



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**APPLICATION OF A BIG DATA PLATFORM FOR
BINARY DATA**

by

Pyojeong Kim

June 2019

Thesis Advisor:
Co-Advisor:
Second Reader:

Arijit Das
Sean P. Kragelund
Glenn R. Cook

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2019	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE APPLICATION OF A BIG DATA PLATFORM FOR BINARY DATA			5. FUNDING NUMBERS	
6. AUTHOR(S) Pyojeong Kim				
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) Using big data is essential for survival in any organization. While most data used in the workplace in the past were structured, the use of unstructured data is dramatically increasing and accounts for most of data growth. Handling unstructured big data is now significant for success in organizations. Sonar files are an example of unstructured big data used in the U.S. Navy. The Navy stores sonar data on separate data storage devices such as the hard disks of laptops and CDs. When users need to make a decision by analyzing the data, they must manually search for the relevant file across the storage system and physically transfer it to the proper applications to be read, which delays decision making. This thesis suggests a way to handle this challenge by using a big data platform. In this thesis, the performance of a big data platform is compared to that of the standalone system. Specifically, the binary data of sonar files were extracted and converted into metadata, which can be used for a query in the Oracle relational database and Application Express (APEX). As an example of big data analytics, a Multilayer Perceptron Regression algorithm was implemented with the metadata. The same environment was established in a Hadoop ecosystem like MapReduce, Hive, and Spark. Finally, this research suggests that a big data platform would be economically beneficial.				
14. SUBJECT TERMS sonar data, machine learning, database, Hadoop, spark, big data, binary data			15. NUMBER OF PAGES 129	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

APPLICATION OF A BIG DATA PLATFORM FOR BINARY DATA

Pyojeong Kim
Major, Republic of Korea Army
Bachelor's degree, Korea Military Academy, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF BUSINESS ADMINISTRATION

from the

**NAVAL POSTGRADUATE SCHOOL
June 2019**

Approved by: Arijit Das
Advisor

Sean P. Kragelund
Co-Advisor

Glenn R. Cook
Second Reader

Glenn R. Cook
Academic Associate, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Using big data is essential for survival in any organization. While most data used in the workplace in the past were structured, the use of unstructured data is dramatically increasing and accounts for most of data growth. Handling unstructured big data is now significant for success in organizations.

Sonar files are an example of unstructured big data used in the U.S. Navy. The Navy stores sonar data on separate data storage devices such as the hard disks of laptops and CDs. When users need to make a decision by analyzing the data, they must manually search for the relevant file across the storage system and physically transfer it to the proper applications to be read, which delays decision making. This thesis suggests a way to handle this challenge by using a big data platform.

In this thesis, the performance of a big data platform is compared to that of the standalone system. Specifically, the binary data of sonar files were extracted and converted into metadata, which can be used for a query in the Oracle relational database and Application Express (APEX). As an example of big data analytics, a Multilayer Perceptron Regression algorithm was implemented with the metadata. The same environment was established in a Hadoop ecosystem like MapReduce, Hive, and Spark. Finally, this research suggests that a big data platform would be economically beneficial.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	RESEARCH QUESTIONS.....	2
B.	SCOPE AND LIMITATIONS.....	2
C.	THESIS OVERVIEW	4
II.	LITERATURE REVIEW	5
A.	QUICK SEARCH FOR SONAR DATA FILES.....	5
1.	Structured Query Language	6
2.	Metadata	6
3.	Sonar	8
B.	SCALE OUT BY BIG DATA PLATFORM	11
1.	Definition and Characteristics of Big Data.....	12
2.	Big Data Analytics.....	13
3.	Hadoop.....	17
4.	Spark	20
C.	EVALUATING ADOPTION OF TECHNOLOGY	23
1.	Cost Benefit Analysis	23
2.	Benefit Cost Analysis	24
III.	METHODOLOGY	25
A.	METADATA EXTRACTION AND STORAGE.....	25
1.	Extraction of Metadata in Binary Form.....	25
2.	RDBMS	26
B.	HANDLING BIG DATA.....	27
1.	MapReduce.....	27
2.	Hive.....	28
C.	DATA MINING	29
1.	Categorization	30
2.	Process.....	31
3.	Case for This Study.....	33
D.	BENEFIT COST ANALYSIS.....	36
1.	Components of Benefits.....	36
2.	Costs and Risk.....	38
IV.	IMPLEMENTATION	41
A.	LOADING DATA IN A STANDALONE LAPTOP.....	41
1.	Extracting Metadata of .mst	42
2.	Loading Data into RDBMS.....	47

3.	Visualizing the Data in APEX.....	47
B.	LOADING DATA INTO THE HADOOP CLUSTER.....	49
1.	MapReduce.....	49
C.	PREDICTING OCEAN DEPTH AT A LOCATION	54
1.	Hadoop Cluster	55
2.	Spark	60
D.	TEST WITH BIG DATA	61
V.	RESULTS	63
A.	EASE OF USE FOR SEARCH.....	63
B.	PERFORMANCE BETWEEN SYSTEMS	65
1.	Sample Data.....	65
2.	Increased Data.....	69
3.	Summary.....	69
VI.	CONCLUSION	71
A.	BENEFITS.....	71
B.	COSTS	73
C.	RISK.....	73
D.	NET PRESENT VALUE	73
E.	SUMMARY AND FUTURE RESEARCH.....	74
	APPENDIX.....	77
A.	JAVA CODE FOR EXTRACTING METADATA FROM SONAR FILES	77
B.	JAVA CODE FOR GETTING PATH	84
C.	JAVA CODE OF LOADING DATA INTO ORACLE DATABASE 11G.....	85
D.	APEX APPLICATION CODE	86
1.	PL/SQL Code under Dynamic Action	86
2.	JavaScript Code under Dynamic Action	86
3.	JavaScript of Source in Regions	86
E.	HADOOP MAPREDUCE CODE	88
F.	IMPLEMENTING MULTILAYER PERCEPTRON REGRESSION	98
1.	Hadoop.....	98
2.	Spark	101
	LIST OF REFERENCES	105
	INITIAL DISTRIBUTION LIST	111

LIST OF FIGURES

Figure 1.	Entire process of research	3
Figure 2.	Active sonar (left) and passive sonar (right). Source: Knight, Pridham, and Kay (1981).	8
Figure 3.	Measuring depth by single beam sonar. Source: U.S. Geological Survey (n.d.).	9
Figure 4.	Depth change at Port Bolivar, Texas. Source: Brennan et al. (2005).	10
Figure 5.	Types of big data analytical tools. Source: Sivarajah et al. (2017).	15
Figure 6.	High-level Hadoop architecture. Source: Holmes (2012).	18
Figure 7.	MapReduce data flow. Source: White (2015).	19
Figure 8.	Hadoop ecosystem in Cloudera. Source: Gerard (2015).	20
Figure 9.	Spark architecture. Source: Chambers and Zaharia (2018).	21
Figure 10.	Comparison between Hadoop and Spark. Source: Achari (2015).	22
Figure 11.	Dynamic Acyclic Graph. Source: Gulati and Kumar (2017).	22
Figure 12.	MapReduce implemented by one job. Source: Holmes (2012).	28
Figure 13.	Example of query and result in Hive. Source: Capriolo et al. (2012).	29
Figure 14.	Categorization of data mining. Source: Martín et al. (2014).	30
Figure 15.	Process of machine learning in Spark. Source: Chambers & Zaharia (2018).	31
Figure 16.	The example of a sine wave. Source: Kong (2017).	33
Figure 17.	Activation function. Source: Reese, Reese, and Grigorev (2017).	35
Figure 18.	The process of multilayer perceptron. Source: Reese et al. (2017).	36
Figure 19.	Classification framework for benefits. Source: Ward, Daniel, and Peppard (2007).	37
Figure 20.	MSTIFF structure. Source: Marine Sonic Technology, Ltd. (2011).	43
Figure 21.	Java code for getting IFD offset.	43
Figure 22.	Hex converter.	44

Figure 23.	Finding Fathometer code	44
Figure 24.	Converting timestamp into the real time.....	45
Figure 25.	APEX workshop page.....	47
Figure 26.	APEX application editor page	48
Figure 27.	Hadoop MapReduce job configuration in main class	50
Figure 28.	nextKeyValue() method in WholeFileRecordReader class	51
Figure 29.	Input key and value in Mapper class.....	52
Figure 30.	MapReduce flow to extract metadata of sonar files.....	53
Figure 31.	Command for loading the metadata from the output of Hadoop MapReduce and creating external table	54
Figure 32.	Process of predicting the depth of seabed.....	55
Figure 33.	Main class of multilayer perceptron regression	56
Figure 34.	compareTo method of StringPair class	57
Figure 35.	compare method in Stringpair_Comparator class.....	58
Figure 36.	DataFrame in multilayer perceptron regression.....	59
Figure 37.	Configuration for multilayer perceptron regression	59
Figure 38.	Training and evaluating the MLP model	60
Figure 39.	Dynamic Acyclic Graph	61
Figure 40.	Result of searching for files in a certain location in SQL Developer	64
Figure 41.	Result of searching data of certain date in APEX.....	64
Figure 42.	Result of searching for files in a certain location in Hive.....	65
Figure 43.	Implementation of extracting the metadata from sample files.....	66
Figure 44.	Time required for an MLP regression depending on the number of reduce tasks	67
Figure 45.	Time required depending on the number of partitions in Spark	69
Figure 46.	Benefits of using a big data platform.....	72

LIST OF TABLES

Table 1.	Description of methods used for extracting the metadata from sonar files.....	46
Table 2.	Time required for an MLP regression with sample data in Spark	68
Table 3.	Time required for each experiment.....	70

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ANN	Artificial Neural Network
APEX	Application Express
API	Application Programming Interface
BCA	Benefit Cost Analysis
CBA	Cost Benefit Analysis
CSV	Comma-separated Value
DAG	Dynamic Acyclic Graph
DCL	Data Control Language
DDL	Data Definition Language
DML	Data Manipulation Language
EDA	Exploratory Data Analysis
GC	Garbage Collection
GUI	Graphic User Interface
HDFS	Hadoop Distributed File System
I/O	Input and Output
IFD	Image File Directory
ITACS	Information Technology and Communications Services
JSON	JavaScript Object Notation
M2M	Machine-to-Machine
Mlib	Machine Learning Library
MLP	Multilayer Perceptron
MSTIFF	Marine Sonic Technology, Ltd. Image Data File Format
NISO	National Information Standards Organization
RDBMS	Relational Database Management System
RDD	Resilient Distributed Dataset
SQL	Structured Query Language
SQUARE	structured queries as relational expressions
TCARI	Tidal Constituent and Residual Interpolation

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, I would like to thank my wife, Hyejin, for standing beside me throughout my career and as I wrote my master's thesis. I also thank my wonderful son, Jaeha, for always making me smile.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The speed of data growth has been dramatically increasing for the past decade and overwhelming the capacity of legacy systems such as the relational database. Big data technology has been used as an alternative, which has improved the performance of data processing and opened its potential power by not only dramatically improving performance of businesses but also opening new possibilities for data usage. Big data allows people to analyze situations as fast as possible by overcoming bounded rationality with minimal time. A person can use information via many sensors even though one cannot recognize the actual information flow. After all, big data helps to converge all types of data and provide new insight about the real world.

For example, Amazon uses big data to provide various recommendations to customers, one being a book recommendation service. Based on analysis of the information about a book that a customer purchased, Amazon recommends other books to customers using big data. Customers cannot search for all the books in the world to find out what they really want to read. They also have bounded rationality, which means that even though they read every book's title and abstract, they would choose the book through considering just a few conditions that they can think of at that moment (Gigerenzer & Selten, 2002). By contrast, Amazon can do all these processes for customers, which means customers partially delegate their decision to choose the books.

The Department of Defense (DoD) also has focused on big data. The DoD invested more than \$7 billion on big data, AI, and cloud services in 2017; from 2012 to 2017, big data accounted for 54.4 percent of DoD spending in these areas (Corrin, 2017). Big data will have a huge impact on the military, just as it has influenced other sectors. Among those diverse data, the U.S. Navy maintains huge datasets of oceanographic data, including sonar files. Since this data has accumulated over a long period of time, users have trouble analyzing the whole set of data in a short time with legacy systems, even though the potential power of using big data platforms would be transparently expected. For example, in order to detect mines via side-scan sonar, machine learning algorithms can be used (Barngrover, 2014). Yet, machine learning algorithms need many repetitive calculations,

which would take a long time. On the other hand, big data platforms would provide a distributed computing capability to enhance the performance of a machine learning algorithm against a huge sonar dataset in less time.

The Naval Postgraduate School (NPS) users of sonar data have a large amount of historical sonar data stored in separate laptops. With this in mind, I expected the big data platform would improve the performance of analysis about sonar data in the NPS team by providing a quick query and search tool. Additionally, this technology would suggest new insights in analyzing sonar data.

The purpose of this study is to show the application of big data technology by using a sample side-scan sonar dataset collected by NPS in a 2014 field experiment. By using Java, Application Express (APEX), and Hive to manage a sample side-scan sonar dataset, this research supports the efficiency of big data technology. In addition, by using a machine learning algorithm in Hadoop, this research also shows the potential of big data technology for analysis in naval applications.

A. RESEARCH QUESTIONS

The research questions we address are:

1. Is searching, sorting, and extracting data of interest more efficient with Hadoop than with standard tools?
2. Can a large set of files be organized into a database more quickly using Hadoop techniques than with standard tools?
3. Is Hadoop data-analysis able to be applied to large datasets?
4. What are the implementation challenges of establishing a big data platform?

B. SCOPE AND LIMITATIONS

To address these research questions, this thesis used a side-scan sonar dataset collected by NPS during a 2014 field experiment. We extracted metadata and created a

database with a map visualization front-end in a standalone system; then we did the same process on a Hadoop cluster using big data tools.

The metadata such as time and location were extracted from the sonar data, and used for querying to quickly search the files in a standalone system by using Java and the Oracle database. Additionally, Oracle APEX, which is compatible to the Oracle Database, was used for visualization through the Google Map application. Next, we performed the same metadata extraction process in a Hadoop cluster. Hive loaded the metadata to provide users with a chance to use Hive-QL, which is similar to a Structured Query Language (SQL). Finally, by using a machine learning method on a big data platform, we demonstrated an ability to predict the water depth in certain areas at certain times, based on previously collected sonar metadata. The entire process of the study is briefly shown in Figure 1.

In our study, we did not transfer the data to Oracle database by using Sqoop since we focused on the methods used to extract metadata from sonar files and handle the files with the metadata within a big data platform.

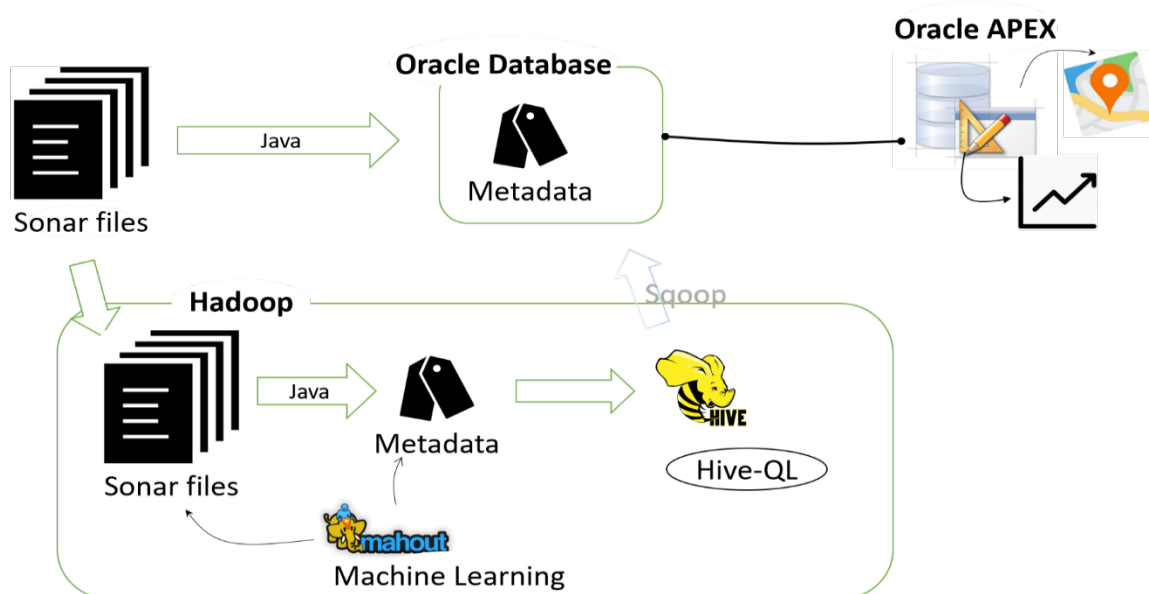


Figure 1. Entire process of research

C. THESIS OVERVIEW

The thesis is organized into six chapters. Chapter I describes the motivation and research questions for this thesis. Chapter II includes background related to the tools and methods used in the research. Chapter III discusses the thesis methodology. Chapter IV describes software implementation. The results are included in Chapter V. Finally, Chapter VI presents conclusions about the research questions and applies the Benefit Cost Analysis (BCA) to the research and gives suggestions for future research.

II. LITERATURE REVIEW

In this chapter, the foundational concepts are defined related to the research questions. To begin with, we describe each related concept to search for desired sonar files by using their metadata and a database. Next, we review the concepts of big data, data analytics and a big data platform, Hadoop. Finally, we study a way to evaluate the technology's economic impact to determine whether to adopt this technology in an organization.

A. QUICK SEARCH FOR SONAR DATA FILES

In any workplace, data is generated for diverse purposes and is stored in a file to access later. To find that data again, one must typically search for the file's name or creation date, and then determine which data to use in the file. To simplify data retrieval, most users store the data in a relational database which uses a Structured Query Language (SQL) system, since it allows them to find the data through simple queries, which ensures ease of use.

However, as the scope of one's work increases, the number of diverse data types, such as videos and images, can complicate finding the desired files. For large datasets, users may want to store and query/extract files of interest based on their descriptive metadata. The data used in this study are side-scan sonar image files stored in the Marine Sonic Technology, Ltd. Image data file format (MSTIFF), with an .mst file name extension (Marine Sonic Technology, Ltd., 2011). These files not only need a special program to read them, but their information content can only be determined after reading the file and parsing its embedded metadata. Therefore, to improve the file search process with these image files, the metadata from each file is used as key words to make a relational database through SQL (Z. Chen, Torson, & Servisetti, 2011). For our sonar data files, the descriptive metadata includes the date, time, and location it was collected.

1. Structured Query Language

SQL is a query language for databases, using relational mapping between the desired output relation from the input relation (Don, 2009). SQL includes data definition language (DDL), data manipulation language (DML), and data control language (DCL) (Don, 2009). SQL is not limited to a specific database system and is not in wide use, even though it was initially used only in IBM relational databases (Don, 2009).

Although structured queries such as relational expressions (SQUARE) was published in 1973, it was difficult for beginners to use due to many mathematical expressions. Based on SQUARE, structured English as query language was developed at IBM in 1974, and SEQUEL 2 was released in 1976, which became known as SQL (Don, 2009).

SQL has data definition and manipulation capabilities as well as a simple query functionality (Don, 2009). It can also be used through online terminals or inserted into programs in diverse languages (Don, 2009). In addition, SQL has a syntax similar to English sentences, which leads to relative ease-of-use, even for beginners (Ben-Gan, 2016).

2. Metadata

SQL provides a quick and easy query function. However, it needs data to be stored in a structured form such as a table since SQL is the standard user and application programming interface (API) (Don, 2009). However, since unstructured data such as images, which consist of binary data, is difficult to store in this format, descriptive information such as date, time, and location derived from the data can be used as a tag for searching (Yee, Swearingen, Li, & Hearst, 2003). Therefore, the metadata is able to be used as a key word during file searches.

a. Definition

The data describing other data of interest is called metadata (Riley, 2017). It improves the quality of search results by providing indexes for the data (Riley, 2017). For example, the metadata can be used as the schema of a table in a Relational Database Management System (RDBMS), as the name of the file, the date of data creation, or a key

to identify the data. Metadata allows users to concentrate on a target area of interest, which not only saves time associated with manually scanning a list of files but also removes time for reading the contents of each file during search (Google, n.d.).

There are several ways to generate metadata. Some metadata are automatically created in a certain format. For instance, the date of creating or accessing a file is stored automatically in the properties of the file. Others metadata can be created based upon the intention of users, such as a tag describing the subject of a picture contained in an image file.

b. Types of Metadata

The National Information Standards Organization (NISO) suggests three types of metadata: descriptive metadata for identification, which includes the title, abstract, author, and keywords; structural metadata for organizing information, just like a book uses chapters; finally, administrative metadata such as the file type, who can access it, etc. for file management (Riley, 2017).

Our sample sonar data is stored in a media file format. The media format includes GIF, JPEG, MP3, TIFF, etc. Examples of the metadata stored in some common media formats are listed below:

- GIF: Comment extension used for descriptions and application extension describing which application it uses (Oracle, n.d.-a)
- JPEG: Information about the digital camera, date and time the picture was taken, distance setting for the camera's focus, and GPS information where the picture was taken (Parameshwaran, 2016)
- TIFF: Date and time created, description of image, equipment to make image, software version of image, creator, and copyright holder (JEITA, 2002)

The sonar image file not only has a date and timestamp like TIFF but it also has latitude, longitude and additional information like depth, which are used as metadata in this study.

3. Sonar

Sonar is an acronym for Sound Navigation and Ranging, which uses sound propagation to locate objects on or under the water. Initial applications of this characteristic of sound was used for war. For instance, an underwater bell was installed in lighthouses to warn of the approach of an enemy's ships. When submarines appeared in World War I, the British developed active sonar technology to detect Germany's submarines. Figure 2 shows two types of sonar, active and passive. Active sonar transmits an acoustic pulse toward an object and listens for its echoes. Passive sonar, on the other hand, only listens for the sound transmitted by other sources. Amongst two types of sonar, active sonar can be used to measure the depth of the water.

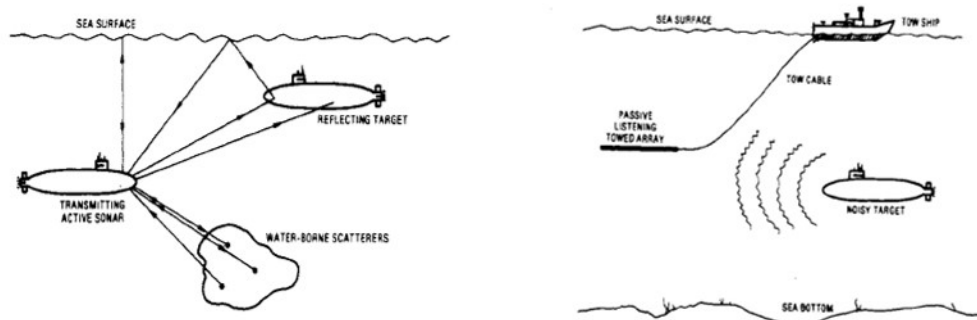


Figure 2. Active sonar (left) and passive sonar (right). Source: Knight, Pridham, and Kay (1981).

a. Building a Seabed Map by Sonar

A seabed map consists of depth information, which can be measured by sonar. For example, Figure 3 shows how single beam sonar can measure water depth to create a seabed map. After the transducer in the ship emits an acoustic pulse, it listens for the pulse to come back. The acoustic energy pulse directly goes down toward the sea floor and is reflected by the sea floor. When the transducer gets the pulse, it calculates the distance by

multiplying the speed of the pulse in the water by the traveling time of the pulse (U.S. Geological Survey, n.d.).

Even though a seabed map can be made this way, it cannot provide information about real time changes of depth since diverse factors such as tidal currents affect depth information (Senet, Seemann, Flampouris, & Ziemer, 2008). Additionally, the magnitude of effects caused by tidal fluctuations differs depending on the location (Hendry, n.d.). As a result, the lowest level of depth is used to make a seabed map, which is the lowest astronomical tide (Jagalingam, Akshaya, & Hegde, 2015).

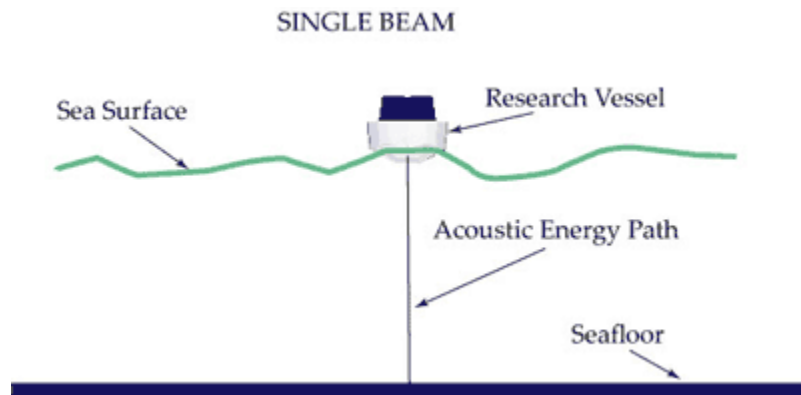


Figure 3. Measuring depth by single beam sonar. Source: U.S. Geological Survey (n.d.).

However, this still cannot tell real time depth information in the seabed. Therefore, the tidal corrector method has been studied, and Tidal Constituent and Residual Interpolation (TCARI) was introduced as a solution in 1995 (Brennan, Hess, Huff, & Gill, 2005). TCARI predicts the depth by using statistical depth data. Figure 4 shows depth variations as a function of time. Since the period of depth changes is not uniform across all locations in the ocean, the depth of all locations should be separately calculated (Brennan et al., 2005).

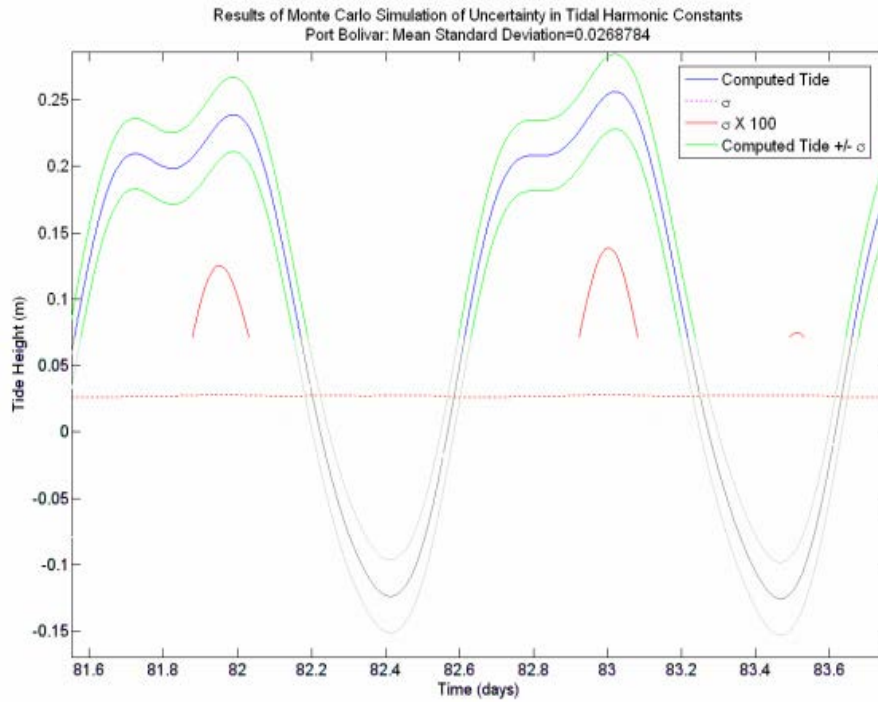


Figure 4. Depth change at Port Bolivar, Texas. Source: Brennan et al. (2005).

b. Structure of a Sonar File

In this study, we analyzed data files collected by a commercial side-scan sonar made by Marine Sonic Technology, Ltd. This sonar generates image data files using the MSTIFF file format (.mst filename extension). The metadata for these sonar images contain navigational data for the sonar's location, depth, etc., at the time when each image was created. The file consists of an image file header, an image file directory, and binary data representing the actual navigational data values (Marine Sonic Technology, Ltd., 2011).

The header starts with an ASCII identifier, "MSTL" and includes the address of the image file directory. The image file directory consists of entry counts, which represent the number of entries; each directory's information tag, which indicates the type of information (e.g., navigation data); and its value, which represents the address of the data within the file (Marine Sonic Technology, Ltd., 2011). Finally, binary data is stored at the address

value's location. The most important information for this study, such as navigation data, was stored in a C programming data structure, which can be easily extracted in binary form.

B. SCALE OUT BY BIG DATA PLATFORM

With the increasing use of PCs and mobile devices, the data generated by users is increasing exponentially. For example, in the past when people bought something at a store, data related to the transaction such as payment method, credit card number, and identification of the ordered item was recorded in the balance sheet of the store. However, now online shopping websites generate and store every record of visitors' activities as data without them realizing it, even if they do not make a purchase. For example, they may record the movement of a user's mouse cursor on their webpage to find out the user's interests for future marketing. In addition, the data generated by the communication between machines called Machine-to-Machine Communication (M2M) is dramatically increasing. A very common example of M2M is a traffic control system. Sensors monitoring traffic on the road send data about the speed and volume of traffic to traffic controllers to optimize traffic flow. The increasing tendency of applying M2M leads to a dramatic growth of digital information.

An average of 2.5 quintillion bytes of data is created every day. Global data sizes are expected to increase to 40 zeta bytes in 2020 (Sivarajah, Kamal, Irani, & Weerakkody, 2017). One zeta byte is 1,000 exabytes, and one exabyte is the amount of information printed by the U.S. Library of Congress (Lyman et al., 2005).

For the application explored in this thesis, sensor data like sonar, which has been collected over several years, has reached an immense volume. Even though RDBMS has been improved to deal with a large quantity of data, it does not have the capability to store and manage big data. To deal with big data, Hadoop is suggested as an alternative (White, 2015).

This section first covers the definition and characteristics of big data and its analytics, then shows big data platforms such as Hadoop and Spark.

1. Definition and Characteristics of Big Data

Big data usually means a massive volume of data. However, as technology of storage and computing develops, this definition has hardly been applied. According to McKinsey Global Institute, big data is used to describe a massive amount of data that traditional data processing systems and software cannot manage (Manyika et al., 2011). For example, Oracle Database 11g, which is a widely used relational database, cannot physically handle more than 65,533 files (Oracle, n.d.-b). Gartner, a global advisory company, defines the dimensions of big data by using 3V acronyms such as high volume, velocity and variety. Information assets with these characteristics can be processed using big data platforms to support decision-making processes (Beyer & Laney, 2012). In addition, IBM added veracity to McKinsey's dimensions of big data and made 4 Vs (University of Wisconsin Data Science, n.d.). IBM defines big data as an opportunity to gain insights that legacy systems never acquired, making big data users more agile to changing business environments. Besides, Anil Jain, MD, who is the vice president and chief medical officer at IBM Watson Health, uses one more V for value to describe big data (Jain, 2016). The 5 Vs of big data cover:

- **Volume:** The size of data. Since data is continually generated by a huge number of devices used throughout the world such as personal computers, mobile phones, diverse sensors on devices, etc., its size is too large to be analyzed with legacy systems (White, 2015).
- **Velocity:** The speed of data generated. The speed of data generated in real time from all the types of devices we use becomes faster with technology development. Even though one device constantly generates data for its own operation, the number of devices interacting with other devices grows dramatically due to modern devices' high interoperability, which leads to the rapid explosion of data growth (White, 2015).
- **Variety:** The diversity of data sources and quality. On the contrary to traditional relational databases, a big data platform can accommodate

diverse data formats as well as structured data such as strings and images (White, 2015).

- Veracity: Uncertainty of data quality. Since a big data platform accommodates any type of data regardless of its accuracy, it may include a lot of false data as a tradeoff for its volume and velocity. As a result, a big data platform needs a process to remove dirty data (Normandeau, 2013).
- Value: Data must be valuable, which includes considerations about the potential usage of data (Jain, 2016).

The definition of big data is still changing since this technology is in the early stages. However, the fact is that it has played a significant role in the improvement of organizational effectiveness and capabilities, even in the public organization (Manyika et al., 2011).

Different fields are benefiting from big data. According to Franceschi-Bicchierai (2014), many intelligence organizations use big data from diverse private sector businesses to analyze and predict future threats. In sports, big data from sensors on people's bodies can help train people by informing them of proper movement and how to immediately adjust (Picks, 2018). Google used big data in Flu Trends to predict the degree of flu in a certain area. The only thing users do is click and see the result. The users not only do not have to pay for experts to get the advice but also do not need to spend much time to analyze the situation.

As can be seen from these examples, big data allows people to analyze situations as fast as possible. Big data helps overcome bounded rationality with minimal time. A person can use information from many sensors even though one cannot observe the actual information flow. After all, big data helps to converge all types of these data and provide new insights about the real world.

2. Big Data Analytics

Big data is not significant by itself. The value of big data lies in using it for analytics. Data mining and analytics change raw data into valuable information. Analyzing

data through algorithms, simulations, experiments, and mathematical modeling techniques has become very popular and is creating new knowledge with insights that often lead to innovation in organizations (Sivarajah et al., 2017). They not only help people to recognize new correlations between data, but also provide new insights to understand phenomena through analytical tools. Analytical tools are distinguished by the source of data such as text, audio, video, and social networks, as well as the purpose of analytics such as predictive analytics (Gandomi & Haider, 2015). Wang et al. (2016) group analytical tools into five categories: descriptive, inquisitive, predictive, prescriptive, and pre-emptive analytics (Figure 5).

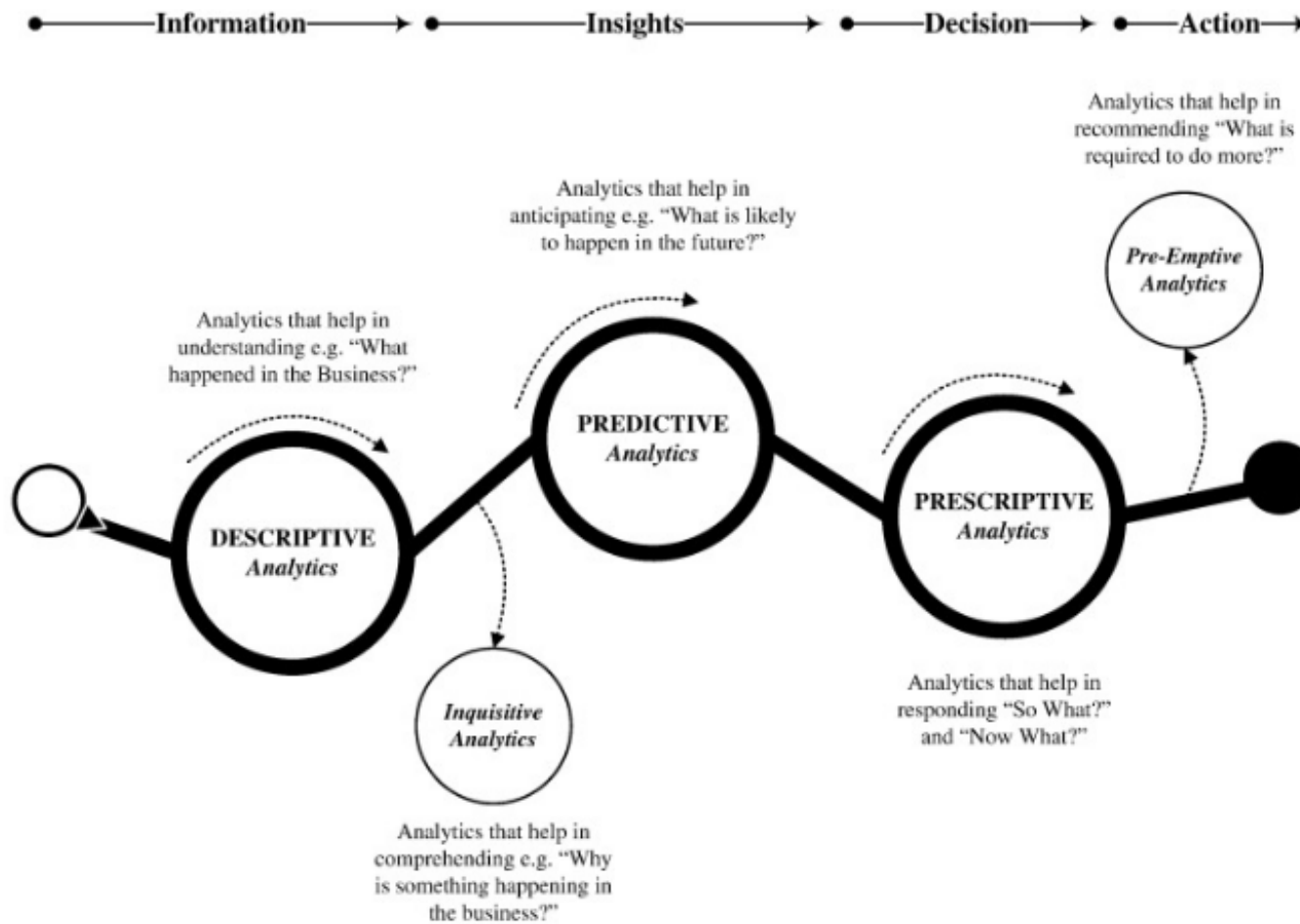


Figure 5. Types of big data analytical tools. Source: Sivarajah et al. (2017).

Depending on one's purpose for using big data, one must choose which analytical tool to use. Descriptive analytics explain a situation by providing patterns and correlations (Joseph & Johnson, 2013). Inquisitive analytics try to determine the underlying causes of a given situation, which leads to validation of the decision's hypotheses (Bihani & Patil, 2014). Predictive analytics anticipate the future of a specific business event based on the pattern found in historical data and statistical methodology (Bihani & Patil, 2014). Prescriptive analytics offer alternatives for solving a predictive problem (Bihani & Patil, 2014). Finally, pre-emptive analytics take precautionary actions to prevent possible problems from emerging (Smith, Szongott, Henne, & von Voigt, 2012).

Although big data offers opportunities to get new insights and knowledge through these analytical tools, utilizing big data also has some challenges. First, the cost of establishing the infrastructure of big data is high for high computing processes (Wang & Wiebe, 2014). On top of that, the type of data is too diverse to integrate as one, which adds many complexities (Gandomi & Haider, 2015). In addition, synchronizing large data is a complex task that places a heavy burden on users (Jin, Wah, Cheng, & Wang, 2015). After all, dramatically scalable, high-performance systems that can accommodate any type of data are demanded for big data analytics.

Even before the concept of big data emerged, some traditional systems could handle large amounts of data. Effective systems were typically expensive, however, and supercomputers were required to process the data. Nonetheless, these existing systems had several limitations for dealing with big data. For example, even though relational database management systems (RDBMS) were widely used in the industry, it became slow to use frequent queries against big data since the improvement of search time in a disk is slower than that of the transfer rate (White, 2015). Besides, RDBMS only handles structured data while the most significant data is generated in diverse formats (White, 2015). Grid computing, while demonstrating great performance for computing-intensive tasks, cannot handle data sizes due to the bottleneck of network bandwidth (White, 2015).

Due to these limitations of existing systems, companies have focused on analyzing only the data that they consider important. Although these existing systems were costly, they still lacked sufficient data storage and computing resources. In addition, companies

lacked relevant technology or engineers. As a result, it was difficult to create an environment where companies could freely analyze the data they provide. Therefore, there has been a constant need for a methodology to process existing data cost-effectively. Hadoop was designed with this need in mind.

3. Hadoop

Hadoop is a technology that handles large amounts of data by combining several inexpensive computers as if they were one (White, 2015). Hadoop's core components are the Hadoop distributed file system (HDFS) and MapReduce. HDFS is a distributed file system that provides the ability to store large files on thousands of distributed devices, and MapReduce is a distributed computing framework that allows you to quickly and easily analyze stored file data using CPU and memory resources on a distributed server (White, 2015). Hadoop is not expensive compared to other systems since it is open source, which does not require expensive software licenses, and uses large quantities of inexpensive commodity hardware combined as a cluster. Hadoop is a scale-out architecture that linearly increases performance as resources are invested. It also provides data recoverability by replicating data in several nodes in preparation of data errors. In addition, the failure of a component does not affect the whole job.

These characteristics of Hadoop are attractive for analyzing large, diverse datasets of accumulated sensor data for science or research. Hadoop also ensures data consistency and recoverability, which relieves concerns about data loss by computer error. In addition, Hadoop can be used to apply big data analytics to a sample collection of sonar files, with the potential to discover new patterns, correlations, and insights in this type of data.

Hadoop utilizes the Hadoop distributed file system (HDFS), and provides MapReduce, a method for parallelizing data processing on the system. Figure 6 illustrates how the storage for each node is partitioned and controlled by an HDFS master node, which provides scalable storage by adding the number of nodes required by the user. The HDFS master also provides fault tolerance by replicating the data in a default number of nodes (Holmes, 2012).

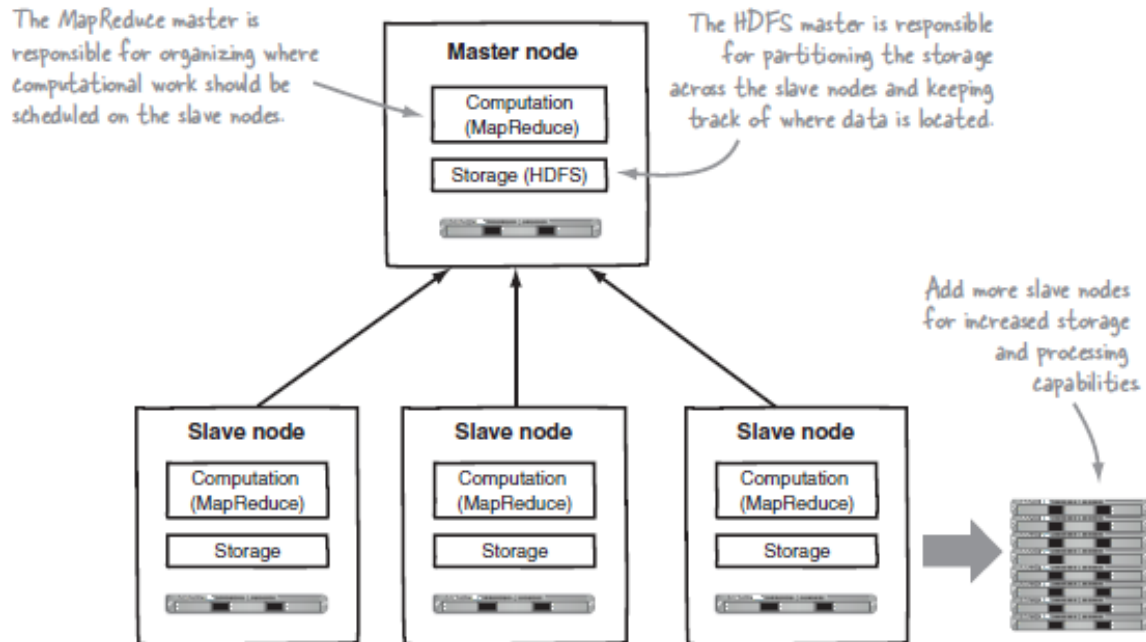


Figure 6. High-level Hadoop architecture. Source: Holmes (2012).

MapReduce is a distributed computing framework that provides parallelization of processing over the whole system. Figure 7 breaks down MapReduce into three parts: mapping, shuffling, and reducing. Initially, input data is divided into three splits that go into each node and are processed to extract data in the form of keys and values in the mapping step. The key and value pairs are passed to the next nodes, which are responsible for reducing jobs in the shuffling step, which gathers data with the same key. In the reducing step, the input data of key and value pairs are used to produce user-desired outputs through diverse functions such as sum, average, or deviations (Holmes, 2012).

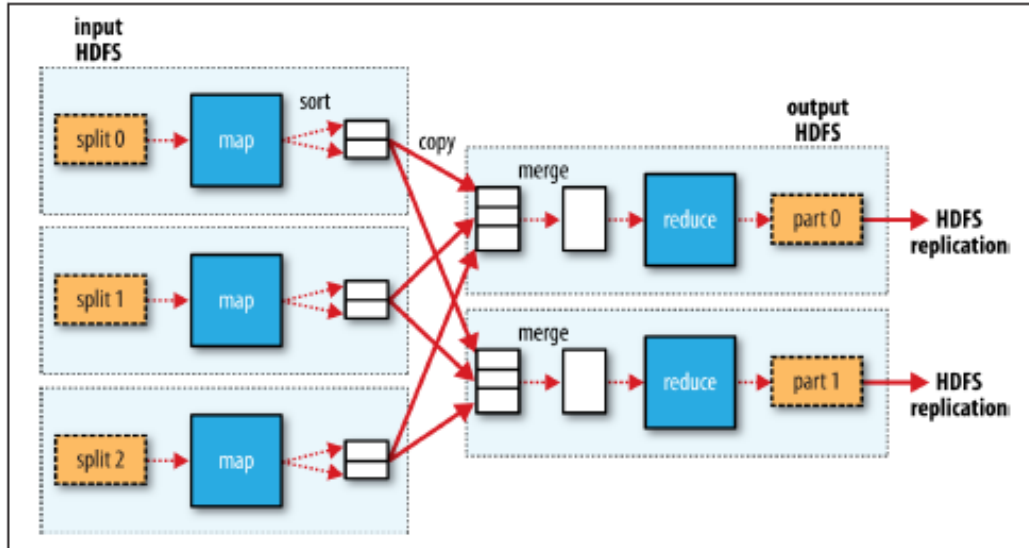


Figure 7. MapReduce data flow. Source: White (2015).

In addition to the main component above, diverse tools have been developed to improve the performance, functionality, and quality for analytics based on Hadoop, which is called the Hadoop ecosystem. Since the Hadoop ecosystem continues to rapidly evolve and varies by vendors, we elected to use a Cloudera Hadoop Cluster for this thesis case study.

Figure 8 illustrates the Hadoop ecosystem in Cloudera. Based on Hadoop's core components, Cloudera provides separate but compatible projects with considerable capabilities for big data analytics. The HBase project plays the role of a Hadoop database for data storage using NoSQL, an acronym for "not only SQL." The Flume and Sqoop projects, meanwhile, are responsible for data transportation between different systems. For example, if the input data is contained in an RDBMS, the user can transfer the data into Hadoop via Sqoop. Flume is useful to transfer streaming data like log files. In addition, the Hive and Impala projects provide SQL-like command line functionality. Pig is a useful project for searching data. Finally, the Mahout project provides a machine learning library for more complicated analysis.

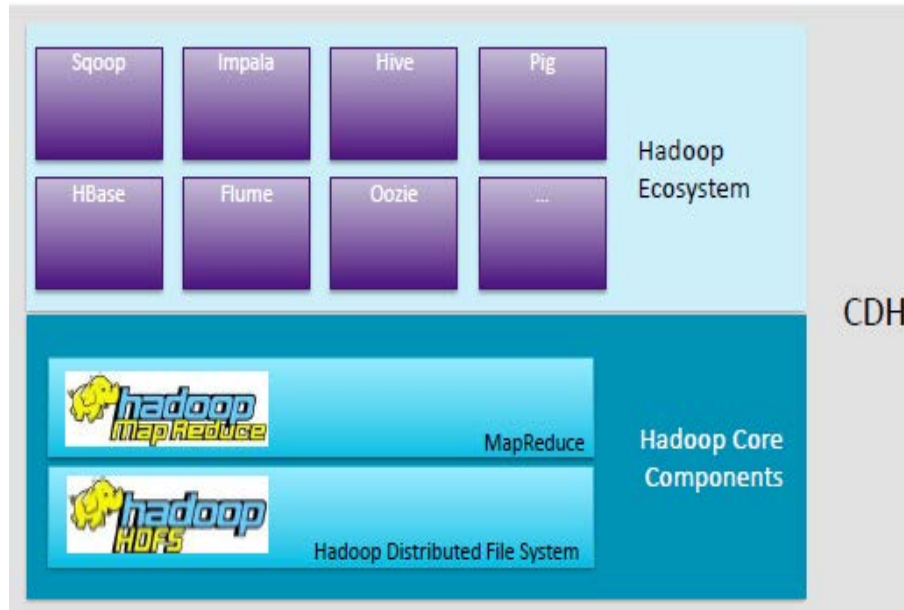


Figure 8. Hadoop ecosystem in Cloudera. Source: Gerard (2015).

4. Spark

As Hadoop consists of master and slave nodes, the Spark distributed processing framework utilizes a driver process and multiple executors. Figure 9 shows how they work together. The driver process is located in the cluster node at the center of the system. It responds to the commands and code made by users, manages the job across executors, and maintains the core information of the Spark application (Chambers & Zaharia, 2018). Located in worker nodes, the executors carry out tasks distributed and managed by the driver process and report the results of their processes (Chambers & Zaharia, 2018). A cluster manager (e.g., YARN in Hadoop) plays a role in allocating resources and managing the system (Chambers & Zaharia, 2018).

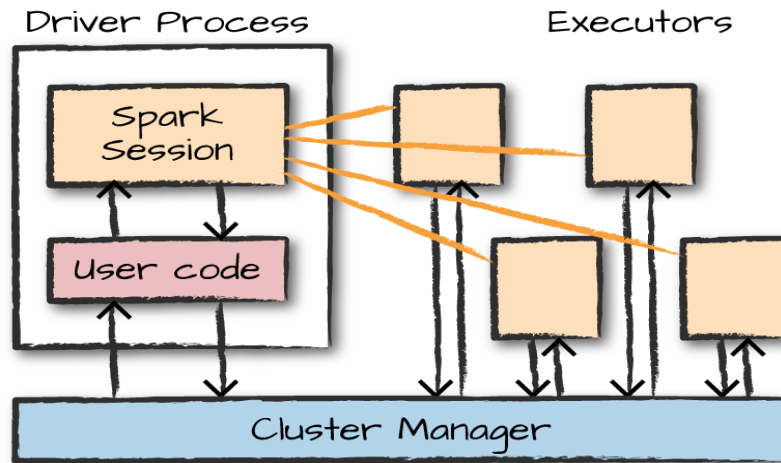
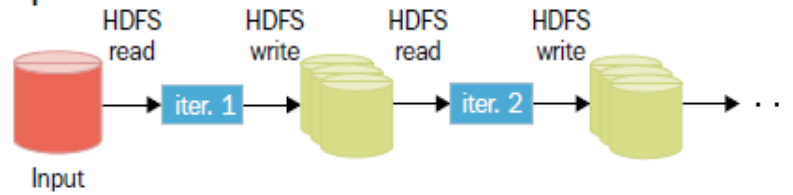


Figure 9. Spark architecture. Source: Chambers and Zaharia (2018).

Spark can process large-scale data rapidly by exploiting the capacity of main memories instead of frequent input and output (I/O) operations, as depicted in Figure 10. Spark shares memory across the cluster and keeps everything in memory as long as possible, thereby avoiding frequent I/O operations. Spark is open source and was developed in UC Berkeley's AMPLab. Spark processes large-scale data faster than MapReduce processing. Therefore, Spark is more suitable to big data analytics with iterative algorithms due to its ability to process intermediate data in the main memory. In addition, Spark can be used with Python, Scala, and Java programming interfaces and can even process real-time streaming data (Achari, 2015).

· Hadoop



· Spark

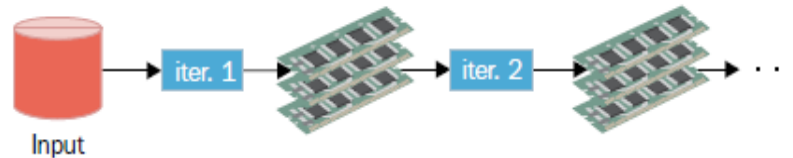


Figure 10. Comparison between Hadoop and Spark. Source: Achari (2015).

Spark's primary data abstraction method, Resilient Distributed Dataset (RDD), processes data in memory on the cluster. (Gulati & Kumar, 2017). RDD is partitioned and stored across the cluster, which allows the cluster to perform parallel computing. RDD ensures the fault tolerance of the cluster by Dynamic Acyclic Graph (DAG), shown in Figure 11, which remembers each stage of data transformation until an action result (e.g., output to a display) commits the data in memory to disk storage (Gulati & Kumar, 2017).

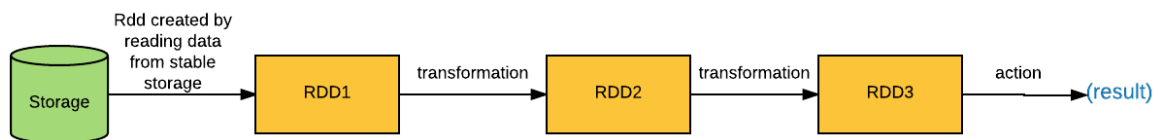


Figure 11. Dynamic Acyclic Graph. Source: Gulati and Kumar (2017).

As mentioned, Spark is better-suited for analytics than MapReduce. Spark also provides a library for analytics called MLlib. One can use advanced analytic tools such as supervised learning, unsupervised learning, recommendation learning, graph analytics and deep learning (Chambers & Zaharia, 2018). Supervised learning produces a model for predicting future values based on historical training data. These models can be used for

classification and regression (Chambers & Zaharia, 2018). Classification predicts discrete values and regression predicts a continuous variable. The recommendation technique provides proper alternatives for users based on their own and similar people's past behaviors. Unlike supervised learning, unsupervised learning does not predict, but finds patterns and relations within a given data set, which can be used for anomaly detection, user segmentation, topic modeling, etc. (Chambers & Zaharia, 2018).

C. EVALUATING ADOPTION OF TECHNOLOGY

As with other business investments, investments in information technology (IT) must be justified. Even though IT has played a key role in the cost reduction of many organization's operations, a number of IT projects have failed. Therefore, many organizations have tried to find out the proper way to evaluate IT projects for successful adoption of new systems. Martens, Walterbusch, and Teuteberg (2012) suggested the total cost of owning cloud computing services in detail. The total cost for adoption of cloud computing is separated by several items such as planning, establishment of system, and maintenance (Martens et al., 2012). Even though the total cost is useful for determining a project's feasibility in a financial perspective, it is not the only business consideration. In addition to the cost, the benefits of adopting new IT systems are necessary to justify proceeding with a project. As a result, cost benefit analysis (CBA) is commonly used for adopting new IT systems (Harrigan, Boyd, Ramsey, Ibbotson, & Bright, 2008).

1. Cost Benefit Analysis

Cost Benefit Analysis (CBA) analyzes the impacts of an investment. It specifies what the impact is, whom it affects, and quantifies the impact through the timeline. Risk analysis using probability and uncertainty is added to CBA. Then, the status quo is compared with this analysis to make a final decision.

The main concept of CBA is how to measure the impact of a dollar. It monetizes benefits and costs by rating how much one is willing to pay for the impact (Arrow et al., 1996). Since benefits and costs are two sides of the same coin, costs are evaluated by how much money is needed to offset negative outputs (Arrow et al., 1996).

Maresova, Sobeslav, and Krejcar (2017) suggested several advantages of using CBA for cloud computing deployment decisions in companies. CBA provides a way to monetize the qualitative values into a sum total benefit and cost, which ensures a way of comparing each alternative (Maresova et al., 2017). According to Maresova et al., it not only provides values from a technology perspective such as the system performance but also suggests diverse impacts related to the operation, size, and geographical position of an organization.

However, CBA also has several constraints. First, CBA easily underestimates the value of the distant future, such as ethical or environmental issues from public goods, since CBA discounts this value by converting it to net present value (Arrow et al., 1996). Second, every dollar has different relative values depending on how much money people have. Finally, if the total dollar value of benefits is bigger than the total dollar value of costs, that alternative is selected. However, the problem of negative effects still exists since benefits cannot always compensate for costs (Arrow et al., 1996).

2. Benefit Cost Analysis

Peppard, Ward, and Daniel (2007) proposed Benefit Cost Analysis (BCA) instead of CBA. They insisted that the least successful cases of IT investment involved CBA with greatly exaggerated benefits: the CBA process focused on the costs, but justified them by the benefits. On the other hand, the most successful cases of IT investment precisely analyzed the benefits. Their costs represented how much they are willing to pay. This method allows people to analyze their business case by using the list of benefits identified through CBA, even after the outset of IT investment, which increases the possibility of success. In addition, these lessons can be applied to the next IT investment decision (Peppard et al., 2007).

III. METHODOLOGY

Scientists and researchers can accumulate a substantial amount of diverse sensor-derived data over years of field experiments. This produces a need to quickly search and extract data files of interest for analysis. In addition, conducting analytics on a large number of datasets may require significant computer processing capabilities to complete these analyses in a practical length of time. This problem motivated this thesis research. To demonstrate the use of metadata and a big data platform to manage, search and analyze a large collection of data files, we utilized a number of side-scan sonar files collected during a one-week field experiment. First, we extracted metadata from MSTIFF files and stored them in a relational database management system (RDBMS) on a standalone computer. The RDBMS utilized SQL to provide a quick file search method. Second, we implemented this process the same way in a Hadoop environment and compared its performance with a standalone system. Third, by applying predictive analytics to this system, we assessed the performance of a Hadoop environment for analytics. Finally, we conducted a benefit cost analysis to determine whether this system would benefit a nominal research team from a business perspective.

A. METADATA EXTRACTION AND STORAGE

In this section, we specify the method to extract metadata from the files and store them in an RDBMS.

1. Extraction of Metadata in Binary Form

Initially, we analyzed metadata contained in the MSTIFF sonar file format, which is similar to the TIFF image file format. Although our study utilized MSTIFF sonar images, other commercial sonar systems may record data files in different, possibly proprietary data formats. Triton Imaging, Inc., for example, developed the XTF image file format (Woods Hole Coastal and Marine Science Center, n.d.). Nevertheless, our methodology can be applied to any image file format that contains metadata.

The MSTL metadata consists of date, time, latitude, and longitude. In addition, we used the file location path to provide quick access to each file. To verify our ability to extract this metadata, we manually analyzed one sonar file using the MSTIFF format description.

This was necessary since we could not use the sonar vendor's sonar-processing software to extract metadata for our needs. Therefore, we first converted the binary data in the file to hexadecimal format, which helped us interpret the metadata fields. Next, we converted the metadata into their correct data type. For example, if a "date" field is described by a *long integer* variable type, we first converted all binary data at the specified file location into long integers and searched for "date" data. Finally, we compared the extracted binary data to the file's known creation date and geographic location.

After validating our file format, we wrote a Java program to automate metadata extraction on the standalone computer. We used Java since it is used in most libraries, reference sources, and most importantly, is also used in Hadoop. This ensured reusability by allowing us to implement code in Hadoop that had already been tested on the standalone computer.

2. RDBMS

After extracting the metadata from each file, it needed to be stored in a database that could be easily searched. Since SQL provides an easy way to search, an RDBMS is used for storing and managing the data from the files. Oracle 11g was selected since it has been commonly used in many organizations, which allows users to access many references (Bihani & Patil, 2014).

For efficient management of the database, the data must be normalized before storing it in the RDBMS (Kroenke & Auer, 2013). To do so, we designed data tables suitable to the normalization rule. Then, we tested several queries to evaluate the performance of searching within the system. In addition to SQL, we used Oracle Application Express (APEX) as a graphic user interface to promote searching performance through enhanced visibility.

B. HANDLING BIG DATA

In this section, we implemented the same program from the standalone system in a big data platform to compare their performance.

The Hadoop ecosystem consists of diverse projects that are still evolving and in active development. The fundamental components of Hadoop, HDFS and MapReduce were the main tools used in this study. In addition, Hive was selected for the study since Hive provides SQL-like query statements. Spark was also used for comparison with MapReduce for big data analytics, since Spark has machine learning libraries and better computing performance than MapReduce.

1. MapReduce

Hadoop processes data using MapReduce, which consists of the mapping and reducing steps shown in Figure 12. First, each mapper processes input data in the form of key and value pairs to generate new key and value pairs. The results from these mappers are distributed and gathered in each reducer according to their keys. Through sorting and shuffling, each reducer gets the key and value pairs it is responsible for, then processes them again to produce new key and value pairs, which comprise the final outputs of MapReduce.

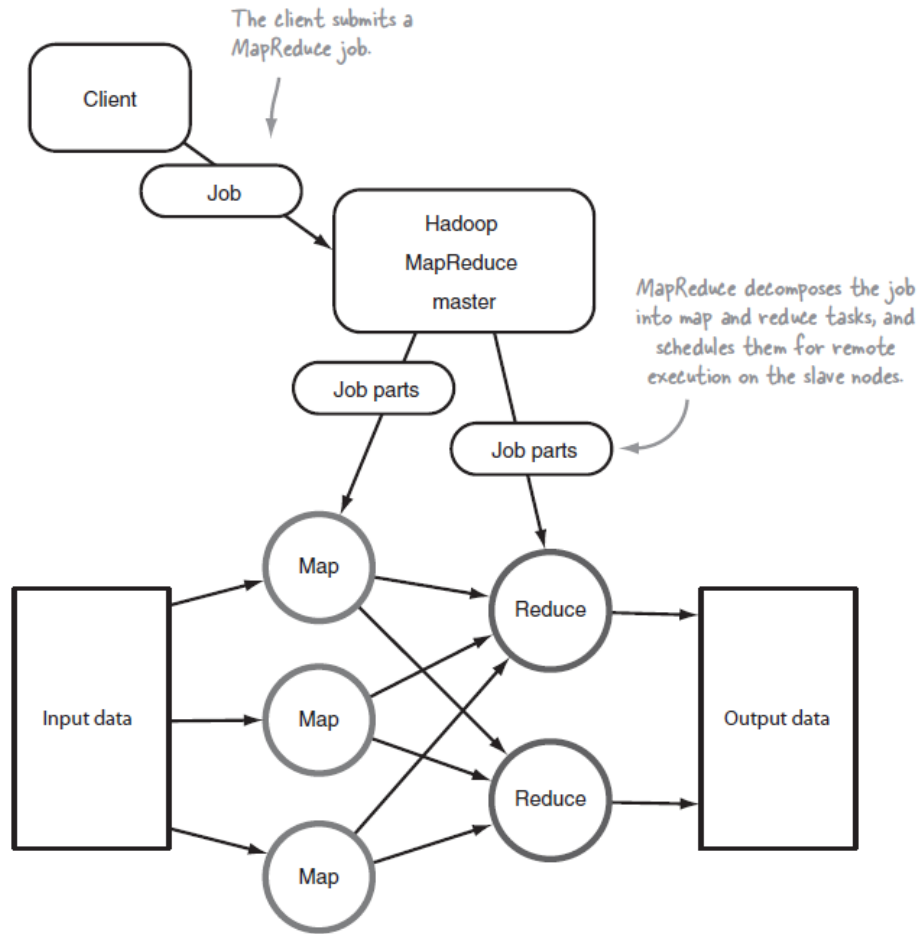


Figure 12. MapReduce implemented by one job. Source: Holmes (2012).

Using SQL-like queries with MapReduce requires substantial effort and customized Java programming knowledge. Therefore, in this study, mapping is used only to extract metadata from the input files among the MapReduce process. Then, after storing this metadata in Hadoop, we compared SQL queries in the RDBMS on a standalone laptop with SQL-like queries in Hive. Later, we tested machine learning algorithms using Spark. As a result, the reducing portion of MapReduce was not necessary for this study.

2. Hive

Hive is one of the high-level data languages used with Hadoop. Since using MapReduce requires Java programming knowledge, it is hard for non-expert analysts to

benefit from Hadoop's capabilities. For this reason, Hive was developed for querying and manipulating data in a big data framework. Users already familiar with SQL can easily access this technology with little additional training, as depicted in Figure 13. In addition, since it needs less code than MapReduce, Hive allows more productivity (Capriolo, Wampler, & Rutherglen, 2012). For example, one can accomplish with five lines of SQL in Hive what would have required more than 200 lines of Java code in MapReduce. In addition, Hive provides many other business intelligence tools to use data in diverse ways.

```
hive> SELECT e.* FROM
> (SELECT name, salary, deductions["Federal Taxes"] as ded,
> salary * (1 - deductions["Federal Taxes"]) as salary_minus_fed_taxes
> FROM employees) e
> WHERE round(e.salary_minus_fed_taxes) > 70000;
```

John Doe	100000.0	0.2	80000.0
Boss Man	200000.0	0.3	140000.0
Fred Finance	150000.0	0.3	105000.0

Figure 13. Example of query and result in Hive. Source: Capriolo et al. (2012).

In this study, the SQL-like queries in Hive were used to compare the search performance between Hadoop and our standalone system. Since the same query could be used in Hive and in the RDBMS on the standalone system, we could make a direct comparison without writing specialized Java code for queries in Hadoop's MapReduce framework.

C. DATA MINING

Data mining is the process of looking for the patterns in data. It typically uses automated processes to examine electronically stored data (Witten, Frank, Hall, & Pal, 2016). Although the dramatic increase of data can hide patterns and make it difficult to understand the relationship between events, data mining can often discover these hidden patterns, leading to new insights for description and prediction.

In this section, we studied the category of data mining techniques and the general process of data mining. Finally, to show the application of data mining techniques to a big data platform, we used the MLP regression model among the techniques with sonar files.

1. Categorization

As depicted in Figure 14, there are diverse techniques in data mining. Data mining is typically divided into the two categories, supervised and unsupervised learning (Weiss & Indurkha, 1998). Supervised learning is a predictive technique that can model a relationship between inputs and outputs like a “white box” allowing users to make predictions about the future (Witten et al., 2016). On the contrary, unsupervised learning is a descriptive technique that does not use any guidance, such as prior theories or discovered relations for learning processes (Martín, Baena, Garach, López, & de Oña, 2014).

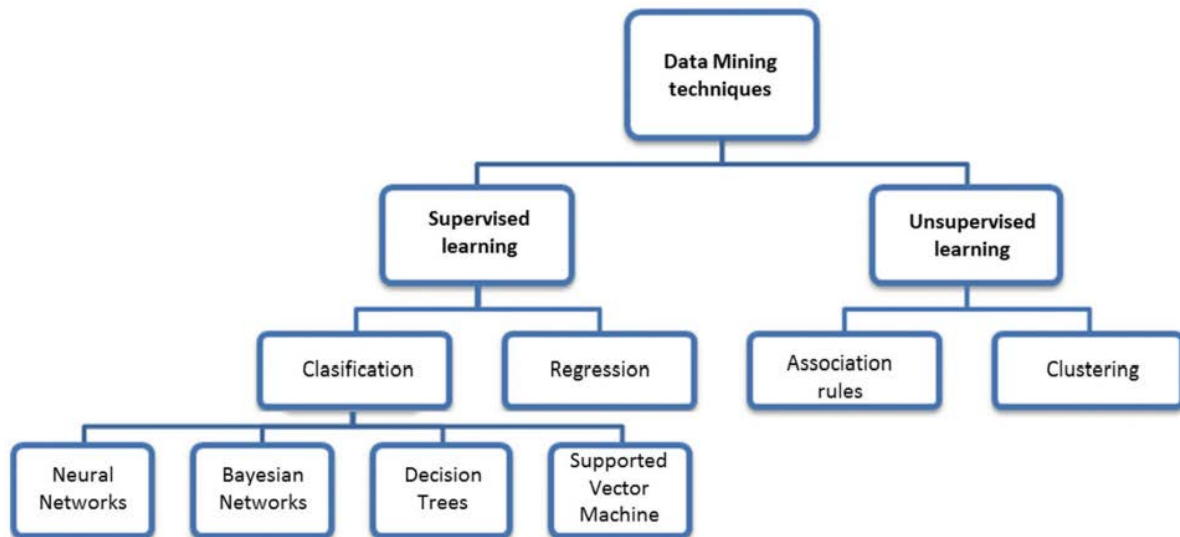


Figure 14. Categorization of data mining. Source: Martín et al. (2014).

Even though the concept of data mining has not changed much over the years, a wealth of data mining tools has been developed since the business community broadly adopted it. Olson (2018) suggested four categories of data mining tools. Two categories,

classification and estimation, are supervised learning techniques. The other two categories, clustering and summarization, are unsupervised learning methods (Olson, 2018).

2. Process

Although the data mining algorithms mentioned above are very powerful, they must be used in conjunction with the other steps of the data mining process. To gain significant knowledge through data mining, the proper data preparation, diverse model testing, and evaluation processes are needed as depicted in Figure 15 (Chambers & Zaharia, 2018).

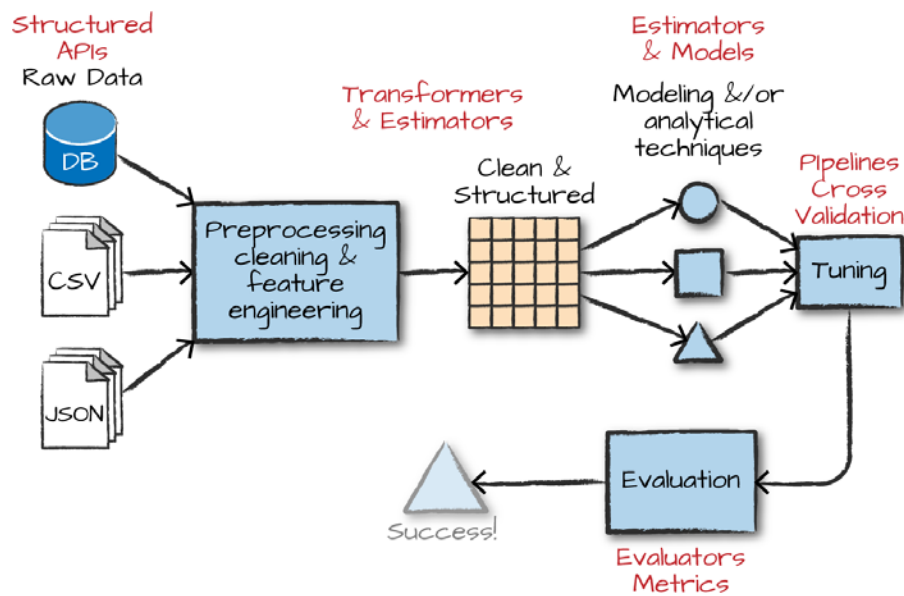


Figure 15. Process of machine learning in Spark. Source: Chambers & Zaharia (2018).

The first step is data collection. Spark has the ability to get data from a variety of data sources. It can deal with comma-separated value (CSV) format files, JavaScript Object Notation (JSON) format files, etc. The next step is data cleaning, which is typically accomplished by exploratory data analysis (EDA) (Chambers & Zaharia, 2018). EDA allows users to remove noise in the data by providing a better understanding of the data's characteristics (e.g., its distribution) via making iterative queries (Chambers & Zaharia, 2018). This "clean" data then undergoes feature engineering, transforming data into

variables which can be used by machine learning algorithms (Chambers & Zaharia, 2018). After feature engineering, the machine learning algorithm train models to predict correct output values by tuning the weighting parameters applied to the input data. (Chambers & Zaharia, 2018). Finally, the model is tested against a dataset different from the training dataset to determine whether it produces desirable output data for previously unseen input data. This is the goal of data mining (Chambers & Zaharia, 2018). Due to the iterative methods used for data preparation and feature engineering, Spark can drastically reduce the amount of time required for machine learning. In addition, it provides distributed computing processes with ease (Chambers & Zaharia, 2018).

Spark provides several functions to support this process through machine learning libraries (MLlib). These functions include transformers, estimators, evaluators, and pipelines (Chambers & Zaharia, 2018). Figure 15 illustrates where each function plays a role in the machine learning process. Transformers change raw data into usable data in MLlib, such as double type numerical data used to represent vectors since Spark uses this type of numerical data (Chambers & Zaharia, 2018). Each estimator serves as a type of starter process for training; estimators provide initial values and use a model started across the entire dataset (Chambers & Zaharia, 2018). Pipelines link the transformers and estimators to automatically generate a tuned model (Chambers & Zaharia, 2018). Finally, evaluators assess how well the trained model performs against data to determine the best model with optimized parameters (Chambers & Zaharia, 2018).

Even though there are a number of different machine learning libraries for Java including Weka, DL4J and ND4L, Spark automatically performs the entire process with MLlib. Almost by definition, users cannot manually analyze big data if the characteristics of big data are considered. In addition, since the data mining process can be iterative, it is possible to return to the data collection process whenever the model needs modified data. This would be exceedingly time consuming if performed manually (Han, Pei, & Kamber, 2011). Therefore, users should be mindful of the problem to be solved and the desired outcome of the machine learning process.

In this study, we compared the performance between Hadoop and Spark for conducting big data analytics on our sonar dataset to predict seawater depth based on historical metadata.

3. Case for This Study

By applying the data mining techniques to our sonar file dataset, the goal of this thesis is to demonstrate the effectiveness and efficiency of a big data platform for this task. In one test case for this study, a regression technique was used on water depth metadata to predict the water depth at other locations not directly represented in the dataset. Before implementing machine learning, we first computed the average depth of each location.

To make a bathymetric map, we matched the depth data to the latitude and longitude coordinates in the metadata, since the sonar data files recorded water depth at each location. However, since the depth of a given location varies with time due to tidal fluctuations, time must also be considered. The relation between time and depth of one location in the sea is expected to have a sinusoidal profile similar to the graph in Figure 16.

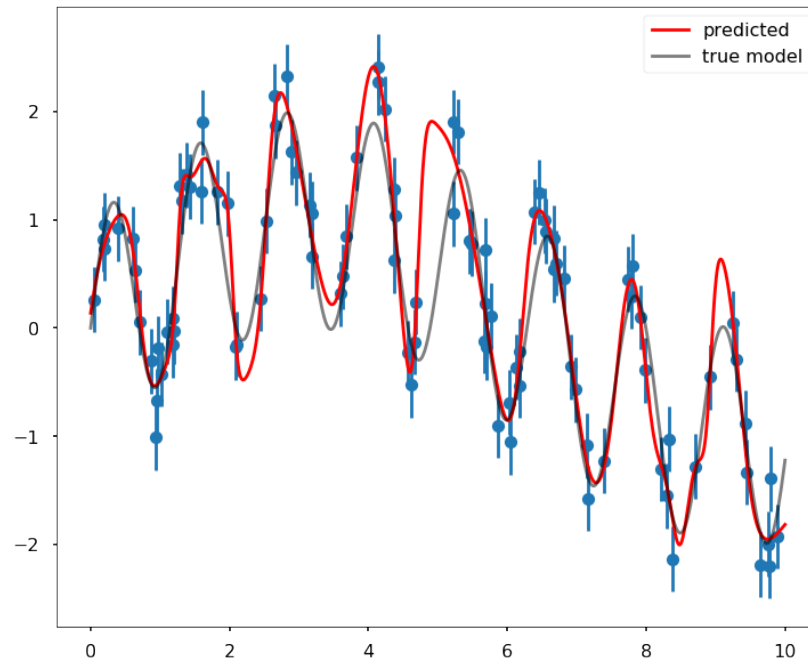


Figure 16. The example of a sine wave. Source: Kong (2017).

Even though we could have simply computed the average depth at each location over the entire time period, we instead applied a multilayer perceptron (MLP) regression method in an Artificial Neural Network (ANN) to predict the form of the sine wave's data (Capilla, 2014).

a. Multilayer Perceptron Network

A neural network is a computing system loosely modeled on the human brain's learning process. As with a human neuron that receives input signals and, if activated, passes it through dendrites, an artificial neuron receives input data and passes it to neurons in the next layer when it is activated (Bulut, 2018). To activate one neuron, diverse inputs from other connected neurons must exceed a given threshold (Bulut, 2018). Figure 17 depicts four inputs, labeled X1 through X4, each weighted by a value labeled W1 through W4, respectively. These inputs enter a single neuron with a summation and activation function (Bulut, 2018). Each input's weighted value has a different degree of impact on the neuron (Bulut, 2018). In the neuron, all inputs are summed and compared against a threshold value, which determines whether the neuron is activated and produces an output (Bulut, 2018).

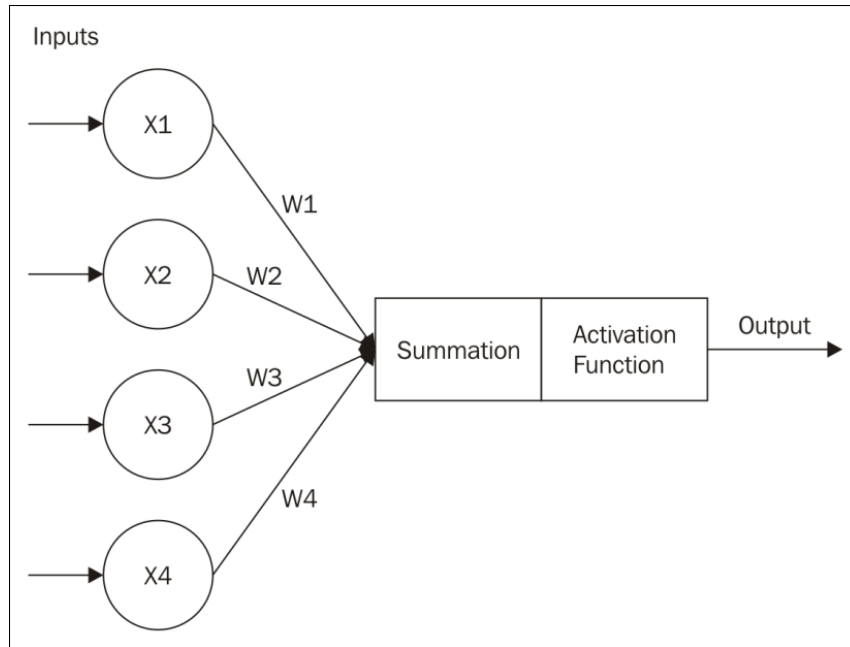


Figure 17. Activation function. Source: Reese, Reese, and Grigorev (2017).

MLP networks consist of a series of layers as shown in Figure 18. These layers consist of an input layer, one or more hidden layers and an output layer. Interactions between the neurons in each layer of an MLP network are similar to those in neural networks described above. Each input enters neurons in each hidden layer. New inputs generated from hidden layer activations proceed to the next hidden layer (if there are more than one) or enter the output layer to activate the neurons inside. When an input moves into another layer, it contributes its weighted value specifying its degree of impact on the neuron. In addition to this forward flow, the difference between the predicted output and actual output is calculated during the learning process. If there is a discrepancy, MLP learning modifies the weights of each neuron to minimize this difference. This is called backpropagation in the learning process (Reese et al., 2017). Unlike clustering, which does not use target variables, training examples consist of predictors and target variables. Predictors can be both categorical and continuous while target variables should be categorical in the classification method (Martín et al., 2014).

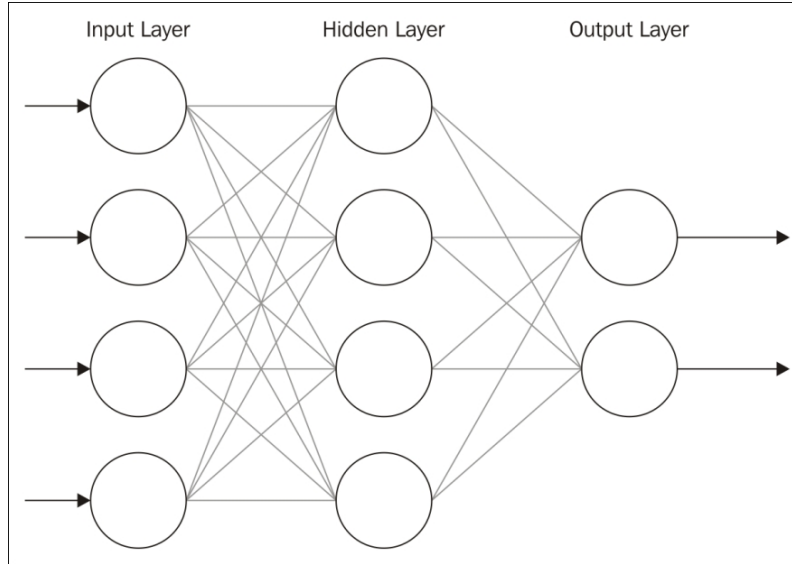


Figure 18. The process of multilayer perceptron. Source: Reese et al. (2017).

In this study, we used MLP regression to train and predict the depth depending on the time for each location. Since each location has its own depth and the depth is independent to the depth of each location, we sorted the data by location and separately made an MLP regression for each location. We used chen004's neural networks library for the standalone system and Hadoop system.

D. BENEFIT COST ANALYSIS

Benefit cost analysis was used for this study rather than cost benefit analysis for two reasons. First, CBA focuses on costs first, then attempts to justify them by using benefits, which can lead to an insufficient analysis about the benefits (Peppard et al., 2007). Second, benefit cost analysis prevents implicit value from being distorted through monetization in CBA.

1. Components of Benefits

Before analyzing the benefits of IT investment, the main objective of the intended investment should first be established. Our main objective for establishing a big data platform for underwater survey research is to provide a quick and easy query system to

search and locate desired files among a large number of historical sonar datasets. This would save time and improve upon current analytic capabilities.

Figure 19 describes a method of classifying benefits by their degree of explicitness and the intended type of business change.

		Type of Business Change		
		Innovation (Do new things)	Improvement (Do things better)	Cessation (Stop doing things)
High ↑ degree of explicitness ↓ Low	Financial Benefits	Financial value can be calculated by applying a cost/price or other valid financial formula to a quantifiable benefit.		
	Quantifiable Benefits	There is sufficient evidence to forecast how much improvement/benefit should result from the changes.		
	Measurable Benefits	Although this aspect of performance is currently measured, or an approximate measure could be implemented, it is not possible to estimate how much performance will improve when changes are implemented.		
	Observable Benefits	By using agreed criteria, specific individuals or groups will use their experience or judgment to decide the extent the benefit will be realized.		

Figure 19. Classification framework for benefits. Source: Ward, Daniel, and Peppard (2007).

a. Type of Business Change

Desired business changes can be broken down into three main types: innovation, improvement, and cessation. Innovation brings new capabilities for doing things that were not possible before the investment. Improvement is focused on doing things better than previously possible, and cessation seeks to stop doing things that are no longer necessary as a result of the investment (Ward et al., 2007). While some benefits such as increasing customer numbers are apparent, there are also some ambiguous benefits. For example, establishing virtual or augmented systems for training employees could motivate employees to socialize and share information in the system. The benefit of sharing

information between employees is an example of “doing new things”. However, the investment may not always achieve the expected result. Therefore, methods for achieving the desired benefits should also be considered (Peppard et al., 2007).

b. Degree of Explicitness

Benefits can also be categorized by their degree of explicitness, such as observable, measurable, quantifiable, and financial benefits (listed in order from least explicit to most explicit). Observable benefits are subjective and qualitative values, and these may even include the opinions of benefit owners or surveys (Ward et al., 2007). Measurable benefits are more explicit than observable benefits since measurable benefits have identified a performance metric for comparison. However, post-investment performance benefits cannot be accurately estimated beforehand (Ward et al., 2007). Quantifiable benefits, however, are easily measured both before and after the investment. For example, the benefits of stopping an operation can be measured by assessing the costs required to continue that operation. For other business changes, benchmarking and pilot tests can be used to determine quantifiable benefits (Ward et al., 2007). Finally, financial benefits assign financial values to quantifiable benefits.

2. Costs and Risk

The total cost of an IT investment consists of both the IT and organizational costs (Ward et al., 2007). IT costs typically include initial and recurring costs. Initial costs are the cost of purchasing the system, hiring new people, etc. Recurring costs are license fees, telecommunications, electricity, maintenance costs, etc. While IT costs are easily recognized, organizational costs caused by business changes arising from an IT investment are difficult to estimate (Ward et al., 2007).

After the total costs are determined, the net benefits of an IT investment can be calculated using financial benefits and costs. However, since expected benefits may not be achieved, the probability of achieving each benefit should be considered using risk assessment in terms of their technical, financial, and organizational risks (Ward et al., 2007).

In this study, we utilized the performance difference between a RDBMS on a standalone system and a big data platform to specify the benefits and costs of our investment. In addition, the ability to apply a machine learning application provided an additional benefit, as this represents a new capability. Since we used existing computing systems for our performance comparison, the IT costs of establishing these systems were not considered as initial costs.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION

Experiments in this research consist of three parts: 1) loading data onto a standalone laptop computer; 2) transferring this data to a Hadoop cluster; and 3) applying machine learning techniques on the distributed Hadoop data. In the first step, the metadata of sonar was extracted from a large number of sonar files and visualized using Oracle Application Express (APEX). In the second step, the sonar data from the NPS laptop was transferred to a Hadoop cluster where the metadata was extracted as in the first step. In the final step, the metadata from the second step was used for two machine learning applications.

A. LOADING DATA IN A STANDALONE LAPTOP

A side-scan sonar dataset was obtained for this experiment. These files were stored using the Marine Sonic Technology, Ltd. Image data file format (MSTIFF) (Marine Sonic Technology, Ltd., 2011). The total volume of this data is 680 MB, is comprised of 577 individual sonar image files distributed in ten subdirectories, and organized by creation date. The size of each file is 1.2 MB. Using the MSTIFF specification document, a Java program was developed to extract the metadata from the sonar image files used in this experiment.

Metadata contains a significant amount of information about the file. Since it can be provided by using the Java program we made without executing the specific software to get the information from the file, extracting the metadata was much faster and efficient. The objective is to build a database from these sonar files so that users can search for files of interest based on stored metadata (i.e., date/time, location, depth). In this chapter, we describe the Java program, written with Java SE Development Kit version 1.8.0_201, used to extract metadata from our dataset. Next, we designed a Relational Database Management System (RDBMS) using Oracle database 11g and Oracle SQL Developer. Finally, we utilized the Oracle APEX web application development tool to develop a map-based graphical user interface for better visualization of location data.

1. Extracting Metadata of .mst

The data used in this thesis was stored in an image data file format developed by Marine Sonic Technology, Ltd. (MST). This format is used within the Sea Scan PC side scan sonar system. The location, or address within the file for each metadata field such as date, location and depth is specified in the file format description. When these data were extracted in Java, the binary data was in little endian format.

a. Description of a Sonar File

According to MST (2011), the structure of MSTIFF files consists of a header, image file directory (IFD), and directory entry. Figure 20 shows the structure of an MSTIFF file. The company explained that a header contains information about the offset for the IFD's entry count. This offset value is stored in the 4 bytes immediately following the 4-byte "MSTL" MSTIFF identifier. It said IFD entry count indicates the number of concatenated IFDs in the sonar file. It described each IFD entry is described by a sequence of fields including its tag, type, count and value offset. The company explained that a "tag" uniquely identifies the data field contained in the IFD entry, and "type" describes that field's data type, whether a single byte, short/long integer, or C-style structure. It also explained that "count" specifies the number of data items, whereas the "value offset" specifies the file location where the field value is found (Marine Sonic Technology, Ltd., 2011).

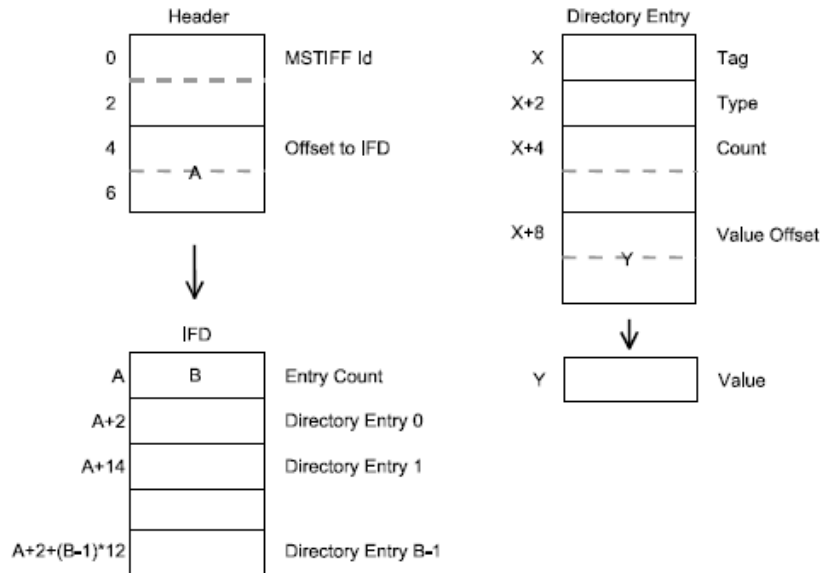


Figure 20. MSTIFF structure. Source: Marine Sonic Technology, Ltd. (2011).

b. Searching the MSTIFF Image File Directory

The IFD's offset is located immediately following the 4-byte "MSTL" MSTIFF identifier. The Java code in Figure 21 is used to extract this data and obtain the set of IFDs in each sonar file. It extracts the offset bytes and fills in the IFD_Address variable.

```
// Extract IFD_Address //
byte[] IFD_Address_tmp = Arrays.copyOfRange(bytes, 4,8);
int IFD_Address = 0;
int i=0;
int shiftBy = 0;
for(shiftBy = 0; shiftBy < 32; shiftBy += 8){
    IFD_Address |= ( (long)( IFD_Address_tmp[i] & 0xff ) ) << shiftBy;
    i++;
}
```

Figure 21. Java code for getting IFD offset

Since the MSTIFF file format distinguishes each tag with a hexadecimal value, the Hex converter code in Figure 22 was used to identify each tag.

```
// Convert Byte to Hex class //
private final static String[] hexSymbols =
{"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};
public final static int BITS_PER_HEX_DIGIT = 4;

public static String toHexFromByte(final byte b){
    byte leftSymbol = (byte)((b >>> BITS_PER_HEX_DIGIT) & 0x0f);
    byte rightSymbol = (byte)(b & 0x0f);
    return(hexSymbols[leftSymbol]+hexSymbols[rightSymbol]);
}
```

Figure 22. Hex converter

The value in the IFD's offset includes the number of concatenated IFDs which consist of tag, type, count and value offset. The metadata of interest in this study consists of time, location and depth, which are only included in the Y2KtimeCorrelation, NavInfo6, and Fathometer2 data tags. For example, Figure 23 shows the Java code for decoding the offset of the Fathometer2 field, which contains information about the water depth where the sonar image was generated. The IFD_HEX variable stores hexadecimal values for the bytes which include this IFD information. The hexadecimal value "2801" is the tag identifier for the Fathometer2 field, and the corresponding file offset is stored in Fatho_Address.

```
for(i= 0; i<(int)file.length()-IFD_Address;i=i+2){
    tag = IFD_HEX[i]+IFD_HEX[i+1];

    if(tag.equals("2801")) {
        Fathometer2_Address = IFD_HEX[i+11] + IFD_HEX[i+10] + IFD_HEX[i+9] + IFD_HEX[i+8];
        Fatho_Address = Integer.parseInt(Fathometer2_Address, 16);
        // System.out.println("Fathometer2's address is " + Fathometer2_Address); //
    }
}
```

Figure 23. Finding Fathometer code

The actual sonar metadata is located at the offsets indicated in each IFD entry. The detailed code for extracting the metadata is provided in Appendix A.

c. *Converting the Data*

The data extracted from the file must now be converted from its specialized storage format into conventional data formats. For example, date and time information is stored as Julian Time and expressed as the number of seconds since midnight. The Julian Time is converted to time in HH:MM:SS format as follows:

$$\text{Hour} = \text{Julian time mod } 3600$$

$$\text{Minute} = (\text{Julian time mod } 3600) / 60$$

$$\text{Second} = \text{Julian time mod } 60$$

The location where the sonar data was collected is stored as latitude and longitude. Since these data are expressed as decimal minute values, they are converted into degrees and decimal minute values by using the formulas:

$$\text{Degrees} = (\text{int}) \text{DecimalMinutes} / 60$$

$$\text{Minutes} = \text{DecimalMinutes} - (\text{Degrees} * 60)$$

In addition to the Julian time format described above, both the Fathometer2 and Nav_Info6 fields contain a timestamp based on the sonar computer's system time. These timestamps can be converted into time-of-day by adding the start time of the system to each timestamp as shown in Figure 24.

```
for(int j=0;j<FathometerCount;j++){  
    Fatho_timestamp_sec[j] = Fatho_timestamp[j]/1000 + start_time_sec;    // Fatho_timestamp : millisecond -> second //  
}
```

Figure 24. Converting timestamp into the real time

To make it easy to search for the files, the path of the files should be extracted as metadata. Each file is stored in the directory named for the sonar file creation date. The code for extracting this metadata is listed in Appendix B. Since the sonar files were stored in subdirectories under a project directory, a double loop was used in the code.

The major methods for extracting metadata, which were also used in Hadoop MapReduce, are briefly described in Table 1.

Table 1. Description of methods used for extracting the metadata from sonar files

Method Name	Return Type	Method Description
toByteFromHex(String s)	Byte array	Converts string type of hexadecimal data into binary data and returns it
toHexFromByte(byte b)	String	Converts binary data into string type of hexadecimal data and returns it
FloatToDegree(float Float)	String	Returns a string after converting location data in decimal minutes floating point format
JulianToTime(long Long)	String	Returns a string of time data after converting Julian time in long integer format
Get_Metadata(String args)	String	Loads the data files in binary format, then searches for date, time, latitude, longitude, and depth data and aggregates them in one string variable upon return
GetFileList()	String array	Generate and return the file path list in string array format
DBUpdate()	Void	Accesses the Oracle JDBC to create a table and inserts the metadata extracted by the Get_Metadata() and GetFileList() methods

2. Loading Data into RDBMS

The above metadata were combined and transferred to Oracle database 11g. The code for combining the metadata is listed in Appendix C. We created a table named “WHOLE_DATA” for storing this metadata in eight columns, including the sonar file paths, date, time, latitude, longitude and depth data.

3. Visualizing the Data in APEX

After loading the data into Oracle database 11g, the metadata could now be displayed in the SQL Workshop page of APEX (see Figure 25). The ability to use APEX is one of the benefits of using Oracle Database. APEX provides a distinguishable graphical user interface (GUI), which allows users to access data in Oracle Database, and also provides users with an Application Builder to develop customized application interfaces connected to the data (Oracle, n.d.-c).

The screenshot shows the Oracle APEX SQL Workshop interface. The browser address bar indicates the URL: localhost:8080/apex/f?p=4500:1001:15003990341009::: The top navigation bar includes tabs for App Builder, SQL Workshop (selected), Team Development, and App Gallery. The Object Browser on the left lists various database objects, with 'WHOLE_DATA' highlighted. The main panel displays the 'WHOLE_DATA' table structure with the following columns:

Column Name	Data Type	Nullable	Default	Primary Key
DATA_ID	NUMBER	No	-	-
DIRECT_DATA	VARCHAR2(50)	Yes	-	-
DATE_DATA	VARCHAR2(20)	Yes	-	-
TIME_DATA	VARCHAR2(20)	Yes	-	-
LAT_DATA	VARCHAR2(20)	Yes	-	-
LONG_DATA	VARCHAR2(20)	Yes	-	-
DEPTH_DATA	VARCHAR2(20)	Yes	-	-
DEPTH_DATA2	VARCHAR2(20)	Yes	-	-

Below the table structure, there is a 'Download' link. The footer of the page includes the text: Copyright © 1999, 2018, Oracle. All rights reserved. and Application Express 18.2.0.00.1.

Figure 25. APEX workshop page

We built an application to show the data on a map using a Google Maps API. We created this application with only a single page, the Home page, to visualize our data on a Google Map. The page in this application consists of Pre-Rendering, Regions, and Post-Rendering components, as shown in the left side of Figure 26. Regions play the role of a container for the contents of the application. We developed hypertext markup language (HTML) for the source code portion of the Regions component, similar to developing a webpage.

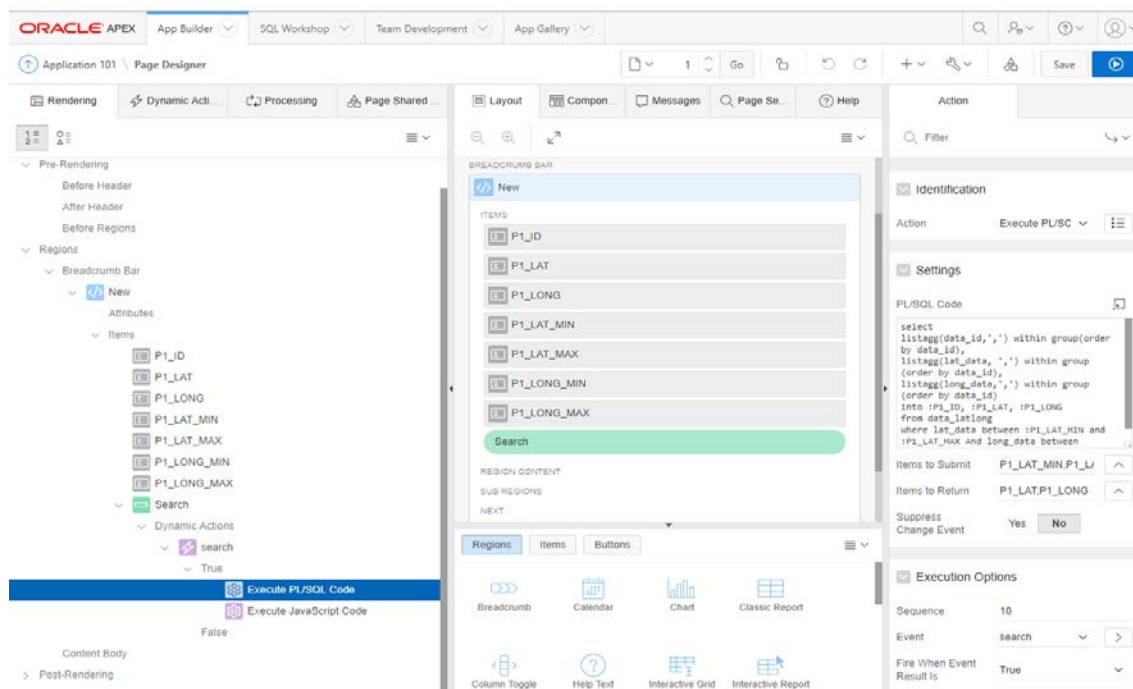


Figure 26. APEX application editor page

Specifically, under Regions, we created items whose names matched the names of the columns in Oracle Database, such as 'P1_ID', 'P1_LAT', 'P1_LONG', including the range limits of the location values. The 'P1' prefix is automatically used by APEX to distinguish the same variable names in other applications. For example, the name of items in a second application would begin with a 'P2' prefix. We also added a 'Search' button to the interface so that users could search for data files of interest. This required adding 'Dynamic actions' under the 'Search' button, and putting the PL/SQL code to display the

requested data in 'P1_ID', 'P1_LAT' and 'P1_LONG'. We also added JavaScript code to plot marks on the map related to the requested location. Figure 26 illustrates the structure of regions, items and buttons. The PL/SQL and JavaScript code for this application are listed in Appendix D.

B. LOADING DATA INTO THE HADOOP CLUSTER

This step utilized Grace, a computer cluster designed for big data research at NPS. A Hadoop distributed file system (HDFS) and Spark in-memory processing system are both installed in the cluster. The version used for this thesis was Hadoop 2.6.0-cdh5.-13.-1. We implemented a MapReduce job for extracting metadata from the files in the HDFS using the same source code used in the standalone system. Since MapReduce in the Hadoop cluster uses the Java programming language, we could recycle the Java code used in the standalone system, saving time and effort. The extracted metadata was imported into Hive to provide easy query functionality. In addition, Hue, a browser-based interface for the cluster, provided access to the HDFS, Hive, and a job tracker.

1. MapReduce

The MapReduce job for extracting the metadata consists of a mapper, reducer, combined files reader, and a main class (see Chapter 3. Section B-1). In general, MapReduce jobs can be written in several different programming languages. The MapReduce codes for this thesis were written in Java since the functions developed for extracting metadata from sonar files were written in this language.

As with Figure 7, determining the input file is the first step for a MapReduce job. Although TextInputFormat, the default input format class, is not able to ingest binary data directly from files, we could read the data by modifying the input format class. Additionally, since the default configuration of MapReduce cannot read files in subdirectories, the addinputpath function was used multiple times by using a loop clause in the main class. Specifically, the FileSystem object's listStatus class shows the list of the paths provided as an input variable. In the loop clause, if the record of the list is a directory, the record is transferred to the addinputpath function. The Job class (consisting of MapReduce) is created in the driver code of the main class, which also sets its parameters.

The driver code in the Main class also sets the format of the output file, as well as setting for the mapper, reducer, and input/output key value classes. We initiated the job and set its configuration as depicted in Figure 27. Note that the GET_MST_Mapper, GET_MST_Reducer, and CombineDocumentFileFormat classes were mainly used for extracting the metadata of the sonar file stored on the Hadoop Cluster.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "GET_MST");
    job.getConfiguration().setLong("mapreduce.input.fileinputformat.split.maxsize", (long) 128*1024*1024);
    job.setJarByClass(GET_MST.class);
    job.setMapperClass(GET_MST_Mapper.class);
    job.setReducerClass(GET_MST_Reducer.class);
    job.setInputFormatClass(CombineDocumentFileFormat.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    Path path = new Path(args[0]);
    FileSystem fs = path.getFileSystem(conf);
    FileStatus[] status = fs.listStatus(path);
    for(FileStatus stat : status) {
        if(stat.isDirectory()) {
            FileInputFormat.addInputPath(job, new Path(stat.getPath().toString()));
        }
    }
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Figure 27. Hadoop MapReduce job configuration in main class

a. *CombineDocumentFileFormat Class*

The input file format class plays a role in determining which format would be accepted and how to split data. As mentioned above, text is the default file format for input in Hadoop. However, the sonar data files used in this thesis are stored in a proprietary binary format. As a result, we coded a customized input file format class, expanded from a WholeFileInputFormat class. This class uses NullWritable as the keys, which does not need to get real keys for the class. The values for this class are binary data, or BytesWritable instances (White, 2015).

However, the WholeFileInputFormat class reads all files in the directory, and each file works as one split, which means each file has one map process. As a result, the number of map processes increase as the number of files increase. Each sonar file's size is relatively small, only 1 to 2 MB, compared to Hadoop's default block size which is 128 MB.

Therefore, it is inefficient to process each file by its own map process because it leads to excessive I/O operations.

To solve this problem, we used CombineFileInputFormat which integrates small file sizes into a block size, usually 128 MB in default. This prevents the number of mapper jobs from increasing to match the number of individual sonar files.

The CombineFileInputFormat class cannot read files in subdirectories. It needs additional code to specify the paths of subdirectories and add those paths via the FileInputFormat.addInputPath class shown in Figure 27.

After the CombineFileInputFormat class packed the sonar files in 128 MB blocks, the blocks were passed to the WholeFileRecordReader class. This split the block into individual records to make key and value pairs, which are the input of the mapper process. As shown in Figure 28, the nextKeyValue() method in the WholeFileRecordReader class separately divided the block, treating each file as a record and storing the path of each file in a path variable. This Text object was called by the getCurrentkey() method, and the binary data of the file was stored in a document variable. This BytesWritable object was called by the getCurrentValue() method in the WholeFileRecordReader class.

```
@Override
public boolean nextKeyValue() throws IOException, InterruptedException {
    if (!read) {
        Path file = inputSplit.getPath(idx);
        FileSystem fs = file.getFileSystem(conf);
        FSDataInputStream input = fs.open(file);
        byte[] bytes = new byte[(int) inputSplit.getLength(idx)];
        int offset = (int) inputSplit.getOffset(idx);
        int length = (int) inputSplit.getLength(idx);
        IOUtils.readFully(input, bytes, offset, length);
        document = new BytesWritable();
        document.set(bytes, offset, length);
        path = new Text(file.toString());
        read = true;
        return true;
    }
}
```

Figure 28. nextKeyValue() method in WholeFileRecordReader class

b. Mapper and Reducer Classes

The mapper is the core of extracting the sonar file metadata in this thesis. The methods used for extraction in section A were reused. The input key of the mapper is the

path of the file, which was used to recognize whether the file is a sonar file by using its .mst extension since files having other extensions were also combined into the block in the WholeFileInputFormat class. The if clause in Figure 29 ensures that only .mst files will be processed by the mapper. The input value of the mapper was the binary data of the file using BytesWritable format in the mapper. The BytesWritable variable was converted and passed into an array of bytes as shown in Figure 29.

```
// Mapper //
public static class GET_MST_Mapper extends Mapper<Text, BytesWritable, Text, Text> {
    public void map(Text key, BytesWritable value, Context context) throws IOException, InterruptedException {
        // Get key from filePath and distinguish acceptable file(only mst) //
        String filePathString = key.toString();
        if(filePathString.matches(".*mst")) {
            // get Value as binary //
            byte[] bytes = value.getBytes();
```

Figure 29. Input key and value in Mapper class

While Java code was needed to generate the file paths of the sonar files on the standalone system, this was not necessary for the HDFS since the path of the file was the input key of the mapper. However, since the input key had already been converted into String format to distinguish mst files using the filePathString variable, this variable was used to generate the output key in Text format via the context.write method. The output value of the mapper was integrated in a String format variable and passed into the context.write method to generate a Text format variable.

The reducer, meanwhile, received the output key and value from the mapper and simply passed them. Processing was not required, since the MapReduce job in this step was used only for extracting the metadata of sonar files.

Figure 30 demonstrates the flow of the combined input file, map, shuffle/sort and reduce processes corresponding to the code of Appendix E. Since our main focus was extracting the metadata, the shuffle/sort and reduce processes did not impact the result in this case. Even though only about 500 files were used for this test case, actual sonar database applications could conceivably need to process files of several orders of magnitude, which would lead to many more key and value pairs.

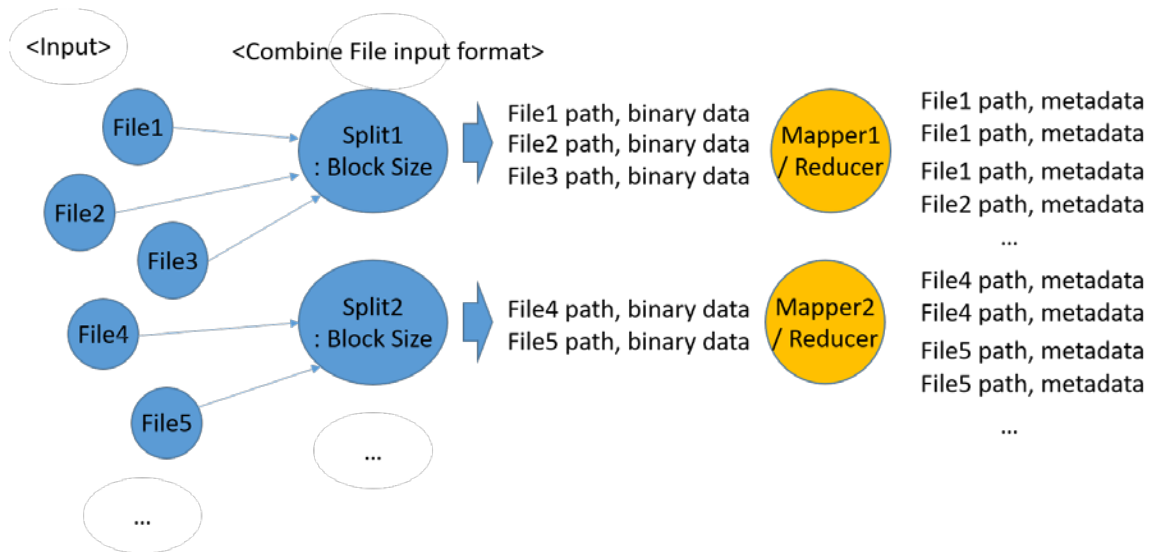


Figure 30. MapReduce flow to extract metadata of sonar files

c. *Hive Loading*

Just as the metadata from the standalone system was inserted into Oracle Database 11g for easier query usage, the metadata extracted by Hadoop MapReduce was loaded into Hive. Hive provides users with a familiar user interface via a SQL-like language called HiveQL.

Hive can handle diverse formats for data storage and search. Even though the sonar files were not directly loaded by Hive, the metadata processed by the Hadoop MapReduce and the raw files could have been. However, to study and compare each step on each system, the metadata output of Hadoop MapReduce from the previous step served as the Hive input file for this thesis.

Figure 31 shows the command used to load the data from the output path of Hadoop MapReduce: `"/output/mst1/."` The create table command is similar to the command used in the Oracle SQL. When the "external" specifier is not used with this command, Hive moves the table data from its original directory to Hive's warehouse directory. Therefore, since the metadata will be used in subsequent analytics (see next section), the "create external table" command was used as shown in Figure 30.

```

create external table metadata (
    Path string,
    Date string,
    Time string,
    Navinfo6 array<struct<timestamp : string,
                        Latitude : string,
                        Longitude : string>>,
    Fathometer array<struct<timestamp : string,
                        Water_Depth : string,
                        Towfish_Depth : string>>)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
    COLLECTION ITEMS TERMINATED BY ','
    MAP KEYS TERMINATED BY ' '
LOCATION '/output/mst1/p*';

```

Figure 31. Command for loading the metadata from the output of Hadoop MapReduce and creating external table

As with the Oracle database 11g RDBMS, Hive's queries operate on this table. We used SELECT command to search for a file's geographic location, creation date, etc.

Hive also has a function to transfer its data to other RDBMS such as Oracle Database, MySQL database, etc. Thus, APEX could be used to provide better visualization by Google Maps through Oracle Database 11g, as described in the previous section for the standalone system.

C. PREDICTING OCEAN DEPTH AT A LOCATION

Within our dataset extracted from the sonar files, predicting the ocean depth at a certain time and location was implemented by using Multilayer Perceptron (MLP) regression. Even though there may be several factors affecting the depth of the seabed at a given location, we applied an MLP regression to predict the depth of the seabed using only time and depth variables. This analytics example was used to compare the performance of a Hadoop cluster and Spark. For this thesis, we assumed that the water depth at each location is independent with respect to every other location. Therefore, an MLP regression must be performed at every location. As a result, after dividing the metadata with a composite key comprised of the latitude and longitude pair in the mapper, each reducer performed an MLP regression with data corresponding to its composite key. Figure 32 shows the process utilized for depth prediction.

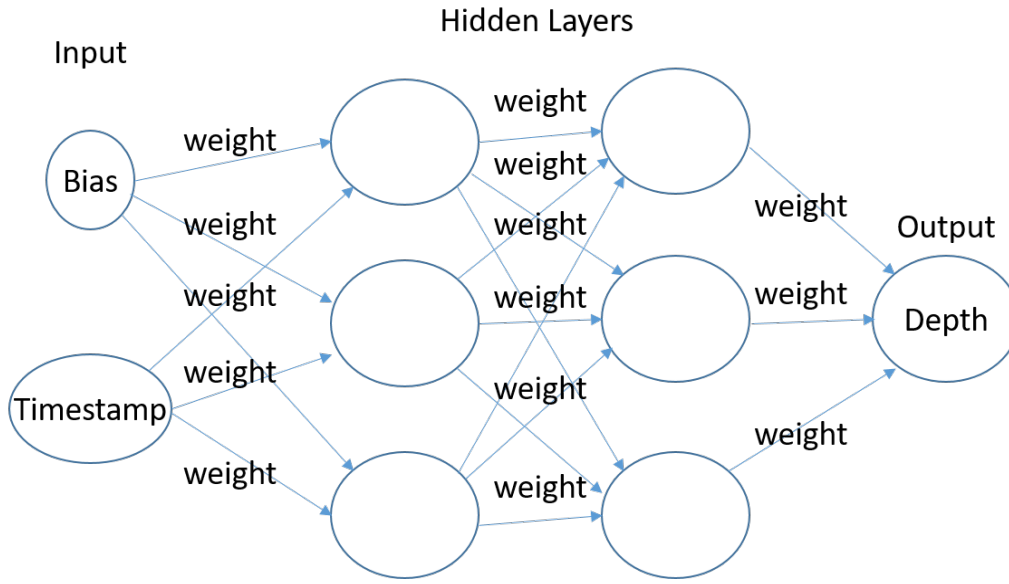


Figure 32. Process of predicting the depth of seabed

In this thesis, we used the java-ann-mlp package available on GitHub (X. Chen, 2017).

1. Hadoop Cluster

In this section, a Multilayer Perceptron (MLP) regression for predicting ocean depth was implemented on the Hadoop cluster. As with extracting the metadata, this process consists of a main class, mapper, and reducer. In addition, the StringPair class was used to group data by composite key, which includes latitude and longitude, to gather data by geographic location. Since we assume that water depth in each location is independent from the depth in other locations, the data should be divided according to location and then used to train a location-specific model. The main class created the Job and specified its configuration, while the mapper sorted data according to StringPair key and the reducer used an MLP Regression to train the model.

a. Main Class

The Main class plays the role of driver, including job and configuration settings for the format of input and output key/value pairs, mapper, reducer, input and output paths,

etc. The default WritableComparable interface sorts the keys passed from the mapper. However, since we used a composite output key for the mapper (i.e., a location consists of its latitude and longitude variables), the output key of the mapper had to be specified with a shuffle and sort process. Figure 33 illustrates how the Job class is configured to use StringPair_Partitioner and StringPair_Comparator as its partitioner and sort comparator classes, respectively. These two classes are specified in the next section.

Since we did not implement an MLP regression during our Java development, we utilized an open source Java implementation of this algorithm, available from GitHub. To use this algorithm, however, we needed to add a class path for its class file, which is a Java Archive (.jar) file. The DistributedCache.addFileToClassPath method provides functionality for-adding class path in a Hadoop cluster. Figure 33 lists the five .jar files used for MLP regression in this thesis.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    // add external jars //
    DistributedCache.addFileToClassPath(new Path("MLP_Jar/java-ann-mlp-1.0.1.jar"),

    DistributedCache.addFileToClassPath(new Path("MLP_Jar/java-data-frame-1.0.9.jar"),

    DistributedCache.addFileToClassPath(new Path("MLP_Jar/log4j-1.2.17.jar"), conf);
    DistributedCache.addFileToClassPath(new Path("MLP_Jar/slf4j-api-1.7.20.jar"), conf);
    DistributedCache.addFileToClassPath(new Path("MLP_Jar/slf4j-log4j12-1.7.20.jar"),

    Job job = Job.getInstance(conf, "MLP_Test");
    job.setJarByClass(MLP_Test.class);
    job.setMapperClass(MLP_Mapper.class);
    job.setPartitionerClass(StringPair_Partitioner.class);
    job.setSortComparatorClass(StringPair_Comparator.class);
    job.setReducerClass(MLP_Reducer.class);
    job.setMapOutputKeyClass(StringPair.class);
    job.setMapOutputValueClass(Text.class);
    job.setOutputKeyClass(StringPair.class);
    job.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Figure 33. Main class of multilayer perceptron regression

b. StringPair Class

The default WritableComparable interface is used for comparison of writable variables such as DoubleWritable, FloatWritable, NullWritable, BytesWritable, etc. (Achari, 2015). In this thesis, since we used a custom output key with two variables (latitude and longitude) for the mapper, the composite key was specified in the name of the StringPair class used in this thesis. The StringPair class consists of reading, writing, printing, and comparing methods similar to the default WritableComparable interface.

The big difference from a default WritableComparable interface is that the StringPair class uses two variables as one composite key, it compares the keys that have two variables to shuffle and sort. Figure 34 shows that the compareTo method initially compares the latitude variable of each key upon a match, it then compares the longitude variable of each key.

```
public int compareTo(StringPair pop) {  
    if (pop == null) return 0;  
    int intcnt = latitude.compareTo(pop.latitude);  
    return intcnt != 0 ? longitude.compareTo(pop.longitude) : intcnt;  
}
```

Figure 34. compareTo method of StringPair class

Since the output key of the mapper was not WritableComparable, it was necessary to register a comparator for the key class. Thus, the StringPair_Comparator method was used in this thesis (White, 2015).

The StringPair_Partitioner class set the partition according to the first field of the key, latitude. Then, the StringPair_Comparator performed sorting operations using the compare method shown in Figure 35.

```

public int compare(WritableComparable w1, WritableComparable w2){
    StringPair k1 = (StringPair) w1;
    StringPair k2 = (StringPair) w2;
    int cmp = k1.getLatitude().compareTo(k2.getLatitude());
    if(cmp !=0){
        return cmp;
    } else{
        return k1.getLongitude().compareTo(k2.getLongitude());
    }
}

```

Figure 35. compare method in Stringpair_Comparator class

c. *MapReduce Classes*

The mapper read the metadata extracted in the previous step as an input by using the LongWritable interface for keys and the Text interface for values. Among several columns of metadata, the latitude, longitude, timestamp and water depth data were processed and transferred to the reducer. The output key was the StringPair form of location data and the output value was the Text form of the timestamp and water depth data.

The reducer is the core component for this step. We used the open source java_ann_mlp project posted in Github (Chen, 2017). This reducer consists of transforming data, setting the configuration, training the model, and finally evaluating the model.

First, the DataFrame instance can contain data such as a comma-separated variable (.csv) file. It also can acquire data via direct input, one row at a time, using the BasicDataFrame() class. The latter method was used for this thesis since data from the mapper used the sequenceFile writable format, which differs from a .csv file. The input value of the reducer had timestamp and water depth data in comma-delimited string format. The row.setCell method passed the timestamp contained in val[1] into a predicting factor, while the row.setTargetCell method set the depth data contained in val[2] as the target variable. The row containing val[1] and val[2] was added into the frame. This process was repeated until the values variable obtained values, then the lock function fixed the DataFrame instance to prevent further modification or addition of rows during training.

```

...
DataFrame frame = new BasicDataFrame();
for(Text itr : values) {
    DataRow row = frame.newRow();
    String val[] = itr.toString().split(",");
    row.setCell("time",Double.valueOf(val[1]));
    row.setTargetCell("depth",Double.valueOf(val[2]));
    frame.addRow(row);
}
frame.lock();

```

Figure 36. DataFrame in multilayer perceptron regression

Figure 37 shows the configuration part in which a TupleTwo instance divides the dataset into a training dataset and an evaluation dataset. In this thesis, we utilized 90% of the data for training, and the rest for evaluation. Then, we instantiated the MLP regression class, which required setting the number of hidden layers and epochs. A hidden layer has neurons whose output is connected to the inputs of other neurons (see CH3. C-3). In the setHiddenLayers method, the number of arguments represents the number of hidden layers to use, while the value of each argument specifies the number of neurons comprising that layer. One epoch means that the neural network model is trained by an entire dataset at one time (Sharma, 2017). Specifying a high number of training epochs can lead to an excessive computational burden and overfitting of the model. In this thesis, we utilized three hidden layers with eight neurons each, and the number of epochs was set to 1,000. Additionally, the batch size used for MiniBatchGradientDescend training set to 30, which means the model was trained to use 30 input records at a time.

```

TupleTwo<DataFrame, DataFrame> miniFrames = frame.shuffle().split(0.9);
DataFrame trainingData = miniFrames._1();
DataFrame crossValidationData = miniFrames._2();
MLPRegression regression = new MLPRegression();
regression.setHiddenLayers(8,8,8);
regression.setEpochs(1000);
regression.setWeightUpdateMode(WeightUpdateMode.MinibatchGradientDescend);
regression.setMiniBatchSize(30);
regression.setLearningRate(0.5);

```

Figure 37. Configuration for multilayer perceptron regression

Figure 38 shows the fit method used to train the MLP regression model. We can check the change in the accuracy of the prediction by comparing the predicted variable and actual variable in this stage. The transformation method of the MLP regression class is used to evaluate the trained model against the evaluation dataset which had been set aside in advance. The output key of the reducer was the same as the input key for the reducer: a String Pair format for a location specified by latitude and longitude. The output value of the reducer was a Text variable, which showed the comparison between the predicted depth and actual depth of the evaluation data set.

```

if(trainingData.rowCount() > 1) {
    regression.fit(trainingData);
    for (int i = 0; i < trainingData.rowCount(); ++i) {
        double predicted = regression.transform(trainingData.row(i));
        double actual = trainingData.row(i).target();
    }
    if(crossValidationData.rowCount() > 1) {
        String[] cmp = new String[crossValidationData.rowCount()];
        for (int i = 0; i < crossValidationData.rowCount(); ++i) {
            double predicted = regression.transform(crossValidationData.row(i));
            double actual = crossValidationData.row(i).target();
            cmp[i] = "predicted: " + predicted + "\t actual: " + actual;
            context.write(key, new Text(cmp[i]));
        }
    }
}

```

Figure 38. Training and evaluating the MLP model

2. Spark

To implement the same MLP regression algorithm in Spark as the Java implementation we used in Hadoop, we used the Eclipse integrated development environment (IDE) to generate a .jar file. Specifically, we used the Apache Maven Project in the Eclipse IDE to create the .jar file. This .jar file was then submitted to the Spark drive and the MLP algorithm in the .jar file was executed across the cluster.

Figure 39 shows the process used to conduct MLP regression in Spark. The same data as used in Hadoop was used as input data. This data was distributed across the cluster by the MapReduce process described in the previous section. Spark then read the input as Resilient Distributed Datasets (RDD). Next, the input RDD was transformed to a Pair RDD

whose key consists of latitude and longitude and whose value consists of timestamp and depth. The Pair RDD was grouped by its key and stored in each partition according to the key. As a result, each partition of grouped RDD has a configured number of keys and a bundle of values compatible to each key. Next, the MLP Regression was executed in each partition. Finally, the result of the MLP Regression model stored as a text file through action. The detailed code is provided in Appendix F.

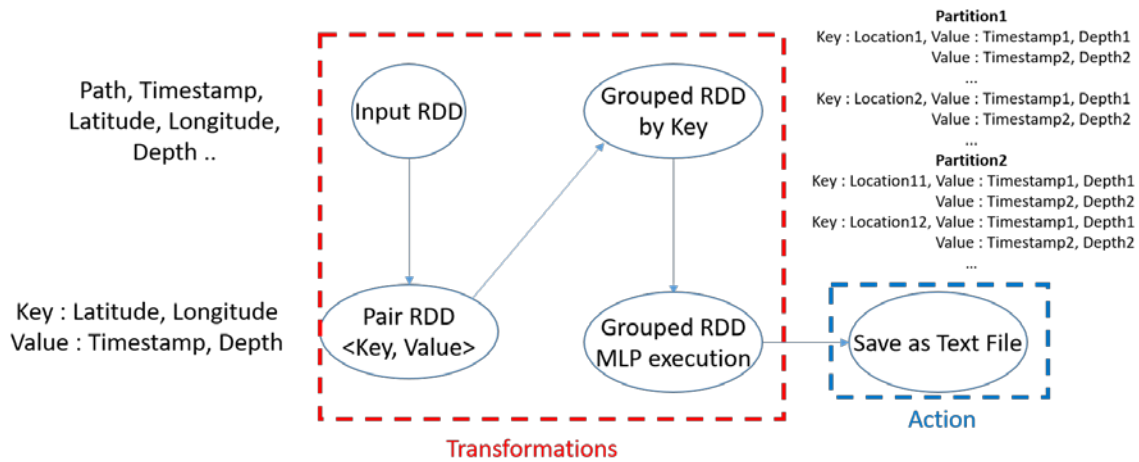


Figure 39. Dynamic Acyclic Graph

D. TEST WITH BIG DATA

A small subset of sample data (approximately 500 MB) was used to develop the code. This sample data was multiplied by two thousand to study the performance of the standalone system comprised of Hadoop and Spark. Extracting the metadata was tested only in the Hadoop cluster since the time required for extracting metadata in the standalone system is directly proportional to input size. In addition, we compared the performance between Hadoop and Spark for implementing the machine learning algorithm, which needs significant computer processing capability to perform large-scale iterative calculations. The number of input files was over 1 million.

Initial attempts using the same code for extracting metadata produced “java.lang.OutOfMemoryError: GC overhead limit exceeded” errors since the Java Virtual

Machine (JVM) tried to prevent OutOfMemoryError in advance. Likewise, we determined that the job for reading more than 1 million files required a larger heap memory size. Therefore, we specified an additional Hadoop client option as shown below:

```
export HADOOP_CLIENT_OPTS="-XX:-UseGCOverheadLimit"
```

The “-XX:-UseGCOverheadlimit” option disables the function that throws an excessive garbage collection (GC) time error when running applications take too much time. This option allows the nodes to load 1 million files.

The MLP regression model based on a larger dataset (simulated by replicating our sample data 2,000 times) required much more time to train than a model based on our sample data alone. The default MapReduce time limit of 6,000,000 milliseconds did not allow enough time to train the model, so we multiplied this time limit by a thousand, using the configuration option of the Main class:

```
conf.set("mapred.task.timeout","6000000000");
```

Similarly, just as more time was needed, much more memory was requested to train the model faster. Since the training process was executed in the reducers, we allocated more memory for this function by setting additional configuration options for the Main class of MapReduce:

```
conf.set("mapreduce.reduce.memory.mb","16384");
```

```
conf.set("mapred.child.java.opts","-Xmx14336m");
```

The mapreduce.reduce.memory.mb configuration controls the size of the physical memory allocation for each Hadoop node. Likewise, the mapred.child.java.opts configuration sets the heap memory allocation for the java process of each mapper and reducer, which is also a java process. That is why the value of mapred.child.java.opts cannot exceed the value of mapreduce.reduce.memory.mb, which allocates the physical memory to Hadoop.

V. RESULTS

This study focused on two objectives: 1) to assess how easily users can retrieve desired information (e.g., data files) by using metadata; and 2) to assess the benefits of using Hadoop and Spark for big data analytics.

To address the first objective, we utilized an example query for searching certain files together with a map application in APEX. To address the second objective, we measured and compared the time required to train an MLP model with each process in each system.

A. EASE OF USE FOR SEARCH

After extracting the metadata and storing it into an Oracle Database, a relational database management system (RDBMS) on a standalone computer, we used an SQL developer to submit a query. Our query example was:

```
'select * from whole_data where lat_data > 24 and long_data > -80.452 and  
date_data>20140727;'
```

Figure 40 shows the result of this query. Since the directory information of the metadata points to the logical location of the file, we can identify the files that match our search criteria by using SQL. Additionally, the user could utilize time and location data to narrow their search to specific files of interest.

DATA_ID	DIRECT_DATA	DATE_DATA	TIME_DATA	LAT_DATA	LONG_DATA	DEPTH_DATA1	DEPTH_DATA2
1	d:\project\140728_3\RDMS234.mat	20140728	17:32:59	24.949532	-80.45188	31.890001	4.94
2	136263.d\project\140728_3\RDMS236.mat	20140728	17:34:13	24.949419	-80.4519	32.41	7.23
3	136299.d\project\140728_3\RDMS236.mat	20140728	17:42:39	24.94938	-80.4519	32.51	7.13
4	136349.d\project\140728_3\RDMS237.mat	20140728	17:50:34	24.9493	-80.45199	32.16	7.18
5	136363.d\project\140728_3\RDMS237.mat	20140728	17:52:51	24.949402	-80.45197	31.419998	7.2
6	136364.d\project\140728_3\RDMS237.mat	20140728	17:54:0	24.949404	-80.45198	32.32	7.26
7	136365.d\project\140728_3\RDMS237.mat	20140728	17:54:19	24.94938	-80.45195	32.4	6.17
8	136400.d\project\140728_3\RDMS237.mat	20140728	17:42:31	24.9493	-80.45197	32.39	7.28
9	136401.d\project\140728_3\RDMS237.mat	20140728	17:42:44	24.949467	-80.45192	32.33	6.84
10	136529.d\project\140728_3\RDMS236.mat	20140728	17:50:34	24.9493	-80.45199	32.16	7.18
11	136414.d\project\140728_3\RDMS237.mat	20140728	17:45:47	24.94932	-80.45199	31.52	5.67
12	136415.d\project\140728_3\RDMS237.mat	20140728	17:46:1	24.94955	-80.45185	31.990002	6.96
13	136416.d\project\140728_3\RDMS237.mat	20140728	17:46:15	24.949467	-80.45185	32.39	6.63
14	136417.d\project\140728_3\RDMS237.mat	20140728	17:46:29	24.94938	-80.45199	32.299999	5.44
15	136451.d\project\140728_3\RDMS236.mat	20140728	17:50:38	24.94938	-80.45197	32.9	7.1
16	136455.d\project\140728_3\RDMS236.mat	20140728	17:53:55	24.94956	-80.451935	31.92	7.08
17	136466.d\project\140728_3\RDMS236.mat	20140728	17:54:19	24.949451	-80.451866	32.34	7.46
18	136467.d\project\140728_3\RDMS236.mat	20140728	17:54:23	24.949348	-80.45199	32.35	5.59
19	136502.d\project\140728_3\RDMS236.mat	20140728	17:42:34	24.949333	-80.451938	32.39	7.16
20	136503.d\project\140728_3\RDMS236.mat	20140728	17:42:49	24.9495	-80.45195	32.24	6.48
21	136516.d\project\140728_3\RDMS236.mat	20140728	17:45:41	24.94945	-80.451938	31.49	6.38
22	136517.d\project\140728_3\RDMS236.mat	20140728	17:46:15	24.949459	-80.45181	32.35	6.56
23	136518.d\project\140728_3\RDMS236.mat	20140728	17:46:19	24.949434	-80.45188	32.45	6.19
24	136553.d\project\140728_3\RDMS239.mat	20140728	17:50:43	24.949419	-80.45199	31.220001	5.8
25	136567.d\project\140728_3\RDMS239.mat	20140728	17:54:0	24.949516	-80.45166	32.04	7.05

Figure 40. Result of searching for files in a certain location in SQL Developer

In addition, the Oracle Database is able to connect to APEX, which provides several applications for using graphs, charts, maps, etc. For this study, we used the Google Map API in APEX to offer better visualization of data files corresponding to a geographic location (Figure 41).

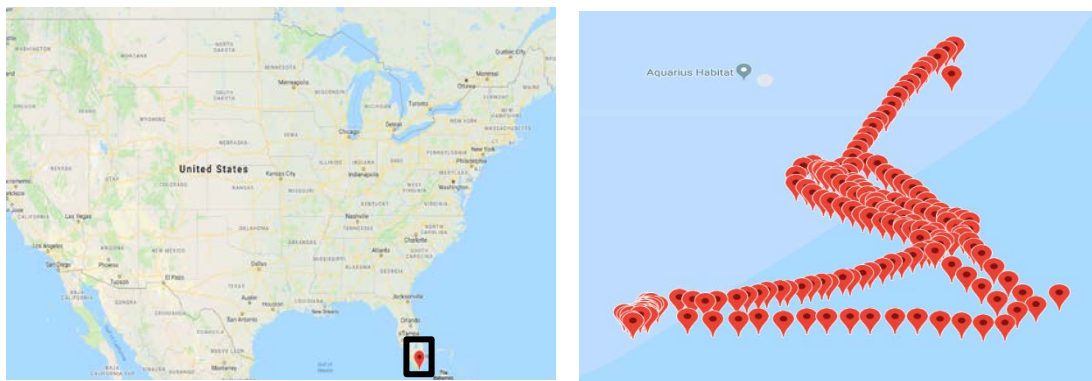
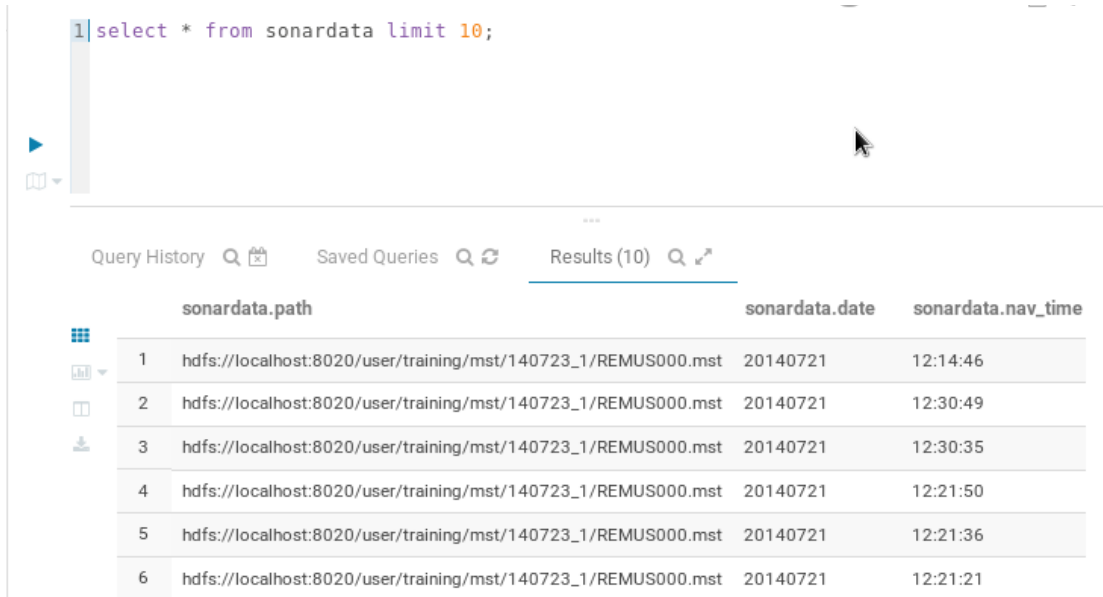


Figure 41. Result of searching data of certain date in APEX

For the Hadoop cluster, we also submitted the query to Hive, which produced the same output as the Oracle Database on the standalone system (Figure 42).



The screenshot shows a Hive query interface. At the top, a SQL query is entered in a text box: `1 select * from sonardata limit 10;`. Below the query box, there are tabs for 'Query History', 'Saved Queries', and 'Results (10)'. The 'Results (10)' tab is selected, displaying a table with three columns: 'sonardata.path', 'sonardata.date', and 'sonardata.nav_time'. The table contains 10 rows of data, but only the first 6 rows are visible in the screenshot. Each row has an index number in the first column, followed by the file path, the date, and the navigation time.

	sonardata.path	sonardata.date	sonardata.nav_time
1	hdfs://localhost:8020/user/training/mst/140723_1/REMUS000.mst	20140721	12:14:46
2	hdfs://localhost:8020/user/training/mst/140723_1/REMUS000.mst	20140721	12:30:49
3	hdfs://localhost:8020/user/training/mst/140723_1/REMUS000.mst	20140721	12:30:35
4	hdfs://localhost:8020/user/training/mst/140723_1/REMUS000.mst	20140721	12:21:50
5	hdfs://localhost:8020/user/training/mst/140723_1/REMUS000.mst	20140721	12:21:36
6	hdfs://localhost:8020/user/training/mst/140723_1/REMUS000.mst	20140721	12:21:21

Figure 42. Result of searching for files in a certain location in Hive

B. PERFORMANCE BETWEEN SYSTEMS

To compare the performance between systems, we implemented metadata extraction and an MLP regression with the sample data as well as increased data since as we use more data, a big data platform would show its performance more clearly.

1. Sample Data

For the standalone system, a Lenovo Ideapad 330 was used to measure the time to extract the metadata and execute the MLP regression. The time required to extract metadata from 641 input files (752 MB total) was 32 seconds (Figure 43).

```
C:\Classes\practice>java DBUpdate
Driver loaded
Database Connect ok
Table created ok
Rows inserted ok
Execution Time : 31.499773217

C:\Classes\practice>_
```

Figure 43. Implementation of extracting the metadata from sample files

For the big data platform, we used Grace, a Hadoop Cluster at NPS (Naval Postgraduate School, n.d.), to implement the metadata extraction and MLP regression. Grace has 22 worker nodes that have two 6-core Xeon processors and a 512 GB memory. Grace's default number of map/reduce tasks is set to 288, since Grace originally had 24 worker nodes that consisted of 288 cores (24 cores/node x 12 nodes in cluster).

Using the Hadoop cluster, the job of extracting metadata took 41 seconds without using the CombineFileInputFormat class. When utilizing this class, however, the Hadoop cluster completed this job in only 28 seconds, about 32% faster, since it sets the block size to 128 MB, reducing the number of blocks. The 128 MB block size required six map tasks ($752 \text{ MB} / 128 \text{ MB} \approx 6$). The time required for processing in Hadoop is affected by its configuration, such as the number of map/reduce tasks in MapReduce and the number of input files and their sizes, as well as the physical capability of the hardware (White, 2015).

To perform an MLP regression with the Hadoop cluster, we experimented using different numbers of reduce tasks to identify the optimum number for our problem because analytics is coded in the reduce task. Each experiment used 288 input files (6.6 MB) derived from the previous extraction process, and performed an MLP regression using a number of reduce tasks arranged from 100 to 1,000. The results in Figure 44 illustrate that the job with 200 reduce tasks completed an MLP regression on our dataset in the least amount of time (47 seconds). It is interesting to note that this number of reduce tasks is close to the actual number of Hadoop nodes available, multiplied by the number of cores per node ($22 \times 12 = 262$).

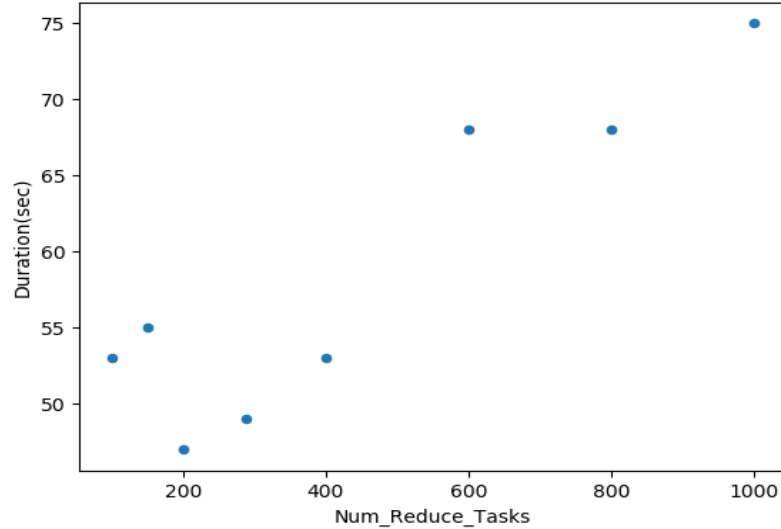


Figure 44. Time required for an MLP regression depending on the number of reduce tasks

The performance of Spark depends on each node's CPU processing speed and memory size (Chambers & Zaharia, 2018). The time required for an MLP regression is not only affected by the number of partitions, but also depends on the number of executors, the number of cores in each executor, and the memory allocated to each executor (Chambers & Zaharia, 2018). After the number of tasks are determined by the number of partitions in the previous stage's resilient distributed dataset (RDD), these tasks are executed concurrently by the number of executors and their cores (Chambers & Zaharia, 2018). First, we set the number of executors, the number of cores per executor, and the memory allocated for each executor as recommended, but also experimented with different figures as shown in Table 2 (Cloudera, n.d.-a). For example, when the number of cores per executor is set at two, one core per node is set aside for the Hadoop Daemon, and one executor is used for the Spark Driver, leaving 120 executors available in total. Finally, the memory for each executor was set at 79 GB (memory for each node / number of executors per node x each executor's ratio of heap overhead: $512 \text{ GB} / 6 \times 0.93 = 79 \text{ GB}$). Since the default memory for an executor is set at 128 GB, and the memory requirements exceed this default setting for more than 2 cores per executor, the memory per executor for these

experiments was set at 116 GB (128 GB x 0.93). Our experiments determined that Spark performed an MLP regression fastest (24 seconds) when the number of executors, the number of cores per executor, and the memory allocated to each executor were set to 15, 15, and 116 GB, respectively.

Table 2. Time required for an MLP regression with sample data in Spark

Cores per Executor	1	2	3	4	5	6	7	8	9	10
Total Executors	241	121	80	60	48	40	34	30	26	24
Memory per executor (GB)	47	79	116	116	116	116	116	116	116	116
Duration (sec)	63	63	56	53	50	49	48	48	46	47

Cores per Executor	11	12	13	14	15	16	17	18	19	20
Total Executors	22	20	18	17	16	15	14	13	12	12
Memory per executor (GB)	116	116	116	116	116	116	116	116	116	116
Duration (sec)	46	46	25	25	24	26	31	31	32	31

Next, using the optimum configuration settings above, we repeated the experiment with different numbers of partitions to determine the optimum number of partitions for MLP regression. These results are shown in Figure 45. Initially, an MLP regression executes faster as the number of partitions increase, up to a value of 200 partitions, which required only 24 seconds to complete. Using more than 200 partitions, however, does not yield faster performance. In fact, using too many partitions (e.g., more than 1,700) actually resulted in a slower execution.

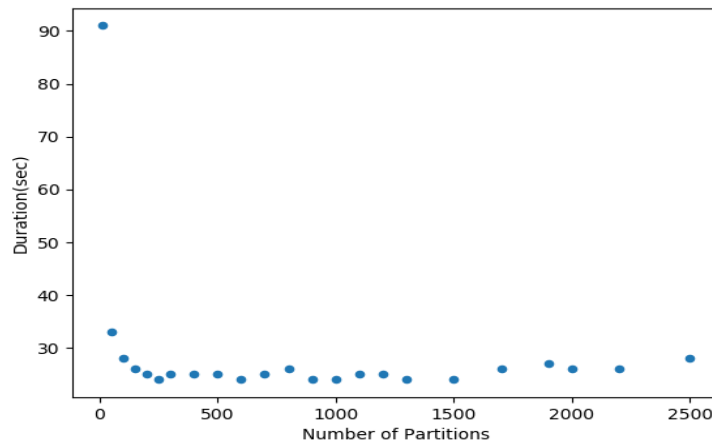


Figure 45. Time required depending on the number of partitions in Spark

2. Increased Data

It was not feasible to store 1 TB of data files on an Oracle Database. Therefore, test results for increased data were obtained using big data platforms only (e.g., Hadoop and Spark).

Extracting metadata from 1,279,967 files (1.42 TB) with MapReduce took 10 minutes and 8 seconds. The size of the resulting output files was 11.63 GB. Implementing an MLP regression on these 11.63 GB input files took 17 hours, 43 minutes, and 59 seconds. The Spark implementation, however, performed an MLP regression much faster, requiring only 4 hours and 50 minutes (a 73% improvement).

3. Summary

The experimental results are summarized in Table 3. With our sample dataset, metadata extraction in Hadoop was slightly faster than on the standalone system (a 12.5% improvement). When executing an MLP regression, however, Hadoop was much faster than the standalone system (a 92.3% improvement). Spark was the fastest solution we tested for performing an MLP regression. Therefore, we concluded that a big data platform is better for conducting analytics, even for a relatively small sample dataset. With increased data, Hadoop only took 10 minutes for the extraction. Assuming that the time required for

metadata extraction on the standalone system is proportional to the size of the dataset, we estimate that the standalone system would take 1,000 times longer than Hadoop when using increased data. Therefore, we conclude that Hadoop is more effective than a standalone system for extracting metadata from larger datasets. Finally, Spark proved to be the best solution we tested for analytics, achieving the fastest execution speed for the MLP regression.

Table 3. Time required for each experiment

Data	Experiment	Standalone	Hadoop	Spark
Sample Data (700 MB)	Extraction	32 sec	28 sec	-
	MLP	10 min 8 sec	47 sec	24 sec
Increased Data (1.4 TB)	Extraction	-	10 min 8 sec	-
	MLP	-	17 hours 44 min	4 hours 50 min

VI. CONCLUSION

Our experimental results highlight some of the advantages of big data platforms. For an organization deciding whether to adopt this technology, however, it is not enough to simply justify all the anticipated implementation costs. Thus, we used benefit cost analysis (BCA) to help determine whether an NPS research team should utilize a big data platform, based on expected benefits.

A. BENEFITS

The classification framework in Figure 19 was useful to analyze the benefits of this case. When it comes to innovation (doing new things), one main benefit is that a big data platform can provide the capacity to perform analytics with statistical methodologies on historical sonar data for diverse purposes, such as predictions and prescriptions. Moreover, an analytic tool developed by the user can be executed on all of the relevant data files stored across the cluster. Following this, the big data platform can assemble the analysis outputs for the user. It also provides analytic tools, such as machine learning libraries and graphical tools, to support different applications. Even though these benefits are apparent, they are categorized as observable benefits, since it is difficult to measure or predict the amount of cost benefits.

In terms of improvement (doing things better), there are two main benefits. First, big data platforms would ensure safe, redundant data storage. In the current system, data files are stored on a number of different computers, some of which are not backed up routinely. However, big data platforms provide data safety via auto-replication of data files across multiple nodes. In addition, since the cluster is being managed by Information Technology and Communications Services (ITACS) at NPS, individual users would benefit from enterprise-level security services (anti-virus, anti-malware, automatic backups. etc.) without additional, duplicative effort and resources. This benefit can be measured by counting the number of accidents that occurred under the current system that could have been prevented by using the Grace cluster. If the cost for recovering from each

accident can be monetized, this cost would represent the financial benefit of using Grace. This is a quantifiable benefit, even though we could not monetize this specific example.

Second, the current system of storing different files on several different computers does not promote usability. Retrieving files of interest is time-consuming and inefficient. A big data platform would allow users to search for these files using SQL, making this process faster and easier than before. This benefit can be measured by the amount of time saved in finding a given file.

The last benefit is that the NPS team would no longer need to purchase and install new devices for data backup. Comparing the time spent under the existing system to the time spent using an NPS big data platform would indicate the benefit of time saved. As a result, users could use the time saved to increase their capacity for other research. Since this benefit can be reflected by the amount of time spent on the search process by a team member, it amounts to a financial benefit.

Objective Type	Doing New Things	Doing Things Better	Stop Doing Things
Financial Benefits		Store data safely - Fault tolerance - ITEC monitors activities in cluster	Stop purchasing and setting new devices for backup
Quantifiable Benefits			
Measurable Benefits		Query to find files fast and easily	
Observable Benefits	Analytics using historical data within significant time		

B : Benefit M : Measure B.O : Benefit Owner

Figure 46. Benefits of using a big data platform

B. COSTS

The main cost of adopting a big data platform would be infrastructure expenses. However, NPS Information Technology and Communications Services (ITACS) already operates several big data platforms for diverse research projects, including Grace. Although costs associated with big data platforms typically include server costs, storage costs, application subscription costs, power costs, and network costs, these costs would likely not be passed onto NPS researchers.

On the contrary, other costs such as consultancy charges, training costs, customization costs, new application installation, and data migration costs should be considered. Since the members of the NPS team are not IT specialists, they should train one person to manage the applications and data in the NPS big data platform. This person would require additional training hours. Likewise, in order to establish customized applications to utilize these big data platforms, the NPS team may need to allocate funds to consult with IT experts, or possibly even hire one. We assumed the NPS team would hire one IT expert and train its members on how to manage their data, which would incur training costs and one IT expert's salary. However, data migration could then be performed by trained team members.

C. RISK

Security issues may be raised by this approach, since the aggregation of unclassified data may require higher-level classification. This risk would require users to obtain the proper security clearances to use the system, which would entail new costs related to proper security processes and storage. Additional risks include the continuity of the system in the event of power outages. Moreover, big data platforms require operating environments with well-regulated humidity, temperature, etc. However, these conditions already exist for the computing resources controlled by ITACS.

D. NET PRESENT VALUE

The net present value (NPV) is a measure of deducted total costs from total benefits. The monetized benefit offsets the cost associated with purchasing and setting up new

devices for backup data. If we assume that users in the NPS team generate 1 TB of sonar data per survey mission, and if the NPS team performs one hundred missions in one year, it would generate 100 TB of sonar data annually. The cost savings would then be approximately \$5,000 per year, assuming that a 1 TB storage device costs \$50. Additionally, they would not have to buy high-performance computers for analytics. We assume their budgets for purchasing new laptops would be \$10,000 per year. As a result, the total savings would be \$15,000 annually.

The total costs consist of one IT expert's wages and training for individual team members. One week of training (Cloudera Data Analyst Training) would be necessary to train members to utilize the big data platform for their needs, which we assume costs \$3,000 per person (Cloudera, n.d.-b). Furthermore, we assume that NPS team consist of five people, so training all team members requires \$15,000 for the first time. Conversely, we estimate that hiring one IT expert would cost \$55,000 annually (PayScale, n.d.). However, instead of hiring a new employee, we assume that the NPS team can consult with experts in the computer science department, resulting in zero costs for IT experts.

As a result, the NPV is calculated to be zero in the first year, and remains positive \$15,000 annually. Based on the positive financial benefit, the other non-monetized benefits and risk would make this decision look more attractive. Even though other possible costs could arise from using big data platforms and the benefit of new analytic capabilities and effective file search/retrieval methods cannot be monetized, the NPS team should choose this option if it prioritizes these benefits over potential security burdens and uncertain risk.

E. SUMMARY AND FUTURE RESEARCH

The accumulation of considerable amounts of sensor data presents both challenges and opportunities for the U.S. Navy, and big data platforms can be employed to address both. This thesis examined the utility of employing big data platforms for searching and analyzing sonar datasets collected by an NPS research team. Our experiments suggest that big data platforms would make NPS researchers more efficient by providing ease-of-use queries and significant analytic capabilities. Moreover, since the NPS ITACS organization

already manages the Grace Hadoop cluster, hardware costs are negligible, while data management and security concerns are mitigated.

However, in order to implement this solution, the NPS research team members would need training to access the big data platform and adopt it for their research. Furthermore, they may need to contract IT consultants to design customized big data platform applications for their specific research needs.

Overall, using a big data platform will enhance productivity and mitigate the risk of managing data. A big data platform will enable faster dataset searches and enhanced data processing. Big data analytics may also produce unexpected insights and knowledge not otherwise possible, therefore providing additional value.

Our experiments have demonstrated metadata extraction from sonar data files to create a database, time and location-based queries to find files of interest, and an example of big data analytics to predict water depth based on historical data. Future research could address the processing needed to extract features from these sonar images such as sea floor characteristics or mine-like objects, which are of great interest to naval operations. These types of big data applications can benefit from the computing power of a Hadoop cluster.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX

A. JAVA CODE FOR EXTRACTING METADATA FROM SONAR FILES

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Arrays;
import java.lang.Integer;

public class Extraction {
// Convert Byte to Hex class //
    private final static String[] hexSymbols =
        {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};
    public final static int BITS_PER_HEX_DIGIT = 4;
    public static String toHexFromByte(final byte b){
        byte leftSymbol = (byte)((b >>> BITS_PER_HEX_DIGIT) & 0x0f);
        byte rightSymbol = (byte)(b & 0x0f);
        return(hexSymbols[leftSymbol]+hexSymbols[rightSymbol]);
    }
// Convert Hex to Byte //
    public static byte[] toByteFromHex(final String s) {
        byte[] b = new byte[s.length() / 2];
        for (int i = 0; i < b.length; i++) {
            int index = i * 2;
            int v = Integer.parseInt(s.substring(index, index + 2), 16);
            b[i] = (byte) v;
        }
        return b;
    }
// Convert the float value to degrees and decimal minutes for Latitude and Longitude //
    public static String FloatToDegree(final float Float) {
        float degree = 0;
        degree= (Float - Float%60) /60 ;
        float min = 0;
        min = Float - (degree*60);
        float sec = min*60;
        min = (sec - sec % 60) / 60;
        sec = sec - min*60;
        String Degree_Min_Sec = degree + "°"+min+"′"+sec+"″";
        return Degree_Min_Sec;
    }
// Convert Julian Time to Time //
// JulianTime= (((Hour * 60) + Minutes) * 60) + Seconds //
    public static String JulianToTime(final long Long) {
        String Time ="";
        Time = (int)Long/3600 + ":" + (int)((Long%3600)/60)+":"+ (int)Long%60;
        return Time;
    }

// Start main //
```

```

public static String[] Get_Metadata(String args) throws IOException {
    try{

//Load the file on Input by the size of a file //
        String LoadFile = args;
        File file = new File(LoadFile);

        FileInputStream Input = new FileInputStream(new File(LoadFile));
        byte[] bytes = new byte[(int)file.length()];

        Input.read(bytes);

// Extract IFD_Address //
        byte[] IFD_Address_tmp = Arrays.copyOfRange(bytes, 4,8);
        int IFD_Address = 0;
        int i=0;
        int shiftBy = 0;
        for(shiftBy = 0; shiftBy < 32; shiftBy += 8){
            IFD_Address |= ( (long)( IFD_Address_tmp[i] & 0xff ) ) << shiftBy;
            i++;
        }

// Extract HEX from IFD //
        byte[] IFD_byte = Arrays.copyOfRange(bytes, IFD_Address, (int)file.length());
        String[] IFD_HEX = new String[(int)file.length() - IFD_Address];

        for(i=0;i<IFD_byte.length;i++){
            IFD_HEX[i] = toHexFromByte(IFD_byte[i]);
        }

// Find out Y2KtimeCorrelation, NavinfoCount, NavInfo6, FathometerCount, Fathometer2 //
        String tag = "";

        String Y2KtimeCorrelation_Hex = "";
        int Y2K_Address = 0;

        String NavinfoCount_Hex = "";
        int NavinfoCount = 0;
        String NavInfo6_Address = "";
        int Nav_Address = 0;

        String FathometerCount_Hex = "";
        int FathometerCount = 0;
        String Fathometer2_Address = "";
        int Fatho_Address = 0;

        for(i= 0; i<(int)file.length()-IFD_Address;i=i+2){
            tag = IFD_HEX[i]+IFD_HEX[i+1];
            if(tag.equals("1D01")) {
                Y2KtimeCorrelation_Hex =
                IFD_HEX[i+11] + IFD_HEX[i+10] + IFD_HEX[i+9] + IFD_HEX[i+8];
                Y2K_Address =
                Integer.parseInt(Y2KtimeCorrelation_Hex, 16);
            }
        }
    }
}

```

```

        if(tag.equals("0A01")) {
            NavinfoCount_Hex = IFD_HEX[i+9] + IFD_HEX[i+8];
            NavinfoCount = Integer.parseInt(NavinfoCount_Hex, 16);
        }
        if(tag.equals("3401")) {
            NavInfo6_Address = IFD_HEX[i+11] + IFD_HEX[i+10] +
IFD_HEX[i+9] + IFD_HEX[i+8];
            Nav_Address = Integer.parseInt(NavInfo6_Address, 16);
        }
        if(tag.equals("1E01")) {
            FathometerCount_Hex = IFD_HEX[i+11] + IFD_HEX[i+10] +
IFD_HEX[i+9] + IFD_HEX[i+8];
            FathometerCount = Integer.parseInt(FathometerCount_Hex,
16);
        }
        if(tag.equals("2801")) {
            Fathometer2_Address = IFD_HEX[i+11] + IFD_HEX[i+10] +
IFD_HEX[i+9] + IFD_HEX[i+8];
            Fatho_Address = Integer.parseInt(Fathometer2_Address, 16);
        }
    }

    // Y2KtimeCorrelation (HEX): Timestamp, Y2K_Date, Y2K_Time //
    byte[] Y2K_info_byte = Arrays.copyOfRange(bytes,
Y2K_Address, Y2K_Address+12);
    String[] Y2K_HEX = new String[12];
    String Y2K_timestamp_Hex = "";
    String Y2K_Date_Hex = "";
    String Y2K_Time_Hex = "";
    int cnt = 0;

    for(int j=0; j<12; j++){
        Y2K_HEX[j] = toHexFromByte(Y2K_info_byte[cnt]);
        cnt++;
    }
    Y2K_timestamp_Hex = Y2K_HEX[0].toString() + Y2K_HEX[1].toString() +
Y2K_HEX[2].toString() + Y2K_HEX[3].toString();
    Y2K_Date_Hex = Y2K_HEX[4].toString() + Y2K_HEX[5].toString() +
Y2K_HEX[6].toString() + Y2K_HEX[7].toString();
    Y2K_Time_Hex = Y2K_HEX[8].toString() + Y2K_HEX[9].toString() +
Y2K_HEX[10].toString() + Y2K_HEX[11].toString();

    // Y2KtimeCorrelation : Transform HEX -> Binary -> Long //
    long Y2K_timestamp = 0;
    byte[] Y2K_timestamp_byte_tmp = new byte[4];
    long Y2K_Date = 0;
    byte[] Y2K_Date_byte_tmp = new byte[4];
    long Y2K_Time = 0;
    byte[] Y2K_Time_byte_tmp = new byte[4];
    i = 0;
    Y2K_timestamp_byte_tmp = toByteFromHex(Y2K_timestamp_Hex);
    Y2K_Date_byte_tmp = toByteFromHex(Y2K_Date_Hex);
    Y2K_Time_byte_tmp = toByteFromHex(Y2K_Time_Hex);

    for(shiftBy = 0; shiftBy < 32; shiftBy += 8){

```

```

        Y2K_timestamp |=
            ( (long)( Y2K_timestamp_byte_tmp[i] & 0xff ) ) << shiftBy;
        Y2K_Date |= ( (long)( Y2K_Date_byte_tmp[i] & 0xff ) ) << shiftBy;
        Y2K_Time |= ( (long)( Y2K_Time_byte_tmp[i] & 0xff ) ) << shiftBy;
        i++;
    }
    long start_time_sec = Y2K_Time - Y2K_timestamp / 1000;
// Navinfo6 (HEX): Timestamp, Latitude, Longitude //
    byte[] Nav_info_byte =
        Arrays.copyOfRange(bytes, Nav_Address, Nav_Address+560*NavinfoCount);
    String[] Nav_HEX = new String[12];
    String[] Nav_timestamp_Hex = new String[NavinfoCount];
    String[] Nav_Lat_Hex = new String[NavinfoCount];
    String[] Nav_Long_Hex = new String[NavinfoCount];
    cnt = 0;
    for(i=0;i<NavinfoCount;i++) {
        for(int j=0;j<12;j++){
            Nav_HEX[j] = toHexFromByte(Nav_info_byte[cnt]);
            cnt++;
        }
        Nav_timestamp_Hex[i] =
Nav_HEX[0].toString() + Nav_HEX[1].toString() + Nav_HEX[2].toString() + Nav_HEX[3].toString();
        Nav_Lat_Hex[i] =
Nav_HEX[4].toString() + Nav_HEX[5].toString() + Nav_HEX[6].toString() + Nav_HEX[7].toString();
        Nav_Long_Hex[i] =
Nav_HEX[8].toString() + Nav_HEX[9].toString() + Nav_HEX[10].toString() + Nav_HEX[11].toString();
        cnt = cnt+548;
    }

// Navinfo6 : Transform HEX -> Binary -> Long or Float //
    long[] Nav_timestamp = new long[NavinfoCount];
    byte[] Nav_timestamp_byte_tmp = new byte[4];

    float[] Nav_Lat = new float[NavinfoCount];
    byte[] Nav_Lat_byte_tmp = new byte[4];
    int Nav_Lat_tmp = 0;

    float[] Nav_Long = new float[NavinfoCount];
    byte[] Nav_Long_byte_tmp = new byte[4];
    int Nav_Long_tmp = 0;

    int Nav_Real_Count =0;
    i = 0;
    for(int j = 0; j < NavinfoCount ; j ++){
        Nav_timestamp_byte_tmp = toByteFromHex(Nav_timestamp_Hex[j]);
        Nav_Lat_byte_tmp = toByteFromHex(Nav_Lat_Hex[j]);
        Nav_Long_byte_tmp = toByteFromHex(Nav_Long_Hex[j]);
        Nav_Lat_tmp = 0;
        Nav_Long_tmp = 0;
        for(shiftBy = 0; shiftBy < 32; shiftBy += 8){
            Nav_timestamp[j] |=
                ( (long)( Nav_timestamp_byte_tmp[i] & 0xff ) ) << shiftBy;
            Nav_Lat_tmp |=
                ( (long)( Nav_Lat_byte_tmp[i] & 0xff ) ) << shiftBy;
            Nav_Long_tmp |=

```

```

        ( (long)( Nav_Long_byte_tmp[i] & 0xff ) ) << shiftBy;
        i++;
    }
    // stop if abnormal data comes out //
    if(j>1) {
        if(Nav_timestamp[j-1] > Nav_timestamp[j]){
            break;
        }
    }
    Nav_Real_Count++;
    Nav_Lat[j] = Float.intBitsToFloat(Nav_Lat_tmp);
    Nav_Long[j] = Float.intBitsToFloat(Nav_Long_tmp);
    i=0;
}
for(i=0;i<NavinfoCount;i++) {
    if(Nav_timestamp[i] == 0) {
        break;
    }
}
long[] Nav_timestamp_sec = new long[i];
for(int j=0;j<i;j++){
    // Nav_timestamp : millisecond -> second //
    Nav_timestamp_sec[j] = Nav_timestamp[j]/1000 + start_time_sec;
}

// Fathometer2 (HEX): Timestamp, WaterDepth, TowfishDepth //
byte[] Fatho_info_byte =
Arrays.copyOfRange(bytes, Fatho_Address,Fatho_Address+12*FathometerCount);
String[] Fatho_HEX = new String[12];
String[] Fatho_timestamp_Hex = new String[FathometerCount];
String[] Fatho_WaterDepth_Hex = new String[FathometerCount];
String[] Fatho_TowfishDepth_Hex = new String[FathometerCount];
cnt = 0;
for(i=0;i<FathometerCount;i++) {
    for(int j=0;j<12;j++){
        Fatho_HEX[j] = toHexFromByte(Fatho_info_byte[cnt]);
        cnt++;
    }
    Fatho_timestamp_Hex[i] = Fatho_HEX[0].toString() +
Fatho_HEX[1].toString() + Fatho_HEX[2].toString() + Fatho_HEX[3].toString();
    Fatho_WaterDepth_Hex[i] = Fatho_HEX[4].toString() +
Fatho_HEX[5].toString() + Fatho_HEX[6].toString() + Fatho_HEX[7].toString();
    Fatho_TowfishDepth_Hex[i] = Fatho_HEX[8].toString() +
Fatho_HEX[9].toString() + Fatho_HEX[10].toString() + Fatho_HEX[11].toString();
}

// Fathometer2 : Transform HEX -> Binary -> Long or Float //
long[] Fatho_timestamp = new long[FathometerCount];
byte[] Fatho_timestamp_byte_tmp = new byte[4];

float[] Fatho_WaterDepth = new float[FathometerCount];
byte[] Fatho_WaterDepth_byte_tmp = new byte[4];
int Fatho_WaterDepth_tmp = 0;

float[] Fatho_TowfishDepth = new float[FathometerCount];

```

```

byte[] Fatho_TowfishDepth_byte_tmp = new byte[4];
int Fatho_TowfishDepth_tmp = 0;

i = 0;
int Fatho_Real_Count = 0;
for(int j = 0; j < FathometerCount ; j ++){
    Fatho_timestamp_byte_tmp =
        toByteFromHex(Fatho_timestamp_Hex[j]);
    Fatho_WaterDepth_byte_tmp =
        toByteFromHex(Fatho_WaterDepth_Hex[j]);
    Fatho_TowfishDepth_byte_tmp =
        toByteFromHex(Fatho_TowfishDepth_Hex[j]);
    Fatho_WaterDepth_tmp = 0;
    Fatho_TowfishDepth_tmp = 0;

    for(shiftBy = 0; shiftBy < 32; shiftBy += 8){
        Fatho_timestamp[j] |=
            ( (long)( Fatho_timestamp_byte_tmp[i] & 0xff ) ) << shiftBy;
        Fatho_WaterDepth_tmp |=
            ((long)(Fatho_WaterDepth_byte_tmp[i] & 0xff) ) << shiftBy;
        Fatho_TowfishDepth_tmp |=
            ((long)(Fatho_TowfishDepth_byte_tmp[i] & 0xff))<< shiftBy;
        i++;
    }
    // stop if abnormal data comes out //
    if(j>1) {
        if(Fatho_timestamp[j-1] > Fatho_timestamp[j]){
            break;
        }
    }
    Fatho_Real_Count++;
    Fatho_WaterDepth[j] = Float.intBitsToFloat(Fatho_WaterDepth_tmp);
    Fatho_TowfishDepth[j] =
        Float.intBitsToFloat(Fatho_TowfishDepth_tmp);
    i=0;
}

long[] Fatho_timestamp_sec = new long[FathometerCount];
for(int j=0;j<FathometerCount;j++){
    // Fatho_timestamp : millisecond -> second //
    Fatho_timestamp_sec[j] = Fatho_timestamp[j]/1000 + start_time_sec;
}

// Join Y2K_timestamp, Nav on Fatho //
// Create one table consisting of Path, Date, Time, Latitude, Longitude, Depth //
// Gap of Timestamps between Nav and Fatho is less than 4 because the interval
of Fatho timestamp is around 4 and that of Nav6 is around 15. //
// base timestamp is Nav6's timestamp //

int Data_Count = 0;
String[] Data = new String[Nav_Real_Count];
for(i=0; i < Nav_Real_Count; i++){
    for(int j = 0; j < Fatho_Real_Count ; j++){
        if(Nav_timestamp_sec[i]==Fatho_timestamp_sec[j]){

```


B. JAVA CODE FOR GETTING PATH

```
import java.io.File;

public class FileList{
    public static String[] GetFileList(){
        String totalFileList[] = new String[1000];
        String dir = "d:\\project";
        File test = new File(dir);
        // System.out.println("The list of folders and files are as below"); //
        int cnt = 0;
        if(test.exists()){
            if(test.isDirectory()){
                String[] testList = test.list();

                for(int i=0;i<testList.length;i++){
                    File test_sub = new File(dir + "\\" + testList[i]);
                    if(test_sub.isDirectory()) {
                        String[] test_subList = test_sub.list();
                        for(int j=0;j<test_subList.length;j++){
                            if(test_subList[j].substring(9,12).equals("mst")){
                                totalFileList[cnt] =
                                    dir + "\\" + testList[i] + "\\" +
                                    test_subList[j];
                                cnt++;
                            }
                        }
                    }
                    }else{
                        totalFileList[cnt] = dir + "\\" + testList[i];
                    }
                }
            } else{
                System.out.println("This is not folder");
            }

        } else{
            System.out.println("There is no folder.");
        }
        String TotalFileList[] = new String[cnt];
        for(int i = 0; i < TotalFileList.length; i++){
            TotalFileList[i] = totalFileList[i];
        }
        return TotalFileList;
    }
}
```

C. JAVA CODE OF LOADING DATA INTO ORACLE DATABASE 11G

```
import java.sql.*;

public class DBUpdate {
    public static void main (String[] args) {
        try {
            // performance:speed = endtime - starttime //
            long start_time = System.nanoTime();
            Class.forName("oracle.jdbc.OracleDriver");
            System.out.println("Driver loaded");
            String url="jdbc:oracle:thin:@localhost:1521:xe";
            String user = "schema";
            String pwd = "1";
            Connection conn;
            conn = DriverManager.getConnection(url,user,pwd);
            System.out.println("Database Connect ok");
            Statement query_stmt=conn.createStatement();
            String query="";

            // create table query
            query="create table Whole_Data (Data_ID Integer NOT NULL, Direct_Data
varchar2(50), Date_Data varchar(20), Time_Data varchar(20), Lat_Data varchar(20), Long_Data
varchar(20), Depth_Data varchar(20), Depth_Data2 varchar(20))";
            query_stmt.executeUpdate(query);
            System.out.println("Table created ok");

            // create sequence query
            query="create sequence Whole_Data_SEQ increment by 1 start with 1";
            query_stmt.executeUpdate(query);

            // insert data into table
            Extraction Ext = new Extraction();
            FileList fileList = new FileList();
            String[] totalFileList = fileList.GetFilesList();
            int Data_Count = 0;
            for(String path : totalFileList){
                String[] File_Data = Ext.Get_Metadata(path);
                for(String Data : File_Data){
                    Data = ""+"+ path + ","+ Data;
                    query =
                    "insert into Whole_Data values(Data_SEQ.nextval," + Data + ")";
                    query_stmt.executeUpdate(query);
                    Data_Count++; //
                }
            }
            System.out.println("Rows inserted ok");
            conn.close();
            // end time check and calculate the performance(speed) //
            long end_time = System.nanoTime();
            System.out.println("Execution Time : " + (end_time - start_time)/1000000000.0);
            query_stmt.close();
        } catch (Exception exp){
            System.out.println("Exception = "+exp);
        }
    }
}
```

D. APEX APPLICATION CODE

1. PL/SQL Code under Dynamic Action

```
select listagg(data_id,',') within group(order by data_id),
       listagg(lat_data, ',') within group (order by data_id),
       listagg(long_data,',') within group (order by data_id)
into :P1_ID, :P1_LAT, :P1_LONG
from data_latlong
where lat_data between :P1_LAT_MIN and :P1_LAT_MAX And
       long_data between :P1_LONG_MIN and :P1_LONG_MAX;
```

2. JavaScript Code under Dynamic Action

```
var a = document.getElementById('P1_LAT').value;
var LAT = a.split(",");
var b = document.getElementById('P1_LONG').value;
var LONG = b.split(",");
var j = document.getElementById('P1_ID').value;
var locations = j.split(",");

var map = new google.maps.Map(document.getElementById('map'), {
    zoom: 14,
    center: new google.maps.LatLng(24.948282, -80.45415),
    mapTypeId: google.maps.MapTypeId.ROADMAP
});

var infowindow = new google.maps.InfoWindow();

var marker, i;

for (i = 0; i < locations.length; i++) {
    marker = new google.maps.Marker({
        position: new google.maps.LatLng(LAT[i], LONG[i]),
        map: map
    });

    google.maps.event.addListener(marker, 'click', (function(marker, i) {
        return function() {
            infowindow.setContent(locations[i]);
            infowindow.open(map, marker);
        }
    })(marker, i));
}
```

3. JavaScript of Source in Regions

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <title>Google Maps Multiple Markers</title>
    <script src="https://maps.googleapis.com/maps/api/js?key=userkey">
```

```

type="text/javascript"></script>

</head>
<body>

<div id="map" style="width: 1000px; height: 600px;"></div>

<script type="text/javascript">
var a = document.getElementById('P1_LAT').value;
var LAT = a.split(":");
var b = document.getElementById('P1_LONG').value;
var LONG = b.split(":");
var j = document.getElementById('P1_ID').value;
var locations = j.split(":");

var map = new google.maps.Map(document.getElementById('map'), {
    zoom: 14,
    center: new google.maps.LatLng(24.948282, -80.45415),
    mapTypeId: google.maps.MapTypeId.ROADMAP
});

var infowindow = new google.maps.InfoWindow();

var marker, i;

for (i = 0; i < locations.length; i++) {
    marker = new google.maps.Marker({
        position: new google.maps.LatLng(LAT[i], LONG[i]),
        map: map
    });

    google.maps.event.addListener(marker, 'click', (function(marker, i) {
        return function() {
            infowindow.setContent(locations[i]);
            infowindow.open(map, marker);
        }
    })(marker, i));
}
</script>
</body>
</html>

```

E. HADOOP MAPREDUCE CODE

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.Arrays;
import java.text.DecimalFormat;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.*;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
public class GET_MST {
    private final static String[] hexSymbols =
        {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};
    public final static int BITS_PER_HEX_DIGIT = 4;
    // Convert Byte to Hex class //
    public static String toHexFromByte(final byte b){
        byte leftSymbol = (byte)((b >>> BITS_PER_HEX_DIGIT) & 0x0f);
        byte rightSymbol = (byte)(b & 0x0f);
        return(hexSymbols[leftSymbol]+hexSymbols[rightSymbol]);
    }
    // Convert Hex to Byte //
    public static byte[] toByteFromHex(final String s) {
        byte[] b = new byte[s.length() / 2];
        for (int i = 0; i < b.length; i++) {
            int index = i * 2;
            int v = Integer.parseInt(s.substring(index, index + 2), 16);
            b[i] = (byte) v;
        }
        return b;
    }
    // Convert the float value to degrees and decimal minutes for Latitude and Longitude //
    public static String FloatToDegree(final float Float) {
        float degree = 0;
        degree = (Float - Float%60) / 60 ;
        float min = 0;
        min = Float - (degree*60);
        float sec = min*60;
        min = (sec - sec % 60) / 60;
        sec = sec - min*60;
        String Degree_Min_Sec = degree + "°"+min+"′"+sec+"″";
        return Degree_Min_Sec;
    }
}
```

```

// Convert Julian Time to Time //
// JulianTime= (((Hour * 60) + Minutes) * 60) + Seconds //
    public static String JulianToTime(final long Long) {
        String Time ="";
        Time = (int)Long/3600 + ":" + (int)((Long%3600)/60)+":"+ (int)Long%60;
        return Time;
    }
// Mapper //
    public static class GET_MST_Mapper extends Mapper<Text, BytesWritable, Text, Text> {
        public void map(Text key, BytesWritable value, Context context) throws IOException,
        InterruptedException {
            // Get key from filePath and distinguish acceptable file(only mst) //
            String filePathString = key.toString();
            if(filePathString.matches(".*mst")) {
                // get Value as binary //
                byte[] bytes = value.getBytes();
                // Extract IFD_Address //
                byte[] IFD_Address_tmp = Arrays.copyOfRange(bytes, 4,8);
                int IFD_Address = 0;
                int i=0;
                int shiftBy = 0;
                for(shiftBy = 0; shiftBy < 32; shiftBy += 8){
                    IFD_Address |= ((long)( IFD_Address_tmp[i] & 0xff ) ) << shiftBy;
                    i++;
                }
                // Extract HEX from IFD //
                byte[] IFD_byte =
                    Arrays.copyOfRange(bytes, IFD_Address, (int)value.getLength());
                String[] IFD_HEX = new String[(int)value.getLength() - IFD_Address];
                for(i=0;i<IFD_byte.length;i++){
                    IFD_HEX[i] = toHexFromByte(IFD_byte[i]);
                }
            }
            // Find out Y2KtimeCorrelation, NavinfoCount, NavInfo6, FathometerCount, Fathometer2 //
            String tag = "";
            String Y2KtimeCorrelation_Hex = "";
            int Y2K_Address = 0;
            String NavinfoCount_Hex = "";
            int NavinfoCount = 0;
            String NavInfo6_Address = "";
            int Nav_Address = 0;
            String FathometerCount_Hex = "";
            int FathometerCount = 0;
            String Fathometer2_Address = "";
            int Fatho_Address = 0;
            for(i= 0; i<(int)value.getLength()-IFD_Address;i=i+2){
                tag = IFD_HEX[i]+IFD_HEX[i+1];
                if(tag.equals("1D01")) {
                    Y2KtimeCorrelation_Hex =
                    IFD_HEX[i+11] + IFD_HEX[i+10] + IFD_HEX[i+9] + IFD_HEX[i+8];
                    Y2K_Address =
                    Integer.parseInt(Y2KtimeCorrelation_Hex, 16);
                }
                if(tag.equals("0A01")) {
                    NavinfoCount_Hex =
                    IFD_HEX[i+9] + IFD_HEX[i+8];

```

```

        NavinfoCount =
            Integer.parseInt(NavinfoCount_Hex, 16);
    }
    if(tag.equals("3401")) {
        NavInfo6_Address =
            IFD_HEX[i+11] + IFD_HEX[i+10] + IFD_HEX[i+9] + IFD_HEX[i+8];
        Nav_Address =
            Integer.parseInt(NavInfo6_Address, 16);
    }
    if(tag.equals("1E01")) {
        FathometerCount_Hex =
            IFD_HEX[i+11] + IFD_HEX[i+10] + IFD_HEX[i+9] + IFD_HEX[i+8];
        FathometerCount =
            Integer.parseInt(FathometerCount_Hex, 16);
    }
    if(tag.equals("2801")) {
        Fathometer2_Address =
            IFD_HEX[i+11] + IFD_HEX[i+10] + IFD_HEX[i+9] + IFD_HEX[i+8];
        Fatho_Address =
            Integer.parseInt(Fathometer2_Address, 16);
    }
}
// Y2KtimeCorrelation (HEX): Timestamp, Y2K_Date, Y2K_Time //
byte[] Y2K_info_byte =
    Arrays.copyOfRange(bytes, Y2K_Address, Y2K_Address+12);
String[] Y2K_HEX = new String[12];
String Y2K_timestamp_Hex = "";
String Y2K_Date_Hex = "";
String Y2K_Time_Hex = "";
int cnt = 0;
for(int j=0;j<12;j++){
    Y2K_HEX[j] = toHexFromByte(Y2K_info_byte[cnt]);
    cnt++;
}
Y2K_timestamp_Hex =
Y2K_HEX[0].toString() + Y2K_HEX[1].toString() + Y2K_HEX[2].toString() + Y2K_HEX[3].toString();
Y2K_Date_Hex =
Y2K_HEX[4].toString() + Y2K_HEX[5].toString() + Y2K_HEX[6].toString() + Y2K_HEX[7].toString();
Y2K_Time_Hex = Y2K_HEX[8].toString() + Y2K_HEX[9].toString()
+ Y2K_HEX[10].toString() + Y2K_HEX[11].toString();
long Y2K_timestamp = 0;
byte[] Y2K_timestamp_byte_tmp = new byte[4];
long Y2K_Date = 0;
byte[] Y2K_Date_byte_tmp = new byte[4];
long Y2K_Time = 0;
byte[] Y2K_Time_byte_tmp = new byte[4];
i = 0;
Y2K_timestamp_byte_tmp = toByteFromHex(Y2K_timestamp_Hex);
Y2K_Date_byte_tmp = toByteFromHex(Y2K_Date_Hex);
Y2K_Time_byte_tmp = toByteFromHex(Y2K_Time_Hex);
for(shiftBy = 0; shiftBy < 32; shiftBy += 8){
    Y2K_timestamp |=
        ( (long)( Y2K_timestamp_byte_tmp[i] & 0xff ) ) << shiftBy;
    Y2K_Date |=
        ( (long)( Y2K_Date_byte_tmp[i] & 0xff ) ) << shiftBy;

```

```

        Y2K_Time |=
            ( (long)( Y2K_Time_byte_tmp[i] & 0xff ) ) << shiftBy;
        i++;
    }
    String Date_tmp = Y2K_Date + "";
    String Date_String =
        Date_tmp.substring(0,4) + "-" + Date_tmp.substring(4,6) + "-" +
        Date_tmp.substring(6,8);
    long start_time_sec = Y2K_Time - Y2K_timestamp / 1000;
    // Navinfo6 (HEX): Timestamp, Latitude, Longitude //
    byte[] Nav_info_byte =
        Arrays.copyOfRange(bytes, Nav_Address,Nav_Address+560*NavinfoCount);
    String[] Nav_HEX = new String[12];
    String[] Nav_timestamp_Hex = new String[NavinfoCount];
    String[] Nav_Lat_Hex = new String[NavinfoCount];
    String[] Nav_Long_Hex = new String[NavinfoCount];
    cnt = 0;
    for(i=0;i<NavinfoCount;i++) {
        for(int j=0;j<12;j++){
            Nav_HEX[j] = toHexFromByte(Nav_info_byte[cnt]);
            cnt++;
        }
        Nav_timestamp_Hex[i] = Nav_HEX[0].toString() +
Nav_HEX[1].toString() + Nav_HEX[2].toString() + Nav_HEX[3].toString();
        Nav_Lat_Hex[i] = Nav_HEX[4].toString() +
Nav_HEX[5].toString() + Nav_HEX[6].toString() + Nav_HEX[7].toString();
        Nav_Long_Hex[i] = Nav_HEX[8].toString() +
Nav_HEX[9].toString() + Nav_HEX[10].toString() + Nav_HEX[11].toString();
        cnt = cnt+548;
    }
    // Navinfo6 : Transform HEX -> Binary -> Long or Float //
    long[] Nav_timestamp = new long[NavinfoCount];
    byte[] Nav_timestamp_byte_tmp = new byte[4];
    float[] Nav_Lat = new float[NavinfoCount];
    byte[] Nav_Lat_byte_tmp = new byte[4];
    int Nav_Lat_tmp = 0;
    float[] Nav_Long = new float[NavinfoCount];
    byte[] Nav_Long_byte_tmp = new byte[4];
    int Nav_Long_tmp = 0;
    int Nav_Real_Count =0;
    i = 0;
    for(int j = 0; j < NavinfoCount ; j++){
        Nav_timestamp_byte_tmp =
            toByteFromHex(Nav_timestamp_Hex[j]);
        Nav_Lat_byte_tmp = toByteFromHex(Nav_Lat_Hex[j]);
        Nav_Long_byte_tmp = toByteFromHex(Nav_Long_Hex[j]);
        Nav_Lat_tmp = 0;
        Nav_Long_tmp = 0;
        for(shiftBy = 0; shiftBy < 32; shiftBy += 8){
            Nav_timestamp[j] |=
                ( (long)( Nav_timestamp_byte_tmp[i] & 0xff ) ) << shiftBy;
            Nav_Lat_tmp |=
                ( (long)( Nav_Lat_byte_tmp[i] & 0xff ) ) << shiftBy;
            Nav_Long_tmp |=
                ( (long)( Nav_Long_byte_tmp[i] & 0xff ) ) << shiftBy;

```

```

        i++;
    }
    // stop if abnormal data comes out //
    if(j>1) {
        if(Nav_timestamp[j-1] > Nav_timestamp[j]){
            break;
        }
    }
    Nav_Real_Count++;
    Nav_Lat[j] = Float.intBitsToFloat(Nav_Lat_tmp);
    Nav_Long[j] = Float.intBitsToFloat(Nav_Long_tmp);
    i=0;
}
for(i=0;i<NavinfoCount;i++) {
    if(Nav_timestamp[i] == 0) {
        break;
    }
}
long[] Nav_timestamp_sec = new long[i];
for(int j=0;j<i;j++){
    Nav_timestamp_sec[j] =
Nav_timestamp[j]/1000 + start_time_sec; // Nav_timestamp : millisecond -> second //
}
// Fathometer2 (HEX): Timestamp, WaterDepth, TowfishDepth //
byte[] Fatho_info_byte =
Arrays.copyOfRange(bytes, Fatho_Address,Fatho_Address+12*FathometerCount);
String[] Fatho_HEX = new String[12];
String[] Fatho_timestamp_Hex = new String[FathometerCount];
String[] Fatho_WaterDepth_Hex = new String[FathometerCount];
String[] Fatho_TowfishDepth_Hex = new String[FathometerCount];
cnt = 0;
for(i=0;i<FathometerCount;i++) {
    for(int j=0;j<12;j++){
        Fatho_HEX[j] =
            toHexFromByte(Fatho_info_byte[cnt]);
        cnt++;
    }
    Fatho_timestamp_Hex[i] = Fatho_HEX[0].toString() +
Fatho_HEX[1].toString() + Fatho_HEX[2].toString() + Fatho_HEX[3].toString();
    Fatho_WaterDepth_Hex[i] = Fatho_HEX[4].toString() +
Fatho_HEX[5].toString() + Fatho_HEX[6].toString() + Fatho_HEX[7].toString();
    Fatho_TowfishDepth_Hex[i] = Fatho_HEX[8].toString() +
Fatho_HEX[9].toString() + Fatho_HEX[10].toString() + Fatho_HEX[11].toString();
}
// Fathometer2 : Transform HEX -> Binary -> Long or Float //
long[] Fatho_timestamp = new long[FathometerCount];
byte[] Fatho_timestamp_byte_tmp = new byte[4];
float[] Fatho_WaterDepth = new float[FathometerCount];
byte[] Fatho_WaterDepth_byte_tmp = new byte[4];
int Fatho_WaterDepth_tmp = 0;
float[] Fatho_TowfishDepth = new float[FathometerCount];
byte[] Fatho_TowfishDepth_byte_tmp = new byte[4];
int Fatho_TowfishDepth_tmp = 0;
i = 0;
int Fatho_Real_Count = 0;

```

```

for(int j = 0; j < FathometerCount ; j ++){
    Fatho_timestamp_byte_tmp =
        toByteFromHex(Fatho_timestamp_Hex[j]);
    Fatho_WaterDepth_byte_tmp =
        toByteFromHex(Fatho_WaterDepth_Hex[j]);
    Fatho_TowfishDepth_byte_tmp =
        toByteFromHex(Fatho_TowfishDepth_Hex[j]);
    Fatho_WaterDepth_tmp = 0;
    Fatho_TowfishDepth_tmp = 0;

    for(shiftBy = 0; shiftBy < 32; shiftBy += 8){
        Fatho_timestamp[j] |=
            ((long)(Fatho_timestamp_byte_tmp[i] & 0xff) ) << shiftBy;
        Fatho_WaterDepth_tmp |=
            ((long)(Fatho_WaterDepth_byte_tmp[i] & 0xff) ) << shiftBy;
        Fatho_TowfishDepth_tmp |=
            ((long)(Fatho_TowfishDepth_byte_tmp[i] & 0xff) ) << shiftBy;
        i++;
    }
    // stop if abnormal data comes out //
    if(j>1) {
        if(Fatho_timestamp[j-1] > Fatho_timestamp[j]){
            break;
        }
    }
    Fatho_Real_Count++;
    Fatho_WaterDepth[j] =
        Float.intBitsToFloat(Fatho_WaterDepth_tmp);
    Fatho_TowfishDepth[j] =
        Float.intBitsToFloat(Fatho_TowfishDepth_tmp);
    i=0;
}

long[] Fatho_timestamp_sec = new long[FathometerCount];
for(int j=0;j<FathometerCount;j++){
    Fatho_timestamp_sec[j] =
        Fatho_timestamp[j]/1000 + start_time_sec;
}
// Join Y2K_timestamp, Nav on Fatho //
int Data_Count = 0;
DecimalFormat GPS_Decimal = new DecimalFormat("###.#####");
DecimalFormat Depth_Decimal = new DecimalFormat("##.##");
String[] Data = new String[Nav_Real_Count];
// Convert DATE+TIME into UNIX TIMESTAMP //
String Date_Time = "";
DateFormat format =
    new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
// divide timestamp(epoch:1/1/1970) by 1000 after reducing it by epoch time of 1/1/2010 //
// since when using MLPRRegression, its double value cannot deal with big number //
long UNIX_timestamp = 0;
for(i=0; i < Nav_Real_Count; i++){
    for(int j = 0; j < Fatho_Real_Count ; j++){
        if(Nav_timestamp_sec[i]==Fatho_timestamp_sec[j]){
            Date_Time =
                Date_String + " " + JulianToTime(Nav_timestamp_sec[i]);

```

```

        try {
            UNIX_timestamp =
format.parse(Date_Time).getTime();
        } catch (ParseException e) {
            e.printStackTrace();
        }
        Data[Data_Count] = (UNIX_timestamp/
1000 + "," + GPS_Decimal.format(Nav_Lat[i] / 60) + "," + GPS_Decimal.format(Nav_Long[i] / 60) + "," +
Depth_Decimal.format(Fathos_WaterDepth[j]) + "," + Depth_Decimal.format(Fathos_TowfishDepth[j]);
context.write(new Text(filePathString), new

        Data_Count++;
        break;
    }else
if(Nav_timestamp_sec[i]==Fathos_timestamp_sec[j]+1){
    JulianToTime(Nav_timestamp_sec[i]);

    Date_Time = Date_String + " " +

    try {
        UNIX_timestamp =

    } catch (ParseException e) {

    }
    Data[Data_Count] = (UNIX_timestamp/
1000 + "," + GPS_Decimal.format(Nav_Lat[i] / 60) + "," + GPS_Decimal.format(Nav_Long[i] / 60) + "," +
Depth_Decimal.format(Fathos_WaterDepth[j]) + "," + Depth_Decimal.format(Fathos_TowfishDepth[j]);
context.write(new Text(filePathString), new

    Data_Count++;
    break;
}
Date_Time = Date_String + " " +

try {
    UNIX_timestamp =

} catch (ParseException e) {

}
Data[Data_Count] = (UNIX_timestamp/
1000 + "," + GPS_Decimal.format(Nav_Lat[i] / 60) + "," + GPS_Decimal.format(Nav_Long[i] / 60) + "," +
Depth_Decimal.format(Fathos_WaterDepth[j]) + "," + Depth_Decimal.format(Fathos_TowfishDepth[j]);
context.write(new Text(filePathString), new

Data_Count++;
break;
}else
if(Nav_timestamp_sec[i]==Fathos_timestamp_sec[j]+2){
    JulianToTime(Nav_timestamp_sec[i]);

    Date_Time = Date_String + " " +

    try {
        UNIX_timestamp =

    } catch (ParseException e) {

    }
    Data[Data_Count] = (UNIX_timestamp/
1000 + "," + GPS_Decimal.format(Nav_Lat[i] / 60) + "," + GPS_Decimal.format(Nav_Long[i] / 60) + "," +
Depth_Decimal.format(Fathos_WaterDepth[j]) + "," + Depth_Decimal.format(Fathos_TowfishDepth[j]);
context.write(new Text(filePathString), new

Text(Data[Data_Count]));

    Data_Count++;
    break;
}
Date_Time = Date_String + " " +

try {
    UNIX_timestamp =

} catch (ParseException e) {

}
Data[Data_Count] = (UNIX_timestamp/
1000 + "," + GPS_Decimal.format(Nav_Lat[i] / 60) + "," + GPS_Decimal.format(Nav_Long[i] / 60) + "," +
Depth_Decimal.format(Fathos_WaterDepth[j]) + "," + Depth_Decimal.format(Fathos_TowfishDepth[j]);
context.write(new Text(filePathString), new

Text(Data[Data_Count]));

    Data_Count++;
    break;
}
if(Nav_timestamp_sec[i]==Fathos_timestamp_sec[j]-1){
    JulianToTime(Nav_timestamp_sec[i]);

    Date_Time = Date_String + " " +

    try {
        UNIX_timestamp =

    } catch (ParseException e) {

    }
    Data[Data_Count] = (UNIX_timestamp/
1000 + "," + GPS_Decimal.format(Nav_Lat[i] / 60) + "," + GPS_Decimal.format(Nav_Long[i] / 60) + "," +
Depth_Decimal.format(Fathos_WaterDepth[j]) + "," + Depth_Decimal.format(Fathos_TowfishDepth[j]);
context.write(new Text(filePathString), new

    Data_Count++;
    break;
}

```

```

Data[Data_Count] = (UNIX_timestamp/
1000 + "," + GPS_Decimal.format(Nav_Lat[i] / 60) + "," + GPS_Decimal.format(Nav_Long[i] / 60) + "," +
Depth_Decimal.format(Fatho_WaterDepth[j]) + "," + Depth_Decimal.format(Fatho_TowfishDepth[j]);
context.write(new Text(filePathString), new
Text(Data[Data_Count]));

Data_Count++;
break;
} else
if(Nav_timestamp_sec[i]==Fatho_timestamp_sec[j]-2){
Date_Time = Date_String + " " +
JulianToTime(Nav_timestamp_sec[i]);
try {
UNIX_timestamp =
format.parse(Date_Time).getTime();
} catch (ParseException e) {
Data[Data_Count] = (UNIX_timestamp/
1000 + "," + GPS_Decimal.format(Nav_Lat[i] / 60) + "," + GPS_Decimal.format(Nav_Long[i] / 60) + "," +
Depth_Decimal.format(Fatho_WaterDepth[j]) + "," + Depth_Decimal.format(Fatho_TowfishDepth[j]);
context.write(new Text(filePathString), new
Text(Data[Data_Count]));

Data_Count++;
break;
}
}
}
}
}
}

public static class GET_MST_Reducer extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> S_tot, Context context) throws IOException,
    InterruptedException {
        int count = 0 ;
        Text a = new Text();
        for(Text t: S_tot) { context.write(key, t);
        }
    }
}

public static class CombineDocumentFileFormat extends CombineFileInputFormat<Text,
BytesWritable> {
    @Override
    protected boolean isSplittable(JobContext context, Path file) {
        return false;
    }
    @Override
    public RecordReader<Text, BytesWritable> createRecordReader(InputSplit split,
TaskAttemptContext context) throws IOException {
        return new CombineFileRecordReader<Text, BytesWritable>
((CombineFileSplit) split, context, WholeFileRecordReader.class);
    }
}

public static class WholeFileRecordReader extends RecordReader<Text, BytesWritable> {
    private CombineFileSplit inputSplit;
    private Integer idx;
    private Text path;

```

```

private Configuration conf;
private BytesWritable document = new BytesWritable();
private boolean read;

public WholeFileRecordReader(CombineFileSplit inputSplit, TaskAttemptContext
context, Integer idx){
    this.inputSplit = inputSplit;
    this.idx = idx;
    this.conf = context.getConfiguration();
    this.read = false;
}
@Override
public void initialize(InputSplit split, TaskAttemptContext context) throws IOException,
InterruptedException {
}
@Override
public boolean nextKeyValue() throws IOException, InterruptedException {
    if (!read) {
        Path file = inputSplit.getPath(idx);
        FileSystem fs = file.getFileSystem(conf);
        FSDataInputStream input = fs.open(file);
        byte[] bytes = new byte[(int) inputSplit.getLength(idx)];
        int offset = (int) inputSplit.getOffset(idx);
        int length = (int) inputSplit.getLength(idx);
        IOUtils.readFully(input, bytes, offset, length);
        document = new BytesWritable();
        document.set(bytes, offset, length);
        path = new Text(file.toString());
        read = true;
        return true;
    } else {
        return false;
    }
}
@Override
public Text getCurrentKey() throws IOException, InterruptedException {
    return path;
}
@Override
public BytesWritable getCurrentValue() throws IOException, InterruptedException {
    return document;
}
@Override
public float getProgress() throws IOException {
    return read ? 1.0f : 0.0f;
}
@Override
public void close() throws IOException {
    // Don't need to do anything //
}
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.set("mapred.textoutputformat.separator", ",");
    Job job = Job.getInstance(conf, "GET_MST");

```

```

        job.getConfiguration().setLong("mapreduce.input.fileinputformat.split.maxsize,"(long)
128*1024*1024);
        job.setJarByClass(GET_MST.class);
        job.setMapperClass(GET_MST_Mapper.class);
        job.setReducerClass(GET_MST_Reducer.class);
        job.setInputFormatClass(CombineDocumentFileFormat.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        Path path = new Path(args[0]);
        FileSystem fs = path.getFileSystem(conf);
        FileStatus[] status = fs.listStatus(path);
        for(FileStatus stat : status) {
            if(stat.isDirectory()) {
                FileInputFormat.addInputPath(job, new Path(stat.getPath().toString()));
            }
        }
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

F. IMPLEMENTING MULTILAYER PERCEPTRON REGRESSION

1. Hadoop

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.mapreduce.Partitioner;

import java.io.FileNotFoundException;
import com.github.chen0040.data.frame.DataRow;
import com.github.chen0040.data.frame.BasicDataFrame;
import com.github.chen0040.data.frame.DataFrame;
import com.github.chen0040.data.frame.DataQuery;
import com.github.chen0040.data.utils.TupleTwo;
import com.github.chen0040.mlp.ann.regression.MLPRegression;

public class MLP_Test {
    public static class StringPair implements WritableComparable<StringPair> {
        private String latitude;
        private String longitude;

        public StringPair(){
        }

        public StringPair(String a, String b){
            this.latitude = a;
            this.longitude = b;
        }

        public void write(DataOutput out) throws IOException {
            WritableUtils.writeString(out, latitude);
            WritableUtils.writeString(out, longitude);
        }

        public void readFields(DataInput in) throws IOException {
            this.latitude = WritableUtils.readString(in);
            this.longitude = WritableUtils.readString(in);
        }

        public int compareTo(StringPair pop) {
            if (pop == null) return 0;
            int intcnt = latitude.compareTo(pop.latitude);
            return intcnt != 0 ? longitude.compareTo(pop.longitude) : intcnt;
        }
    }
}
```

```

    }
    public String getLatitude() {
        return latitude.toString();
    }
    public String getLongitude() {
        return longitude.toString();
    }
    @Override
    public String toString(){
        return latitude.toString() + "\t" + longitude.toString();
    }
}

public static class MLP_Mapper extends Mapper<LongWritable, Text, StringPair, Text> {

    public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {

        String record = value.toString();
        String[] parts = record.split(",");
        StringPair keys = new StringPair(parts[2],parts[3]);
        String values = parts[0] + "," + parts[1] + "," + parts[4];
        context.write(keys, new Text(values));
    }
}

public static class StringPair_Partitioner extends Partitioner<StringPair, Text> {
    @Override
    public int getPartition(StringPair keys, Text values, int numPartitions) {
        return Math.abs(keys.getLatitude().hashCode()*127)% numPartitions;
    }
}

public static class StringPair_Comparator extends WritableComparator {
    public StringPair_Comparator() {
        super(StringPair.class, true);
    }

    @Override
    public int compare(WritableComparable w1, WritableComparable w2){
        StringPair k1 = (StringPair) w1;
        StringPair k2 = (StringPair) w2;
        int cmp = k1.getLatitude().compareTo(k2.getLatitude());
        if(cmp !=0){
            return cmp;
        } else{
            return k1.getLongitude().compareTo(k2.getLongitude());
        }
    }
}

public static class MLP_Reducer extends Reducer<StringPair, Text, StringPair, Text> {
    public void reduce(StringPair key, Iterable<Text> values, Context context) throws
FileNotFoundException, IOException, InterruptedException {
        // Create a data frame manually by using Cheen0040's DataFrame //

```

```

        DataFrame frame = new BasicDataFrame();
        for(Text itr : values) {
            DataRow row = frame.newRow();
            String val[] = itr.toString().split(",");
            row.setCell("time,"Double.valueOf(val[1]));
            row.setTargetCell("depth,"Double.valueOf(val[2]));
            frame.addRow(row);
        }
        frame.lock();

//      System.out.println(frame.rowCount());
//      System.out.println(frame.getLevels());
        TupleTwo<DataFrame, DataFrame> miniFrames = frame.shuffle().split(0.9);
        DataFrame trainingData = miniFrames._1();
        DataFrame crossValidationData = miniFrames._2();
        MLPRegression regression = new MLPRegression();
        regression.setHiddenLayers(3);
        regression.setEpochs(1000);

        if(trainingData.rowCount() > 1) {
            regression.fit(trainingData);
//          System.out.println("Training:");
            for (int i = 0; i < trainingData.rowCount(); ++i) {
//              double[] array = trainingData.row(i).toArray();
                double predicted = regression.transform(trainingData.row(i));
                double actual = trainingData.row(i).target();
//              System.out.format("predicted: %f\texpected: %f\n", predicted,
actual);

            }

//          System.out.println("Exit");
//          System.out.println("Cross-validation:");
            if(crossValidationData.rowCount() > 1) {
                String[] cmp = new String[crossValidationData.rowCount()];
                for (int i = 0; i < crossValidationData.rowCount(); ++i) {
                    double predicted =
regression.transform(crossValidationData.row(i));
                    double actual = crossValidationData.row(i).target();
//                    System.out.format("predicted: %f\texpected: %f\n",
predicted, actual);

                    cmp[i] = "predicted: " + predicted + "\t actual: " +
actual;

                    context.write(key, new Text(cmp[i]));
                }
            }
        }

        }

    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        // add external jars //
        DistributedCache.addFileToClassPath(new Path("MLP_Jar/java-ann-mlp-1.0.1.jar"),
conf);
    }

```

```

        DistributedCache.addToClassPath(new Path("MLP_Jar/java-data-frame-1.0.9.jar"),
        conf);

        DistributedCache.addToClassPath(new Path("MLP_Jar/log4j-1.2.17.jar"), conf);
        DistributedCache.addToClassPath(new Path("MLP_Jar/slf4j-api-1.7.20.jar"), conf);
        DistributedCache.addToClassPath(new Path("MLP_Jar/slf4j-log4j12-1.7.20.jar"),
        conf);

        Job job = Job.getInstance(conf, "MLP_Test");
        job.setJarByClass(MLP_Test.class);
        job.setMapperClass(MLP_Mapper.class);
        job.setPartitionerClass(StringPair_Partitioner.class);
        job.setSortComparatorClass(StringPair_Comparator.class);
        job.setReducerClass(MLP_Reducer.class);
        job.setMapOutputKeyClass(StringPair.class);
        job.setMapOutputValueClass(Text.class);
        job.setOutputKeyClass(StringPair.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

2. Spark

```

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.PairFunction;

import com.github.chen0040.data.frame.BasicDataFrame;
import com.github.chen0040.data.frame.DataFrame;
import com.github.chen0040.data.frame.DataRow;
import com.github.chen0040.data.utils.TupleTwo;
import com.github.chen0040.mlp.ann.regression.MLPRegression;
import com.github.chen0040.mlp.enums.WeightUpdateMode;

import scala.Tuple2;

public class MLP {
    public static void main(String[] args) {
        SparkConf conf = new SparkConf().setMaster("yarn").setAppName("MLP");
        JavaSparkContext javaSparkContext = new JavaSparkContext(conf);

        javaSparkContext.addJar("/home/pyojeong.kim.ks/Downloads/MLP/Spark/
        Jar/java-ann-mlp-1.0.6.jar");
        javaSparkContext.addJar("/home/
        pyojeong.kim.ks/Downloads/MLP/Spark/Jar/java-data-frame-1.0.11.jar");
        javaSparkContext.addJar("/home/pyojeong.kim.ks/Downloads/MLP/Spark/Jar/log4j-
        1.2.17.jar");
        javaSparkContext.addJar("/home/pyojeong.kim.ks/Downloads/MLP/

```

```

Spark/Jar/slf4j-api-1.7.26.jar");          javaSparkContext.addJar("/home/
pyojeong.kim.ks/Downloads/MLP/Spark/Jar/slf4j-log4j12-1.7.26.jar");
    JavaRDD<String> inputData = javaSparkContext.textFile(args[0]);

    JavaPairRDD<String, String> pairs = inputData.mapToPair(new
PairFunction<String, String, String>() {
        @Override
        public Tuple2<String, String> call(String input) throws
Exception {
            return new Tuple2<String, String>(input.split(",")[2]+","+
input.split(",")[3], input.split(",")[1]+","+input.split(",")[4]);
        }
    });

    JavaPairRDD<String, Iterable<String>> group = pairs.groupByKey(300);

    JavaPairRDD<String, Tuple2<String, String>> results =
        group.mapValues((Iterable<String> values) -> {
            String cmpdepth = "";
            String experiment = "";
            DataFrame frame = new BasicDataFrame();
            for(String value : values) {
                String[] val = value.split(",");
                DataRow row = frame.newRow();
                row.setCell("time,"
Double.valueOf(val[0]));
                row.setTargetCell("depth,"
Double.valueOf(val[1]));
                frame.addRow(row);
            }
            frame.lock();
            TupleTwo<DataFrame, DataFrame> miniFrames =
frame.shuffle().split(0.9);
            DataFrame trainingData = miniFrames._1();
            DataFrame crossValidationData = miniFrames._2();
            MLPRRegression regression = new MLPRRegression();
            regression.setHiddenLayers(8,8,8);
            regression.setEpoches(1000);

            regression.setWeightUpdateMode(WeightUpdateMode.MiniBatchGradientDescend);
            regression.setMiniBatchSize(30);
            regression.setLearningRate(0.5);
            if(trainingData.rowCount() > 1) {
                regression.fit(trainingData);
                if(crossValidationData.rowCount() > 1) {
                    for (int i = 0; i <
crossValidationData.rowCount(); ++i) {
                        double predicted =
regression.transform(crossValidationData.row(i));
                        double actual =
crossValidationData.row(i).target();
                        System.out.format("predicted:
%f\texpected: %f\n," predicted, actual);

```

```

        crossValidationData.row(i)+"";
        tmp.substring(5,18);
        actual;

        String tmp =
        experiment =
        cmpdepth = predicted+ ,""+

    }
    }
    }
    return new Tuple2<String,
String>(experiment,cmpdepth);
    });
    results.saveAsTextFile(args[1]);
}
}

```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- Achari, S. (2015). *Hadoop essentials*. Birmingham, UK: Packt Publishing Ltd.
- Arrow, K. J., Cropper, M. L., Eads, G. C., Hahn, R. W., Lave, L. B., Noll, R. G., ... Smith, K. (1996). *Benefit-cost analysis in environmental, health, and safety regulation*. Washington, DC: American Enterprise Institute.
- Barngrover, C. M. (2014). *Automated detection of mine-like objects in side scan sonar imagery* (Doctoral dissertation). Retrieved from <https://escholarship.org/uc/item/4gw8g426>
- Ben-Gan, I. (2016). *T-SQL Fundamentals* (3rd ed.). Redmond, WA: Microsoft Press.
- Beyer, M. A., & Laney, D. (2012). *The importance of “Big Data”: A definition* (Report No. G00235055). Retrieved from <https://www.gartner.com/en/documents/2057415>
- Bihani, P., & Patil, S. T. (2014). A comparative study of data analysis techniques. *International Journal of Emerging Trends & Technology in Computer Science* 3(2), 95–101.
- Brennan, R., Hess, K., Huff, L. C., & Gill, S. (2005). The design of an uncertainty model for the tidal constituent and residual interpolation (TCARI) method for tidal correction of bathymetric data. Paper presented at the Biennial International Symposium of the Hydrographic Society, Rotterdam, The Netherlands. Retrieved from http://ushydro.thsoa.org/hy05/09_2a.pdf
- Bulut, Y. E. (2018). *AI for data science: artificial intelligence frameworks and functionality for deep learning, optimization, and beyond*. Basking Ridge, NJ: Technics Publications.
- Capilla, C. (2014). Multilayer perceptron and regression modelling to forecast hourly nitrogen dioxide concentrations. *WIT Transactions on Ecology and the Environment*, 183, 39–48.
- Capriolo, E., Wampler, D., & Rutherglen, J. (2012). *Programming hive* (1st ed). Sebastopol, CA: O'Reilly & Associates.
- Chambers, B., & Zaharia, M. (2018). *Spark: The definitive guide big data processing made simple*. Sebastopol, CA: O'Reilly Media, Inc.
- Chen, X. (2017, June 12). Package provides java implementation of multi-layer perceptron neural network with back-propagation learning algorithm [Blog post]. Retrieved from <https://github.com/chen0040/java-ann-mlp>

- Chen, Z., Torson, J., & Servisetti, S. (2011). Reexamining database keyword search: Beyond performance. *International Symposium on Innovations in Information and Communications Technology*, 94–99. <https://doi.org/10.1109/ISIICT.2011.6149603>
- Cloudera. (n.d.-a). Tuning Spark applications – 5.9.x – Cloudera documentation [Concept]. Retrieved April 10, 2019, from https://www.cloudera.com/documentation/enterprise/5-9-x/topics/admin_spark_tuning.html#spark_tuning__spark_tuning_resource_allocation
- Cloudera. (n.d.-b). Cloudera data analyst training. Retrieved May 6, 2019, from <https://www.cloudera.com/about/training/courses/data-analyst-training.html#?classType=instructor-led&country=US&city=San+Francisco>
- Corrin, A. (2017, December 6). DoD spent \$7.4 billion on big data, AI and the cloud last year. Is that enough? Retrieved from <https://www.c4isrnet.com/it-networks/2017/12/06/dods-leaning-in-on-artificial-intelligence-will-it-be-enough/>
- Don, C. (2009). SQL. In L. Liu, and M. T. Ozsü (Eds.), *Encyclopedia of Database Systems* (pp. 2753–2760). Boston, MA: Springer.
- Franceschi-Bicchierai, L. (2014, June 5). Edward Snowden: The 10 most important revelations from his leaks. Retrieved from <https://mashable.com/2014/06/05/edward-snowden-revelations/>
- Gandomi, A., & Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2), 137–144. <https://doi.org/10.1016/j.ijinfomgt.2014.10.007>
- Gerard. (2015, February 4). Hadoop essentials—The eight things you need to know. Retrieved from <http://workinganalytics.com/hadoop-essentials-by-cloudera-eight-lessons-learned/>
- Gigerenzer, G., & Selten, R. (Eds.). (2002). *Bounded rationality: The adaptive toolbox*. Cambridge MA: MIT Press.
- Google. (n.d.). External metadata indexing guide. Retrieved January 2, 2019, from <https://support.google.com/gsa/answer/6329257?hl=en>
- Gulati, S., & Kumar, S. (2017). *Apache Spark 2.x for Java developers*. Retrieved from <https://books.google.com/books?id=8-ZDDwAAQBAJ>
- Han, J., Pei, J., & Kamber, M. (2011). *Data mining: concepts and techniques*. Waltham, MA: Morgan Kaufmann, Elsevier.
- Harrigan, P. O., Boyd, M. M., Ramsey, E., Ibbotson, P., & Bright, M. (2008). The development of e-procurement within the ICT manufacturing industry in Ireland. *Management Decision*, 46(3), 481–500.

- Hendry, J. (n.d.). Why are tides important in satellite mapping and managing our coasts? Retrieved March 22, 2019, from <https://www.tcarta.com/blog/why-are-tides-important-in-satellite-mapping-and-managing-our-coastt>
- Holmes, A. (2012). *Hadoop in practice*. Shelter Island, NY: Manning.
- Jagalingam, P., Akshaya, B. J., & Hegde, A. V. (2015). Bathymetry mapping using Landsat 8 satellite imagery. *Procedia Engineering*, 116, 560–566. <https://doi.org/10.1016/j.proeng.2015.08.326>
- Jain, A. (2016, September 17). The 5 Vs of big data. Retrieved from Watson Health Perspectives website: <https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/>
- Japan Electronics and Information Technology Industries Association (JEITA). (2002). *Exchangeable image file format for digital still cameras: Exif Version 2.2* (JEITA CP-3451). Retrieved from EXIF website: <https://www.exif.org/Exif2-2.PDF>
- Jin, X., Wah, B. W., Cheng, X., & Wang, Y. (2015). Significance and challenges of big data research. *Big Data Research*, 2(2), 59–64.
- Joseph, R. C., & Johnson, N. A. (2013). Big data and transformational government. *IT Professional*, 15(6), 43–48. <https://doi.org/10.1109/MITP.2013.61>
- Knight, W. C., Pridham, R. G., & Kay, S. M. (1981). Digital signal processing for sonar. *Proceedings of the IEEE*, 69(11), 1451–1506. <https://doi.org/10.1109/PROC.1981.12186>
- Kong, Q. (2017, July 15). Machine learning 15: Using scikit-learn Part 3 – Regression [Blog post]. Retrieved from <http://qingkaikong.blogspot.com/2017/07/machine-learning-15-using-scikit-learn.html>
- Kroenke, D. M., & Auer, D. J. (2013). *Database processing* (15th ed.). New York, NY: Prentice Hall.
- Lyman, P., Varian, H. R., Swearingen, K., Charles, P., Good, N., Jordan, L. L., & Pal, J. (2005). How much information? 2003. Retrieved from <http://Www2.Sims.Berkeley.Edu/Research/Projects/How-Much-Info-2003>
- Marine Sonic Technology, Ltd. (2011). MSTIFF Marine Sonic Technology LTD. Image data file format. White Marsh, VA: Marine Sonic Technology Ltd.
- Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Byers, A. H. (2011). *Big data: The next frontier for innovation, competition, and productivity*. Retrieved from https://bigdatawg.nist.gov/pdf/MGI_big_data_full_report.pdf

- Maresova, P., Sobeslav, V., & Krejcar, O. (2017). Cost–benefit analysis – Evaluation model of cloud computing deployment for use in companies. *Applied Economics*, 49(6), 521–533. <https://doi.org/10.1080/00036846.2016.1200188>
- Martens, B., Walterbusch, M., & Teuteberg, F. (2012). Costing of cloud computing services: a total cost of ownership approach. *2012 45th Hawaii International Conference on System Sciences*, 1563–1572. <https://doi.org/10.1109/HICSS.2012.186>
- Martín, L., Baena, L., Garach, L., López, G., & de Oña, J. (2014). Using data mining techniques to road safety improvement in Spanish roads. *Procedia - Social and Behavioral Sciences*, 160, 607–614. <https://doi.org/10.1016/j.sbspro.2014.12.174>
- Naval Postgraduate School. (n.d.). Information Technology and Communications Services at the Naval Postgraduate School. Retrieved April 3, 2019, from <https://my.nps.edu/web/technology/hpc>
- Normandeau, K. (2013, September 12). Beyond volume, variety and velocity is the issue of big data veracity. Retrieved from <https://insidebigdata.com/2013/09/12/beyond-volume-variety-velocity-issue-big-data-veracity/>
- Olson, D. L. (2018). *Data mining models* (2nd ed.). New York, NY: Business Expert Press.
- Oracle (n.d.-a). GIF metadata format specification. Retrieved January 3, 2019, from https://docs.oracle.com/javase/7/docs/api/javax/imageio/metadata/doc-files/gif_metadata.html
- Oracle (n.d.-b). Database reference. Retrieved March 22, 2019, from https://docs.oracle.com/cd/B28359_01/server.111/b28320/limits002.htm#i287915
- Oracle (n.d.-c). Tutorials. Retrieved March 22, 2019, from Oracle APEX website: <https://apex.oracle.com/en/learn/tutorials/>
- Parameshwaran, I. (2016, July 1). Impact of metadata on Image Performance – Dexecure. Retrieved from <https://dexecure.com/blog/impact-of-metadata-on-image-performance/>
- PayScale. (n.d.). Information technology specialist salary. Retrieved April 11, 2019, from https://www.payscale.com/research/US/Job=Information_Technology_Specialist/Salary
- Peppard, J., Ward, J., & Daniel, E. (2007). Managing the realization of business benefits from IT investments. *MIS Quarterly Executive*, 6(1).
- Reese, R. M., Reese, J. L., & Grigorev, A. (2017). *Java: Data science made easy*. Birmingham, UK: Packt Publishing Ltd.

- Riley, J. (2017). Understanding metadata: What is metadata, and what is it for? Retrieved from National Information Standards Organization website: https://groups.niso.org/apps/group_public/download.php/17446/Understanding%20Metadata.pdf
- Senet, C. M., Seemann, J., Flampouris, S., & Ziemer, F. (2008). Determination of bathymetric and current maps by the method DiSC based on the analysis of nautical X-Band radar image sequences of the sea surface (November 2007). *IEEE Transactions on Geoscience and Remote Sensing*, 46(8), 2267–2279. <https://doi.org/10.1109/TGRS.2008.916474>
- Sharma, S. (2017, September 23). Epoch vs. batch size vs. iterations. Retrieved from <https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9>
- Sivarajah, U., Kamal, M. M., Irani, Z., & Weerakkody, V. (2017). Critical analysis of big data challenges and analytical methods. *Journal of Business Research*, 70, 263–286. <https://doi.org/10.1016/j.jbusres.2016.08.001>
- Smith, M., Szongott, C., Henne, B., & von Voigt, G. (2012). Big data privacy issues in public social media. *2012 6th IEEE International Conference on Digital Ecosystems and Technologies (DEST)*, 1–6. <https://doi.org/10.1109/DEST.2012.6227909>
- University of Wisconsin Data Science (n.d.). What is big data? Retrieved January 8, 2019, from <https://datasciencedegree.wisconsin.edu/data-science/what-is-big-data/>
- U.S. Geological Survey. (n.d.). Sea-floor mapping technology, bathymetry. Retrieved March 22, 2019, from <https://woodshole.er.usgs.gov/operations/sfmapping/bathy.htm>
- Wang, Y., & Wiebe, V. J. (2014). Big data analytics on the characteristic equilibrium of collective opinions in social networks: *International Journal of Cognitive Informatics and Natural Intelligence*, 8(3), 29–44. <https://doi.org/10.4018/IJCINI.2014070103>
- Ward, J., Daniel, E., & Peppard, J. (2007). Building better business cases for IT investments. Retrieved from <https://pdfs.semanticscholar.org/990c/22c22576e1783e5b06cdee377581eda614ee.pdf>
- Weiss, S. M., & Indurkha, N. (1998). *Predictive data mining: A practical guide*. San Francisco, CA: Morgan Kaufmann.
- Woods Hole Coastal and Marine Science Center, (n.d.). Data format library. Retrieved May 5, 2019, from https://woodshole.er.usgs.gov/operations/sfmapping/sonar_xtf.htm

- White, T. (2015). *Hadoop: the definitive guide* (4th ed.). Beijing: O'Reilly.
- Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). *Data mining: Practical machine learning tools and techniques*. San Francisco, CA: Morgan Kaufmann.
- Yee, K.-P., Swearingen, K., Li, K., & Hearst, M. (2003). Faceted metadata for image search and browsing. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 401–408.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California