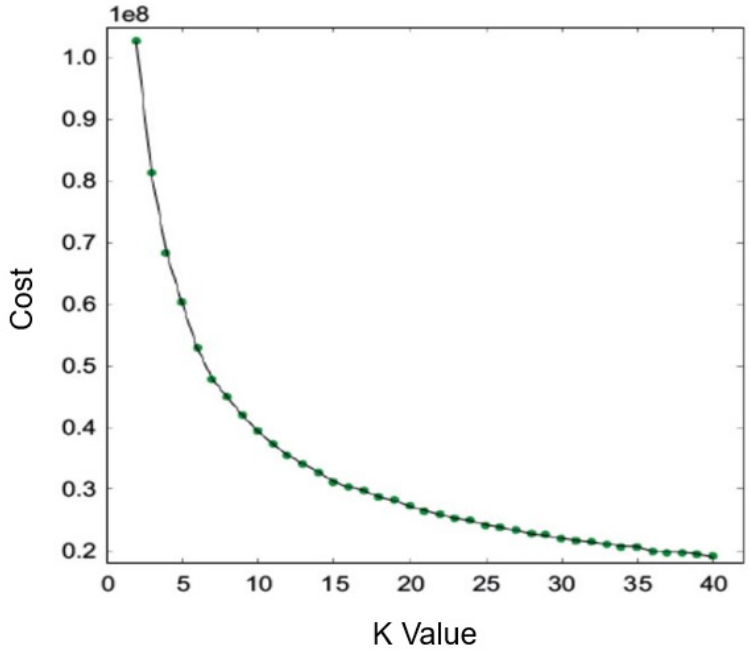

Figure 8. Example Cost Analysis of Dataset Models Trained from K values 2 to 40. Adapted from [14].

D. AMAZON WEB SERVICES (AWS)

AWS offers online cloud computing services which include various tools and services which can be used for Big Data analysis. This section provides an overview of the specific services available from AWS that were used for this research.

(1) Amazon Elastic Compute Cloud (EC2)

Amazon EC2 provides a virtual computing capability within the AWS cloud. EC2 also manages the various aspects of scalability by managing computer cluster configurations. As with the YARN resource managers found in Hadoop, the EC2 automated computer cluster management allows the user to focus on the applications within while AWS handles the rest. The virtual environments provided within EC2 are referred to as instances [23].

(2) Amazon Elastic Map Reduce (EMR)

Amazon EMR is a platform offered through AWS which provides users with a customizable large-scale cluster computing capability fully managed by AWS [24]. Within the EMR framework, Amazon includes many data storage and data analytics tools fully installable onto a singular computer cluster, or multiple computer clusters, customizable by the user. These tools include but are not limited to Apache Hadoop, HBase, Phoenix, and Spark. There are a multitude of other tools which can be installed within an EMR computer cluster, however they will not be discussed within this thesis [17].

EMR computer clusters are comprised of multiple EC2 instances, where each of the instances is referred to as a node. The EMR computer cluster is divided into Master nodes, Core nodes, and Task nodes. The Master node is responsible for software management, data distribution, and maintaining computer cluster health. Within these responsibilities, the Master node also assigns the analytics and processing of data amongst the other nodes. The Core node executes tasks and stores data within HDFS within the computer cluster. The Task nodes only perform designated tasks and do not store any data. If a multi-node computer cluster is chosen for launch, it will contain at a minimum one Master node and one Core node, whereas the Task node is optional [17].

(3) Amazon Simple Storage Solution (S3)

Amazon S3 is a secure and scalable AWS-hosted storage service that is available and integrated within Amazon EMR. Amazon S3 also provides easy-access solution for data uploading, downloading, and storage within the cloud and is available to all installed tools within the computer cluster. While HDFS storage memory within the EMR computer cluster is cleared upon deactivation of a computer cluster, S3 maintains data storage even when a computer cluster is not running, making it a more viable storage solution for the experimentation conducted for this thesis [25].

THIS PAGE INTENTIONALLY LEFT BLANK

III. DESIGN AND IMPLEMENTATION

This chapter explains the experimental setup and design of the tools utilized within this thesis, complete with details regarding the process in which each of the tools were implemented.

A. DESIGN GOAL

The experimental design for this thesis begins with a large dataset of signal metadata which requires increasingly scalable storage and query access to various subsets of data within the dataset. Once stored, the data must be accessible from a data analytics tool by which machine learning algorithms can be applied to the selected dataset.

Spark was chosen as the data analytics tool due to the extensive data analytics and machine learning capability of Spark [14]. HBase was chosen as the scalable unstructured storage solution for the signal dataset, however HBase by itself is neither easily queried from nor easily accessed by Spark [8]. Therefore, Phoenix was selected as a layer on top of the HBase storage to provide both an ability to query the data via SQL and to load the data into Spark [26]. A design flow chart can be seen in Figure 9.

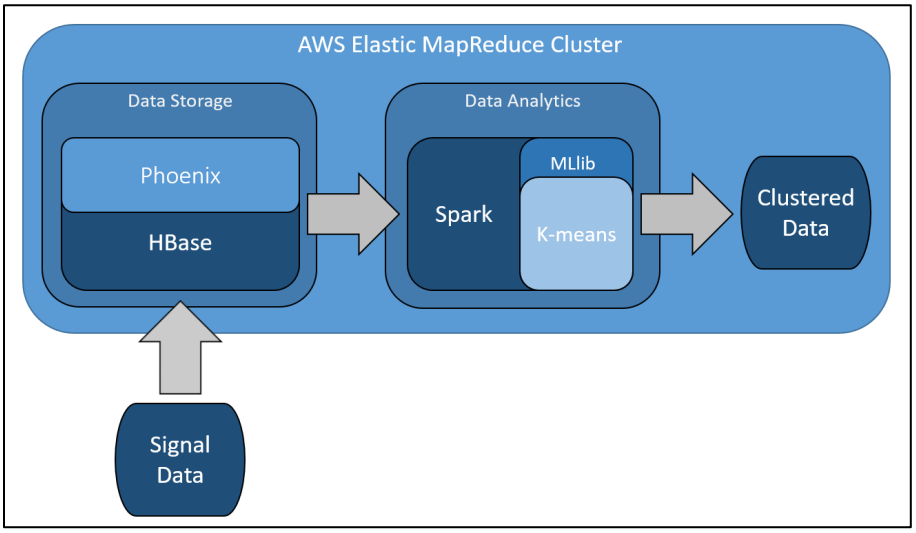

Figure 9. Flow Chart of Experimental Design.

B. DATA SET

The experiment conducted in this thesis required a dataset that would be representative of actual RF signal data. Therefore, the dataset used for this thesis began with 10 signals randomly generated in MATLAB with calculated values for nine parameters: duration, average amplitude, average power, peak to average power ratio, signal to noise ratio (SNR), SNR in decibels (dB), 3-dB bandwidth, 10-dB bandwidth, and noise equivalent bandwidth [27].

This set of signal metadata was then expanded into a larger dataset. To ensure that the expanded dataset would be representative of the original dataset, it was necessary that the correlation between each of the metadata values be considered across the set of signals [28]. To accomplish the upscaling of the dataset, the Multivariate Normal Random Number Generator in MATLAB was used. As described in [29], the Multivariate Normal Random Number Generator function takes the covariance matrix and mean vector as input along with the initial dataset to be expanded. This allows for an output of any specified number of test signals, each with the same nine metadata parameters, with the same correlation between each of the parameters [29]. Of note, due to the generation method, the dataset is not realistic for the real world where a signal database would likely contain more diversity in signal information while representing multiple types of signals from multiple types of emitters. However, this dataset proved sufficient to demonstrate the proof of concept for this thesis.

To resemble a dataset large enough to necessitate the need for a cluster of computers, the dataset was generated on the most powerful computer system available containing a 3.20 GHz processor with 128 GB of RAM. On this system, the generation of 10^8 signals, a file size of 15.2 GB, was the largest which the computer could generate within RAM capability.

C. AWS EMR SETUP

AWS was chosen for the cloud-based cluster computing capability including access to HBase, Phoenix, and Spark; all available through EMR [17]. This section describes the setup and implementation of EMR computer clusters used for this thesis.

(1) AWS Command Line Interface (CLI)

The AWS CLI provides integration and management of AWS computer clusters from within the command line of the user's computer [30] and therefore required installation. Appendix A provides a detailed description of how to install the AWS CLI.

(2) Amazon S3

Amazon S3 was chosen for secure and scalable storage of the AWS EMR data used in this thesis. As described in [25], using S3 allows for ease of uploading of the generated dataset, and the downloading and storage of data for analysis. S3 also provides the backup storage for HBase when installed on the computer cluster [31]. A singular S3 storage folder in AWS is referred to as a "bucket" [25]. Appendix A provides a detailed description of the process used to create an S3 bucket.

(3) Launching an EMR Computer Cluster

In launching an EMR computer cluster, there are a multitude of customization options available as described in [17]. Some important design configurations considered were node, for instance, size and capability. For this thesis, a general-purpose node size of m4.2xlarge was selected. Each m4.2xlarge instance contains 8 virtual processors (vCPUs) and 32 GB of memory [32]. This size was found to be the smallest size capable of launching HBase, Phoenix, and Spark all on the same computer cluster with a single node. While one could inherently get more capability from nodes with more vCPUs and/or memory, for consistency in experimentation, all nodes or instances launched were launched as m4.2xlarge nodes. In addition to selecting instance size, there was the requirement to create an Amazon EC2 Key Pair for login encryption prior to launching a computer cluster[33]. Appendix A provides a detailed description of EC2 Key Pair creation and the instructions creating and launching an EMR computer cluster with HBase, Phoenix, and Spark installed.

(4) Hadoop Command Line

The Hadoop command line was used to access and launch HBase, Phoenix and Spark. As described in [34], to access the Hadoop command line, a connection to the

Master Node must be established through an SSH connection. Additionally, PuTTY and PuTTYgen software were downloaded and installed. PuTTYgen was used to transform the Key Pair created to the format necessary to launch the Hadoop command line via PuTTY [34]. Appendix B provides a detailed procedure for setting up an SSH connection, downloading and installing PuTTY and PuTTYgen, and launching the Hadoop CLI window.

D. HBASE DESIGN AND IMPLEMENTATION

HBase was implemented as the data storage for the signal dataset because of the allowance for any type of structured or unstructured dataset and the upward scalability and load balancing capability [9]. This section describes the HBase schema designed for this thesis, how the designed table was created, and the method chosen to upload the dataset into HBase.

1. HBase Schema

The HBase schema designed for this thesis consists of a singular column family which contains nine columns for the metadata as listed in section B of this chapter [8]. In a real-world application, additional column families could be added to store I and Q data snippets or other pertinent signal metadata such as location or time information. However for the purpose of this thesis, only a single column family was necessary to store the large dataset which was queried and then used for K-means machine learning analysis.

The Row Key assignment to each signal placed into the HBase table should be unique to the signal and there are various ways to assign Row Key identifiers [8]. Various combinations of data to include location data, timing data, emitter type, or other signal identifiers could be used in Row Key creation, however, for this thesis, incrementing integers starting with the number 1 were used as Row Key identifiers primarily for flexibility, ease of sorting, cluster assignment identification, and querying. The HBase schema designed for this thesis can be seen in Figure 10.

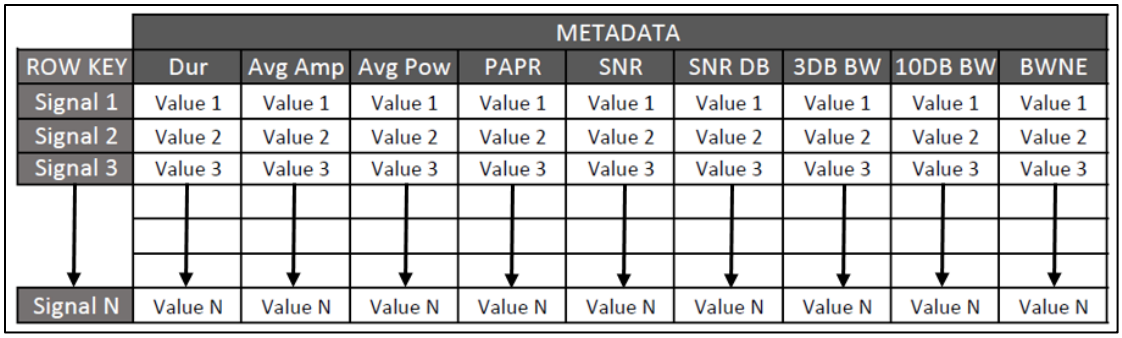

Figure 10. Design of Thesis HBase Schema. Adapted from [9].

2. HBase Implementation

To gain access to HBase on EMR, the HBase command line was launched from the Hadoop command line. Once launched, the HBase table was created, and the dataset was uploaded to the table [35]. This section provides the procedure and code implemented for launching HBase, table creation, and uploading data into HBase.

a. Launching HBase

To launch the HBase window, a Hadoop command line window was launched in accordance with the instructions provided in Appendix B. In the Hadoop command line, the command "hbase shell" was entered, launching the HBase shell window within the Hadoop command window [9].

b. Table Creation

As described in [9], there are many options that can be configured in HBase table creation. The one option that was utilized, aside from table name and column family name, was the number of versions for the table in creation. HBase has the capability to create multiple versions of data based on the timestamp of when the data was uploaded [9]. For this thesis the entire dataset was uploaded at once and therefore only necessitated a single version of data, therefore the number 1 was selected for the versions option. The code used to create the HBase table can be seen in Appendix C.

c. Dataset Loading

As described in [8], There are multiple methods that can be used to load data into HBase, however for this thesis the built-in bulk-load functionality of HBase was used to load in a CSV file of the MATLAB-generated dataset. The ImportTSV function was used, with a value separator specified as a comma, instead of a tab, allowing for CSV file upload. For this function, the first column of the CSV file for upload must contain the Row Key identifiers. Additionally, the column family input to the function must match that which was identified during table creation and each individual column name must be identified. Finally, the ImportTSV command takes the S3 bucket path location of the stored CSV file as the final input [8]. An ImportTSV implementation command example can be seen in Appendix C.

E. PHOENIX DESIGN AND IMPLEMENTATION

Phoenix was used in this thesis for the ease of transition between the HBase storage and the data analytics of Spark. Phoenix also provided the added query capability of an SQL layer on top of HBase [26]. This section describes the method used to map Phoenix to the data storage table in HBase.

(1) Launching Phoenix

To launch the Phoenix window, a Hadoop CLI window was launched in accordance with the instructions in Appendix B [36]. The command used to launch the Phoenix command line window can be seen in Appendix C.

(2) Mapping Phoenix to HBase Table

As described in [37], HBase tables can be created from a Phoenix command window or a Phoenix table can be mapped to an already existing HBase table. A table associated with a previously existing HBase table is referred to as a "View."

When a View is created in Phoenix, the data is accessible through Phoenix while the data storage is maintained within HBase. Therefore, no data transfer occurs between HBase and Phoenix. This is important to note as there can be monetary and performance

costs associated with transferring large amounts of data. This lack of data transfer allows for upward scalability to large datasets stored in HBase and accessed through Phoenix without performance or monetary costs associated.

Since the HBase table was previously created and the data was uploaded into HBase, a View was created in Phoenix, mapping the HBase data to a Phoenix table of the same name. From there, data can be queried from the table and placed into another table as necessary. To correctly create the View, the table, column family, and individual column names must be input precisely as created in the HBase table, to include case [37]. The code and commands used to create a View in Phoenix can be seen in Appendix C.

F. SPARK DESIGN AND IMPLEMENTATION

Spark was implemented in this thesis due to the extensive data analytics capability and machine learning library [15]. More specifically, Spark was used to query and load data from Phoenix, stored in HBase, and perform K-means machine learning with that data. This section describes the methods used to implement Spark data loading, data preparation, and machine learning.

1. Launching Spark

To launch the Spark command window, a Hadoop command window was launched in accordance with the instructions in Appendix B. There are different commands to launch Spark for use with Python, SQL, or Scala [15]. For this thesis, the Spark Scala API was used, as this was the original Spark language and therefore provided the most extensive resource availability. Properly configuring the number of executor cores, executors, and amount of executor memory can affect Spark performance and completion time [13]. After reviewing [38] and some trial and error, for the computer cluster sizes implemented in this thesis, three executor cores, an executor memory of 10 GB, and the number of executors equal to twice the number of launched worker nodes was chosen [38]. The command to launch Spark from the Hadoop CLI can be seen in Appendix C.

2. Loading Data into Spark from Phoenix

Loading data from Phoenix into Spark requires the use of the exact table, column family, and column names which were used in both Phoenix and HBase tables to include case sensitivity [39]. Due to the method by which Spark reads in the data from Phoenix, an additional step was found to be necessary prior to any data analysis. When the data was loaded into a Spark Dataframe, directly followed by the implementation of K-means on the loaded data, Spark would not achieve parallelization across larger computer clusters. As the computer cluster size increased, there was no notable change in execution time to perform K-means clustering. To counter this issue, once data was loaded into a Spark Dataframe from Phoenix, the data was written out as a CSV file stored in the S3 bucket. Subsequently, in a new Spark command line window, the CSV file was then loaded back into Spark from the same S3 bucket location [15]. This allowed for Spark to achieve parallelization across the computer clusters as can be seen in Chapter IV.A of this thesis. The code used to import the data from Phoenix into a Spark Dataframe can be seen in Appendix C.

3. K-means Machine Learning

To apply K-means machine learning algorithms to data within a Spark Dataframe, data preparation must take place beforehand [15]. This section describes the methods and code used for data preparation and the application of K-means to the dataset.

a. Data Preparation

The numerical values within each of the different parameters were of varying quantity, where one parameter may have a value less than one, and another a value of 500. Therefore, the dataset needed to be scaled to ensure that each of the parameters were weighted equally during K-means implementation. It was decided that scaling each parameter of the dataset to values ranging from -1 to 1 would be efficient. To accomplish data scaling, the MinMaxScaler function was used, which allows the user to identify the minimum and maximum ranges for the data to be scaled [15]. Furthermore, as described in [15], the data columns to which K-means was applied, referred to as features, were

combined into a single vector array in Spark. The code used to execute the above tasks can be seen in Appendix C.

b. K-means Implementation

Once the features are combined into a single vector and scaled appropriately, the K-means algorithm can be applied. First the K-means model is trained, taking the input of the number of clusters K and the maximum number of iterations [15]. Through trial and error, and to ensure that the K-means successfully clustered the dataset, the maximum iterations is set arbitrarily high to a value of 1000 to ensure that test datasets of various sizes would have an iteration allowance large enough to complete clustering. Once trained, the model is then applied to the vector of features, and the cluster assignments are generated [15]. The code used to execute the K-means implementation can be seen in Appendix C.

c. Ideal K Value

As described in Chapter II.C.2.c of this thesis, an effective method to identity a reasonable value for K is by use of the "elbow method" to determine an approximate value of K in which the change in total error, referred to as cost, begins converging to zero [14]. To identify a reasonable K value for this thesis, code was developed that would run multiple rounds of K-means against the dataset, incrementing the K value and computing the cost at each incremental K value. Once the most recent calculated cost value is greater than 95% of previously calculated cost, the code finishes execution, and the K value is output. This method was used against a test data size of 10^6 signals, and the program identified 14 as a reasonable K value. A plot of K value versus cost can be seen in Figure 11 with the value 14 identified. Furthermore, for variable consistency throughout K-means testing of varying data sizes, a K value of 14 was used throughout the experimentation. The code for identifying a reasonable K value was written in Scala, implemented within Spark, and can be seen in Appendix C.

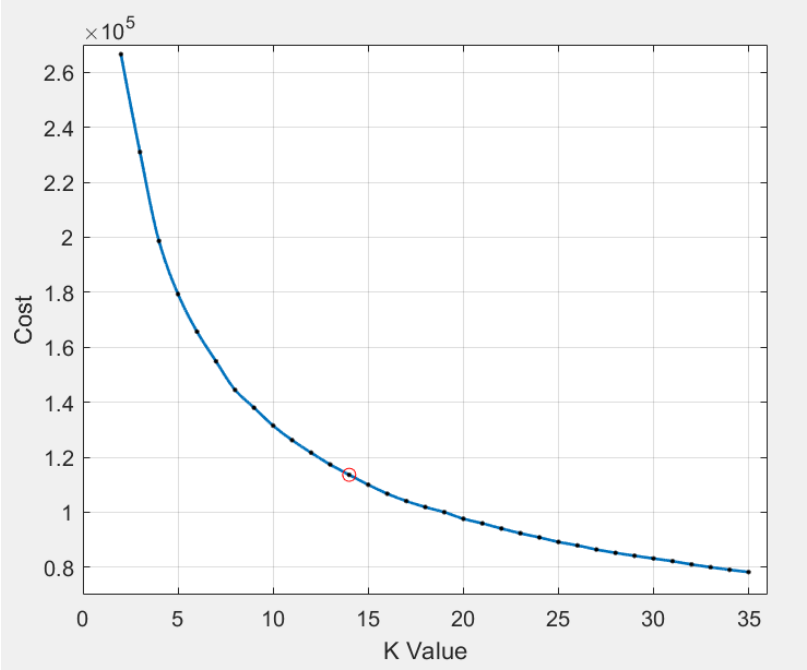

Figure 11. Cost Analysis Plot of K Value vs. Cost for 10^6 Signals.

G. TEST IMPLEMENTATION

The code and procedure for this thesis was researched and generated sequentially from HBase, to Phoenix, to Spark. Once the process was designed and prior to implementation on the generated large dataset, there was a test dataset implemented for which the results were already known, since the larger dataset would prove too large to appropriately verify. By implementing this test dataset, the process was verified to ensure that no part of data transfer or preparation process would affect the outcome and that the Spark machine learning was being implemented correctly. This section describes the dataset used for process verification.

(1) Iris Data

A very commonly known set of multivariate data which can be found within the pattern recognition literature is Fisher's Iris data set, which was introduced by British statistician Ronald Fisher in 1936 [40]. As seen in [41], this dataset is comprised of 3 classes of 50 instances, in which each class is a different type of Iris flower. The dataset

provides features consisting of Sepal width, Sepal length, Petal width, and Petal length for each of the three types of Iris flowers [41].

(2) Test Phase

To verify the process ultimately implemented in this research, the dataset was first uploaded into HBase as described in section D of this chapter. A Phoenix table was then mapped to the existing HBase table and Spark queried the data into a Dataframe. Once in Spark, the data was pre-processed as listed in section F.3.a of this chapter and K-means was implemented with a K value of 3.

Figure 12 shows the two-dimensional plot of the original data for Sepal length and width, color coded for the three types of Iris plants as described in [41]. Figure 13 shows the resulting two-dimensional plot of the clustered data for the same two features, color-coded the same for the three clusters created.

The Iris setosa plant, which was linearly separated from the other two as can be seen in Figure 13, clustered into a separate group with 100% accuracy. The second cluster was grouped with 94% accuracy, while the third cluster was grouped with only 72% accuracy. While this can be seen by the plot, it was also verified by comparison of cluster values after K-means implementation. K-means was implemented on the same exact dataset within MATLAB, with identical clustering assignments. Therefore, whatever the reason for the lower accuracy in the third cluster assignments, it is likely related to the correlations within the dataset itself and not a result of Spark K-means implementation. The identical K-means results between Spark and MATLAB provided a demonstration that untrained, the process implemented will upload, store, transfer, scale, and cluster the data as intended.

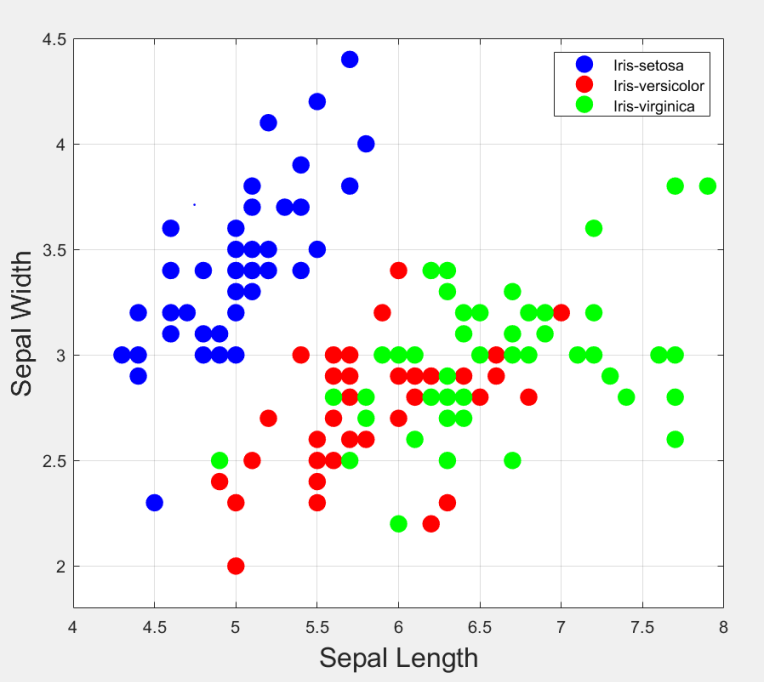

Figure 12. Plot of Iris Data Sepal Length vs. Sepal Width. Adapted from [41].

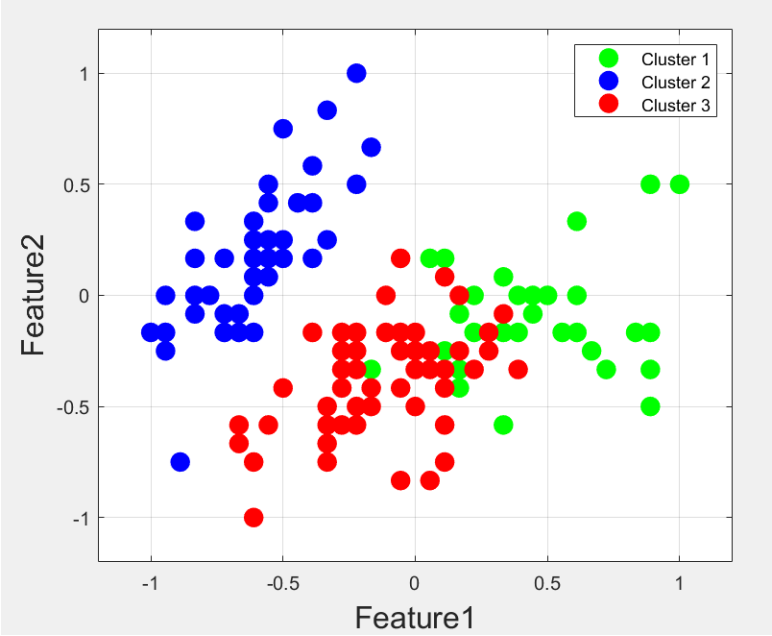

Figure 13. Plot of Iris Data Spark K-means Clustering Results.

IV. RESULTS

This chapter provides the results of the experiment conducted as described in Chapter III of this thesis.

A. K-MEANS PERFORMANCE RESULTS

Throughout this thesis experiment, clusters were launched with 2, 5, 10, 15, 20, 25, and 50 general-purpose EC2 worker nodes [32]. Across the different computer cluster variations, K-means clustering was applied to signal datasets of 1 x 10^5, 1 x10^6, 0.5 x 10^7, 1 x 10^7, 2 x 10^7, 3 x 10^7, 4 x 10^7, 5 x 10^7, and 1 x 10^8 signals. The list of signal data sizes and the equivalent data file size for each set of signals can be seen in Table 1. A K value of 14, which was found to be a reasonable value for the 10^6 data size [14], was set for all K-means implementations to ensure consistency of performance testing and results.

Table 1. Number of Signals Tested and Associated Data Size.

Number of Signals	Data Size
1x10^5	6.8MB
1x10^6	68.5MB
0.5x10^7	776.7MB
1x10^7	1.5GB
2x10^7	3GB
3x10^7	4.6GB
4x10^7	6.1GB
5x10^7	7.6GB
1x10^8	15.3GB

It was found that for the smallest two datasets, 1 x10^5 and 1 x 10^6 signals, the execution time did not vary as computer cluster size was increased. However, once the dataset was increased to 0.5 x 10^7 signals and beyond, there was a noticeable decrease in

execution time from a two-node computer cluster to a five-node computer cluster, then less significant decrease for larger computer clusters. Additionally, as the data size increased within the same size computer cluster, the execution time increased for K-means. The recorded K-means execution times across multiple computer cluster sizes can be seen in Table 2.

Table 2. Execution Time(s) across Various Signal Dataset Sizes and Computer Cluster Sizes.

Number of Signals \ Computer Cluster Size	2	5	10	15	20	25	50
1x10^5	21.8	21.6	21.6	21.2	----------	----------	----------
1x10^6	307.0	291.5	290.4	294.9	----------	----------	----------
0.5x10^7	452.6	80.7	78.5	60.7	40.8	36.9	30.5
1x10^7	1209.7	328.9	124.9	81.0	95.4	89.6	43.5
2x10^7	2734.4	757.9	259.3	96.9	89.1	90.1	68.8
3x10^7	3624.1	1011.1	499.5	405.8	209.8	111.6	74.3
4x10^7	4200.4	1125.9	553.8	480.2	261.4	324.5	90.1
5x10^7	4480.6	1195.4	643.4	438.5	417.6	507.8	98.5
1x10^8	>36000	7691.5	3398.7	2106.5	2038.7	1442.9	442.6

Figure 14 displays the increase in execution time as the number of signals increases. Of note, at datasets of 5 x 10^7 and below, with 15 nodes enabled, the execution time drops below 10 minutes. Additionally, as the number of signals increases, the execution time increases on what resembles an exponential rate for the datasets implemented on a computer cluster size of five nodes.

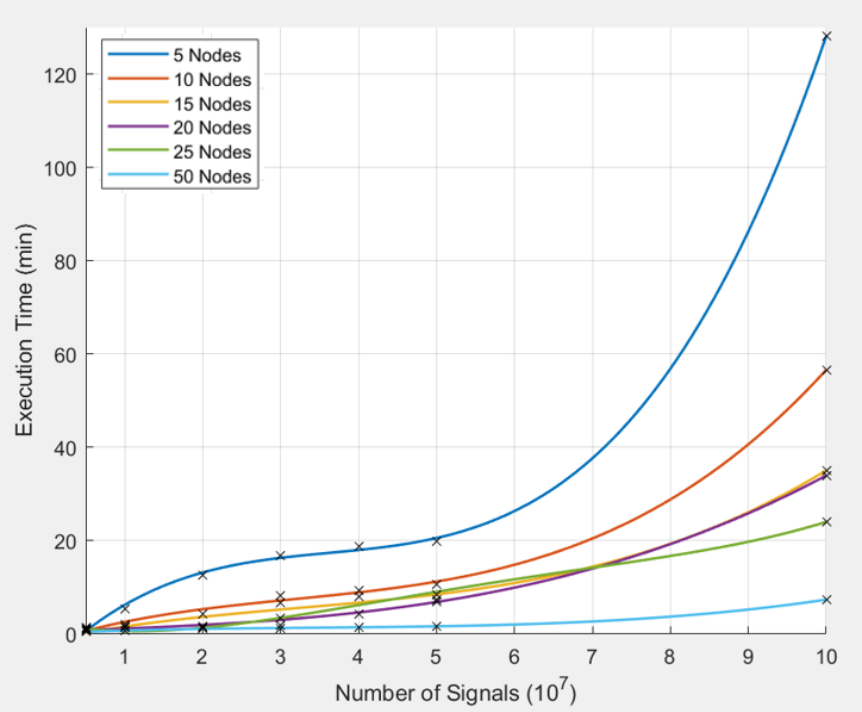

Figure 14. Plot of Number of Signals vs. Execution Time (min).

It is understood that the change in execution time in relation to the variables assigned within the K-means algorithm, referred to as the time complexity, is defined as $O(n,d,K,i)$. The time complexity O is a function of the number of data points n, clusters K, feature dimensions d, and iterations i [42]. In this thesis research, the number of clusters and the number of feature dimensions for all K-means runs were each a constant value. Therefore, the number of iterations and the number of data points were the only variables affecting time complexity as defined. Of note, this definition does not consider any unknown effects specifically caused by implementing K-means across a cluster of computers. While it is expected that an increase in data size causes an increase in iterations, both of which increasing the overall execution time, this does not directly support an exponential increase in execution time [43]. With only one of the tested computer clusters implemented with a large enough dataset to demonstrate what resembles an exponential increase, it is undetermined whether larger datasets run on larger computer cluster sizes would have the same or similar trends.

If this trend were to prove consistent across larger computer cluster sizes and K-means implementation eventually results in an exceptionally large execution time on any dataset size, an identifiable pattern or threshold level may be discovered. The identification of a pattern or threshold level would provide an important design consideration factor for future experiments and real-world implementation. Additionally, while the number of iterations varies on each K-means run, as a result of the first random cluster assignment [15], a method by which to measure and record the change in iterations as data sizes increase would assist in determining the cause of this trend.

Figure 15 shows the decrease in K-means execution time as the computer cluster size increases. Figure 16 shows the same plot zoomed in for an execution time ranging from zero to 40 minutes. Of note, for all data sizes, the execution time of K-means is decreased by at least 90% within the first 15 nodes, however the execution time does not fall below 10 minutes until approximately 40 nodes are implemented.

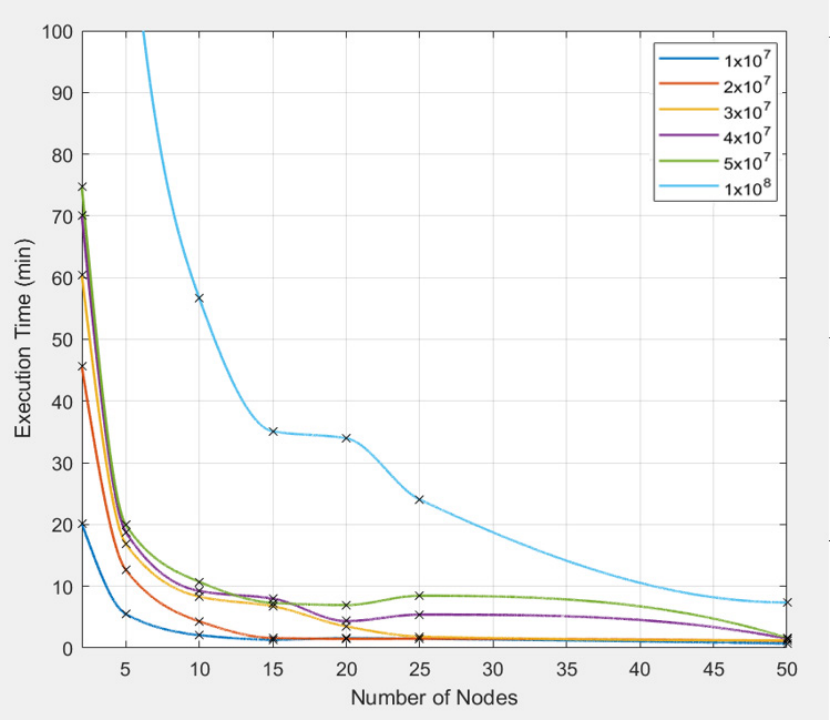

Figure 15. Plot of Computer Cluster Size vs. Execution Time (min).

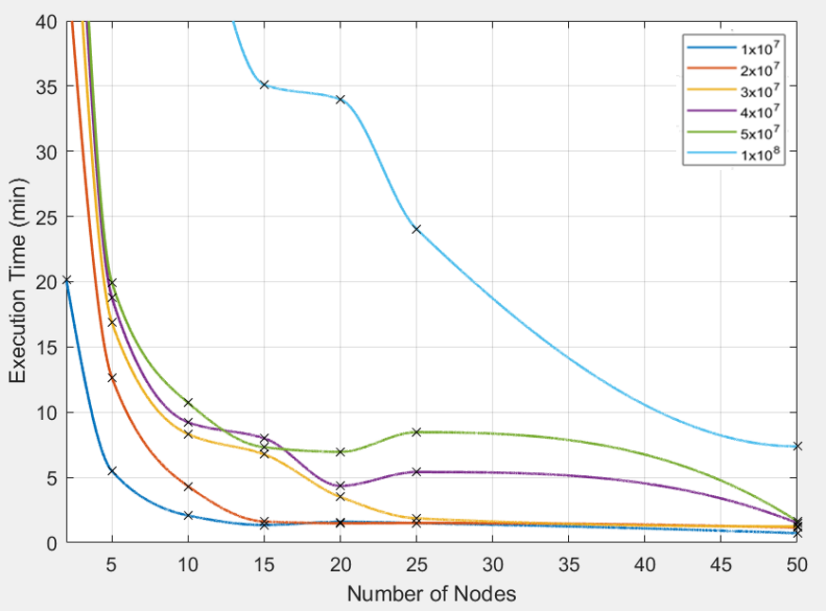

Figure 16. Plot of Computer Cluster Size vs. Execution Time (min), Execution Time Less Than 40 Minutes.

Intuitively, one would expect a more linearly decreasing trend in execution time as the computer cluster size increases, however this was not the case with these results. This may be a result of internal cluster operations, the configuration of the cluster itself, or how the dataset is distributed across the cluster. However, it is unknown as to precisely what causes the execution time to taper off after the initial sharp decrease within the first 15 nodes implemented. Further investigation into the cluster configuration should be conducted in future work in an attempt to determine the cause of this result and how to counter it, if possible, for better performance as computer cluster sizes increase.

B. CLUSTER VERIFICATION

To verify K-means implementation on the large signal datasets provided consistent clustering across sets of data tested, a method of comparison was devised. It was expected that the resulting 14 cluster centroids from each K-means run would be relatively close to the corresponding 14 centroids for each of the datasets tested. Therefore, the distance between the 14 cluster centroids was compared for multiple K-means runs of varying dataset sizes to ensure that they were relatively similar.

Each cluster centroid is a nine-dimensional coordinate vector, representative of the nine features of signal data [15]. The distance was calculated between corresponding centroids for two K-means runs, then the average distance between the corresponding centroid pairs was calculated. This process was repeated for 10 K-means runs using varying sizes of datasets. The results can be seen in Table 3.

Table 3. Centroid Distance Comparison across Multiple Iterations

Trial	Average Distance
1	0.227
2	0.166
3	0.233
4	0.197
5	0.187
6	0.242
7	0.232
8	0.208
9	0.184
10	0.195
Overall Average	0.207

Across the 10 trials, the distance between coordinating cluster centroids ranged from 0.166 to 0.242, with an overall average of 0.207. Since the range of the dataset was scaled from -1 to 1 during the K-means pre-processing [15], the maximum distance that any two points can be from one another is six. Therefore, the results demonstrate that the centroids remain relatively the same across multiple iterations and varying data sizes of the same data type. Of note, the cluster centroids were also compared on a separate 10 iterations of the same exact dataset to assess for any variance from one K-means iteration to the next. It was observed that across all iterations, the cluster centroids were consistently identical to each other after K-means clustering.

C. INFERENCES

Based on the data obtained from this experiment, for computer clusters to be useful for K-means, the data size should be larger than 10^6 signals or 65.5 MB. While datasets less than this can be implemented and clustered effectively, there is no observed performance benefit as the computer cluster size increases. However, if the signal database were to be extended to contain more columns of metadata and therefore more features to cluster by, this result could change and a smaller set of signals may benefit from K-means implementation across a computer cluster. While datasets of 10^6 signals, or 65.5 MB, and smaller can be effectively clustered, there is no observed benefit in using a computer cluster.

All of the dataset sizes from 1 x 10^7 signals and above had reduced execution time by approximately 90% as computer cluster size increased to 15 and then tapered off as the computer cluster size continued to increase demonstrating that there was minimal gain from using a computer cluster size larger than 15 nodes, dependent on the performance need.

Finally, the experiment results are representative of the specific node chosen for this experiment. The m4.2xlarge EC2 general purpose instance node chosen for this test contains 8 vCPUs and 32 GB of memory [32]. With change to chosen node type, the results should vary, and computer cluster size should be chosen based on vCPU and memory capacity.

D. AWS CLUSTER PERFORMANCE LESSONS LEARNED

Running computer clusters within AWS, while the prescribed method for implementing large-scale machine learning on signal metadata, at times yielded inconsistent results. There were times throughout the experimentation process when computer clusters were seemingly performing inconsistently, specifically in execution time, with prior K-means runs of clusters of the same size. On various occasions, a cluster would to be manually terminated, then relaunched to ensure result consistency. It was observed that while performance time can be measured with moderate consistency, there

is still a margin of error where computer cluster performance is concerned. Therefore, the results were moderately specific to the performance of the individual cluster launched.

Another notable observation about cluster performance was the varying results in K-means execution time. While all the obtained results fell within the overall trend as seen in Figures 13, 14, and 15, to ensure consistent execution times, each dataset test on each node variation was run a minimum of three times due to an occasional performance outlier. Therefore, while overall results were consistent, the experiment required multiple iterations at every level to accurately observe cluster performance.

V. CONCLUSION

This chapter provides a summary of this thesis and recommends work to be considered in the future.

A. SUMMARY

The purpose of this research was to explore the use of cluster computing to implement large-scale storage and machine learning onto large sets of signal data. Experimentation was completed with the use of open-source data storage and analytics tools, all installed across a single computer cluster in AWS EMR [17]. By running various sizes of datasets through the same K-means clustering implementation, execution times were recorded and the results demonstrated a distinct benefit for more computer nodes as the data size increases. Additionally, the results showed a notable decrease in benefit for computer clusters with more than 15 nodes. As identical datasets were run on varying computer cluster sizes, the results demonstrated the efficiency of increased computer cluster sizes for executing K-means on larger sets of signal data.

HBase was selected as the non-relational, no-SQL database storage solution used to store the signal metadata. HBase was chosen primarily due to the up-scalability of HBase and the capability to store various unstructured datatypes [19]. Phoenix was selected as an SQL layer on HBase as it provided a streamlined method for querying the HBase data from Spark, while also providing a simpler query capability than that provided in HBase [37]. Spark was selected as the data analytics tool because of its high-performance machine learning capability and design for analyzing large sets of data across multiple computer nodes [15]. Finally, AWS was used as a means to implement the aforementioned open-source tools all on one EMR cluster, while capable of adjusting cluster size [17].

Ultimately, this thesis demonstrated a proof of concept that cluster computing and distributed data storage and analytics tools could prove effective for storage and machine learning application onto signal datasets too large to be implemented on a single computer system.

B. FUTURE WORK

This section describes suggestions for follow-on work to this thesis.

1. Multi-Emitter Dataset with More Features

The dataset used for testing within this thesis was not representative of a real world signal database as described in Chapter 3. While the dataset generated for this thesis was sufficient for testing performance consistency throughout the experiment, a more diverse signal dataset would prove useful in a more realistic test of not only the performance capability, but the follow-on analysis of how the signals are separated and clustered. Additionally, a follow-on dataset could contain more metadata features. By increasing the number of features, there are more identifiers by which to compare the signals and could thereby affect the outcome of the K-means clustering.

2. Connection from Spark to HBase

The process implemented in this thesis to appropriately load the signal data into a Spark Dataframe required that once loaded from Phoenix, the data be written out to a CSV file in the S3 bucket, then re-loaded back into Spark. Otherwise parallelization across the various nodes was not achieved and there was no change in performance as the number of nodes was increased. Therefore, an alternative method should be explored for loading data into Spark from either HBase or Phoenix in such a manner that the data is loaded properly for parallelization. Another open-source tool available which may prove useful is Apache Hive, which may be able to provide query capability from HBase tables and a connection to Spark [44].

3. Other Database Options

While the experimentation was accomplished with the use of HBase as the data storage, HBase proved to be not very user-friendly. In addition to the data loading problem listed in the previous section, the ability to make data queries directly from HBase is very limited. Other database options should be explored for ease of use and data querying capability, while also providing a more stream-lined connection to Spark. For one, Apache

Cassandra is another scalable open-source no-SQL database that can be used in connection to Spark [45].

4. Analysis of Larger K Values

For the experimentation in this thesis, a K value of 14 was used for all K-means implementations to provide a clear picture of how the performance and execution time was affected by varying data size and number of nodes. With better knowledge of how many nodes must be launched to minimize K-means execution time, running a process to identify a reasonable K value for each data size can be better explored [14]. Furthermore, increasing the K value will likely affect execution time which can be measured for better overall analysis of K-means implementation on large signal datasets.

5. MATLAB Comparison

In this research, multiple K-means clustering implementations were run solely in Spark and compared against each other. For further analysis and comparison, the same set of data can be run through K-means clustering, or another machine learning algorithm, in both MATLAB and then Spark. The results of these runs could then be compared to verify if both MATLAB and Spark yield the same results.

6. Larger Datasets

The largest dataset implemented within this research was $1x10^8$ signals. A continuation of this work could include uploading a significantly larger dataset to test the utility of cluster computing on much larger datasets than the ones used in this thesis. Additionally, the results of K-means implementation with larger datasets would provide an ability to better analyze the performance trends as the data size is increased for each of the computer cluster configurations.

7. Other Machine Learning Options

While K-means is a very popular and widely used unsupervised machine learning algorithm [15], there are other machine learning algorithms that should be explored. There are many machine learning algorithms available in the Spark MLlib. For one, Bisecting K-

means is another variant of K-means clustering which, while similar to the K-means algorithm implemented in this thesis, performs clustering differently and is similar to hierarchical clustering [15]. Other machine learning algorithms either alone or in combination with K-means may prove useful in further analysis of large datasets of signal data.

APPENDIX A. LAUNCHING AN AWS CLUSTER

To launch a Cluster on AWS, there are various procedural steps that must be followed. These steps include the downloading and installation of the AWS Command Line Interface (CLI), creating an S3 bucket, creating an Amazon EC2 Key Pair [46]. Many of the general instructions for setting up an AWS cluster can also be found in [17]. The following outline the specific steps taken for launching the clusters used in this thesis.

(1) Install the AWS CLI

The AWS CLI allows the user to integrate and manage the AWS clusters and tools from within the user command line [17]. The following steps provide directions for the download and installation of the AWS CLI for Windows.

- For Windows OS, Download the 64-bit or 32-bit installer at https://s3.amazonaws.com/aws-cli/AWSCLI64PY3.msi or https://s3.amazonaws.com/aws-cli/AWSCLI32PY3.msi [17].

(2) S3 Bucket Creation

- Sign-in to Amazon Web Services https://aws.amazon.com/
- Under the **Services** tab, under Storage, Select **S3** (See Figure 17)

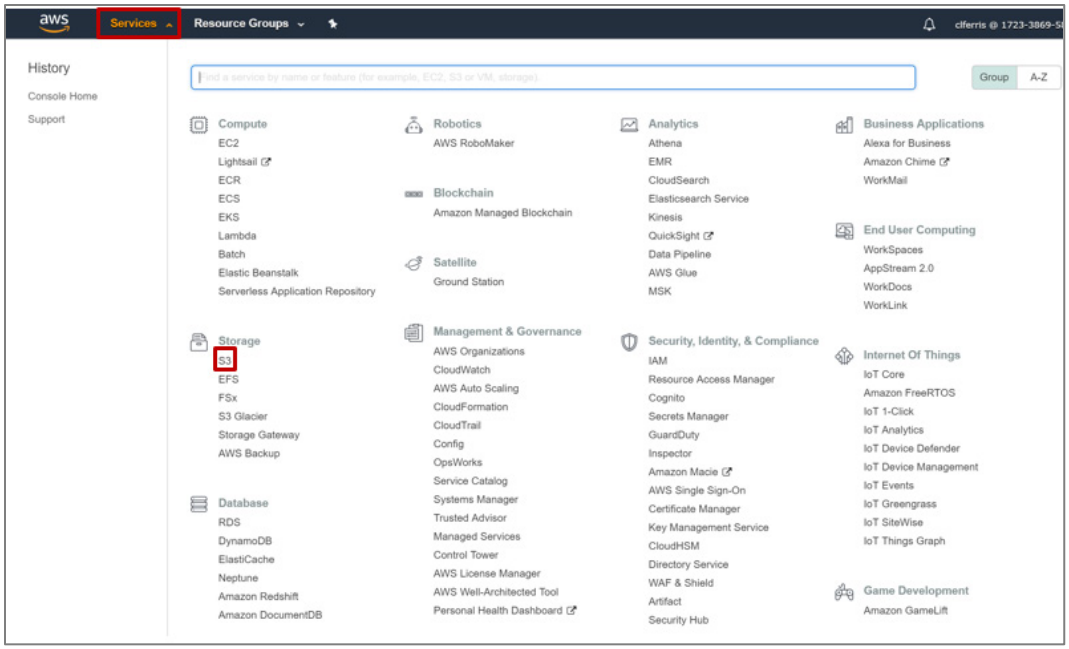

Figure 17. Screen Image of AWS Management Console with Services/ S3 Selected.

- Select **Create bucket:** Enter Name and Region, Select **Create** (See Figure 18)

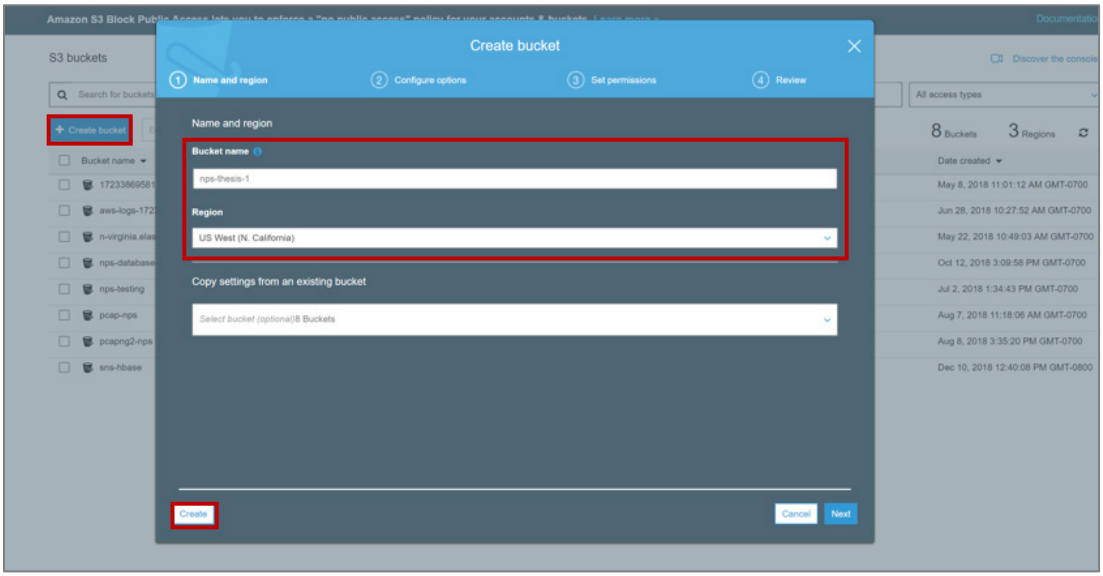

Figure 18. Screen Image of AWS Create Bucket.

(3) EC2 Key Pair Creation:

- Launch Amazon EC2 console at https://console.aws.amazon.com/ec2/
- In the Navigation Pane on the left side of the screen, under **NETWORK & SECURITY**, select **Key Pairs** [17]
- Select **Create Key Pair,** enter desired Key pair name, Select **Create** (See Figure 19)

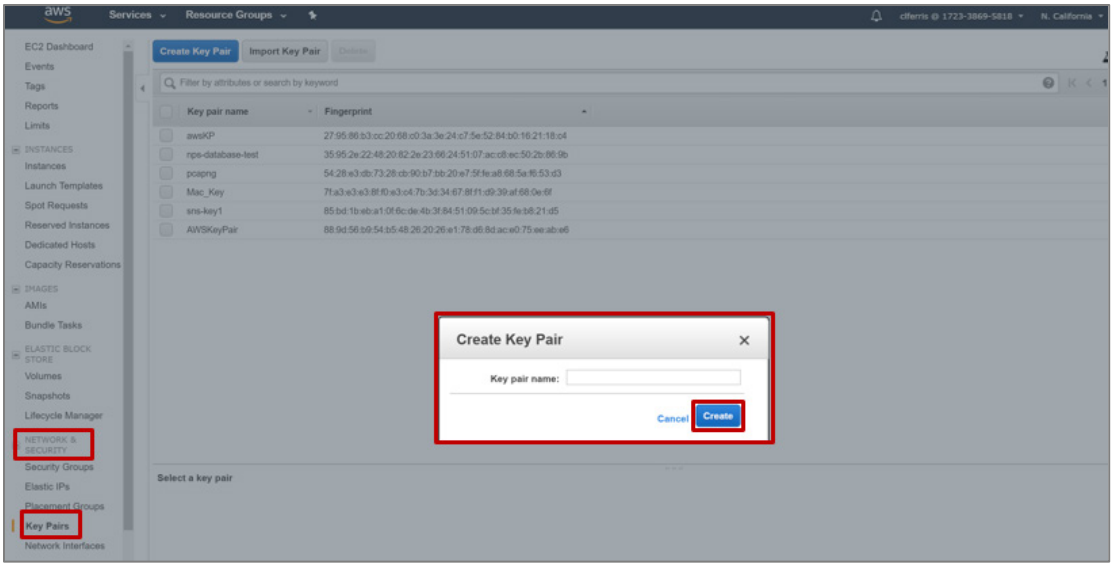

Figure 19. Screen Image of AWS Create Key Pair.

- Save the private key where it can be safeguarded and accessed in the future

(4) Launch an EMR Cluster with HBase, Phoenix and Spark

- Sign-in to Amazon Web Services https://aws.amazon.com/
- Under the **Services** tab, under Analytics, Select **EMR** (See Figure 20)

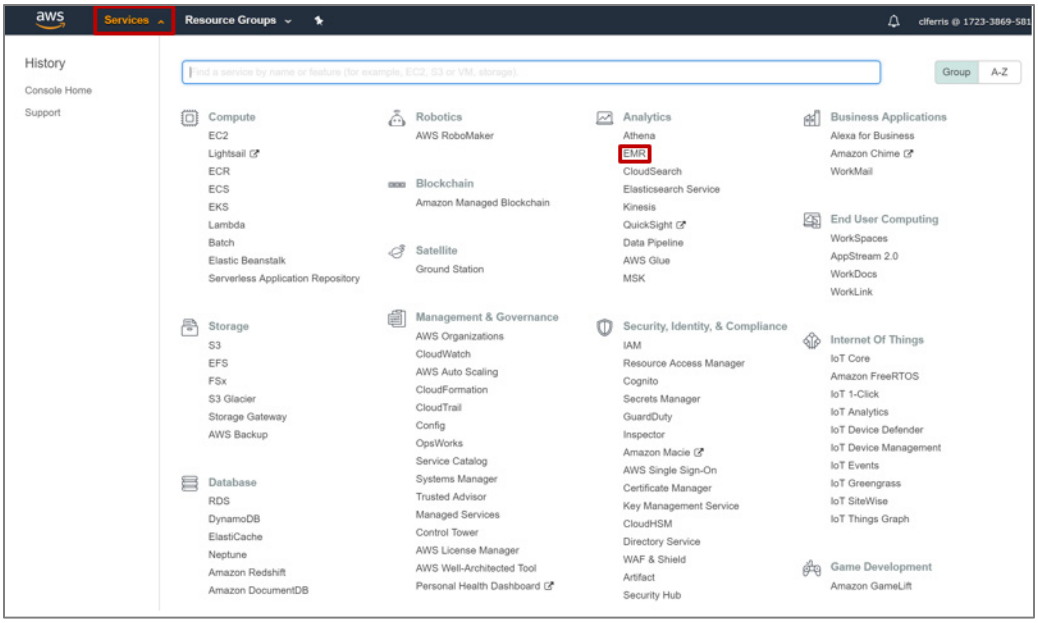

Figure 20. Screen Image of AWS Management Console with Services/ EMR Selected.

- Select **Create Cluster** (See Figure 21)

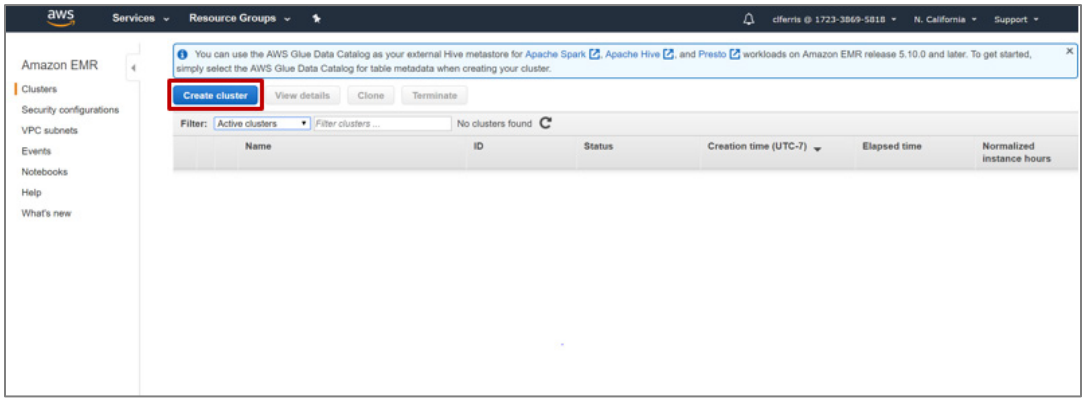

Figure 21. Screen Image of AWS Create Cluster.

- On the Quick Options page, select Go to advanced options (See Figure 22)

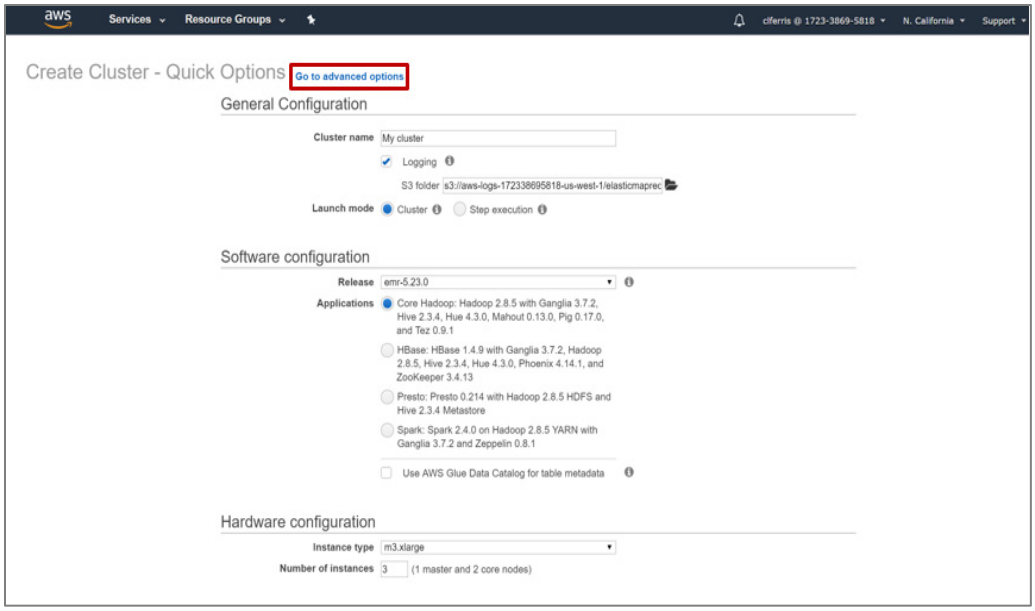

Figure 22. Screen Image of AWS Create Cluster – Quick Options.

- Under Software Configuration, ensure Hadoop, HBase, Phoenix and Spark are selected
- Under HBase storage settings, Select S3 and enter file path to your previously created S3 bucket, then click next (See Figure 23)

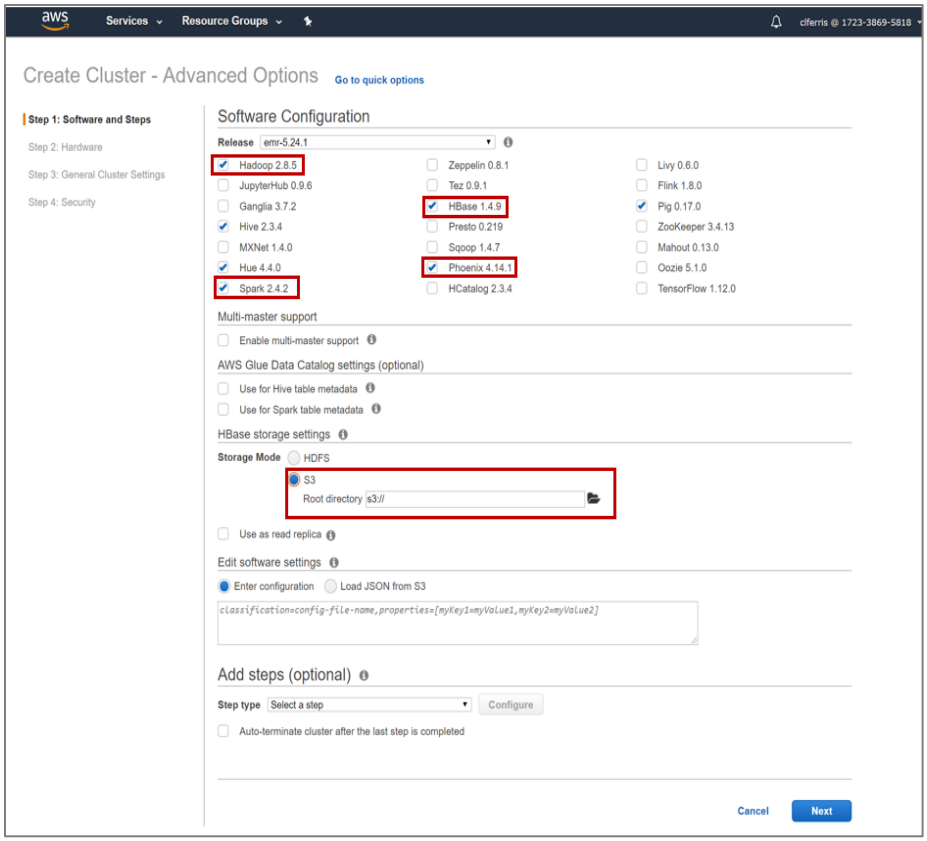

Figure 23. Screen Image of AWS Create Cluster – Advanced Options – Software.

- Under **Instance Type**, next to the Master Node, click the pencil icon to the right of **m4.2xlarge** (See Figure 24)

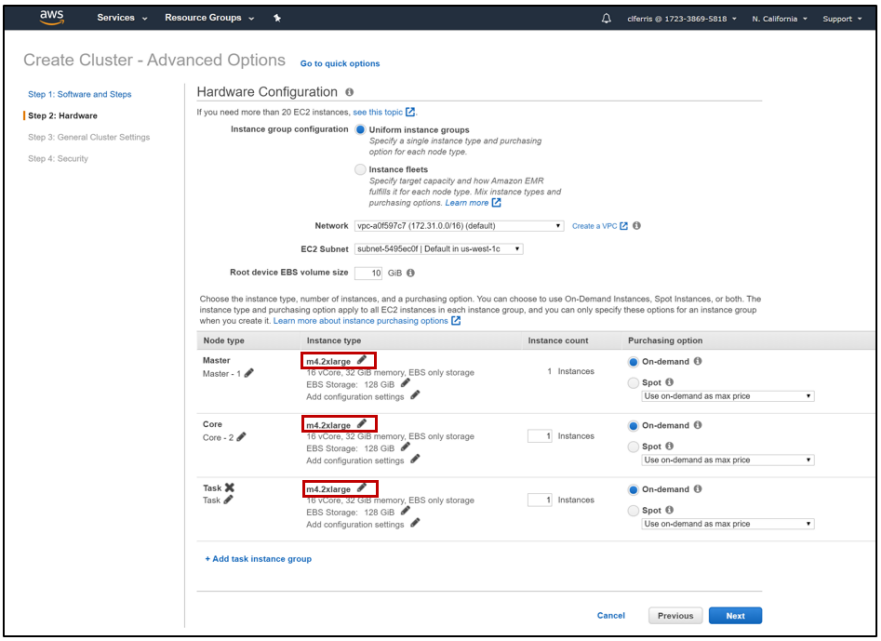

Figure 24. Screen Image of AWS Create Cluster – Advanced Options – Hardware.

- Select **m4.2xlarge,** click **Save** (See Figure 25)

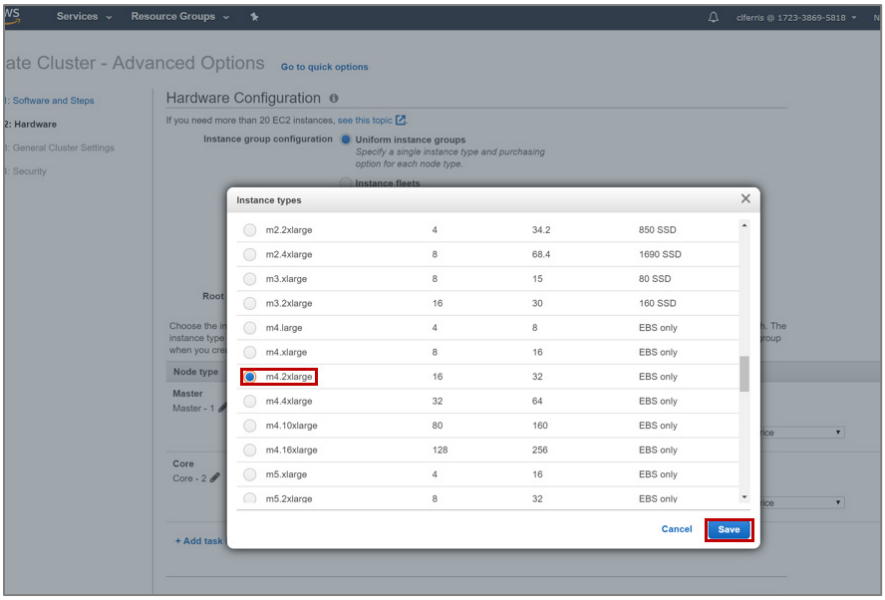

Figure 25. Screen Image of AWS Create Cluster – Advanced Options – Hardware 2.

- Follow the same steps for the Core Node and Task Node under **Instance Type.**
- Click **Next**
- Under **General Cluster Settings**, Enter a Cluster name, Click **Next** (See Figure 26)

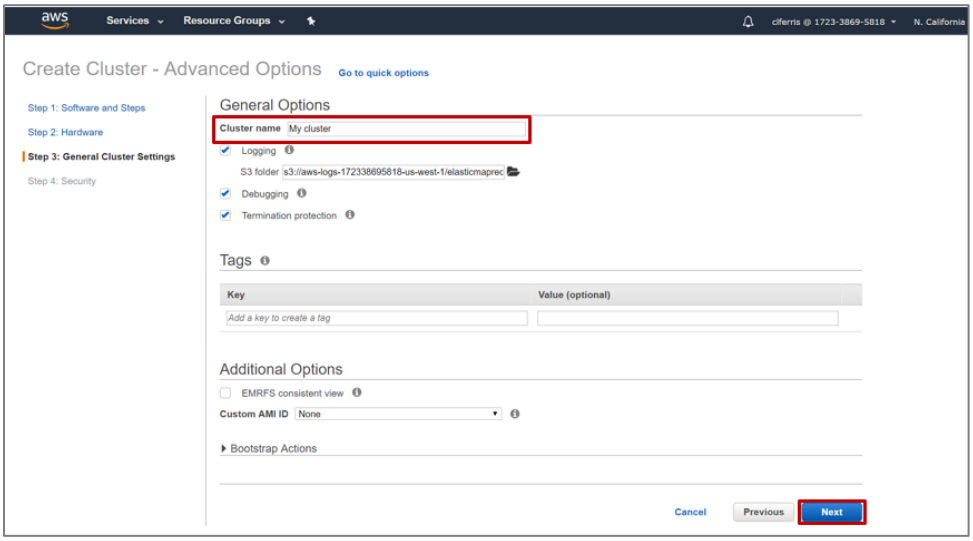

Figure 26. Screen Image of AWS Create Cluster – Advanced Options – General.

- Under Security, select previously created EC2 key pair, click **Create cluster** (See Figure 27)

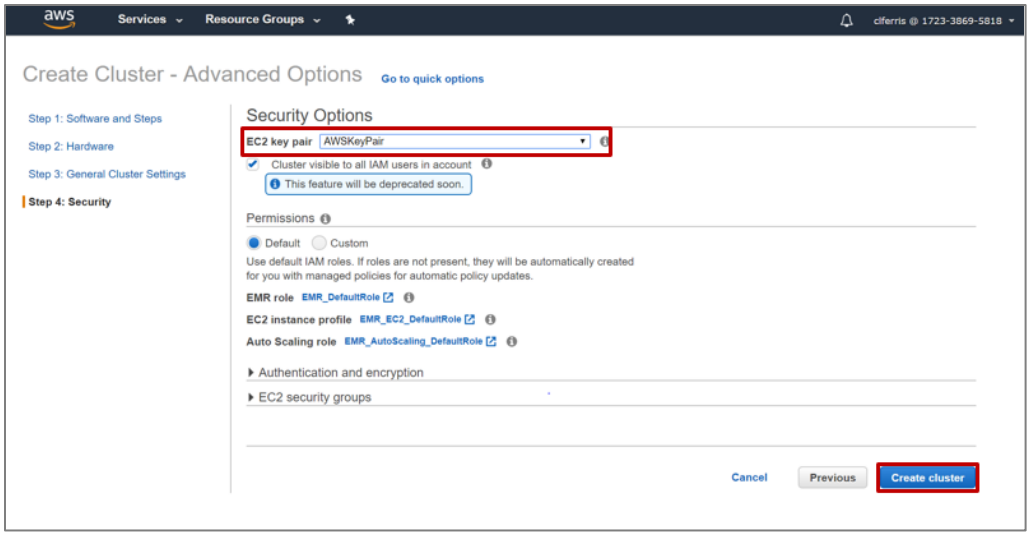

Figure 27. Screen Image of AWS Create Cluster – Advanced Options – Security.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. LAUNCHING THE HADOOP CLI

Prior to launching a Hadoop CLI on AWS, the steps for setting up and launching a cluster must have already been completed as outlined in Appendix A of this thesis and a cluster must be launched and in the "Waiting" status [17]. Once this is complete, there are procedural steps that must be followed including the set-up of an SSH connection, the download and installation of PuTTY and PuTTYgen [34]. Many of the general instructions for launching the Hadoop CLI can be also found in [17]. The following outline the specific steps taken for launching the clusters used in this thesis.

The following may be used as a procedural outline for launching the Hadoop CLI.

(1) Set up an SSH connection

- Launch Amazon EMR at https://console.aws.amazon.com/elasticmapreduce/
- Select **Clusters** from Navigation Pane
- Select the running cluster (See Figure 28)

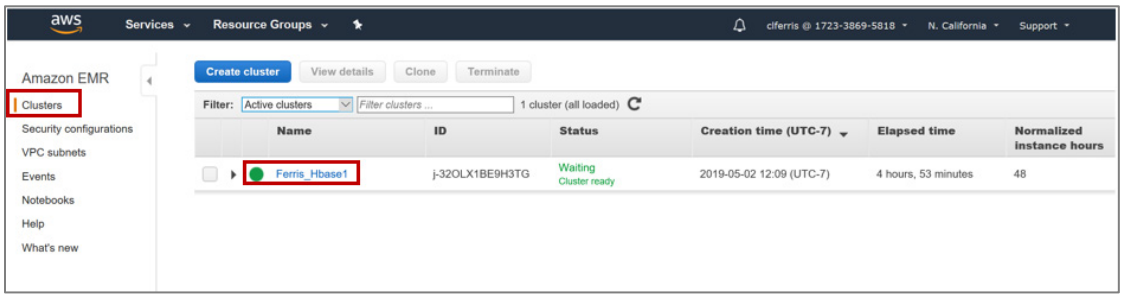

Figure 28. Screen Image of AWS EMR Console with Running Cluster Selected.

- Under **Security and access**, select **Security groups for Master** link (See Figure 29)

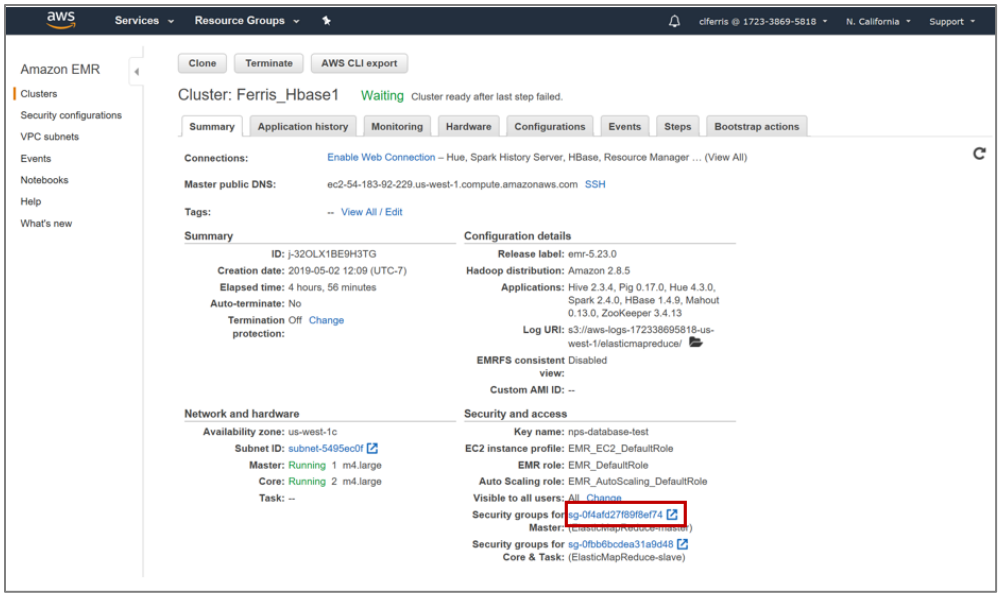

Figure 29. Screen Image of Security Groups Link Selected.

- Select ElasticMapReduce-master
- Under the Actions drop-down tab, select Edit inbound rules (See Figure 30)

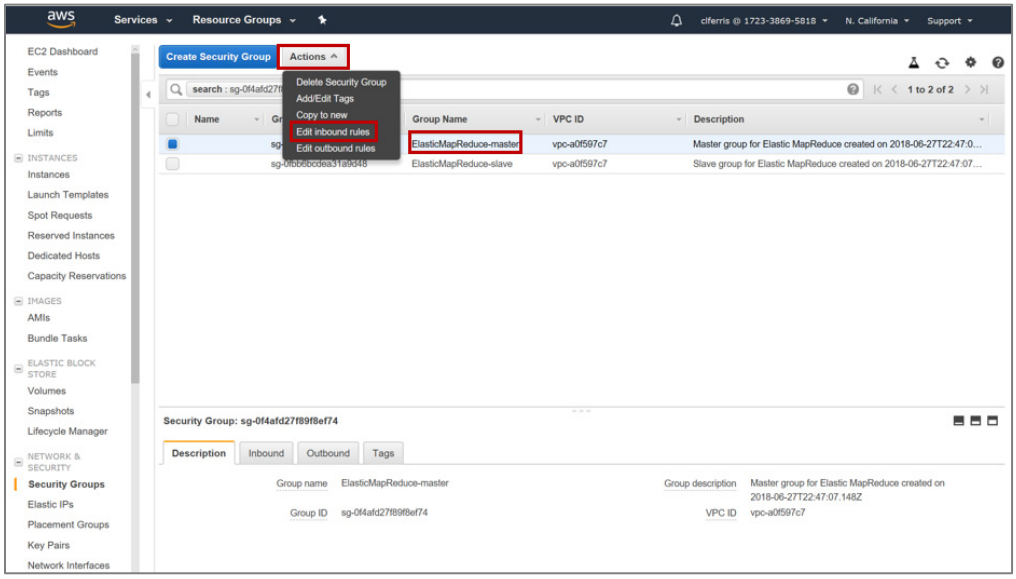

Figure 30. Screen Image of Actions and Edit Inbound Rules Selected.

- Select Add Rule
- Select SSH
- Under Source, select My IP
- Select Save (See Figure 31)

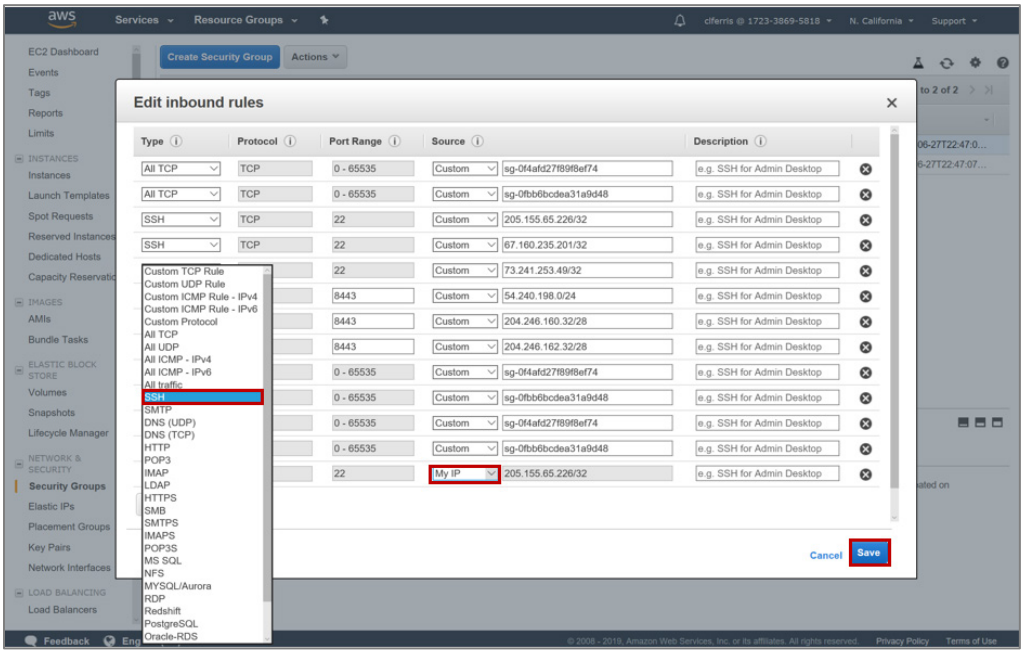

Figure 31. Screen Image of Edit Inbound Rules Options with SSH and My IP Selected.

- Select ElasticMapReduce-slave and repeat steps f through j

(2) Download PuTTYgen to Create .ppk Key

- Download PuTTYgen.exe at http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
- Launch PuTTYgen.exe

- Select **Load**
- Navigate to and select the Amazon EC2 Key Pair (.pem),
- Under **Type of key to generate**, select **SSH-1**
- Select **Save private key**
- Save the new key under the same name as the Amazon EC2 Key Pair (See Figure 32)

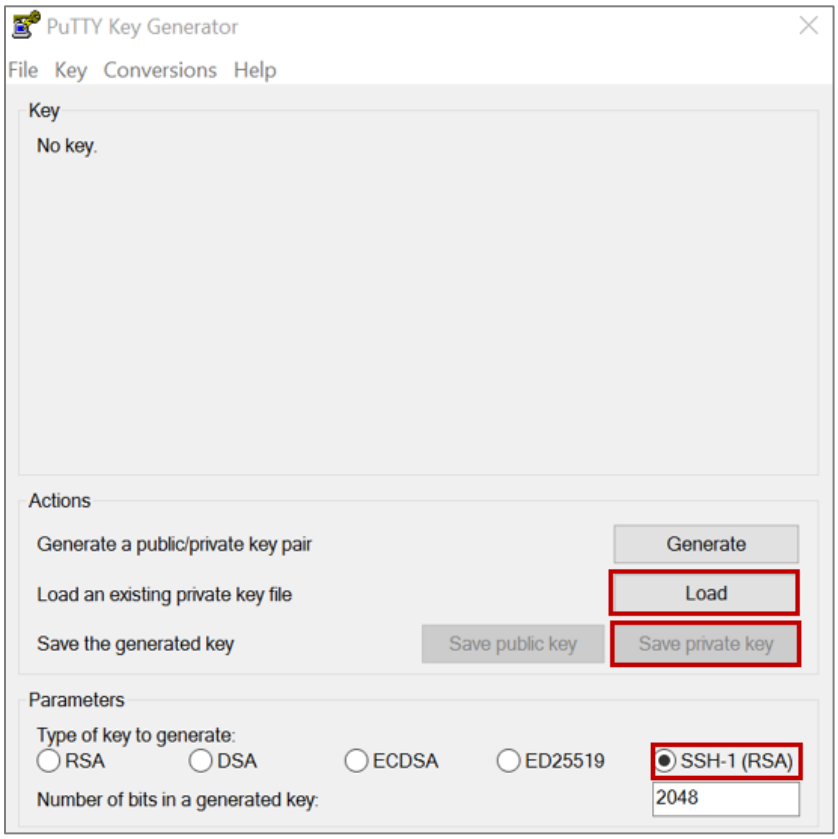

Figure 32. Screen Image of Edit Inbound Rules Options with SSH and My IP Selected.

(3) Launch the Hadoop CLI:

- Launch Amazon EMR at https://console.aws.amazon.com/elasticmapreduce/
- Select **Clusters** from Navigation Pane
- Select the **Name** of the running cluster (as performed in Step 1 above)
- Under **Master public DNS**, select the **SSH** link and follow the listed instructions with the running cluster Host Name and created user private key (See Figure 33)

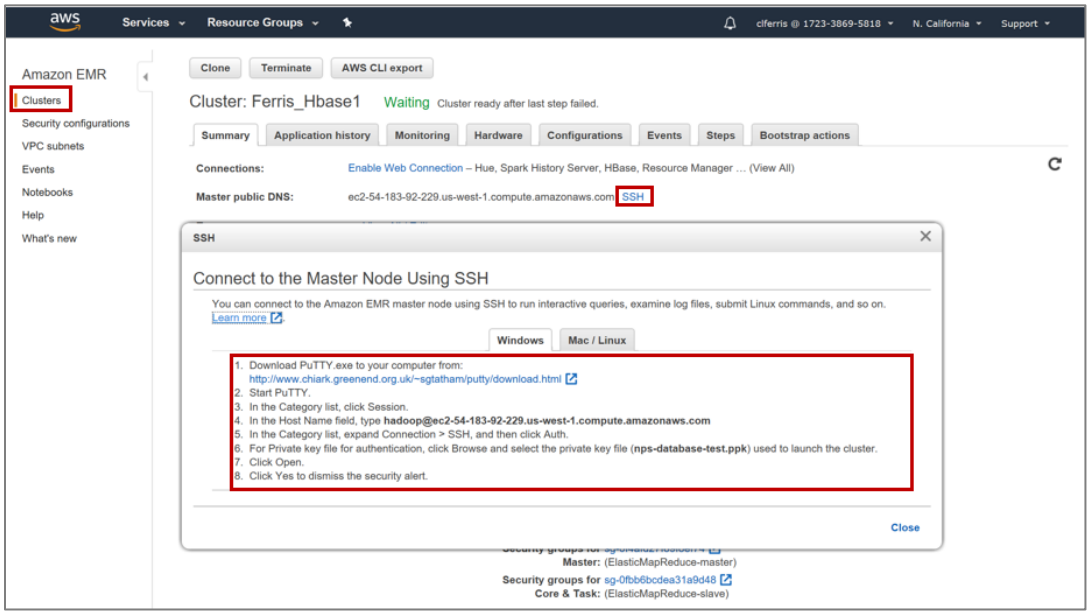

Figure 33. Screen Image of AWS SSH – Connect to Master Node.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. AMAZON EMR COMMANDS AND CODE

This Appendix provides the various commands and lines of code used in this thesis work as referenced in Chapter III.

(1) HBase Table Creation

The following command creates an HBase table from within the HBase shell with the name “TABLE” with the column family “CF.”

```
create 'TABLE’, {NAME => 'CF’, VERSIONS => 1 }
```

(2) Loading Data into HBase from S3 Bucket

The following code loads the CSV file csvFile.csv from the S3 bucket location s3n://bucket-name/csvFile.csv into the previously created HBase table, TABLE, with column family CF, and assign names of the nine columns of data to “COL1”, “COL2”, “COL3”, “COL4”, “COL4”, “COL6”, “COL7”, “COL8”, and “COL9”. Note that this must be entered from the Hadoop CLI and not the HBase shell [8].

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv
-Dimporttsv.separator="," -Dimporttsv.columns=
HBASE_ROW_KEY,CF:COL1,CF:COL2,CF:COL3,CF:COL4,CF:COL5,CF:C
OL6,CF:COL7,CF:COL8,CF:COL9 ‘TABLE' s3n://bucket-name/csvFile.csv
```

(3) Launching phoenix

The following command launches the Phoenix CLI window when input into the Hadoop CLI window.

```
/usr/lib/phoenix/bin/sqlline-thin.py http://localhost:XXXX
```

(4) Mapping Phoenix to HBase Table

The following code creates a View in Phoenix and must be the same name as the table name of the HBase table previously created. Note that the column family names and individual column names must match the HBase table and are case-sensitive.

```
CREATE VIEW TABLE (
rowkey VARCHAR PRIMARY KEY,
"CF"."COL1" VARCHAR,
"CF"." COL2" VARCHAR,
"CF"."COL3" VARCHAR,
"CF"."COL4" VARCHAR,
"CF"."COL5" VARCHAR,
"CF"."COL6" VARCHAR,
"CF"."COL7" VARCHAR,
"CF"."COL8" VARCHAR,
"CF"."COL9" VARCHAR);
```

(5) Launching Spark

The following command launches the Spark shell when executed from the Hadoop CLI window.

```
spark-shell --executor-cores 3 --num-executors 4 --executor-memory 10G --jars
/usr/lib/phoenix/phoenix-spark-4.14.1-HBase-1.4.jar,/usr/lib/phoenix/phoenix-
client.jar
```

(6) Spark code

The following code loads the necessary functions into Spark, load the data from Phoenix into a Spark Dataframe, perform the necessary data preparation prior to implementing K-means on the data, and execute K-means clustering on the dataset.

```
import org.apache.hadoop.conf.Configuration
import org.apache.phoenix.spark._
import org.apache.spark.sql.functions._
import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.ml.feature.{VectorAssembler, MinMaxScaler}
import org.apache.spark.sql.{SparkSession, SQLContext}
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.sql.DataFrame
```

```
import org.apache.spark.ml.linalg.{Vector, DenseVector}
import org.apache.spark.mllib.linalg.{VectorUDT, Vectors, Vector}
import org.apache.spark.sql.types.{StructField, StructType}

//Load table as a Dataframe with use of a configuration object
sc.stop()
val configuration = new Configuration()
val sc = new SparkContext("local", "phoenix-load")
val sqlContext = new SQLContext(sc)

//Write data out into S3 bucket location
df.coalesce(1).write.csv("s3://path-to-bucket/file.csv")

//Read data out into S3 bucket location in separate Spark window
val df = spark.read.format("csv").option("inferSchema",true).load(
"s3://path-to-bucket/file.csv")

//Create a Dataframe with name 'df' containing data loaded from Phoenix
val df = sqlContext.phoenixTableAsDataFrame(
"TABLE", Array("ROWKEY","CF.COL1","CF.COL2", "CF.COL3","CF.COL4",
"CF.COL5","CF.COL6","CF.COL7","CF.COL8","CF.COL9"), conf =
configuration)

//Create an array of the columns containing the features for K-means
val columns = Array(
"DUR","AVG_AMP","AVG_POW","PAPR","SNR","SNR_DB","BW_3DB","B
W_10DB","BWNE")

//Combine the input columns array as "assembler" [15]
val assembler = new VectorAssembler().setInputCols(columns).
setOutputCol("featuresin")

val df = assembler.transform(featureDf)

//Scale feature data into values between -1 and 1
val scaler = new MinMaxScaler().setInputCol("featuresin").
setOutputCol("features").setMin(-1).setMax(1)

val scalerModel = scaler.fit(df)
val scaledData = scalerModel.transform(df)

//Run K-means model, specifying K value, Seed, and Max Iterations [15]
val kmeans = new KMeans().setK(2).setSeed(1L).setMaxIter(100)
val model = kmeans.fit(scaledData)
```

```
val predictions = model.transform(scaledData)
```

(7) Identify an ideal K value

The following code runs multiple iterations of K-means on the data loaded into Spark, each time calculating the cost. Once the change in cost from one iteration to another falls below a specified threshold, the code stops and output the value for K which was determined as an approximately ideal value.

```
object FindK {
  def main(args: Array[String]) {
    // Local variable declaration/run first iteration of Kmeans

    var k = 2;
    var Cost = new Array[Double](50)
    val kmeans = new KMeans().setK(k).setSeed(1L).setMaxIter(100)
    val model = kmeans.fit(scaledData)
    Cost(0) = model.computeCost(scaledData)
    var k = 3;

    // perform loop execution to find K value associated to minimal chance in cost
    do {
        val kmeans = new KMeans().setK(k).setSeed(1L).setMaxIter(100)
      val model = kmeans.fit(scaledData)
      Cost(k-2) = model.computeCost(scaledData)
      println( "Value of k: " + k);
      println( "Value of cost: " + Cost(k-2));
      k = k + 1;
    }
    while( Cost(k-3) < 0.95* Cost(k-4))
    var kReas = k - 2;
    println("K value: " + kReas);
    }
```

LIST OF REFERENCES

[1] S. Zheng *et al.*, "Big data processing architecture for radio signals empowered by deep learning: Concept, experiment, applications and challenges," *IEEE Access*, vol. 6, pp. 55907–55922, 2018.

[2] R. A. Romero, A. Rios, and T. T. Ha, "Signals of interest recovery with multiple receivers using reference-based successive interference cancellation for signal collection applications," *IEEE Access*, vol. 2, pp. 725–756, 2014.

[3] S. Nie and D. Sun, "Research on counter-terrorism based on big data," in *2016 IEEE International Conference on Big Data Analysis (ICBDA)*, 2016, pp. 1–5.

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

[5] P. A. Flach, *Machine learning: The art and science of algorithms that make sense of data*. Cambridge; New York: Cambridge University Press, 2012.

[6] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, Jun. 2010.

[7] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the Hadoop Ecosystem," *J. Big Data*, vol. 2, no. 1, p. 24, Nov. 2015.

[8] L. George, *HBase: The definitive guide: random access to your planet-size data*. Sebastopol, CA, USA: O'Reilly, 2011.

[9] S. Shirparv, *Learning HBase*. Birmingham, B3 2PB, UK: Packt Publishing Ltd, 2014.

[10] T. Gunarathne, *Hadoop MapReduce v2 cookbook*. Birmingham, B2 2PB, UK: Packt Publishing Ltd, 2015.

[11] E. McNulty, "SQL vs. NoSQL- what you need to know," *Dataconomy*, 01-Jul-2014. [Online]. Available: https://dataconomy.com/2014/07/sql-vs-nosql-need-know/.

[12] "NoSQL Databases Explained," *MongoDB*. [Online]. Available: https://www.mongodb.com/nosql-explained.

[13] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark*. Sebastopol, CA, USA: O'Reilly, 2015.

[14] P. Zečević and M. Bonaći, *Spark in Action*. Shelter Island, NY: Manning Publications Co, 2016.

[15] B. Chambers and M. Zaharia, *Spark: The Definitive Guide: Big Data Processing Made Simple*, 1st ed. Sebastapol, CA: O'Reilly Media, 2018.

[16] S. Ryza, Ed., *Advanced Analytics with Spark: Patterns for Learning from Data at Scale*, 2nd ed. Sebastopol, CA, USA: O'Reilly, 2017.

[17] Amazon Web Services, "Amazon EMR – Management Guide," 2019 [Online] Available: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-mgmt.pdf.

[18] "What Is Big Data?." [Online]. Available: https://www.oracle.com/big-data/guide/what-is-big-data.html.

[19] T. White, *Hadoop: The Definitive Guide*, 4th ed. Sebastopol, CA, USA: O'Reilly, 2015.

[20] "Overview | Apache Phoenix." [Online]. Available: http://phoenix.apache.org/.

[21] "Apache Phoenix," *Hortonworks*. [Online]. Available: https://hortonworks.com/apache/phoenix/.

[22] "K-means Clustering - MATLAB Kmeans." [Online]. Available: https://www.mathworks.com/help/stats/kmeans.html.

[23] "What is Amazon EC2? – Amazon Elastic Compute Cloud." [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

[24] "What is Amazon EMR? – Amazon EMR." [Online]. Available: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-what-is-emr.html.

[25] "Cloud Object Storage | Store & Retrieve Data Anywhere | Amazon Simple Storage Service," Amazon Web Services. [Online]. Available: https://aws.amazon.com/s3/.

[26] Hadoop Online Tutorials, "Apache Phoenix - An SQL Layer on HBase." [Online]. Available: http://hadooptutorial.info/apache-phoenix-hbase-an-sql-layer-on-hbase.

[27] S. S. Haykin, *Introduction to Analog and Digital Communications*, 2nd ed. Hoboken, NJ: Wiley, 2007.

[28] C. W. Therrien, *Discrete Random Signals and Statistical Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1992.

[29] "Multivariate Normal Random Numbers – MATLAB mvnrnd." [Online]. Available: https://www.mathworks.com/help/stats/mvnrnd.html.

[30] "AWS Command Line Interface," Amazon Web Services. [Online]. Available: https://aws.amazon.com/cli/.

[31] "HBase on Amazon S3 (Amazon S3 Storage Mode) – Amazon EMR." [Online]. Available: https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hbase-s3.html.

[32] Amazon Web Services, "Amazon EC2 Instance Types – Amazon Web Services," [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[33] "Amazon EC2 Key Pairs – Amazon Elastic Compute Cloud." [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html.

[34] "Connect to the Master Node Using SSH – Amazon EMR." [Online]. Available: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-master-node-ssh.html.

[35] "Using the HBase Shell – Amazon EMR." [Online]. Available: https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hbase-connect.html.

[36] "Installation." [Online]. Available: https://phoenix.apache.org/installation.html.

[37] "Overview." [Online]. Available: http://phoenix.apache.org/.

[38] R. Pedapatnam, "Understanding Resource Allocation configurations for a Spark application," Clairvoyantsoft, December 11, 2016. [Online]. Available: http://site.clairvoyantsoft.com/understanding-resource-allocation-configurations-spark-application/.

[39] "Apache Spark Plugin | Apache Phoenix." [Online]. Available: https://phoenix.apache.org/phoenix_spark.html.

[40] "The Use of Multiple Measurements in Taxonomic Problems – Fisher – 1936 – Annals of Eugenics – Wiley Online Library." [Online]. Available: https://onlinelibrary.wiley.com/doi/epdf/10.1111/j.1469-1809.1936.tb02137.x.

[41] "UCI Machine Learning Repository: Iris Data Set." [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Iris.

[42] S. Ghosh and S. Kumar, "Comparative analysis of K-Means and Fuzzy C-Means algorithms," *Int. J. Adv. Comput. Sci. Appl.*, vol. 4, no. 4, 2013. [Online]. doi:10.14569/IJACSA.2013.040406

[43] X. Dongkuan and Y. Tian, “A comprehensive survey of clustering algorithms,” *An. of Data. Sci.,* vol. 2, no.2, pp. 165–193, Jun. 2015 [Online]. doi:10.1007/s40745-015-0040-1.

[44] “Apache Hive TM.” [Online]. Available: https://hive.apache.org/.

[45] “Apache Cassandra.” [Online]. Available: http://cassandra.apache.org/.

[46] “Step 2: Launch Your Sample Amazon EMR Cluster – Amazon EMR.” [Online]. Available: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs-launch-sample-cluster.html.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California