

東海大學

資訊工程研究所

碩士論文

指導教授：楊朝棟博士

雲端城市交通狀態評估系統應用巨量資料的架構

Cloud City Traffic State Assessment System using a Novel  
Architecture of Big Data

研究生：顏尹臻

中華民國一零四年六月

東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 顏 尹 臻 所提之論文

雲端城市交通狀態評估系統應用巨量資料

的架構

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

召 集 人

林迺衛

簽章

委

員

黃向辰

時文中

呂芳輝

指 導 教 授

楊朝棟

簽章

中華民國 104 年 6 月 30 日

# 摘要

巨量資料的應用蓬勃發展，本論文透過政府的 Open Data 公車動態資訊，取得公車的即時定位，透過公車的定位資訊評估道路交通狀態。本論文建置了一個雲端城市交通狀態評估系統，透過了 Big Data 架構來實作系統，其中運用高可用性的雲端套件 Apache Hadoop 及 Apache Spark，提出一個有效的架構用於處理巨量資料，套用於台灣大道的公車定位資訊，並且透過即時計算公車的即時平均速度與歷史的平均速度做比較，在找尋塞車點的部分應用 Fuzzy C-Means, K-Means, DBSCAN 找出交通堵塞及繁忙的集中點，藉此評估道路的交通狀態。在實作上，本論文透過實驗比較 Hadoop 與 Spark 的差異後選擇 Spark 作為本論文系統之運算架構。數據儲存的部分經由實驗比較 HDFS 中不同 replication 數量的讀寫速度找出最適合設定套用於系統。找尋塞車點部份經由實驗找尋最佳的分群方法，並經由實驗結果選擇最適合本系統之分群方法 K-Means 做為評估方式。在即時評估的實作本系統透過移動平均的概念實作即時交通狀態的評估，在介面部分使用了網頁前端技術呈現雲端城市交通狀態評估系統，本論文系統實際應用於台灣大道並能成功達成交通狀態的評估。

關鍵字: 雲端運算，巨量資料，Spark，HBase，Fuzzy C-Means.

# Abstract

Recently, big data are widely applied to different field. This work presents a cloud city traffic state assessment system using a novel architecture of big data. The proposed system provides the real-time busses location and real-time traffic situation, especially the real-time traffic situation nearby, through open data, GPS, GPRS and cloud technologies. With the high-scalability cloud technologies, Hadoop and Spark, the proposed system architecture is first implemented successfully and efficiently. Next, we utilize three clustering methods, DBSCAN, K-Means, and Fuzzy C-Means to find the area of traffic jam in Taichung city and moving average to find the area of traffic jam in Taiwan Boulevard which is the main road in Taichung city. Finally, several experiments are test. The first experiment indicates that the computing ability of Spark is better than that of Hadoop. The second experiment compares the HDFS processing speed under different number of replication. In the last experiment, we compare the clustering performance of DBSCAN, K-Means, and Fuzzy C-Means so that K-Means is adopted in the proposed system. Based on these experiments, the provided services are present via an advanced web technology.

**Keywords:** Cloud Computing, Big Data, Spark, HBase, Fuzzy C-Means.

## 致謝詞

研究所兩年的生活讓我收穫很多，透過研究所課程的學習以及在論文研究中的精進，讓我對雲端運算及巨量資料的處理技術有深入的了解，並能實際的將這些技術應用到生活中，讓我能實作出一個系統。

完成這篇論文要感謝的人有很多，首先，謝謝我的指導老師楊朝棟教授帶領我進入雲端與巨量資料處理的領域，教導我完成這篇論文，謝謝老師研究所兩年的指導，在學習的過程中不時的督促、討論及指點我正確的方向，讓我獲益匪淺。除了學習外，也讓我能做自己想做的論文題目，雖然過程中遭遇到相當多的困難與挑戰，但是老師總能在我遇到困難時給予意見及指導，如果不是楊老師的提拔及給予的挑戰，我想我可能無法完成這篇論文。也感謝老師讓我能夠在畢業前去北海道參加研討會發表論文成果，學習新事務，拓展國際視野。再來，要感謝陳碩聰老師指導我論文中的數學部分，並在評估方式給予我很多知識與建議，如果沒有碩聰老師我的論文可能無法這麼嚴謹。

另外，感謝抽空前來參加論文口試的委員們，謝謝系上呂芳懌老師對我的研究提供了很多的意見、指導和鼓勵。謝謝林迺衛老師、黃國展老師及時文中老師，給了我很多專業上的想法與建議，因為有您們的意見讓本來較不明確的內容，在重整之後，讓我的論文能更加完整及嚴謹。也感謝兩年來在實驗室遇見的學長姊、同學、學弟、不同實驗室的同學，讓我在這兩年的研究生涯過的很充實。

最後要感謝我的父母，如果沒有他們的支持我沒辦法完研究所的學業，感謝他們從小到大的栽培，讓不是很會讀書的我能走到今天，也謝謝我的姊姊在這之中給予我很多的鼓勵，因為有妳幫忙負擔家計讓我能安心的讀完研究所。

東海大學資訊工程學系 高效能實驗室 顏尹臻 104 年 07 月

# Table of Contents

摘要	I
Abstract	II
致謝詞	III
Table of Contents	IV
List of Figures	VI
List of Tables	VIII
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Contributions . . . . .	5
1.3 Thesis Organization . . . . .	5
<b>2 Background and Related Works</b>	<b>6</b>
2.1 Cloud Computing and Big Data . . . . .	6
2.1.1 Cloud Computing . . . . .	6
2.1.2 Big Data . . . . .	9
2.1.3 NoSQL . . . . .	10
2.2 Hadoop Ecosystem . . . . .	11
2.2.1 Hadoop . . . . .	11
2.2.2 HDFS . . . . .	11
2.2.3 HBase . . . . .	13
2.2.4 HBase Master and Region Servers . . . . .	14
2.2.5 HBase Data Model . . . . .	14
2.2.6 HBase Row Keys . . . . .	15
2.2.7 HBase Column Family . . . . .	16
2.2.8 Cloudera CDH . . . . .	17
2.2.9 Apache Spark . . . . .	18
2.2.10 Spark Application . . . . .	18
2.3 Mathematical Model and Algorithms . . . . .	20
2.3.1 Moving Average . . . . .	20
2.3.2 Fuzzy C-Means . . . . .	20

---

2.3.3	DBSCAN . . . . .	21
2.3.4	K-Means Clustering . . . . .	22
2.4	Related Works . . . . .	22
<b>3</b>	<b>System Design and Implementation</b>	<b>24</b>
3.1	Cloud System Architecture . . . . .	24
3.2	Provided Services . . . . .	25
3.2.1	City Traffic State Assessment System Service Design . . . . .	27
3.2.2	Data Collection Service . . . . .	28
3.2.3	Real Time Process Service . . . . .	30
3.2.4	Data Analysis Service . . . . .	31
3.2.5	Web Application Service . . . . .	32
3.3	System Implementation . . . . .	33
3.3.1	Cluster Deployment . . . . .	34
<b>4</b>	<b>Experimental Results</b>	<b>39</b>
4.1	Experimental Environment . . . . .	39
4.2	Spark and Hadoop MapReduce Performance Comparison . . . . .	40
4.3	Experiment for HDFS read and write speed under various replication	41
4.4	Using Spark to process bus location data under different number of executors . . . . .	46
4.5	Comparison of different clustering ways to find traffic jams grouping	51
4.5.1	DBSCAN . . . . .	51
4.5.2	K-Means . . . . .	52
4.5.3	Fuzzy-C-Means . . . . .	54
4.5.4	Results Compared . . . . .	55
4.6	Cloud City Traffic State Assessment System . . . . .	56
<b>5</b>	<b>Conclusions and Future Work</b>	<b>61</b>
5.1	Concluding Remarks . . . . .	61
5.2	Future Work . . . . .	62
	<b>References</b>	<b>63</b>
	<b>A Hadoop Installation</b>	<b>68</b>
	<b>B HBase Installation</b>	<b>72</b>
	<b>C Cloudera Manager Installation</b>	<b>74</b>
	<b>D Experimental-Spark and Hadoop MapReduce Performance Com- parison Source Code</b>	<b>75</b>
	<b>E Experiment-HDFS read and write speed under various replication Data</b>	<b>78</b>

# List of Figures

1.1	Bus positioning and data transmission schematic . . . . .	2
1.2	The dawn of big data from IDC . . . . .	3
1.3	Taichung City bus Dynamic Positioning . . . . .	4
2.1	Cloud computing service model . . . . .	8
2.2	Big data-4V . . . . .	9
2.3	HDFS architecture . . . . .	12
2.4	The components of HBase . . . . .	13
2.5	HBase service architecture . . . . .	15
2.6	Row of data based on key alphabetically sorted . . . . .	16
2.7	The rows and columns in HBase . . . . .	17
2.8	Cloudera Manage Interface . . . . .	17
2.9	Spark Applicatione . . . . .	19
3.1	Bus positioning and data transmission schematic . . . . .	25
3.2	Block division schematic . . . . .	26
3.3	XML Format . . . . .	27
3.4	Block division schematic apply to Taiwan Boulevard . . . . .	27
3.5	City Traffic State Assessment System Service . . . . .	28
3.6	Real Time Data collection service . . . . .	29
3.7	History Data Collection Service . . . . .	29
3.8	Real Time Process Service . . . . .	30
3.9	Data Analysis Service . . . . .	31
3.10	Web applications service . . . . .	32
3.11	System Deployment Architecture Diagram . . . . .	34
3.12	Cloudera manager status . . . . .	35
3.13	Cloudera manager hosts . . . . .	35
3.14	Hadoop NameNode information . . . . .	36
3.15	MapReduce JobTracker . . . . .	37
3.16	HBase region server status . . . . .	37
3.17	Spark History Server . . . . .	38
3.18	Spark Job . . . . .	38
4.1	Spark and Hadoop Computing cluster . . . . .	41
4.2	Spark and Hadoop MapReduce Performance Comparison . . . . .	41
4.3	The flow chart of HDFS read and write test . . . . .	42



---

4.4	Read and write time duration for MAP=12,24,36,48,60 in various replication number under the case 12GB . . . . .	43
4.5	Read and write time duration for MAP=12,24,36,48,60 in various replication number under the case 60GB . . . . .	44
4.6	Read and write time duration for MAP=12,24,36,48,60 in various replication number under the case 120GB . . . . .	45
4.7	All sizes in 12 map read and write speed . . . . .	46
4.8	All sizes in 12 map read and write average speed of 1GB . . . . .	46
4.9	The flow chart of convBus . . . . .	47
4.10	The flow chart of comBus . . . . .	48
4.11	Using convBus and comBus processing 1-days of Bus data . . . . .	49
4.12	Using convBus and comBus processing 2-days of Bus data . . . . .	49
4.13	Using convBus and comBus processing 3-days of Bus data . . . . .	50
4.14	DBSCAN in epsilon=0.001 Clustering results . . . . .	51
4.15	DBSCAN in epsilon=0.002 Clustering results . . . . .	52
4.16	K-MEANS in Iterations=100 Clustering results . . . . .	53
4.17	K-MEANS in Iterations=1000 Clustering results . . . . .	53
4.18	Fuzzy-C-Means in Iterations=50 Clustering results . . . . .	54
4.19	Fuzzy-C-Means in Iterations=100 Clustering results . . . . .	55
4.20	Comparison of DBSCAN,K-Means,Fuzzy-C-Means . . . . .	56
4.21	Cloud City Traffic State Assessment System functions . . . . .	57
4.22	Web UI Function Menu . . . . .	57
4.23	History Bus Travel each block of time use waveform display . . . . .	58
4.24	History Bus Travel each block of time use Use long bar graph display	58
4.25	Real-time assessment traffic in Taiwan Boulevard using WEB presentation . . . . .	59
4.26	Real-time assessment traffic in Taiwan Boulevard using WEB presentation . . . . .	59
4.27	Use DBSCAN clustering on web views . . . . .	60
4.28	Use K-Means clustering on web views . . . . .	60

# List of Tables

3.1	Software Specification . . . . .	33
4.1	Experimental environment . . . . .	40
E.1	Write 12G in HDFS [Map-Size-Replication-R/W] . . . . .	78
E.2	Read 12G in HDFS [Map-Size-Replication-R/W] . . . . .	80
E.3	Write 60G in HDFS [Map-Size-Replication-R/W] . . . . .	83
E.4	Read 60G in HDFS [Map-Size-Replication-R/W] . . . . .	85
E.5	Write 120G in HDFS [Map-Size-Replication-R/W] . . . . .	87
E.6	Read 120G in HDFS [Map-Size-Replication-R/W] . . . . .	90

# Chapter 1

## Introduction

To conform the growth of Big Data[1] application, almost all major industries and companies of the world invested the area of Big Data analysis. With the technology of Big Data processing, the companies can analyze and classify the massive data they have, finally come out the valuable information. For analyzing the condition of transport, the government through GPS positioning to record the relevant map location of most urban public transport systems, coupled with the back-end processing of data transmission via GPRS or 3G to tracking the status of transport. This service provides users the information of location and waiting time, the most commonly utilized for bus service, many cities offer similar services.[2] This research collects the open source of transportation data the government provided to proceed the processing of Big Data and then analyzing instantly via Cloud Computing[3] architecture. Through the calculation of GPS coordinates bus, the research can assess the extent of road congestion and provide the information of passers-road conditions. Figure 1.1 shows bus positioning and data transmission schematic.[4]

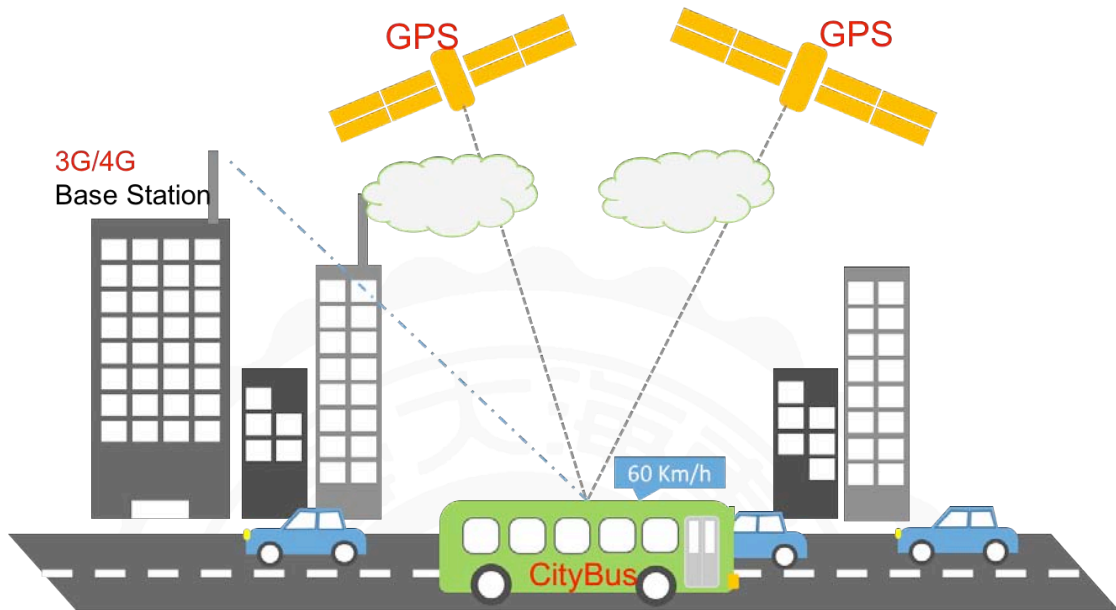


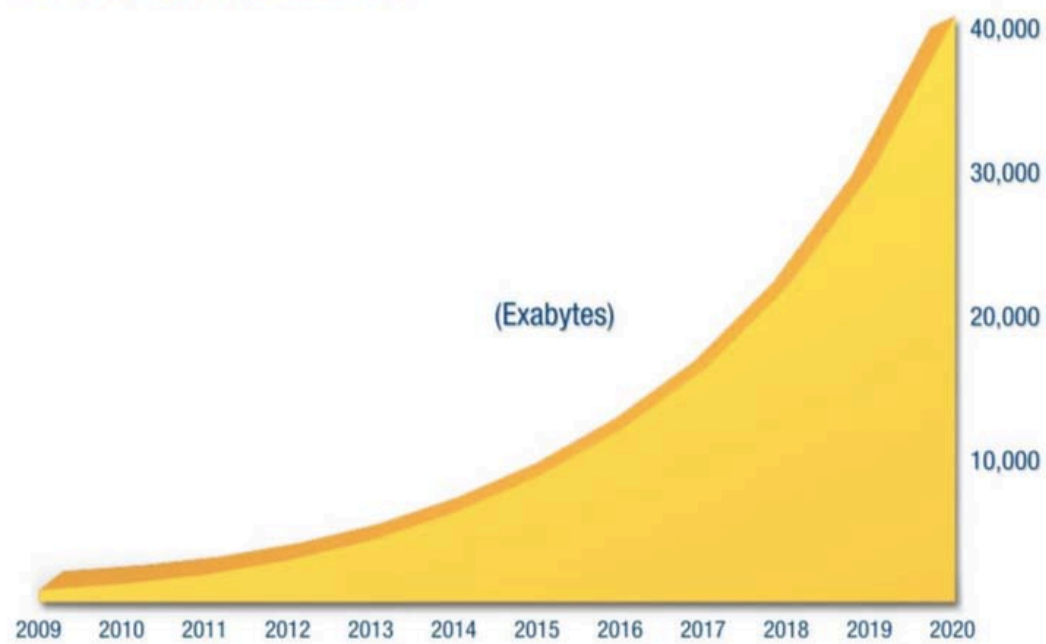
FIGURE 1.1: Bus positioning and data transmission schematic

## 1.1 Motivation

To reflect the booming of big data application, nowadays many industries such as healthcare, manufacturing, telecommunications, retail, energy, transportation, automotive, security and other applications market imports the technology of Big Data analysis. Through Big Data processing technology, those industries can filter out the useless data that is hardly to perform in the past, then gain the advantage of valuable information. IDC's Digital Universe Study[5] analysis that global data amount continue to grow rapidly. It is expected the data amount would breakthrough 40000Exabyte (EB, 1EB = 1 Million Terabyte) in the year of 2020. shown in Figure 1.2. Data analysis is a quite common concept in life that performed by collecting all the information and data, analyze then come out the result as the basis of future action and decision.

In recent years, processing and analysis of data has become more complex, the amount of data is getting bigger, and more and more diverse species. It is more harder to gather useful information from the data. Hence, to analyze transportation data and gain the valuable information for providing passengers or even

## The Digital Universe: 50-fold Growth from the Beginning of 2010 to the End of 2020



Source: IDC's Digital Universe Study, sponsored by EMC, December 2012

FIGURE 1.2: The dawn of big data from IDC

government transportation planning department the information becomes a important issue. Many countries now have many city public transportation methods, such as subway, light rail tram, bus ... and so on. The highest density of road network is bus. Bus goes through most of the city's routes, and most of city bus has been imported intelligent transportation solutions that through GPS positioning technology to locate buses in the city. With the positioning of the bus, user can access the bus arrive time and immediate location update. This research collects open data from public bus information then analyzes the data to know the traffic condition in the bus routes, includes information of the extent of congestion, average speed, estimated time for exercise, etc.

This research considers the bus runs density for keeping update the traffic condition within specific route, and through real-time motion analysis for passers-by to access the traffic condition of the section they select immediately. With this analysis, the passers-by can evaluate the traveling time. Also, this research through the data collection and analysis of long-term route positioning to assess

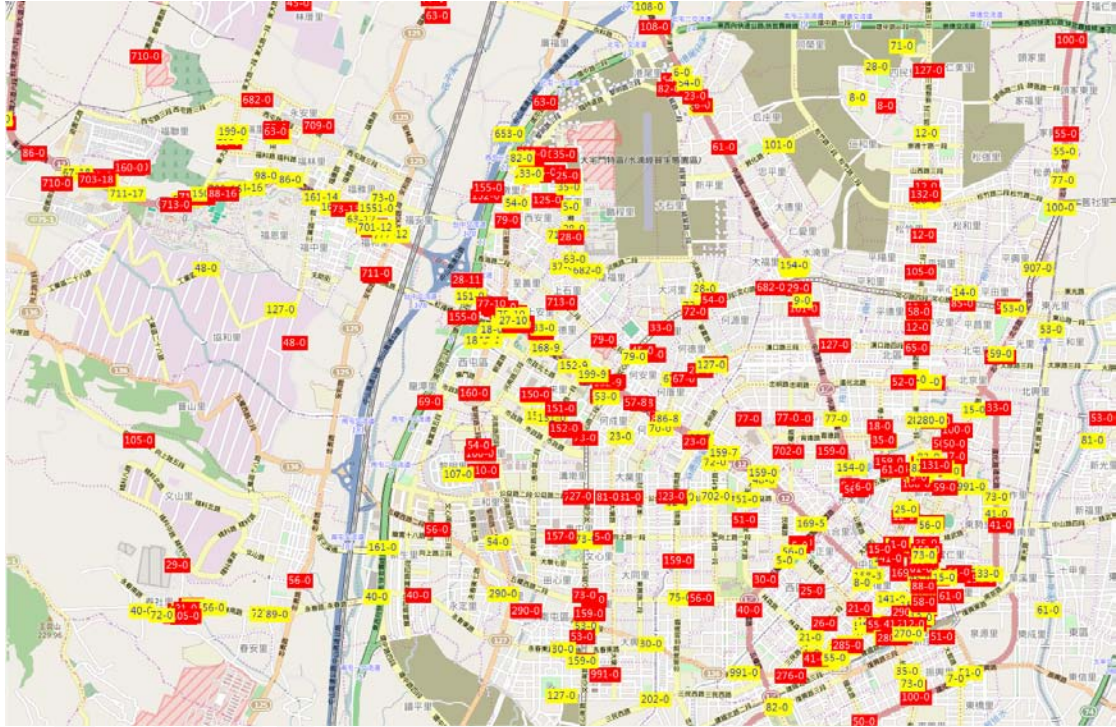


FIGURE 1.3: Taichung City bus Dynamic Positioning

the long-term road traffic conditions, provides benefit for the Ministry of Transportation to improve the traffic situation. Because location-based information is continuously updated and bus lines in the city is quite complicated, it requires a good architecture to perform the processing of big data immediately and analyzes the long-term collection data. Therefore, this research is based on cloud computing platform, through the big data analysis technology to deal with real-time information and massive data collected . This research through Big Data processing platform on cloud to handle the open data of bus information and distributed Apache Hbase cloud architecture to dispersed data storage on several hosts, which is able to balance the massive data stored. The platform in this research contains a algorithm design to calculate bus runs speed via movement of bus positioning. The result can be used to evaluate the traffic situation whereby sections. Furthermore, this research through Apache Spark[6, 7] architecture to Implementing algorithms. With distributed processing framework, it' s able to deal with a massive real-time information effectively and handle big data Statistics and Analysis.

## 1.2 Contributions

In this article we build a cloud system to processing and analysis of a huge amount of Bus Dynamic data. We implement a distributed computing and analysis system based on cloud computing architecture to processing big data. In this work, we present a cloud city traffic state assessment system using a novel architecture of big data. With the high-scalability cloud technologies, Hadoop and Spark, the proposed system architecture for big data and services is implemented . Based on the comparison between real-time buses average velocity and the past average velocity, Fuzzy C-Means , K-Means, DBSCAN is applied to find out the heavy traffic area or traffic jam in each block of Taiwan Boulevard.

## 1.3 Thesis Organization

In Chapter 2, we will describe some background information, including Cloud Computing, Big Data, NoSQL, Apache Hadoop, HDFS, Apache HBase , Apache Spark and Cloudera CDH. In Chapter 3, we will introduce our experimental environment and system architecture, experimental methods and the overall implementation of system. Chapter 4 shows the experimental results and analyses. Chapter 5 provides conclusions and future work of this thesis.

# Chapter 2

## Background and Related Works

### 2.1 Cloud Computing and Big Data

#### 2.1.1 Cloud Computing

Cloud computing is an Internet-based computing which provides services of computers and other devices on demand via computer networks hardware and software. Cloud computing describes a new Internet-based IT service model, usually involves with Internet to provide dynamic extended and virtualized resources. Users no longer need to know the details of the Cloud or have the appropriate expertise, but also have no direct control. A good starting point for a definition of cloud computing is the definition issued by the U.S. National Institute of Standards and Technology (NIST) September, 2011. It starts with: Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

Five essential characteristics:

- Provide services according to the needs.



- Anytime, anywhere access by any network device.
- Users share resource pool.
- Quick redeployment.
- Can be monitored and measured.

Three service models as shown in Figure 2.1:

- Software as a Service (SaaS): Consumers use software deployed in the cloud or data stored in the cloud, without managing cloud infrastructure and programming execution environment. Consumers no longer need to install software on their computer, therefore reducing maintenance and software support issues.
- Platform as a Service (PaaS): PaaS provides a computing platform typically includes the operating system, database, programming language execution environment and web server. Application developers can develop and run their software solutions on a cloud platform without the cost and complexity of buying and managing the underlying hardware and software layers.
- Infrastructure as a Service (IaaS): As the most basic cloud service model, IaaS provides physical machines, virtual machines or other resources; consumers can fully control the allocated resources, but cannot control the cloud infrastructure. [3].

Four deployment models:

- Public Cloud: Third-party cloud infrastructure for use in general public or a large industry group. The organization has to sell its cloud services and system services, allowing consumers to rent for deployment and use of cloud services.

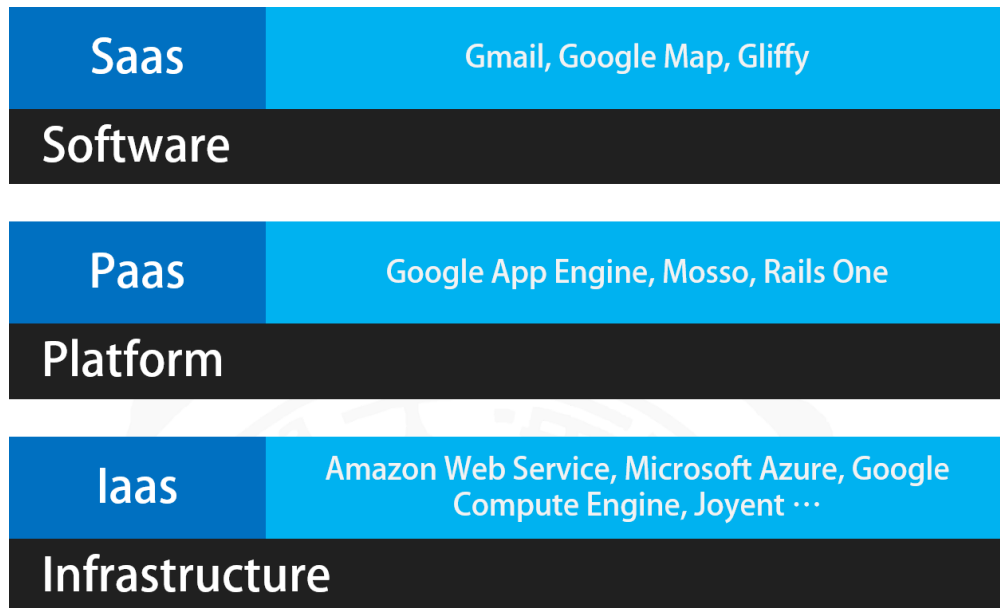


FIGURE 2.1: Cloud computing service model

- **Private Cloud:** The cloud infrastructure and software resources are built within the firewall, for all departments within an organization or enterprise to share resources in the data center. Private cloud infrastructure is fully operational for a specific organization; its managers may be in the organization itself, or a third party; it may be controlled within the organization, but may also be outside the organization.
- **Community Cloud:** Community cloud shared by several organizations to support a particular community with common concerns, such as security requirements, policy, and compliance considerations. Managers may be in the organization itself, but also a third party; it may be managed within the organization, but may also be outside the organization.
- **Hybrid Cloud:** A cloud may consist of two or more cloud infrastructures, which includes a private cloud, community cloud, public cloud and so on. These systems keep their own independence, but combine standardized or enclosed proprietary technologies with each other to ensure portability of data and applications.

## 2.1.2 Big Data

Big data is a broad term for data sets so large or complex that traditional data processing applications are inadequate. The development of big data has four directions:

- Volume: Production, processing, preservation for large amounts of data.
- Velocity: Generation speed of data and processing speed of computer.
- Variety: Data type including structural data or non-structural data, text, video, web pages, streaming.
- Veracity: Data reliability and data quality.

Big data is a large and complicate problem so that we face many challenges including analysis, capture, curation, search, sharing, storage, transfer, visualization, and information privacy. Analysis of big data collected from small data sets can find new correlations, to "spot business trends, increase income, enhance the competitive power, adjust research results, prevent diseases, combat crime, report real-time traffic and so on. as shown in Figure 2.2[1]:

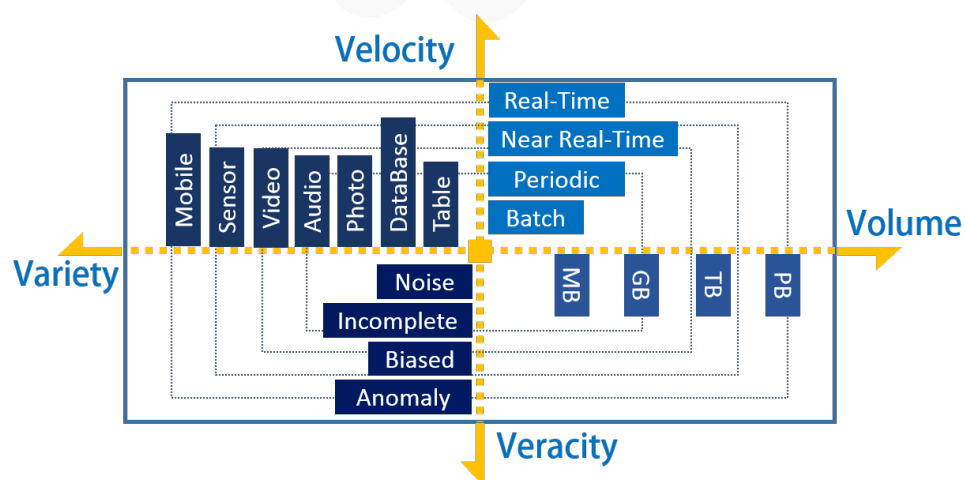


FIGURE 2.2: Big data-4V

### 2.1.3 NoSQL

NoSQL is next generation database. Carlo Strozzi used the term NoSQL in 1998 to name his lightweight, open-source relational database that did not expose the standard SQL interface. NoSQL mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable. It provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Motivations for this approach include simplicity of design, horizontal scaling, and finer control over availability. The data structures used by NoSQL databases (e.g. key-value, graph, or document) differ from those used in relational databases, making some operations faster in NoSQL and others faster in relational databases. The particular suitability of a given NoSQL database depends on the problem it must solve.

NoSQL databases are increasingly used in big data and real-time web applications. NoSQL systems also support SQL-like query languages. Many NoSQL stores compromise consistency (in the sense of the CAP theorem) in favor of availability and partition tolerance. Barriers to the greater adoption of NoSQL stores include the use of low-level query languages, the lack of standardized interfaces, and huge investments in existing SQL. Most NoSQL stores lack true Atomicity, Consistency, Isolation, Durability (ACID) transactions.

Strozzi suggests that, as the current NoSQL movement departs from the relational model altogether; it should therefore have been called more appropriately 'NoREL' ; referring to 'No Relational'. Eric Evans reintroduced the term NoSQL in early 2009 when Johan Oskarsson of Last.fm organized an event to discuss open-source distributed databases. The name attempted to label the emergence of an increasing number of non-relational, distributed data stores. Most of the early NoSQL systems did not attempt to provide atomicity, consistency, isolation and durability guarantees, contrary to the prevailing practice among relational database systems. [8].

## 2.2 Hadoop Ecosystem

### 2.2.1 Hadoop

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures. [9, 10].

The project includes these modules:

- Hadoop Common: The common utilities that support the other Hadoop modules.
- Hadoop Distributed File System (HDFS): A distributed file system that provides high-throughput access to application data.
- Hadoop YARN: A framework for job scheduling and cluster resource management.
- Hadoop MapReduce: A YARN-based system programming model for massive data processing of large data.

### 2.2.2 HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost

hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is now an Apache Hadoop subproject. HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode. Figure 2.3 shows the architecture of HDFS[11].

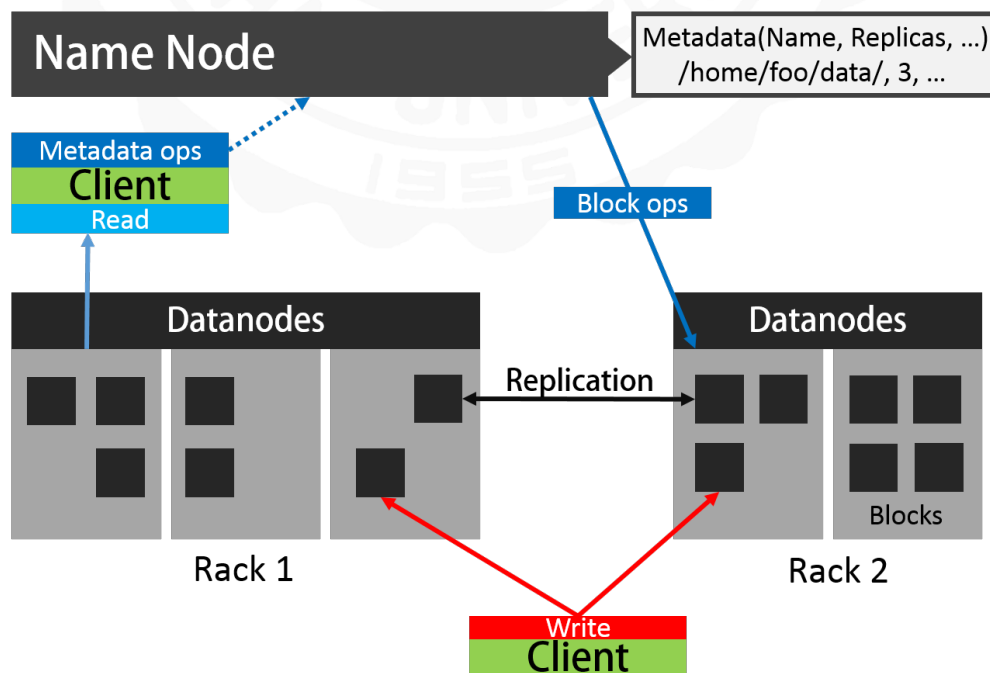


FIGURE 2.3: HDFS architecture

### 2.2.3 HBase

HBase, is an open source, high reliability, high performance, scalable, and not based on the relational model based on distributed repository for storing large scale unstructured data. [12, 13, 14, 15, 16]. HBase is a distributed, versioned, non-relational database (i.e., NoSQL) modeled after Google's Bigtable: A Distributed Storage System for Structured Data. Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS. HBase data actually stored in HDFS. Figure 2.4 shows the different components of HBase, and how it can work with existing systems.

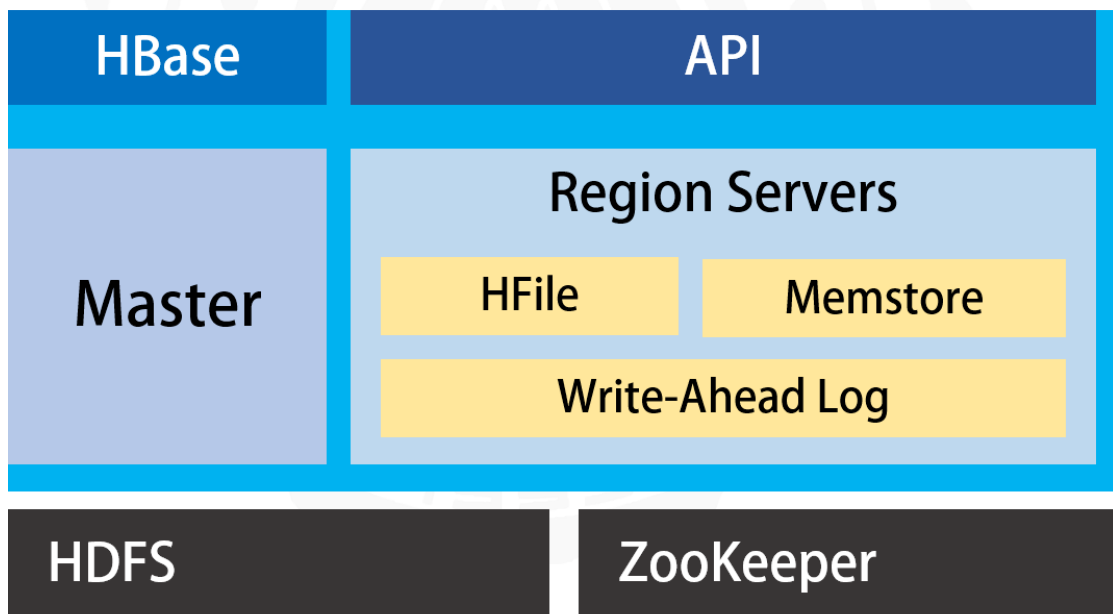


FIGURE 2.4: The components of HBase

Features[17, 18]:

- Strongly consistent reads/writes: HBase is not eventual consistent data storage; thus, it is ideal for the task of gathering, such as for high-speed counter.
- Automatic sharding: HBase tables are distributed over the cluster through regions that can be automatic separated and allocated as data grow.
- Hadoop/HDFS Integration: HBase is in support of HDFS as its distributed file system.

- MapReduce: HBase supports massively parallel processing through MapReduce by using HBase as sources and sinks.
- Java Client API: HBase supports easy-to-use Java APIs for programming.
- Thrift/REST API: HBase supports REST and Thrift as non-Java front ends.
- Block Cache and Bloom Filters: HBase supports block caches and Bloom filters for high-capacity query optimization.
- Operational Management: HBase provides insight into the operation and JMX metrics by the built-in web page.

#### 2.2.4 HBase Master and Region Servers

Master server (master) region is also responsible for the entire region server load balancing, unloading the busy servers, and moving a busy region to the region was less occupied by the server. Primary server is not part of the actual data storage or retrieval path, which is responsible for coordinating and maintaining load balancing cluster state, but does not provide any services to the region information server or client, therefore, can actually reduce its load. In addition, it is to deal with the changes and outline other metadata operations, such as: data tables and establishment of column family. Responsibility of region server is to serve their region, responsible for all requests to read and write, and is divided over the size threshold setting range of quality regional client to communicate directly with them to deal with all the work related to information. . Figure 2.5 shows the architecture of HBase service[19].

#### 2.2.5 HBase Data Model

Column (row) is the basic unit of data model. One or more rows form a (row), each row is identified through a unique row key. Several columns (rows) will form a data table (table) or a lot of data sheets. There may be many versions of each



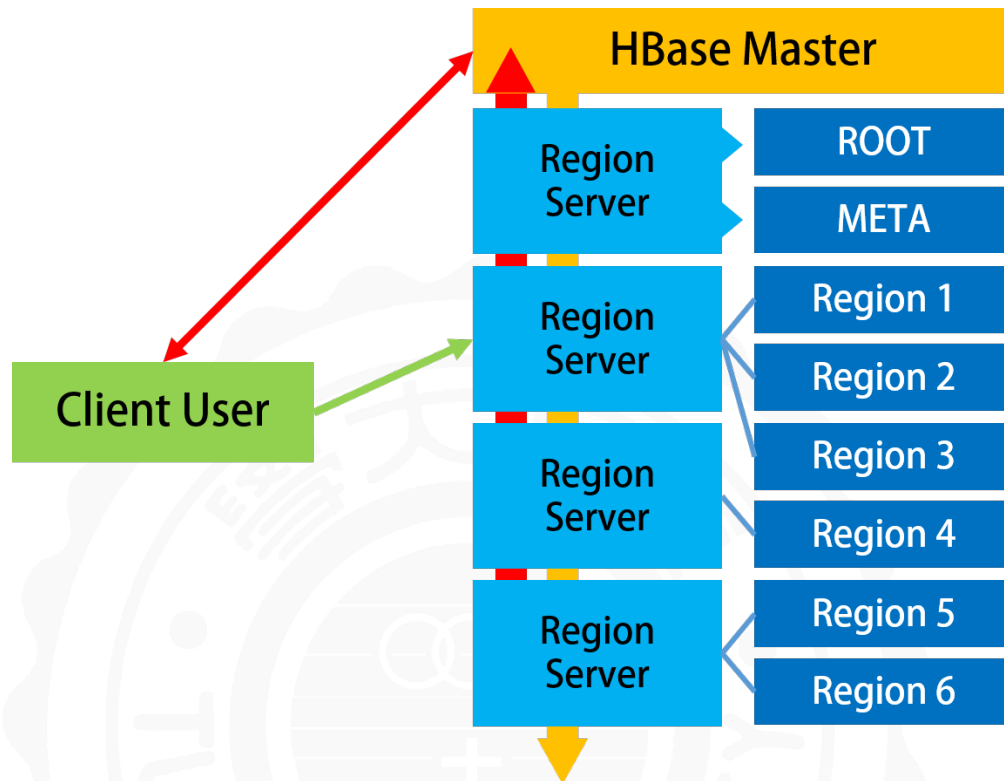
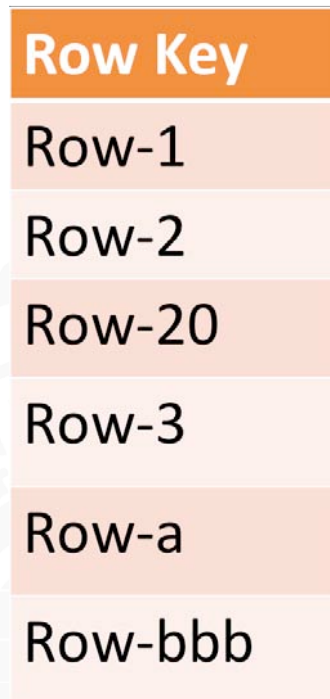


FIGURE 2.5: HBase service architecture

row, each with a unique value will be stored on a separate storage unit which looks like a description of a traditional database, but added an extra dimension (dimension) to allow each storage unit has multiple versions. Data access mode is as follows: Table, RowKey, Family, Column, Timestamp  $\rightarrow$  Value.[19].

### 2.2.6 HBase Row Keys

Row key is an array of bytes from HBase is considered, but it must be a string. In alphabetical order, each key will use binary code to do more, from left to right verbatim yuan for comparison, the length of the short string sorting in front, as \* because row0 ... less than row1 ..., so both What followed, it will be sorted to the top. Data will be sorted according to row key. It acts like a connected database management system type master Kam and is also unique. Each row key is unique, otherwise it will be updated to the same data row. Row key can consist of any characters. As shown in Figure2.6[19].



Row Key
Row-1
Row-2
Row-20
Row-3
Row-a
Row-bbb

FIGURE 2.6: Row of data based on key alphabetically sorted

### 2.2.7 HBase Column Family

Row is composed by columns, which can be classified into the respective column family. This helps to establish the scope of the semantic or partial information between certain functions and give them (example: compression) or just that they remain in the memory. All columns in a column family are stored on the same low-level files called HFile. One needs to define column family when establishing data table and should be to reduce the modifier and should not define too many column family. Column family name must consist of the characters can be displayed, and use other names or values, there are significantly different. \* It can be seen in Figure, row data library in general, with respect to the difference in HBase column-oriented design. . In Figure 2.7 we can see the difference of design of rows in the general database as opposed to that of column-oriented HBase[19].

	column A (int)	column B (varchar)	column C (boolean)	column D (date)
row A				
row B				
row C			NULL?	
row D				

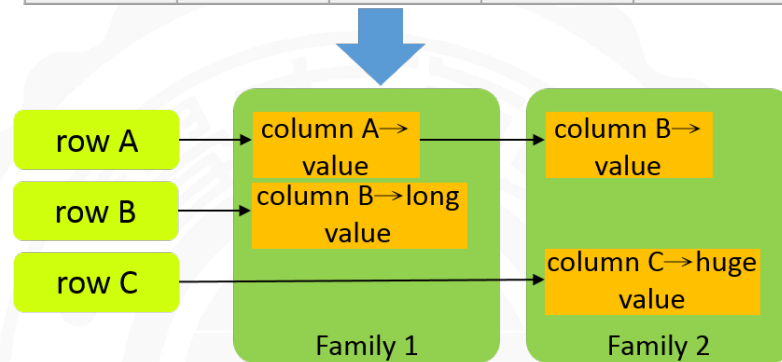


FIGURE 2.7: The rows and columns in HBase

## 2.2.8 Cloudera CDH

CDH is the world's most complete, tested, and popular distribution of Apache Hadoop and related projects. CDH is 100% Apache-licensed open source and is the only Hadoop solution to offer unified batch processing, interactive SQL, and interactive search, and role-based access controls. More enterprises have downloaded CDH than all other such distributions combined[20, 21]. As shown in Figure 2.8.

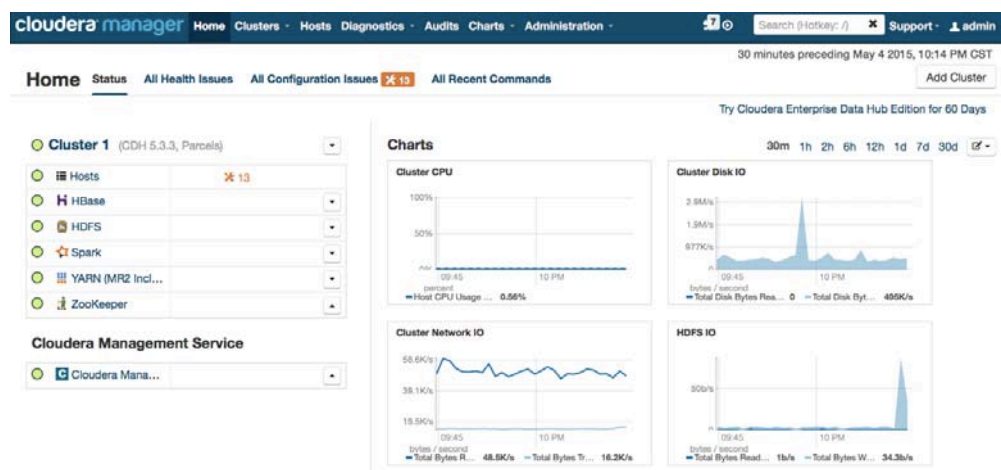


FIGURE 2.8: Cloudera Manage Interface

### 2.2.9 Apache Spark

Apache Spark is an open-source cluster computing framework originally developed in the AMPLab at UC Berkeley. In contrast to Hadoop's two-stage disk-based MapReduce paradigm, Spark's in-memory primitives provide performance up to 100 times faster for certain applications. By allowing user programs to load data into a cluster's memory and query it repeatedly, Spark is well suited to machine learning algorithms. Spark requires a cluster manager and a distributed storage system. For cluster management, Spark supports standalone (native Spark cluster), Hadoop YARN, or Apache Mesos. For distributed storage, Spark can interface with a wide variety, including Hadoop Distributed File System (HDFS), Cassandra, OpenStack Swift, and Amazon S3. Spark also supports a pseudo-distributed local mode, usually used only for development or testing purposes, where distributed storage is not required and the local file system can be used instead; in this scenario, Spark is running on a single machine with one executor per CPU core. Spark has over 465 contributors in 2014, making it the most active project in the Apache Software Foundation and among Big Data open source projects[7].

### 2.2.10 Spark Application

Spark Application is the applications executed on a computing architecture of Spark. The architecture of Spark Application consists of two main components: driver program (SparkContext) and executor. Spark can be performed in the machine as well as clusters, but most of all operations are in the cluster. Application can be performed by Spark in three modes: Spark Standalone, YARN, and mesos. These three clustering models can provide computing resources for Spark and be able to manage these resources, which are available to the executor and the driver program to use. Based on the execution of the driver program in Spark Application is focused on the cluster, Spark Application can be divided into Cluster mode

and Client mode. As shown in Figure 2.9

Spark terms:

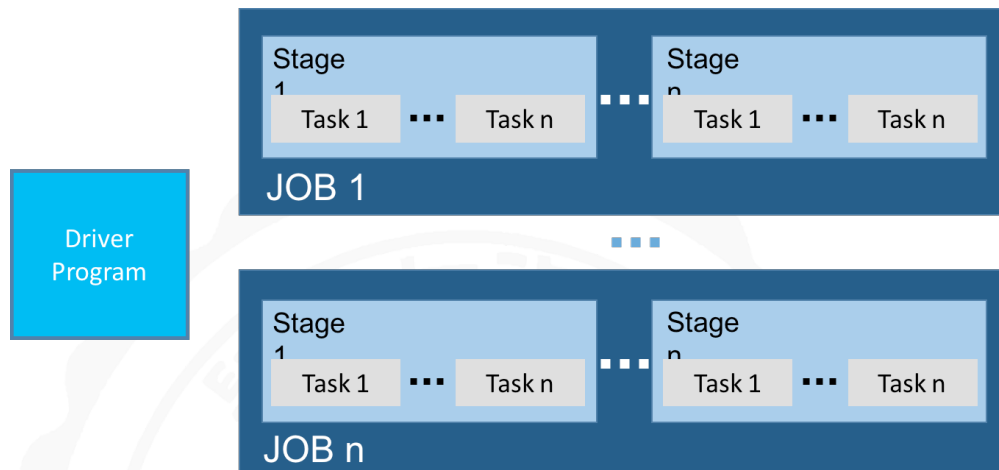


FIGURE 2.9: Spark Application

- Application: Operations on the user program on the Spark, which contains a driver program and operate in a centralized cluster executor.
- Driver Program: Driver program is the application execution of main () function which produces SparkContext.
- Executor: Executor is the Application Process executed on each worker node, and this is responsible for the implementation of Process task, and is responsible for the data stored in the memory and hard drive. Each one will have a executors Application.
- Cluster Manager: Cluster manager is the requirement of external service resources on each cluster (for example: Standalone, Mesos, Yarn)
- Worker Node: The nodes that can perform application.
- Task: The unit work performed on the executor.
- Job: Composition comprising a plurality of parallel computing in Task.
- Stage: Each Job is divided into a lot of task and each group task is called Stage, also called TaskSet.
- RDD: Basic computational unit of Spark

## 2.3 Mathematical Model and Algorithms

### 2.3.1 Moving Average

Suppose

$$S = \{x_i, x_{i+1}, x_{i+2}, \dots, x_{N+(i-1)}\}$$

is a subset of sample values and the size of the subset is set to N. Then, a new series of

$$\{A_1, A_2, \dots, A_i, \dots\}$$

is called the moving average of S which is obtained by the following calculation:

$$A_i = \frac{x_i + x_{i+1} + x_{i+2} + \dots + x_{N+(i-1)}}{N}$$

### 2.3.2 Fuzzy C-Means

The Fuzzy C-Means (FCM)[22] attempts to partition a finite collection of n elements

$$X = \{x_1, \dots, x_n\}$$

into a collection of c fuzzy clusters with respect to some given criterion. Given a finite set of data, the algorithm returns a list of p cluster centres

$$C = \{c_1, \dots, c_p\}$$

and a partition matrix

$$U = [u_{ij}]_{p \times n}$$

where each element  $u_{ij}$  tells the degree to which element  $x_j$  belongs to cluster  $c_i$

Then FCM aims to minimize an objective function J:

$$J(U, c_1, c_2, \dots, c_p) = \sum_{i=1}^p \sum_{j=1}^n (u_{ij})^m \text{dist}(c_i, x_j)^2$$

where

$$m \in [1, \infty)$$

represent weighting;

$$\text{dist}(c_i, x_j)$$

denotes the distance between  $c_i$  and  $x_j$ . By introducing Lagrange multiplier  $\lambda_i$  the above objective function is rewritten as

$$\begin{aligned} & J_{new}(U, c_1, c_2, \dots, c_p, \lambda_1, \dots, \lambda_n) \\ &= J(U, c_1, c_2, \dots, c_p) + \sum_{j=1}^n \lambda_j \left( \sum_{i=1}^p u_{ij} - 1 \right) \\ &= \sum_{i=1}^p \sum_{j=1}^n (u_{ij})^m \text{dist}(c_i, x_j)^2 + \sum_{j=1}^n \lambda_j \left( \sum_{i=1}^p u_{ij} - 1 \right) \end{aligned}$$

and the optimal cluster is

$$c_i = \frac{\sum_{j=1}^n (u_{ij})^m x_j}{\sum_{j=1}^n (u_{ij})^m}$$

### 2.3.3 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a density-based data clustering algorithm. DBSCAN requires two parameters: (eps) and the minimum number of points required to form a dense region (minPts). It starts with an arbitrary starting point that has not been visited. This point's  $\epsilon$ -neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as noise. Note that this point might later be found in a sufficiently sized  $\epsilon$ -environment of a different point and hence be made part of a cluster. If a point is found to be a dense part of a cluster, its  $\epsilon$ -neighborhood is also part of that cluster. Hence, all points that are found within the  $\epsilon$ -neighborhood are added, as is their own  $\epsilon$ -neighborhood when they are also dense. This process continues until the density-connected cluster is completely

found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.[23, 24]

### 2.3.4 K-Means Clustering

k-Means clustering is a method of cluster analysis which aims to partition  $n$  observations into  $k$  clusters in which each observation belongs to the cluster with the nearest mean. Given a set of observations  $(x_1, x_2, \dots, x_n)$  where each observation is a  $d$ -dimensional real vector, k-means clustering aims to partition the  $n$  observations into  $k$  ( $k \leq n$ ) sets  $\{s_1, s_2, \dots, s_k\}$  so as to minimize the within-cluster sum of squares

$$\arg \min \sum_{i=1}^k \sum_{x_j \in s_i} \|x_j - \mu_i\|^2,$$

where  $\mu_i$  is the mean of points in  $s_i$ . [25, 26]

## 2.4 Related Works

In recent years, the road traffic flow are rapidly increasing and city's traffic situation is worsening so that the traffic congestion has become a common problem in the world. To solve this problem, many researches are proposed. Lee et al. 2006[27] proposed and test the new method of calculating the optimal link speed for the collected traffic information from probe cars. Zeng et al. 2014[28] developed a multisensor traffic state assessment system providing a novel and robust algorithms to solve the problem that the sensors usually acquire incomplete data of traffic data. The traffic state assessment based on the fusion decision model of rough sets and cloud is applied to the actual road traffic condition, and the evaluation accuracy is above 98%.

In this work, we apply big data to provide the service of real-time traffic situation and then solve the traffic congestion. Since the big data are large and



complex, challenges of traditional data processing include analysis, capture, curation, searching, sharing, storage, transfer, visualization, and so on. To solve these challenges, it is necessary to introduce technologies of big data including high-volume, high-velocity and high-variety technologies.

Barbierato et al, 2014[29]. Adopted NoSQL to store big data since NoSQL provides a mechanism for storage and retrieval of data that is better than the tabular relations used in relational databases. Zhang et al, 2013[30] used HBase to store big data since HBase provides the distributed data storage cluster through HDFS in Hadoop. Their experiments indicate a good performance. Thus, both NoSQL and HBase have advantages in storing big data.

Gu et al, 2014[31] applied Hadoop MapReduce to process big data successfully. However, for processing real-time big data, Hadoop MapReduce is unable to show superiority due to the fact that Hadoop MapReduce needs more time on starting JOB and then distribute JOB to each node. To improve this drawback, the authors adopted the IN memory of Spark to achieve high-speed computation since Spark's in-memory primitives provide performance up to 100 times faster for certain applications. In addition, Spark requires a cluster manager and a distributed storage system. For cluster management, Spark supports standalone (native Spark cluster), Hadoop YARN, or Apache Mesos. For distributed storage, Spark can interface with a wide variety, including Hadoop Distributed File System (HDFS), Cassandra, OpenStack Swift, and Amazon S3. Thus, our proposed architecture will combine Spark with the distribution computation of Hadoop YARN to enhance performance.[32]

## Chapter 3

# System Design and Implementation

This thesis develops a cloud city traffic state assessment system to provide drivers real-time traffic situation and traffic assessment. The proposed system provides drivers the real-time information of location and real-time traffic situation, especially the real-time traffic situation nearby, through open data, GPS, GPRS and cloud evaluated bus state. Furthermore, the proposed system also provide the traffic assessment and push traffic improvement ahead through these real-time information collected in advance. Because the real-time data collected is huge and from different attributes, this thesis utilizes a novel cloud architecture of big data to store, process, and analyze a huge amount of real-time data and thus provides useful information.

### 3.1 Cloud System Architecture

Based on the high-scalability cloud technologies, Hadoop and Spark, the proposed system architecture is shown in Figure 3.1. The architecture can be mainly divide into two parts: data storage and data analysis and computation. In data storage, we used Hadoop HDFS to be a cloud storage basis and then a NoSQL database

for big data, namely HBase, is utilized to establish the cluster of distributed data storage including structured and unstructured data based on the Hadoop HDFS. In data analysis and computation, we adopted Spark to meet the requirement of the high-speed real-time computation since Spark can quickly assess and analyze the data stored in HBase.

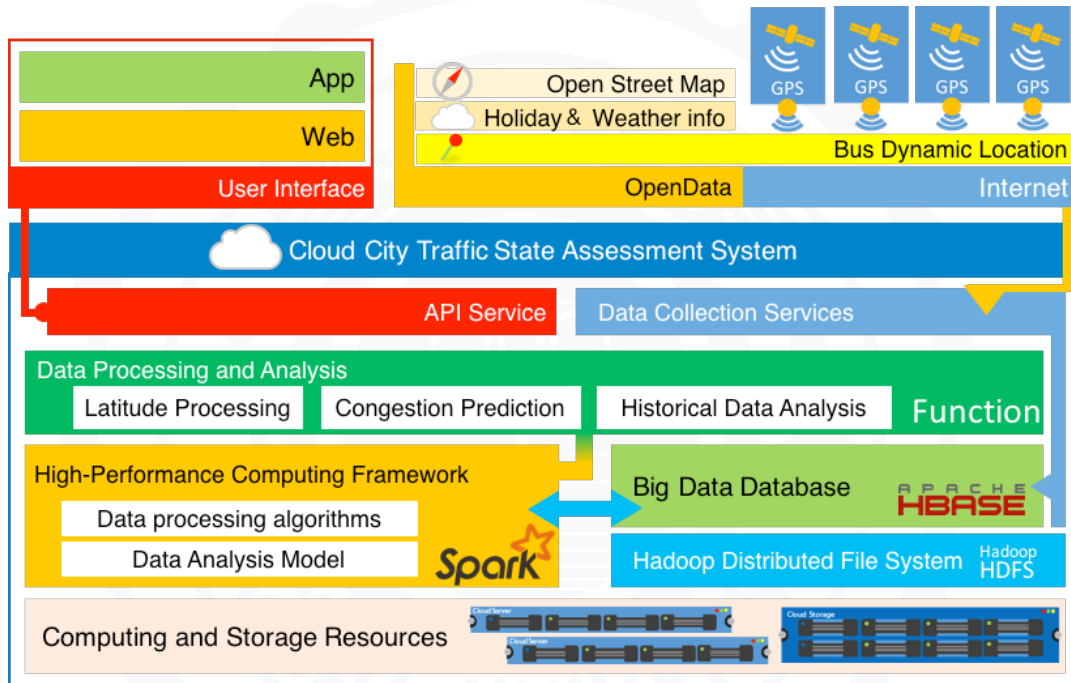


FIGURE 3.1: Bus positioning and data transmission schematic

## 3.2 Provided Services

In Taichung City Bus Dynamic System, there are 245 bus routes and over 1000 bus drive in these route at rush hour. Taichung City Bus Dynamic System provides each bus information in XML format including bus position, bus number, bus route, bus velocity, and so on. In addition, XML format is shown in Figure 3.3 the Taichung City Bus Dynamic System updates each bus position every 20 seconds. For an instance, this thesis implement the proposed Cloud City Traffic State Assessment System in Taiwan Boulevard Taichung, Taiwan. First of all, the proposed system used python programming to captures the bus position updating data per second into our storage basis in Hadoop HBase. In this step, the host

road is randomly divide into several blocks according to the intersection with other roads is shown in Figure 3.2 and Figure 3.4. We apply moving average in subsection 2.3.1 to evaluate the real-time traffic state in three levels, Jam、Normal、Smooth, within a block. Suppose  $x_n$  denotes the duration of nth bus within ith block with size N and

$$A_{t+k} = \frac{1}{n} \sum_n x_n^{t+k} \quad (3.1)$$

is the average of n durations within ith block at time t+k. Then, a new series of

$$S_i^t = \{A_t, A_{t+1}, \dots, A_{t+k}\} \quad (3.2)$$

is called the moving average within ith block and at time t. If  $S_i^t$  is an increasing set and the historical data also show the bad traffic state, the traffic state of the block is Jam. If  $S_i^t$  is a decreasing set and the historical data also show the smooth traffic state, the traffic state of the block is Smooth.

To precisely determine the area with traffic congestion in Taichung city, we then cluster the buses using fuzzy c-means in subsection 2.3.2 as follow:

$$c_i = \frac{\sum_{j=1}^n (u_{ij})^m x_j}{\sum_{j=1}^n (u_{ij})^m} \quad (3.3)$$

where  $m \in [1, \infty)$  represent weighting;  $dist(c_i, x_j)$  denotes the distance between cluster center  $c_i$  and bus  $x_j$ ;  $u_{ij}$  tells the degree to which bus  $x_j$  belongs to cluster  $c_i$ .

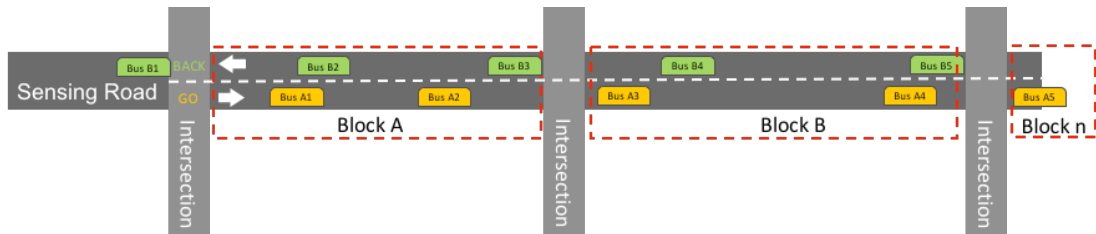


FIGURE 3.2: Block division schematic

```

- <BusDynInfo>
- <EssentialInfo>
- <Location>
- <name>Maxwin</name>
- <CenterName>
- <Location>
- <UpdateTime>2015-04-01 05:08:48</UpdateTime>
- <CoordinateSystem>
- </EssentialInfo>
- <BusInfo>
- <BusData ProviderID="3" BusID="903-FE" DutyStatus="0" BusStatus="0" RouteID="86" GoBack="1" Longitude="120.621948" Latitude="24.179197" Speed="9.0" Azimuth="136" DataTime="2015-04-01 05:08:42"/>
- <BusData ProviderID="3" BusID="866-US" DutyStatus="0" BusStatus="0" RouteID="86" GoBack="2" Longitude="120.685020" Latitude="24.137680" Speed="0.0" Azimuth="248" DataTime="2015-04-01 05:08:43"/>
- </BusInfo>
- </BusDynInfo>

```

FIGURE 3.3: XML Format



FIGURE 3.4: Block division schematic apply to Taiwan Boulevard

### 3.2.1 City Traffic State Assessment System Service Design

The proposed system architecture is introduced in this section. First of all, we develop Python Program to capture open data including real-time data and non real-time data from government server and then store these data in Big Data Distributed DB for Real Time Process Service through a data transmission service, namely Data Collection Service. To use cloud computing, the data in Big Data Distributed DB is transferred to Cloud Storage and then processed on cloud to provide the real-time traffic situation. In the processing, Data Analysis Service is introduced to analysis the data. Finally, these results of analyzing real-time traffic situation is friendly present by web application service through several web technologies, java script, html5, and css3 combined with d3.js、jquery shown in Figure 3.5.

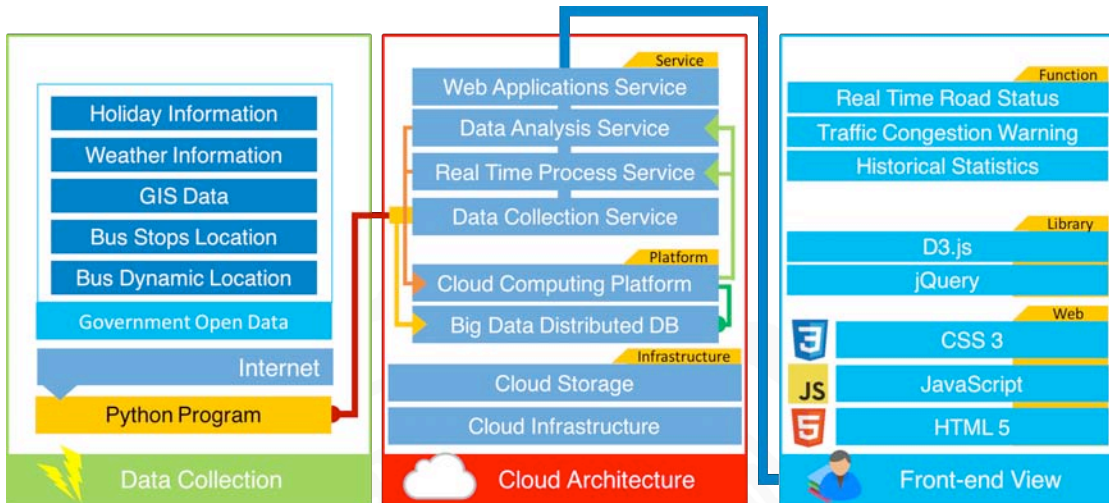


FIGURE 3.5: City Traffic State Assessment System Service

### 3.2.2 Data Collection Service

Data Collection Service in the proposed system is responsible for collecting open data including real-time data and historical data. This service utilize Python Program to capture these open data. The processing detail of these two open data is as follows:

#### Real Time Data Collection Service

The collection of real-time data is important for traffic situation evaluation, especially bus dynamic location and weather information. We utilize Python Program to capture these real-time data from government server and then store these data in a file system HDFS managed by Big Data Distributed Database, Hbase, through a tool Hbase API. Figure 3.6 show the Real Time Data Collection Service.

#### History Data Collection Service

Generally, historical data including bus GPS data is huge and with different format. To solve these two problems, we first utilize Python Program to capture these data respectively and then process the different format of these data by

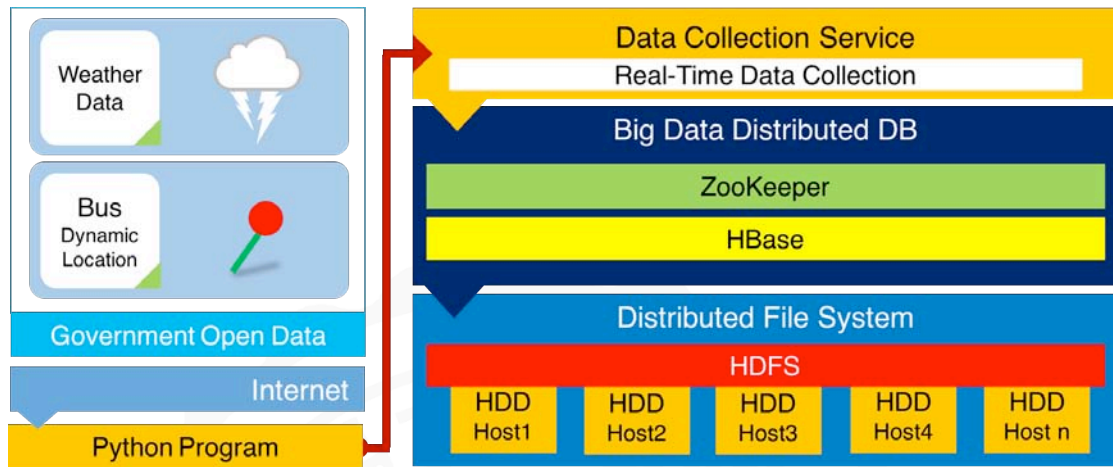


FIGURE 3.6: Real Time Data collection service

Apache Spark. Finally, the processed data are stored in HDFS or Hbase. Figure 3.7 shows the History Data Collection Service.

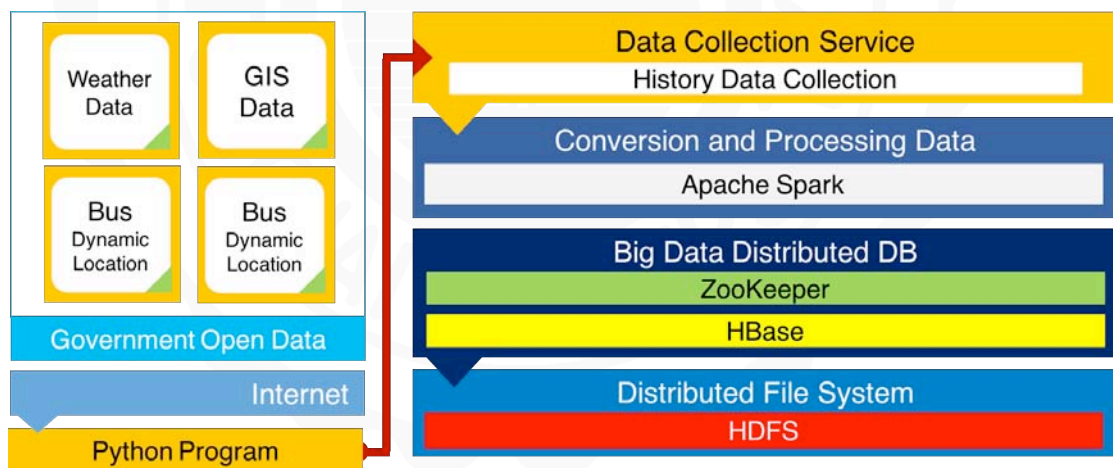


FIGURE 3.7: History Data Collection Service

### 3.2.3 Real Time Process Service

Real Time Process Service is responsible for real-time processing of data. Based on the open data collected by Data Collection Service, the proposed system real-time process, analyze, and assess them using Apache Spark. In addition, Apache Spark transmits the results to front-end and stores these results in Hbase at the same time . Figure 3.8 shows the Real Time Process Service.

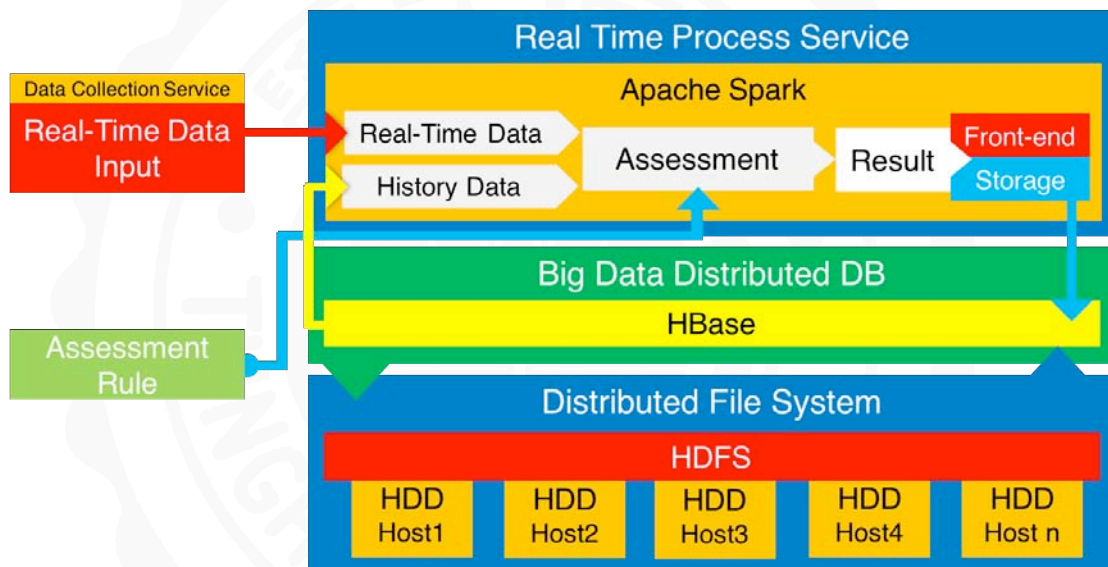


FIGURE 3.8: Real Time Process Service



### 3.2.4 Data Analysis Service

Data Analysis Services are mainly responsible for analyzing the data stored in the system. In order to quickly analyze data, this service uses Apache Spark as an analytical tool. In addition, Apache Spark transmits the results to front-end and stores these results in Hbase at the same time. Figure 3.9 shows the Data Analysis Services.

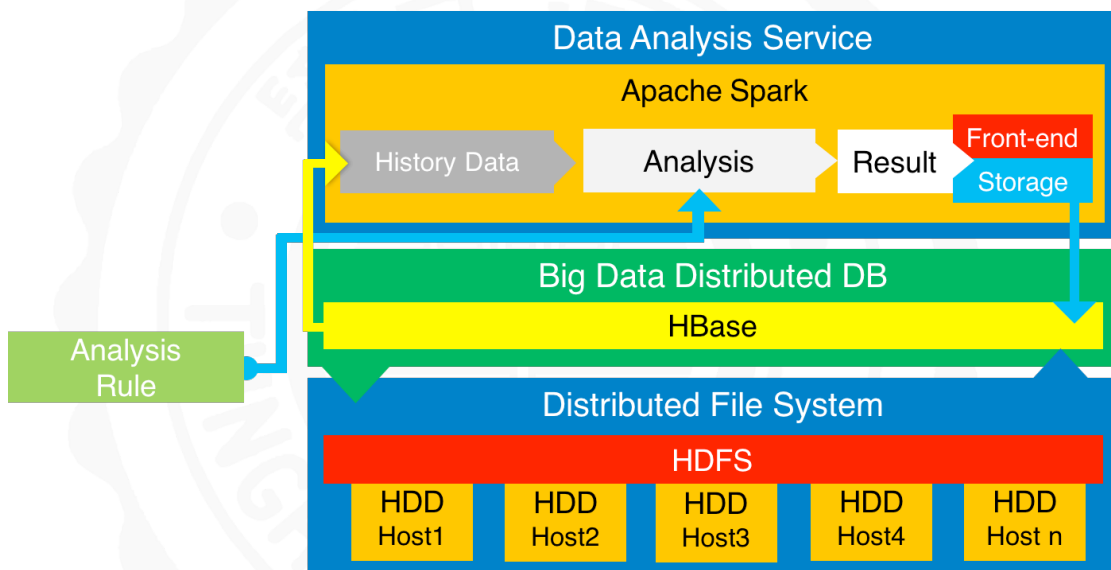


FIGURE 3.9: Data Analysis Service

### 3.2.5 Web Application Service

The last service in the proposed system is the Web Applications Service. Web Applications Service call back the results from Real Time Process Service and Data Analysis Service to front-end through HTTP GET/POST. For exchange of information, we adopt JSON format in Web Applications Service. Figure 3.10 show the Web applications service.

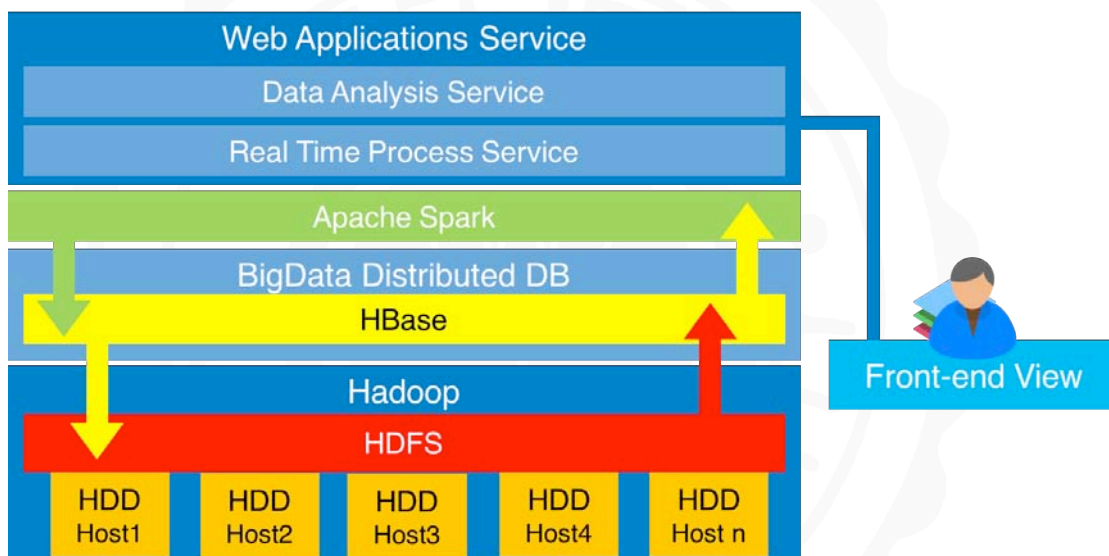


FIGURE 3.10: Web applications service

### 3.3 System Implementation

Fourteen nodes were used to build a cloud cluster platform by using Cloudera Manager, Two nodes as master, Twelve node as the computing node to set up Apache ZooKeeper, Apache Hadoop HDFS, Apache Hadoop YARN, Apache HBase, and Apache Spark. Table 3.1 shows the software specification.

TABLE 3.1: Software Specification

	Version
Cloudera CDH	5.3.3
ZooKeeper	3.4.5
Hadoop	2.5.0
HDFS	2.5.0
YARN	2.5.0
HBase	0.98.6
Spark	1.2.0

### 3.3.1 Cluster Deployment

On the deployment,platform environment using two servers as master, and using 10 Gigabit Ethernet connection. computing nodes using 1 Gigabit Ethernet, each node as DataNode, NodeManage, and RegionServer, where three computing nodes as ZooKeeper, as shown in Figure 3.11.

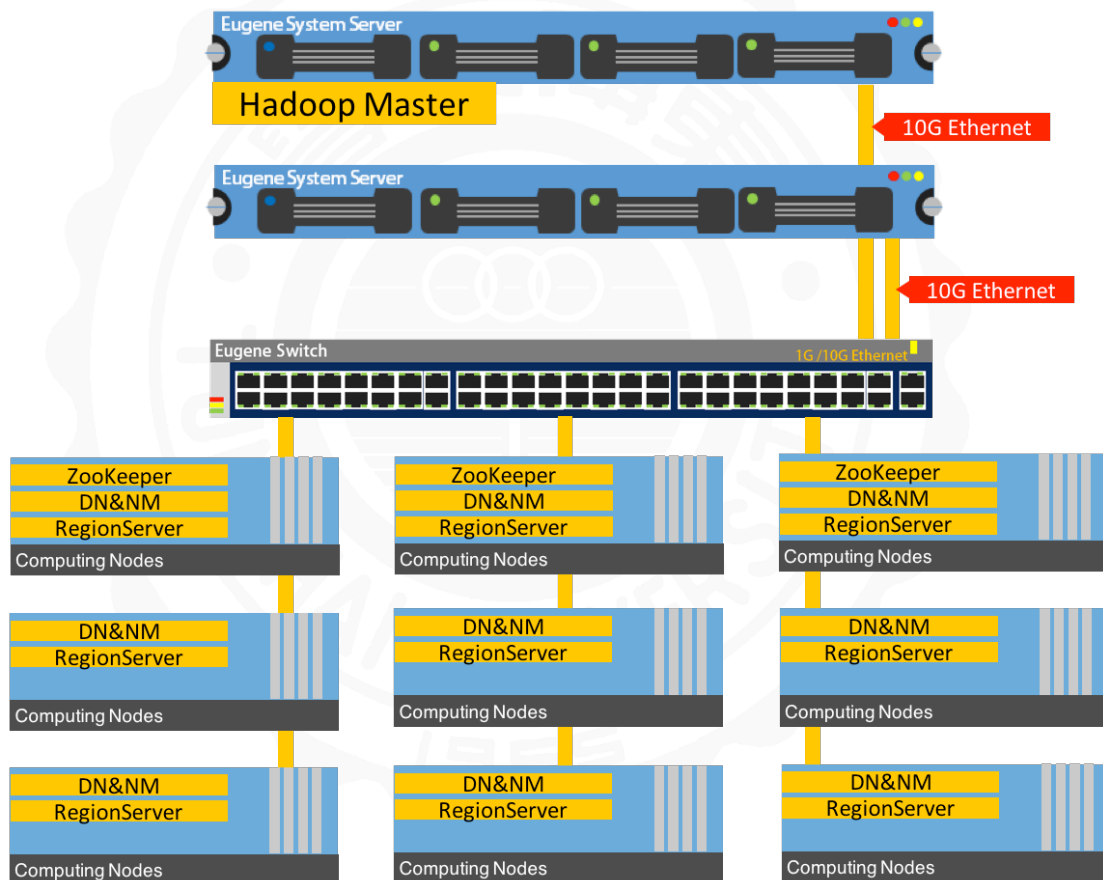


FIGURE 3.11: System Deployment Architecture Diagram

Cloudera Manager is used to monitor service states and system loading, in service states such as Spark, HBase, HDFS, YARN and ZooKeeper service status, or in system loading such as CPU usage, Memory usage, Disk I/O, network and Disk usage. Figure 3.12 shows the Cloudera Manager monitor interface. Cloudera Manager can also monitor the status of each node, confirming normal connections of each host. Cloudera Manager checks at regular intervals, and it will warn if connections are abnormal or the connection quality is poor. Cloudera Manager can

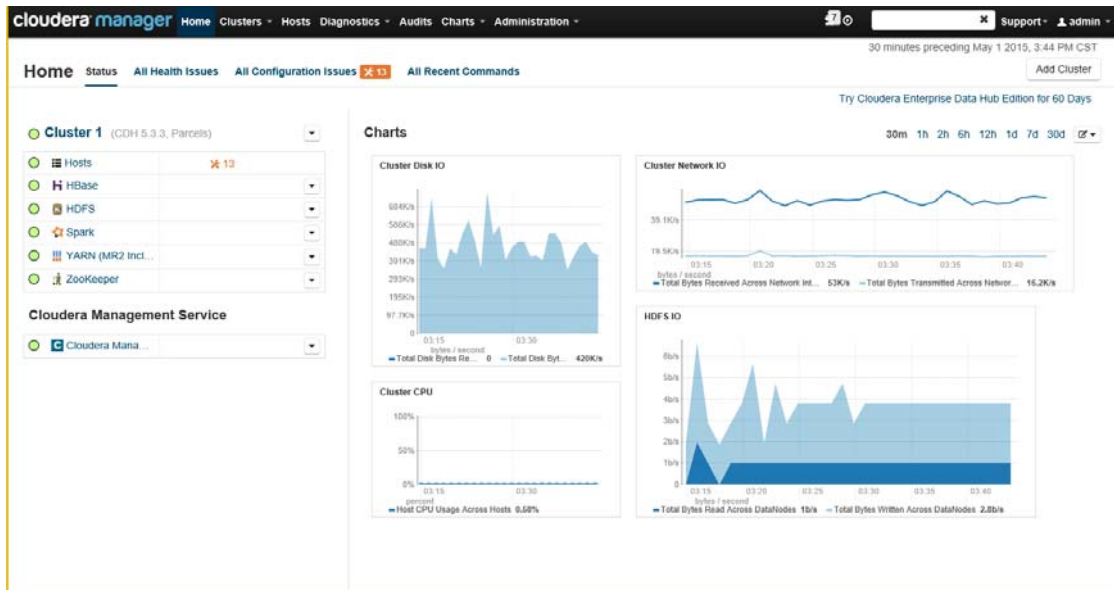


FIGURE 3.12: Cloudera manager status

remove nodes at any time, adding or removing nodes into or out of the cluster. Figure 3.13 shows the states of Fourteen hosts.

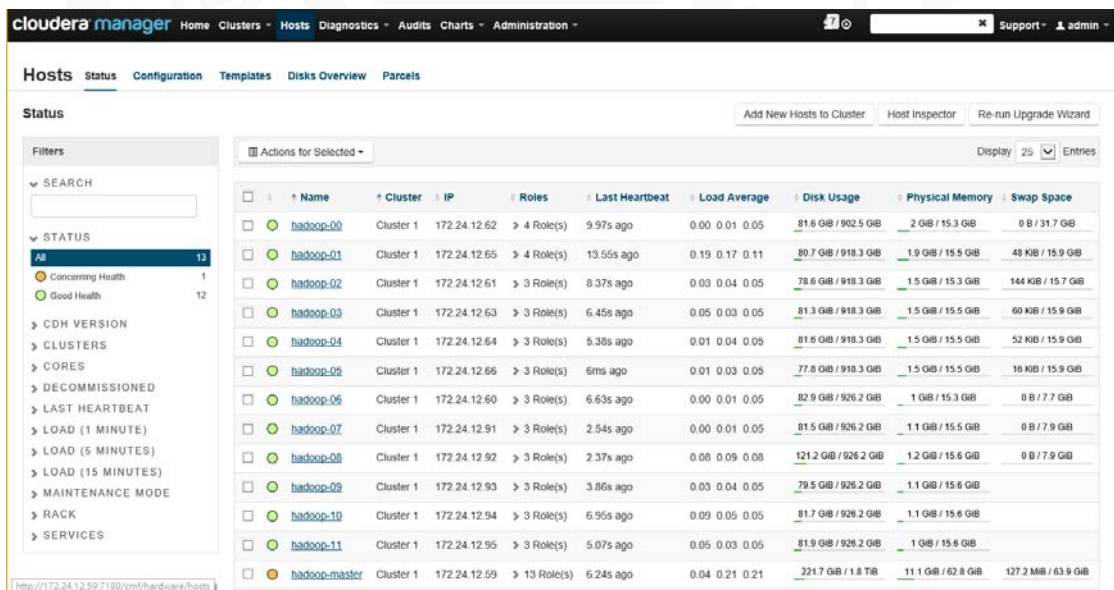


FIGURE 3.13: Cloudera manager hosts

Through the Fourteen hosts, i.e., the two NameNode and Twelve DataNodes, the Hadoop HDFS NameNode Web Interface shows that the cluster provides 10.47 TB of big data storage space. This information also shows how many live DataNodes are functioning shown in Figure 3.14. Cloudera Manager can also link to YARN ResourceManager Web Interface All Applications page to query nodes

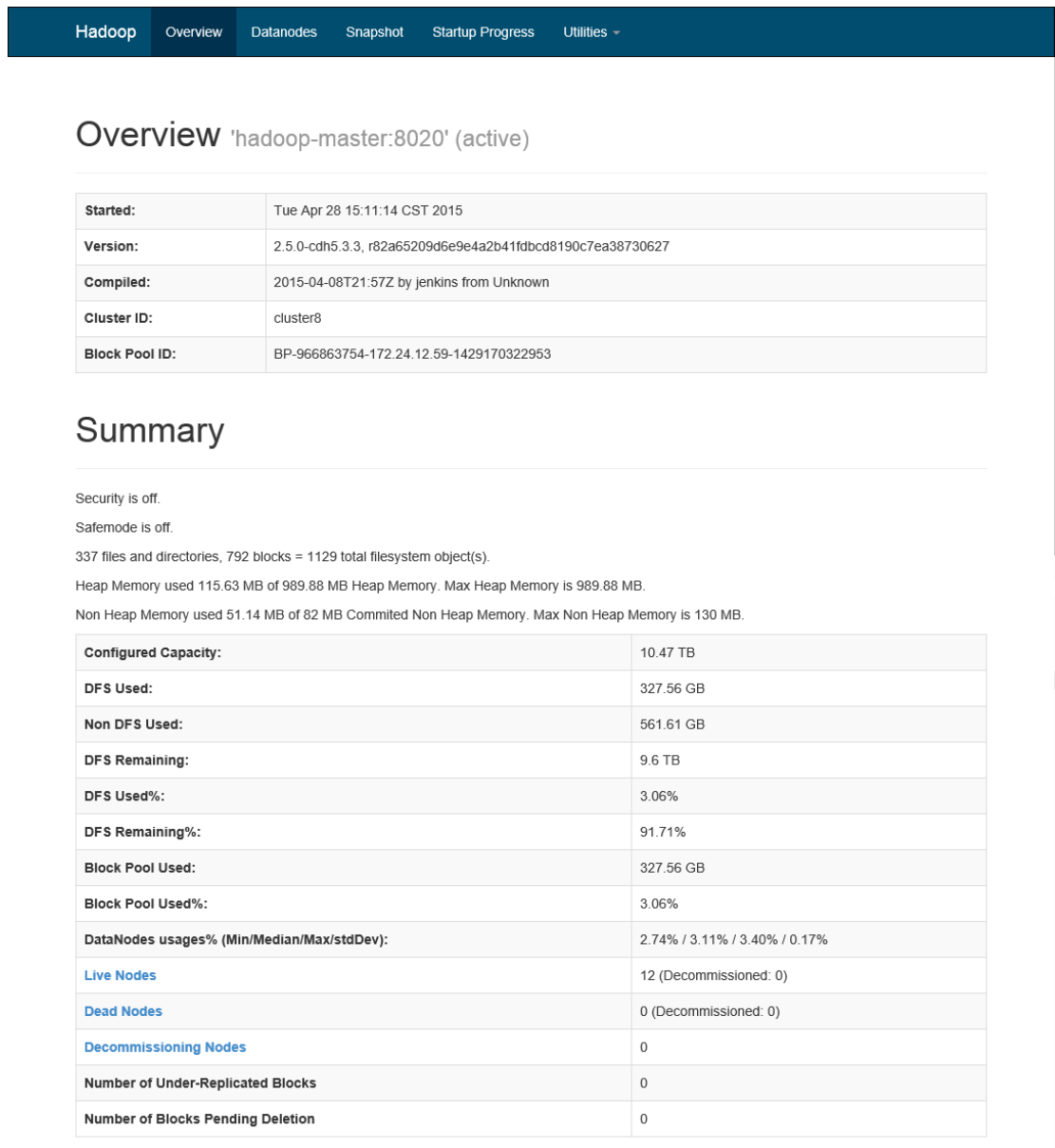


FIGURE 3.14: Hadoop NameNode information

state. Figure 3.15 shows thirteen nodes are currently in operation and the Applications running status or history. Figure 3.16 provides working states of HBase Region Server, e.g., from Requests per Second of hadoop-00 ~ hadoop-11 of, one can check whether the Region Servers are busy ,also shows how many live Region Server are functioning.

Figure 3.17 provides Spark History Server Web Interface, e.g., Spark Application running status and time.also provides Spark Jobs details of the implementation period,as shown in Figure 3.18.

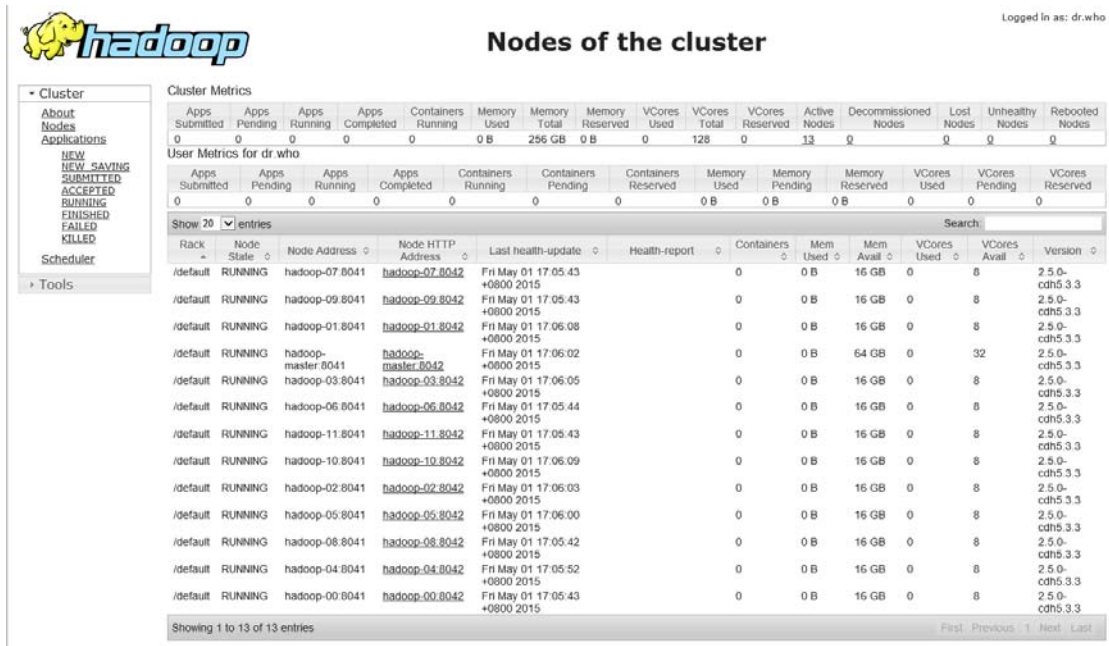


FIGURE 3.15: MapReduce JobTracker

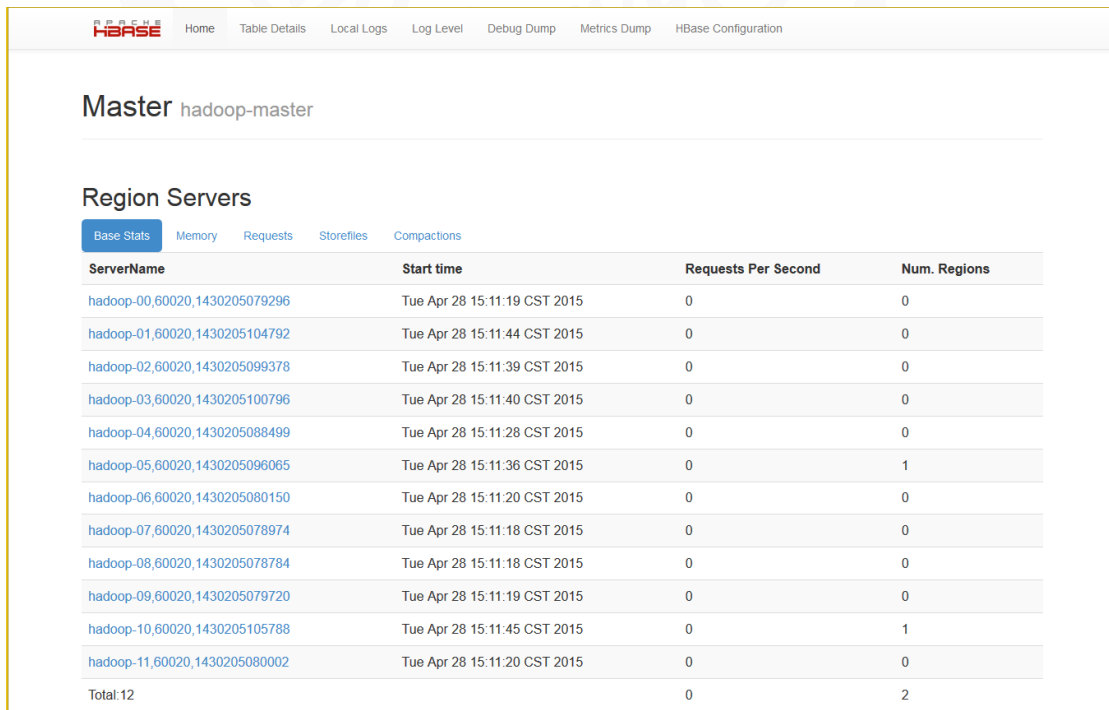


FIGURE 3.16: HBase region server status

**Spark History Server**

Event log directory: hdfs://hadoop-master:8020/user/spark/application-history

Showing 1-15 of 15

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated
<a href="#">application_1429173244917_0468</a>	Spark Count hdfs	2015/04/21 01:37:50	2015/04/21 01:38:32	42 s	root	2015/04/21 01:38:33
<a href="#">application_1429173244917_0467</a>	Spark Count hdfs	2015/04/21 01:34:31	2015/04/21 01:35:22	51 s	root	2015/04/21 01:35:24
<a href="#">application_1429173244917_0465</a>	Spark Count hdfs	2015/04/21 01:27:23	2015/04/21 01:27:59	37 s	root	2015/04/21 01:28:00
<a href="#">application_1429173244917_0464</a>	Spark HBase	2015/04/20 15:38:03	2015/04/20 15:39:15	1.2 min	root	2015/04/20 15:39:16
<a href="#">application_1429173244917_0463</a>	Spark HBase	2015/04/20 15:35:14	2015/04/20 15:36:27	1.2 min	root	2015/04/20 15:36:28
<a href="#">application_1429173244917_0462</a>	Spark HBase	2015/04/20 15:33:59	2015/04/20 15:34:13	14 s	root	2015/04/20 15:34:14
<a href="#">application_1429173244917_0461</a>	Spark HBase	2015/04/20 15:32:58	2015/04/20 15:33:22	24 s	root	2015/04/20 15:33:23
<a href="#">application_1429173244917_0460</a>	Spark HBase	2015/04/20 15:31:56	2015/04/20 15:32:06	10 s	root	2015/04/20 15:32:07
<a href="#">application_1429173244917_0459</a>	Spark HBase	2015/04/20 15:25:52	2015/04/20 15:26:18	26 s	root	2015/04/20 15:26:19
<a href="#">application_1429173244917_0458</a>	Spark HBase	2015/04/20 15:22:13	2015/04/20 15:22:38	25 s	root	2015/04/20 15:22:39
<a href="#">application_1429173244917_0455</a>	Spark Count hdfs	2015/04/20 14:43:32	2015/04/20 14:44:14	42 s	root	2015/04/20 14:44:15
<a href="#">application_1429173244917_0454</a>	Spark HBase	2015/04/20 14:36:25	2015/04/20 14:36:49	24 s	root	2015/04/20 14:36:51
<a href="#">application_1429173244917_0452</a>	Spark HBase	2015/04/20 14:26:52	2015/04/20 14:27:06	14 s	root	2015/04/20 14:27:08
<a href="#">application_1429173244917_0451</a>	Spark HBase	2015/04/20 14:25:27	2015/04/20 14:25:43	15 s	root	2015/04/20 14:25:44
<a href="#">application_1429173244917_0450</a>	Spark Count hdfs	2015/04/20 14:20:00	2015/04/20 14:20:23	23 s	root	2015/04/20 14:20:24

Spark 1.2.0-SNAPSHOT

FIGURE 3.17: Spark History Server

**Spark** Jobs Stages Storage Environment Executors Spark HBase (application\_1429173... application UI)

**Spark Jobs (?)**

Scheduling Mode: FIFO  
 Active Jobs: 0  
 Completed Jobs: 1  
 Failed Jobs: 0

**Active Jobs (0)**

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
--------	-------------	-----------	----------	-------------------------	---

**Completed Jobs (1)**

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	saveAsTextFile at convbus.scala:71	2015/05/05 00:04:55	18 ms	2/2	20/20

**Failed Jobs (0)**

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
--------	-------------	-----------	----------	-------------------------	---

Spark 1.2.0-SNAPSHOT

FIGURE 3.18: Spark Job



# Chapter 4

## Experimental Results

In this section, experimental environment and results with respect to the proposed Cloud City Traffic State Assessment System are present. In subsection 4.1, we first introduce the proposed novel architecture for the system and then implement the architecture respectively. Subsections 4.2-4.5 give some performance tests to verify the efficiency of the system

### 4.1 Experimental Environment

This subsection introduces our environmental environment including hardware and software. To implement the proposed system, we use 12 physical servers connected by Gigabit Ethernet to establish a cluster. In hardware, each physical server is Intel Core i7 CPU with 16GB Memory and 1TB HD. In software, Ubuntu 14.04 is adopted as our operating systems. Also, Cloudera Express 5.2.0 、Hadoop 2.5.0 、HBase 0.98.6 、Spark 1.1.0 、Zookeeper 3.4.5 are installed, as shown in Table 4.1 and Figure 4.1.

TABLE 4.1: Experimental environment

ID	CPU	RAM	HDD	NIC
1	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	1Gb Ethernet
2	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	1Gb Ethernet
3	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	1Gb Ethernet
4	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	1Gb Ethernet
5	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	1Gb Ethernet
6	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	1Gb Ethernet
7	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	1Gb Ethernet
8	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	1Gb Ethernet
9	Intel® Core™ i7-4790@3.60GHz	16GB DDR3	1TB	1Gb Ethernet
10	Intel® Core™ i7-4790@3.60GHz	16GB DDR3	1TB	1Gb Ethernet
11	Intel® Core™ i7-4790@3.60GHz	16GB DDR3	1TB	1Gb Ethernet
12	Intel® Core™ i7-4790@3.60GHz	16GB DDR3	1TB	1Gb Ethernet
13	Intel® Xeon® E5-2630v3@2.40GHz*2	64GB DDR4	2TB	10Gb Ethernet
14	Intel® Xeon® E5-2630v3@2.40GHz*1	8GB DDR4	2TB	10Gb Ethernet

## 4.2 Spark and Hadoop MapReduce Performance Comparison

To verify that Spark adopted in our system has better performance than Hadoop MapReduce, this subsection gives a comparison test for Spark and Hadoop using WordCount programming. First, ten test files of size 1GB 10GB are randomly generated and then put these files into HDFS. Next, the ten files in HDFS are executed by Spark and Hadoop individually. As shown in Figure 4.2, Spark cost less time regardless of file size. It is worth mentioning that the difference between them increases when the test file size increases.



FIGURE 4.1: Spark and Hadoop Computing cluster

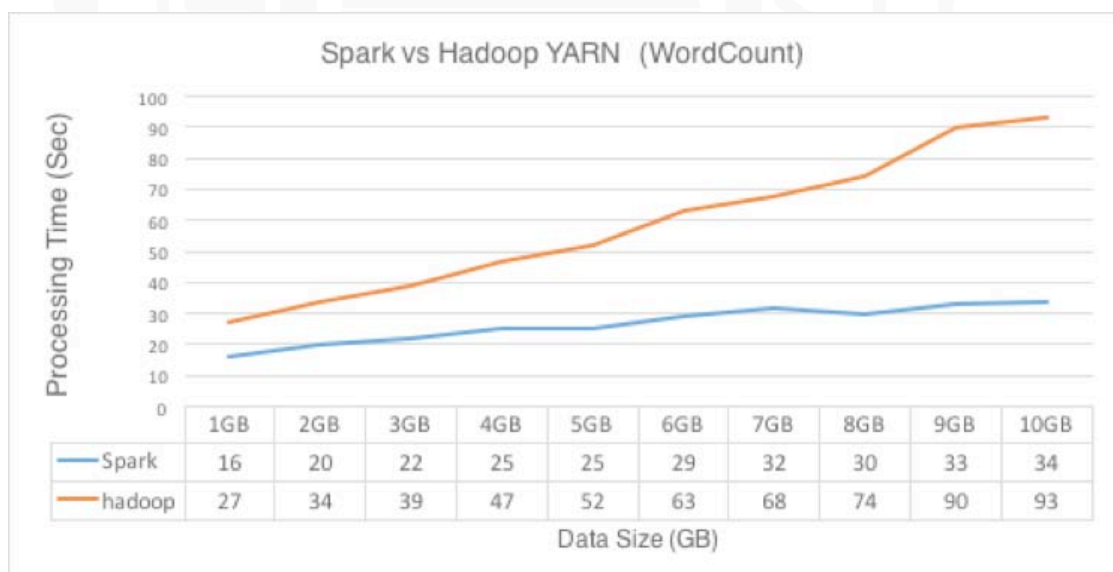


FIGURE 4.2: Spark and Hadoop MapReduce Performance Comparison

### 4.3 Experiment for HDFS read and write speed under various replication

Data processing and analysis are important to the proposed system. Before data analysis, the proposed system needs to read and write data. In other words, data read and data write are two important parts in data processing. Since the proposed system store data in HDFS, we adopt Hadoop TESTDFSIO to test the read and

write performance by changing replication number and MAP number in different data size to obtain the best solution. The detail is as follows. First of all, we test data read speed and data write speed in three data sizes, 12G, 60G, and 120G. In each data size, we increase the replication number from 1 to 12 to observe the data read speed and data write speed. At the same time, we adjust TESTDFSIO arguments to observe the data read speed and data write speed under various MAP number which is a multiple of number of cluster nodes, especially  $(1 \sim 5) * \text{number of nodes}$ . Since TESTDFSIO is a MapReduce program, the test procedure is given as follows. As shown in Figure 4.3:

- Step 1. Input user parameters in beginning.
- Step 2. Arrange the number of MAP and the data size for each MAP when reading and writing data
- Step 3. Read and write data into node by MAP.
- Step 4. Estimate MAP execution time and collect corresponding results.
- Step 5. Output the results.

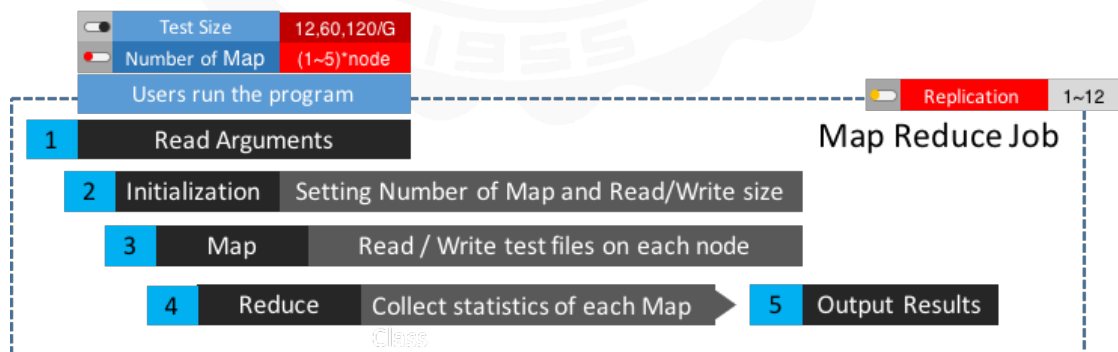
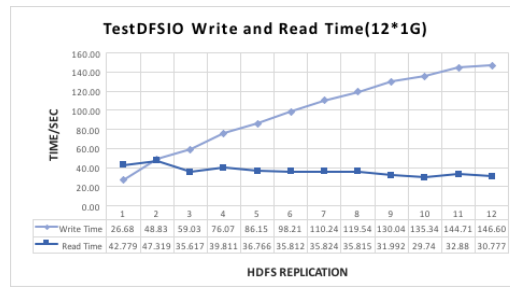


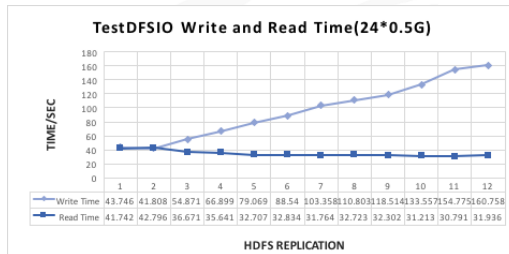
FIGURE 4.3: The flow chart of HDFS read and write test

## Experimental Results

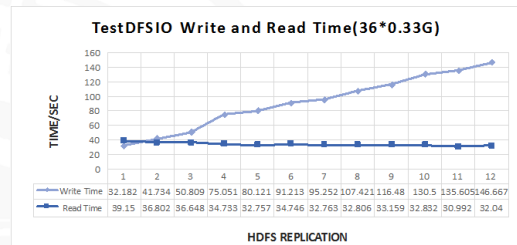
In the results. Figure 4.4 Read and write time duration for MAP=12,24,36,48,60 in various replication number under the case 12GB.



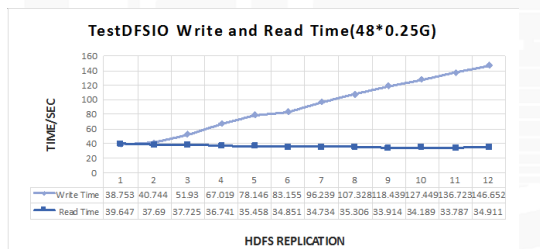
(a) map=12



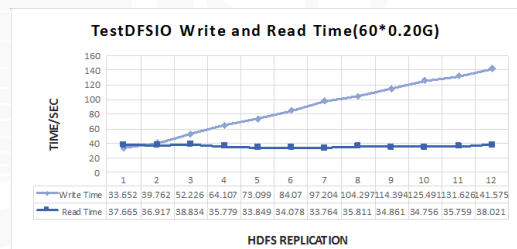
(b) map=24



(c) map=36



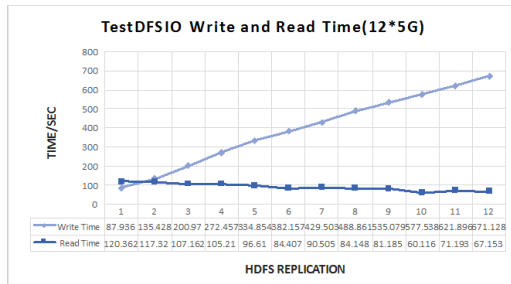
(d) map=48



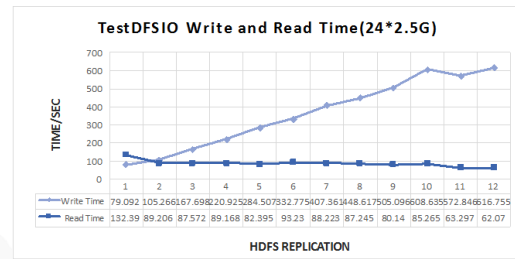
(e) map=60

FIGURE 4.4: Read and write time duration for MAP=12,24,36,48,60 in various replication number under the case 12GB

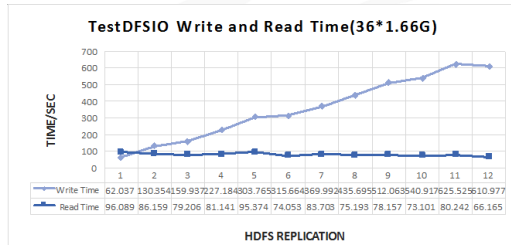
Figure 4.5 Read and write time duration for MAP=12,24,36,48,60 in various replication number under the case 60GB.



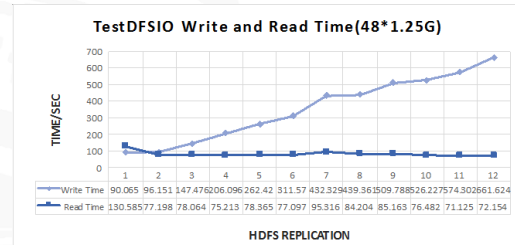
(a) map=12



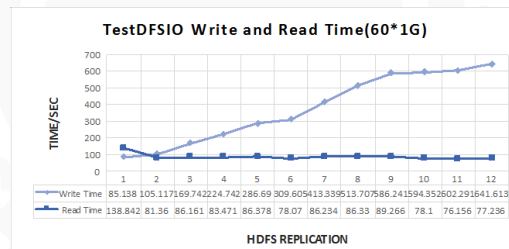
(b) map=24



(c) map=36



(d) map=48



(e) map=60

FIGURE 4.5: Read and write time duration for MAP=12,24,36,48,60 in various replication number under the case 60GB

Figure 4.6 Read and write time duration for MAP=12,24,36,48,60 in various replication number under the case 120GB.

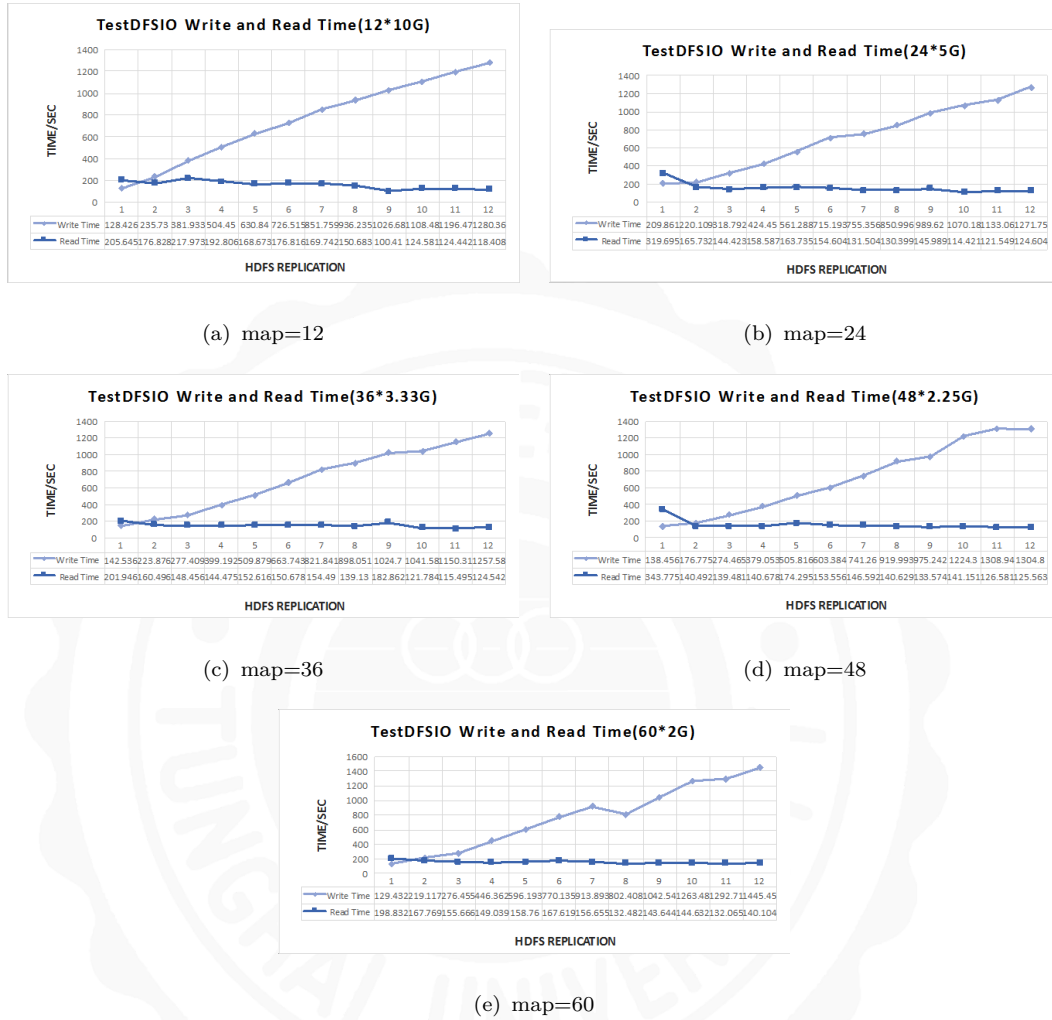
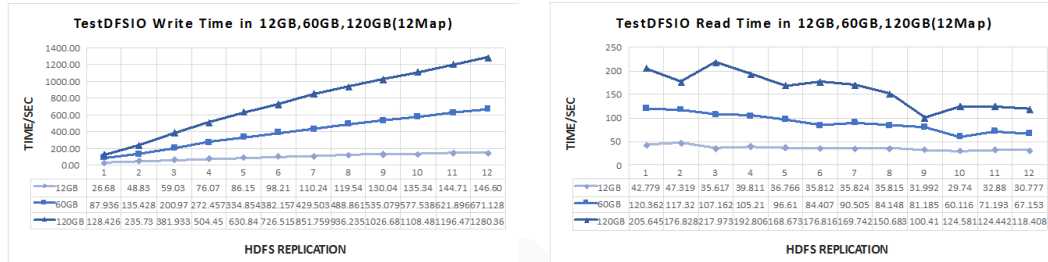


FIGURE 4.6: Read and write time duration for MAP=12,24,36,48,60 in various replication number under the case 120GB

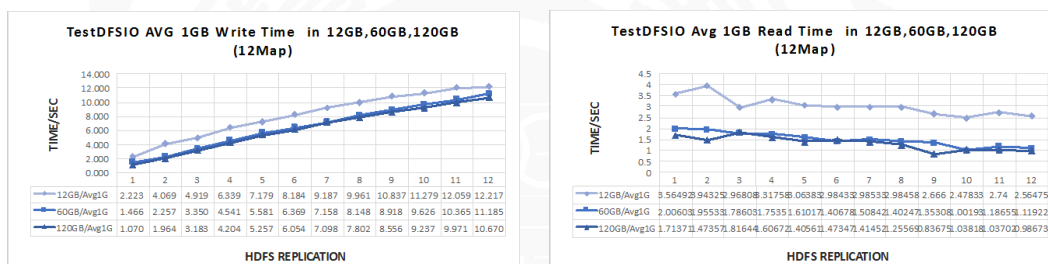
From above figures, one can observe that more replication number more writing time under various data size and various MAP but reading time slightly decreases when replication number increases, as shown in Figure 4.7. In conclusion, replication 2 has the best result. In the point of view on different test size, writing time increases stably but reading time decreases unstably, as shown in Figure 4.8. In addition, one can observe that both reading time and writing time are short for bigger data file. It means once reading or writing a big data file is efficient.



(a) Read

(b) Write

FIGURE 4.7: All sizes in 12 map read and write speed



(a) Read

(b) Write

FIGURE 4.8: All sizes in 12 map read and write average speed of 1GB

## 4.4 Using Spark to process bus location data under different number of executors

Most input data of the proposed system is obtained through the Government OPEN DATA. These input data collected every two seconds from open data are about 5GB one day. They will be a big historical data. Accordingly, the processing and calculation of these big data are achieved through Spark Application. To process and analyze the big data efficiently, two Spark Applications are tested. Twelve executors are set to each Spark Application to test one-day, two-day, four-day data and six memory size.

The first Spark Application, namely convBus, is mainly to remove unwanted data in BUS GPS information and duplicate data in accordance with the update



time so as to find the Block of the given latitude and longitude coordinates. The detail procedure is summarized as follows. As shown in Figure 4.9:

- Step 1. Read BUS record data file.
- Step 2. Filter necessary BUS information and remove XML format data.
- Step 3. Use MAP to group Update Time and cluster coordinate into proper Block.
- Step 4. Remove duplicate data.
- Step 5. Output data to HDFS.

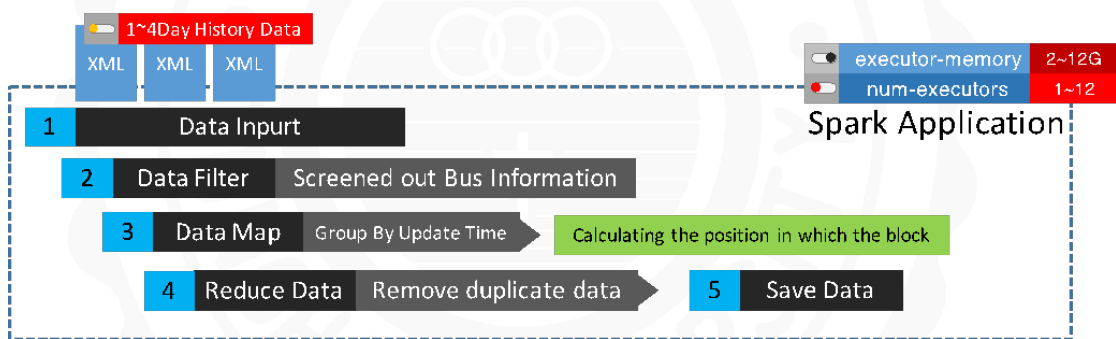


FIGURE 4.9: The flow chart of convBus

The second Spark Application, namely comBus, has almost the same procedures with the first Spark Application. The difference between them is that comBus has a permutation for Block in last two steps. The detail procedure is summarized as follows. As shown in Figure 4.10:

- Step 1. Read BUS record data file.
- Step 2. Filter necessary BUS information and remove XML format data.
- Step 3. Use MAP to group Update Time and cluster coordinate into proper Block.
- Step 4. Remove duplicate data.
- Step 5. Sort by Block.
- Step 6. Output data to HDFS.

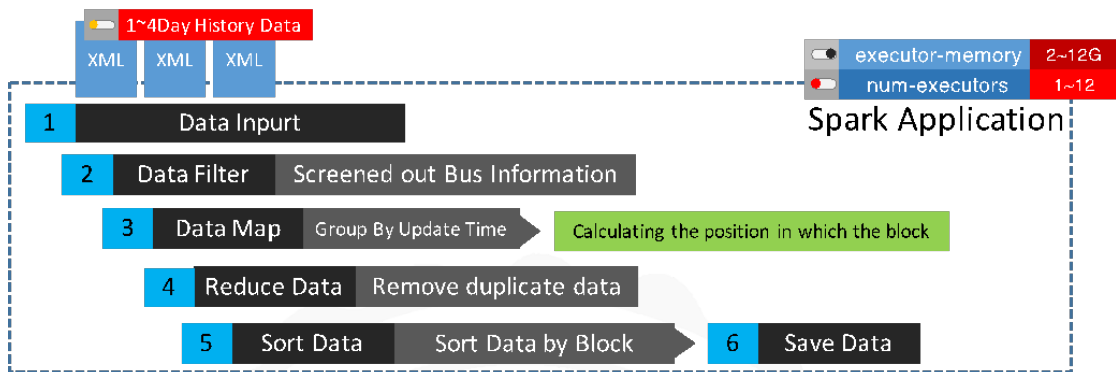


FIGURE 4.10: The flow chart of comBus

## Experimental Results

Figures 4.11,4.12,4.13 show the execution time of using both convBus and comBus to process one-day, two-day and three-day data under different number of executors and different memory. One can observe that processing time decreases when executors increase under fixed memory. However, processing time is similar when memory increases, especially when executors are greater than three. We also observe that the execution time of comBus is greater than the execution time of convBus. To reduce their processing time, increase of the number of executors is a consideration.

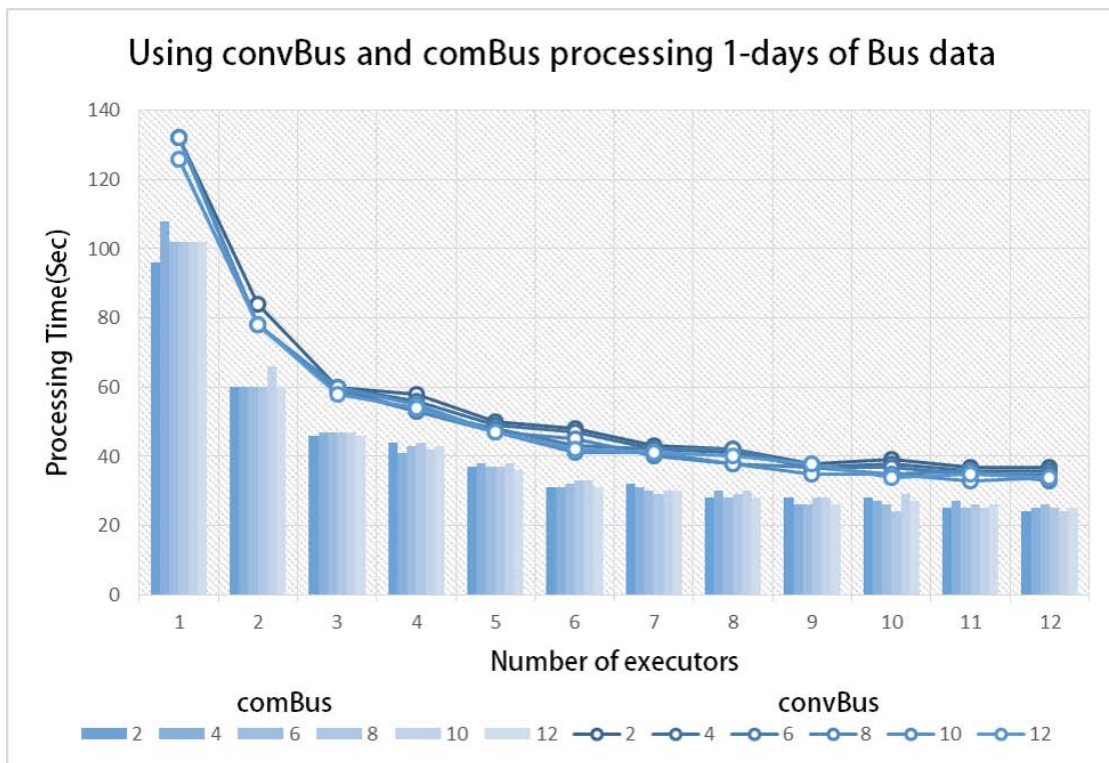


FIGURE 4.11: Using convBus and comBus processing 1-days of Bus data

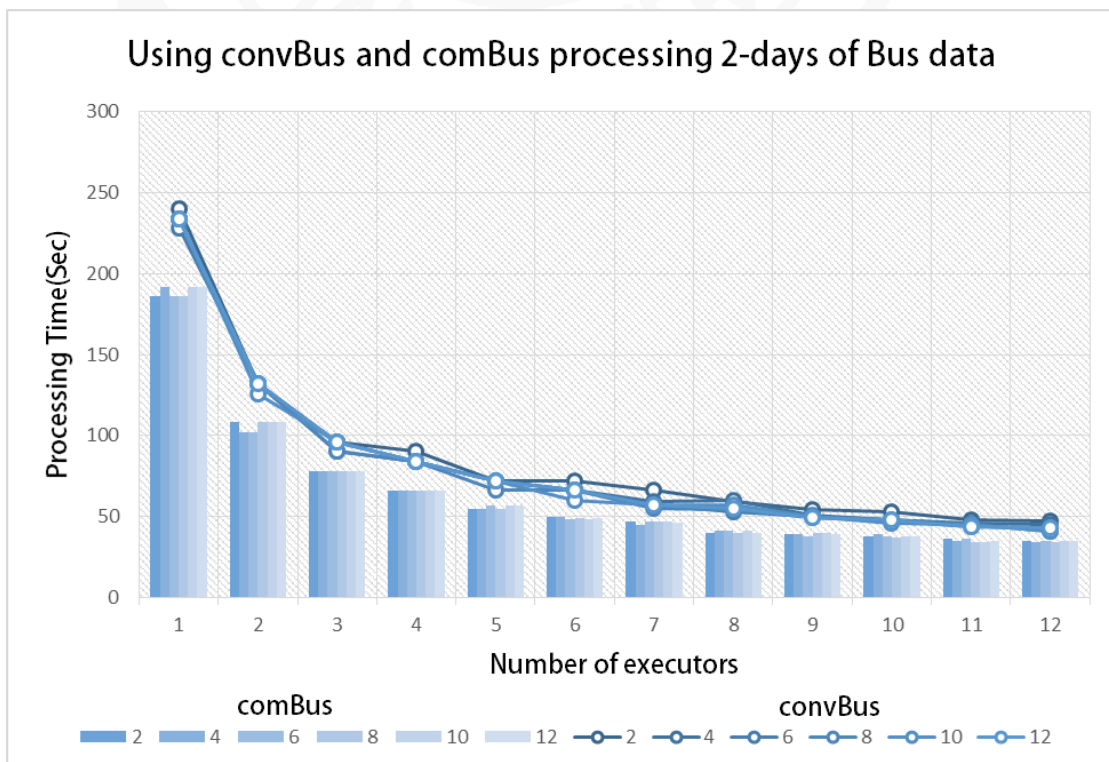


FIGURE 4.12: Using convBus and comBus processing 2-days of Bus data

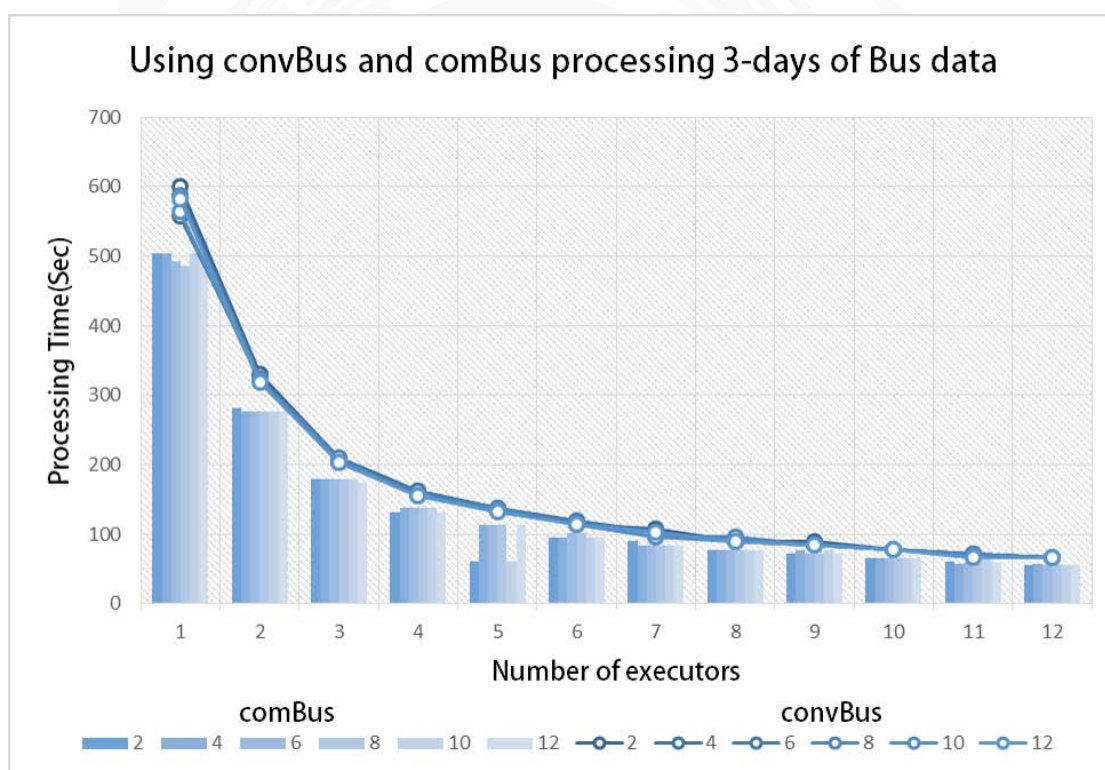


FIGURE 4.13: Using convBus and comBus processing 3-days of Bus data

## 4.5 Comparison of different clustering ways to find traffic jams grouping

This section finds traffic jams by three popular cluster methods: DBSCAN、K-means、Fuzzy-C-Means. Apache Spark is utilized to compute the cluster methods efficiently. The results are summarized in the following:

### 4.5.1 DBSCAN

DBSCAN clustering applies two parameters, epsilon and minPts, to control searching area and minimum cluster components. We utilize two epsilons and three minPts to test the performance of DBSCAN clustering. Figure 4.14 (a)-(d) show the results in cases original, PTS=3, PTS=4, PTS=5 under epsilon=0.001. Figure 4.15 (a)-(d) show the results in cases original, PTS=3, PTS=4, PTS=5 under epsilon=0.002. One can observe that the cluster decreases when PTS increases and the searching area enlarges when epsilon increases.[40]

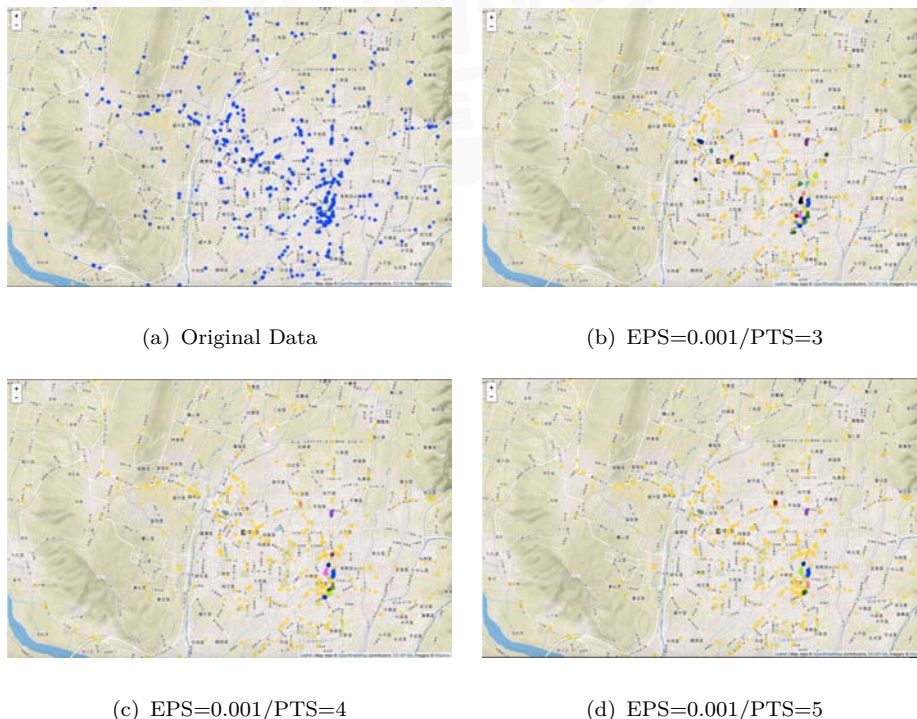


FIGURE 4.14: DBSCAN in epsilon=0.001 Clustering results

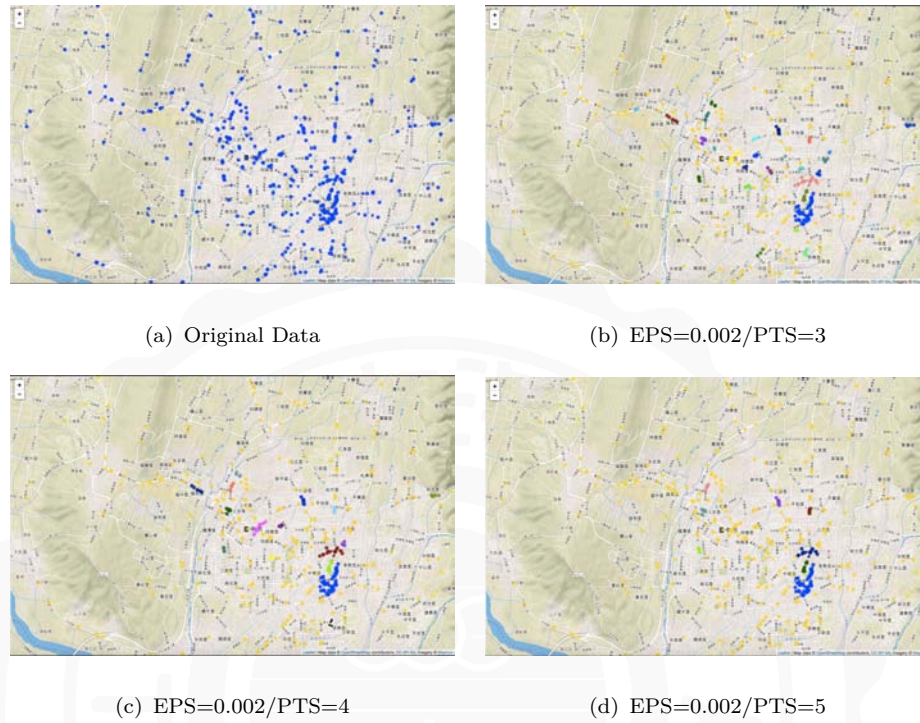


FIGURE 4.15: DBSCAN in epsilon=0.002 Clustering results

### 4.5.2 K-Means

K-Means clustering applies two parameters, k-value and iteration, to control cluster and iteration number. We utilize three k-values and two iterations to test the performance of K-MEANS clustering. Figure 4.16 (a)-(d) show the results in cases original, k=10, k=50, k=100 under iteration=100. Figure 4.17 (a)-(d) show the results in cases original, k=10, k=50, k=100 under iteration=1000. One can observe that the clustering fails when k is very small and works when k is sufficient.[41]

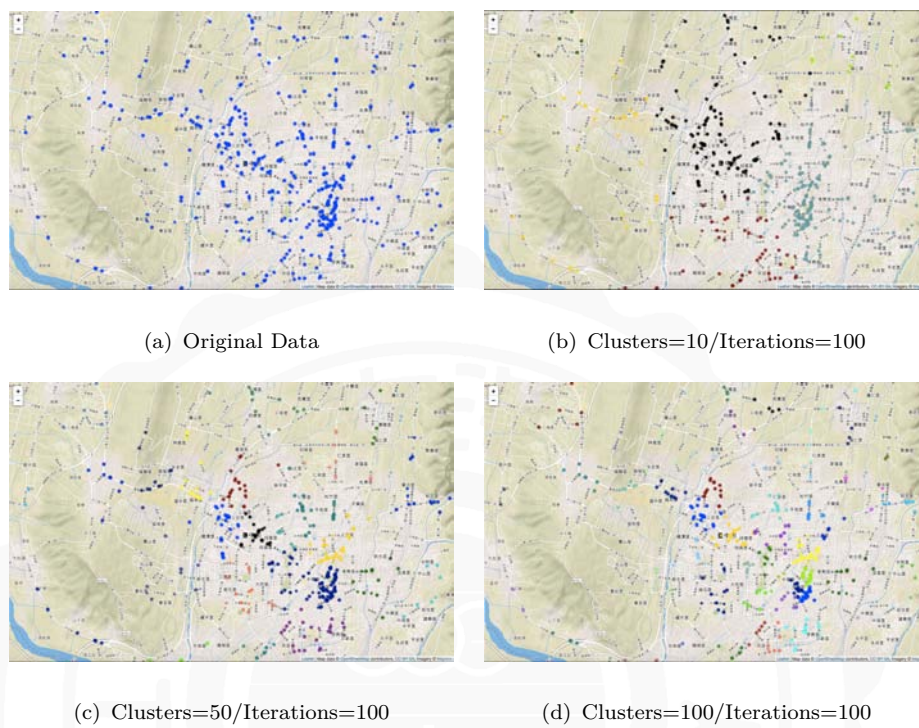


FIGURE 4.16: K-MEANS in Iterations=100 Clustering results

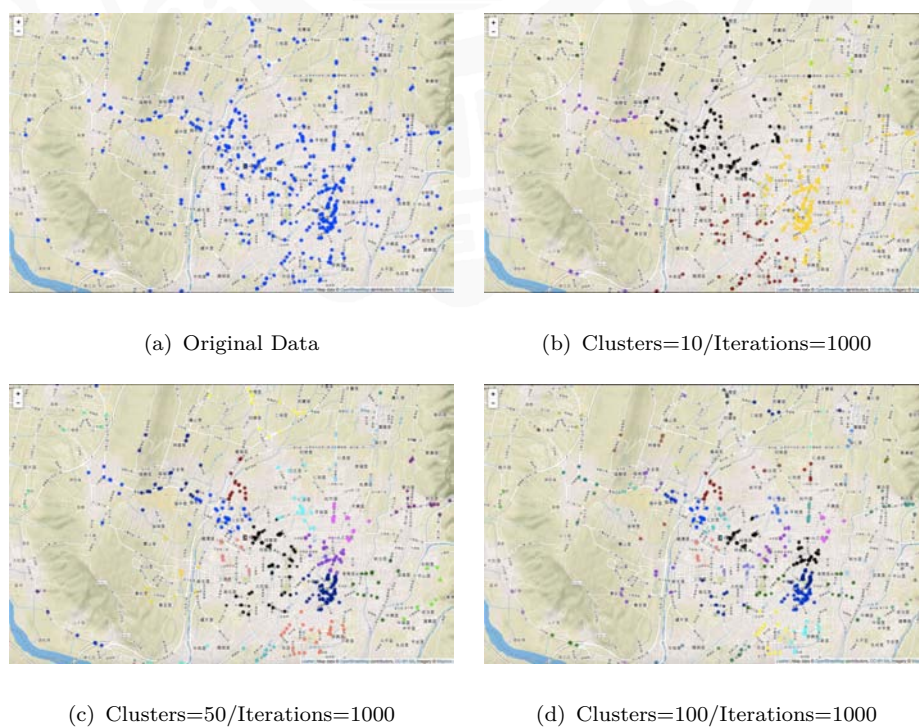


FIGURE 4.17: K-MEANS in Iterations=1000 Clustering results

### 4.5.3 Fuzzy-C-Means

Fuzzy-C-Means clustering applies two parameters, k-value and iteration, to control cluster and iteration number. We utilize three k-values and two iterations to test the performance of K-MEANS clustering. Figure 4.18 (a)-(d) show the results in cases original, k=10, k=50, k=100 under iteration=50. Figure 4.19 (a)-(d) show the results in cases original, k=10, k=50, k=100 under iteration=100. One can observe that the clustering fails when k is very small and works when k is sufficient.[42]

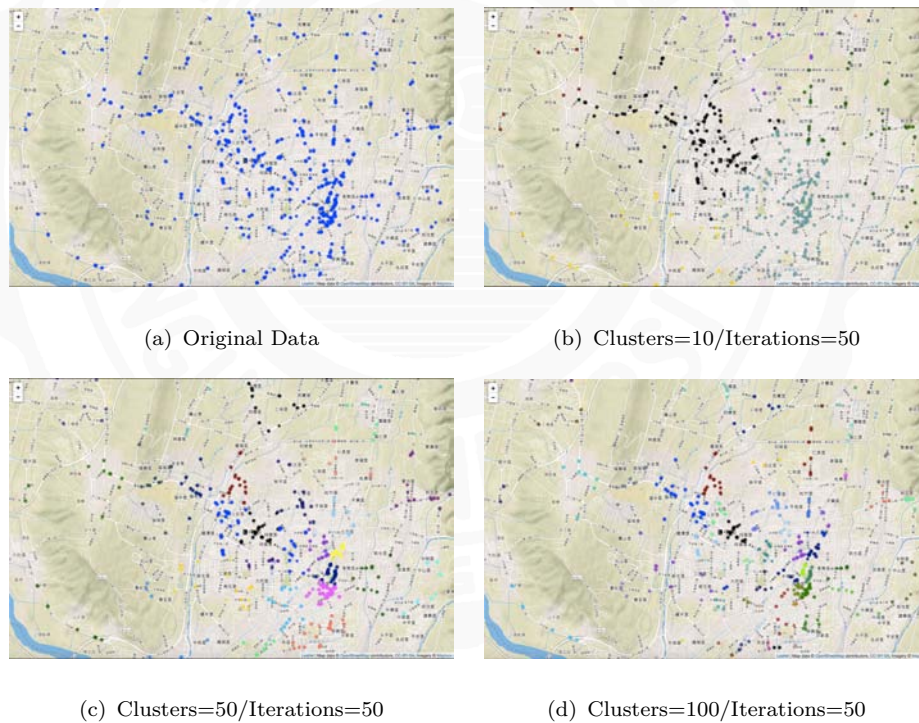


FIGURE 4.18: Fuzzy-C-Means in Iterations=50 Clustering results



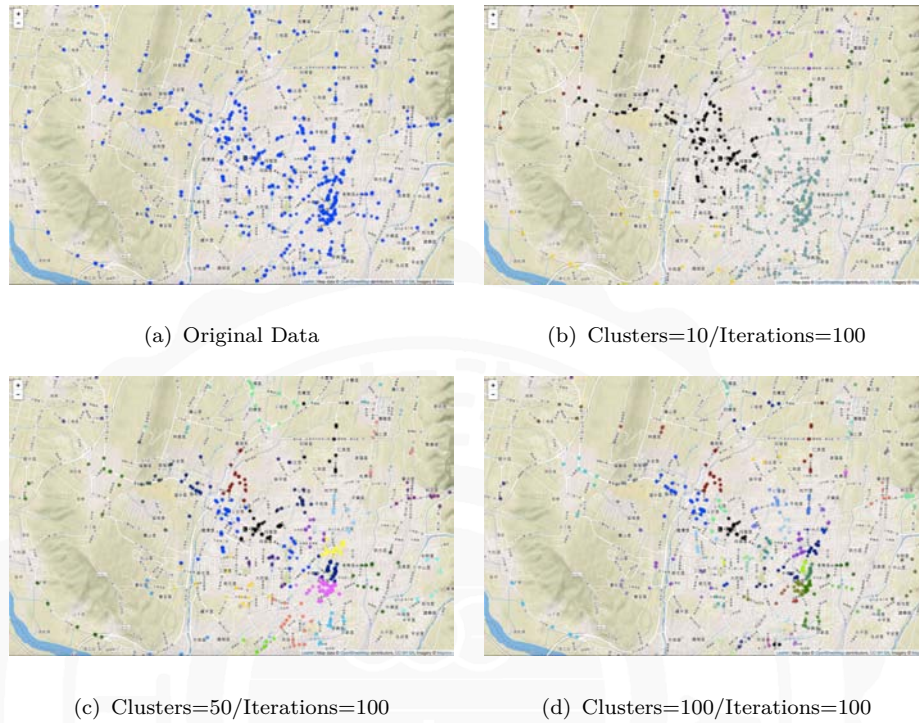


FIGURE 4.19: Fuzzy-C-Means in Iterations=100 Clustering results

#### 4.5.4 Results Compared

Figure 4.20 (a)-(d) show the clustering results in cases original, DBSCAN with  $EPS=0.001/PTS=5$ , K-means with  $k\text{-value}=100/Iteration=100$ , Fuzzy-C-Means with  $k\text{-value} = 100/Iterations=50$ . From the results in Figure 5, we know that Fuzzy-C-Means has the best clustering but k-means and DBSCAN use less time.

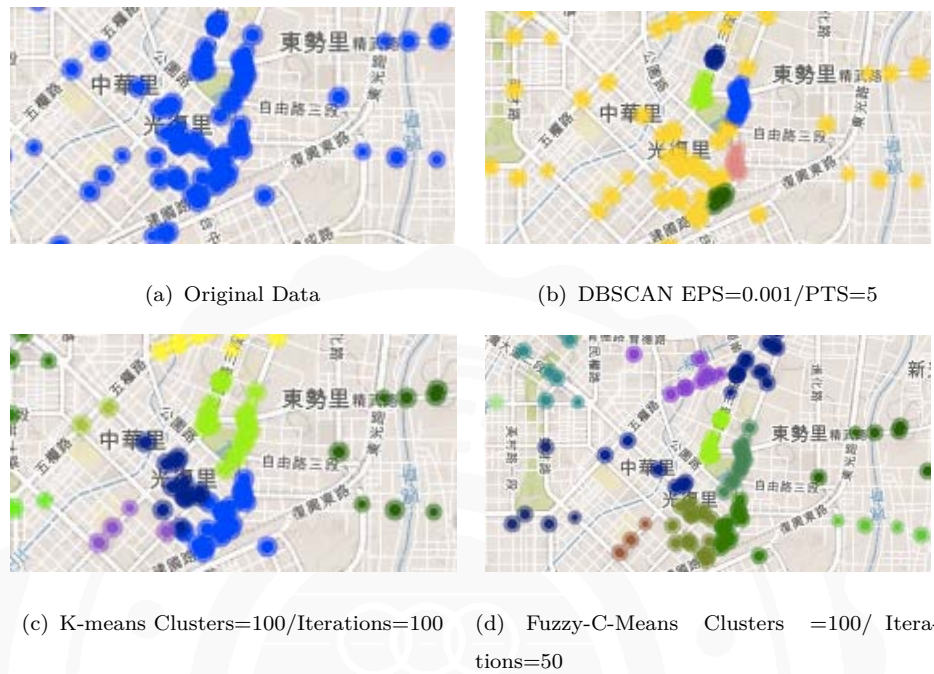


FIGURE 4.20: Comparison of DBSCAN,K-Means,Fuzzy-C-Means

## 4.6 Cloud City Traffic State Assessment System

In this thesis, the system Cloud City Traffic State Assessment System, offers the user to understand the Traffic State through a Web presentation in the User-friendly interface. The Web using Html5, CSS3, JavaScript, and JQuery with semantic Front End, the proposed Cloud City Traffic State Assessment System provides a web-based user-friendly interface to offer many features, as shown in Figure 4.21 and the left side of Figure 4.22.

- Historical data: provide the average speed of a bus and the area of traffic jam in the past by a line chart, as shown in Figures 4.23,4.24.
- Real-time evaluation: provide real-time traffic state, bus speed, and distribution of busses, as shown in Figures 4.26,4.26.
- Clustering results: provide the clustering results of DBSCAN and K-Means, as shown in Figures 4.27,4.28.

Moreover, we use AJAX and Openstreetmap to update web-page information and Map information.

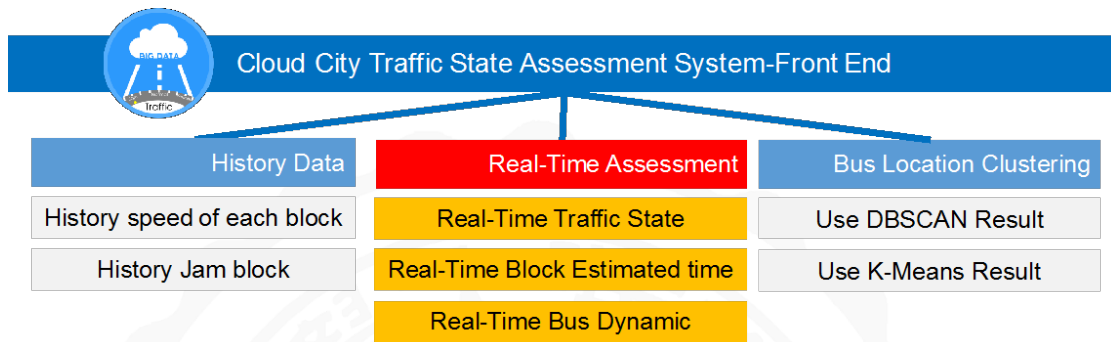


FIGURE 4.21: Cloud City Traffic State Assessment System functions

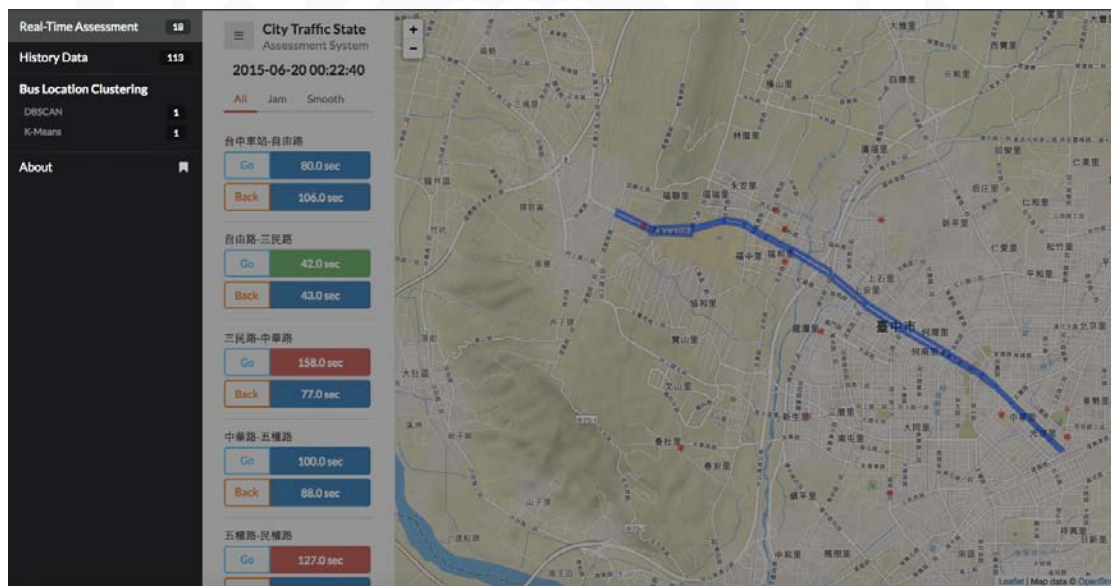


FIGURE 4.22: Web UI Function Menu



FIGURE 4.23: History Bus Travel each block of time use waveform display



FIGURE 4.24: History Bus Travel each block of time use Use long bar graph display

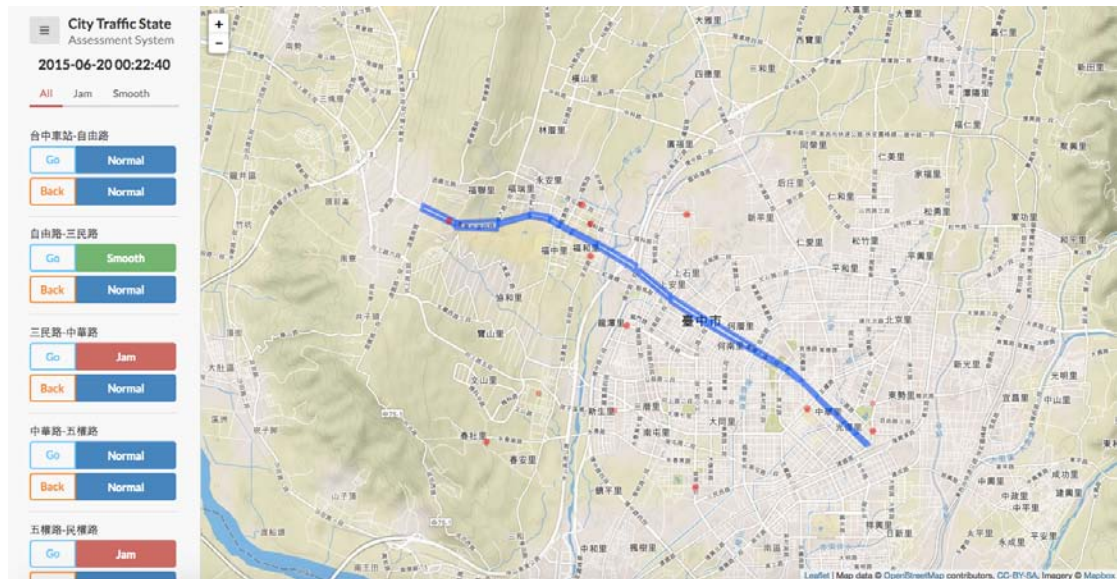


FIGURE 4.25: Real-time assessment traffic in Taiwan Boulevard using WEB presentation

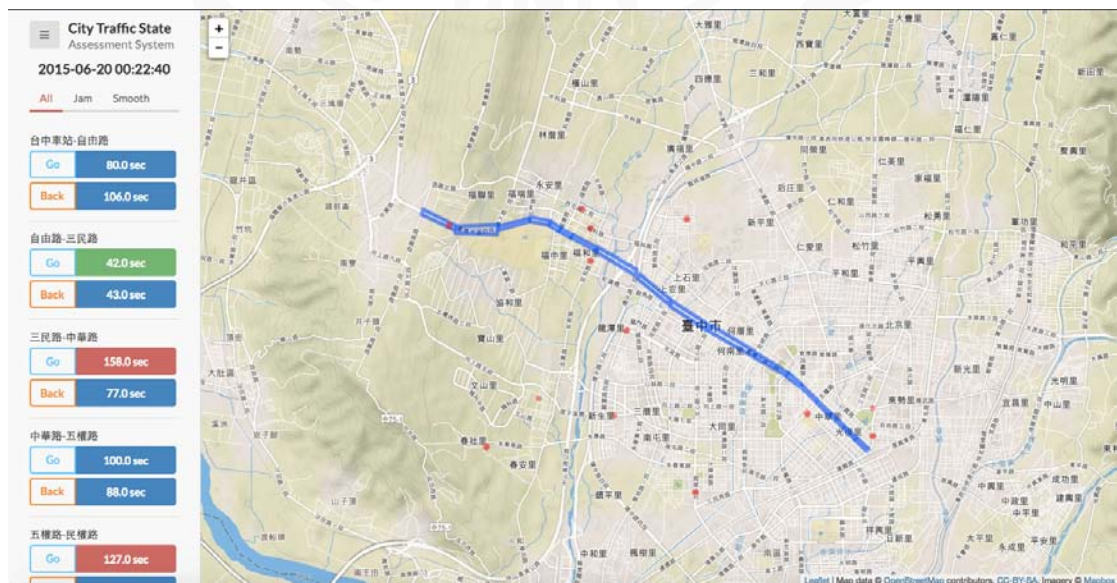


FIGURE 4.26: Real-time assessment traffic in Taiwan Boulevard using WEB presentation

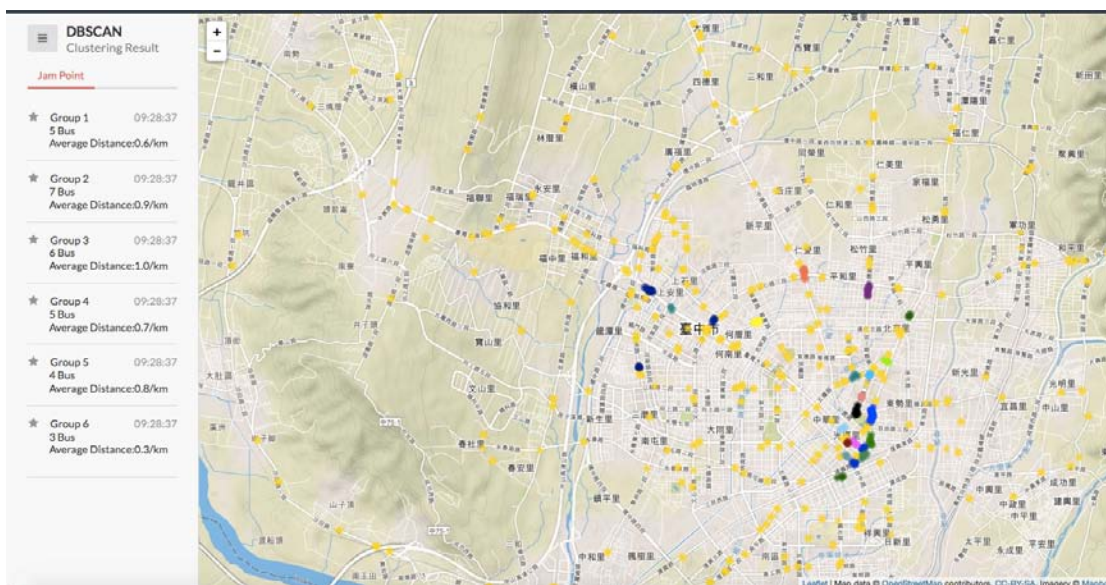


FIGURE 4.27: Use DBSCAN clustering on web views

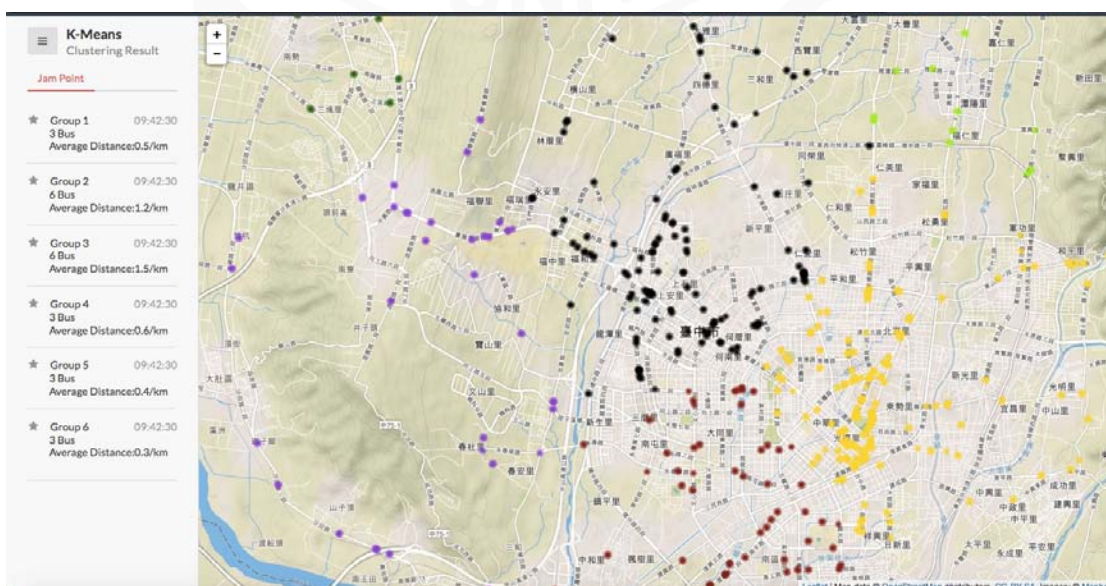


FIGURE 4.28: Use K-Means clustering on web views

# Chapter 5

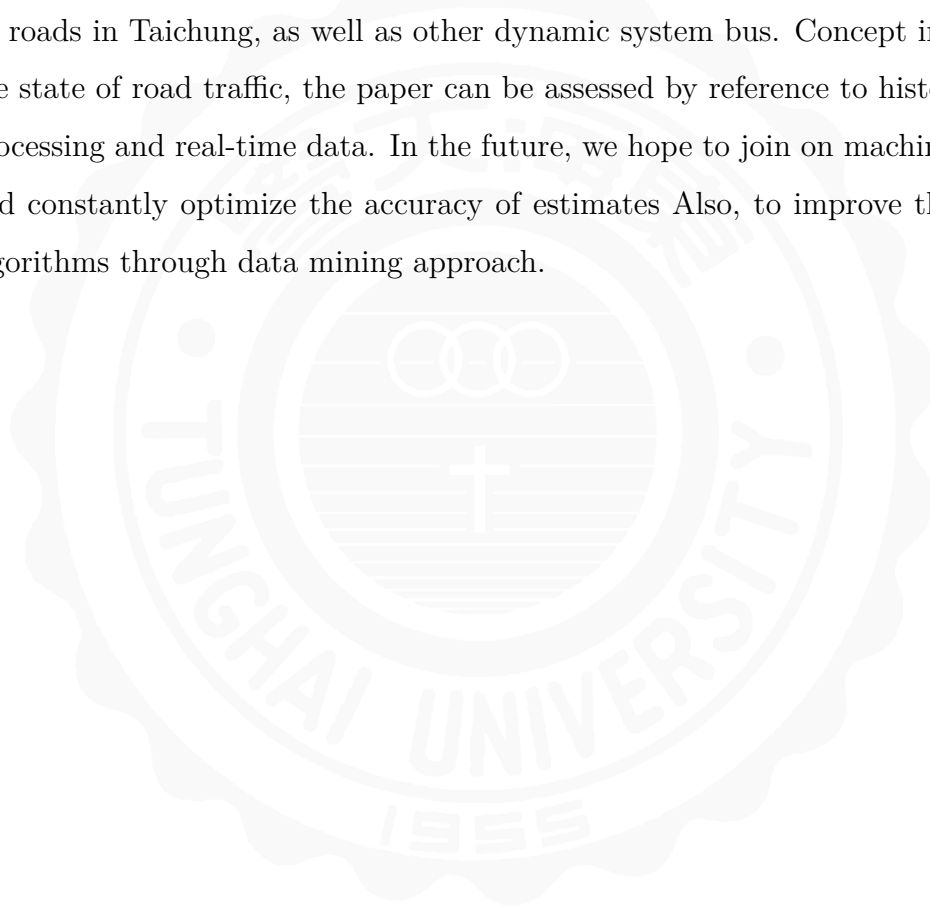
## Conclusions and Future Work

### 5.1 Concluding Remarks

In this work, we present a cloud city traffic state assessment system using a novel architecture of big data. With the high-scalability cloud technologies, Hadoop and Spark, the proposed system architecture for big data and services is implemented in Taiwan Boulevard efficiently. In addition, experimental results show that Spark adopted in our system has better performance than Hadoop MapReduce. Based on the comparison between real-time buses average velocity and the past average velocity, Fuzzy C-Means , K-Means, DBSCAN is applied to find out the heavy traffic area or traffic jam in each block of Taiwan Boulevard. In the system interface, this work has designed a front-end web interface, providing users to view traffic status in Taiwan Boulevard, the user can know the real-time traffic state, and view the history of traffic status.

## **5.2 Future Work**

In this study, Cloud City Traffic State Assessment System has successfully collected the result through the traffic state of big data to assess the road, and applied to Taichung, Taiwan Boulevard. In the future, we hope to apply this system to all roads in Taichung, as well as other dynamic system bus. Concept in assessing the state of road traffic, the paper can be assessed by reference to historical data processing and real-time data. In the future, we hope to join on machine learning and constantly optimize the accuracy of estimates. Also, to improve the existing algorithms through data mining approach.





# References

- [1] Big data, 2014. [http://en.wikipedia.org/wiki/Big\\_data](http://en.wikipedia.org/wiki/Big_data).
- [2] C. Dobre and F. Xhafa. Intelligent services for big data science. *Future Generation Computer Systems*, 37(0):267 – 281, 2014. Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing  
Special Section: Semantics, Intelligent processing and services for big data  
Special Section: Advances in Data-Intensive Modelling and Simulation  
Special Section: Hybrid Intelligence for Growing Internet and its Applications.
- [3] Cloud computing, 2014.  
[http://en.wikipedia.org/wiki/Cloud\\_computing](http://en.wikipedia.org/wiki/Cloud_computing).
- [4] Jianting Zhang, S. You, and L. Gruenwald. High-performance spatial query processing on big taxi trip data using gpgpus. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 72–79, June 2014.
- [5] The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east, 2012. <http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf>.
- [6] Apache spark, 2015. <https://spark.apache.org/>.
- [7] Apache spark, 2015. <http://en.wikipedia.org/w/index.php?title=Spark&oldid=654641608>.
- [8] Nosql, 2014. <http://en.wikipedia.org/wiki/NoSQL>.
- [9] Hadoop, 2014. [http://en.wikipedia.org/wiki/Apache\\_Hadoop](http://en.wikipedia.org/wiki/Apache_Hadoop).

- 
- [10] Apache hadoop, 2014. <http://hadoop.apache.org/>.
- [11] Dhruba Borthakur. The hadoop distributed file system: Architecture and design, 2007. [http://hadoop.apache.org/docs/r0.18.0/hdfs\\_design.pdf](http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf).
- [12] Yang Jin, Tang Deyu, and Zhou Yi. A distributed storage model for ehr based on hbase. In *Information Management, Innovation Management and Industrial Engineering (ICIII), 2011 International Conference on*, volume 2, pages 369–372, Nov 2011.
- [13] Haijie Ding, Yuehui Jin, Yidong Cui, and Tan Yang. Distributed storage of network measurement data on hbase. In *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, volume 02, pages 716–720, Oct 2012.
- [14] Jun Bai. Feasibility analysis of big log data real time search based on hbase and elasticsearch. In *Natural Computation (ICNC), 2013 Ninth International Conference on*, pages 1166–1170, July 2013.
- [15] M.N. Vora. Hadoop-hbase for large-scale data. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, volume 1, pages 601–605, Dec 2011.
- [16] Lizhi Cai, Shidong Huang, Leilei Chen, and Yang Zheng. Performance analysis and testing of hbase based on its architecture. In *Computer and Information Science (ICIS), 2013 IEEE/ACIS 12th International Conference on*, pages 353–358, June 2013.
- [17] Hbase, 2014. <http://wiki.apache.org/hadoop/Hbase>.
- [18] Apache hbase, 2014. <https://hbase.apache.org/>.
- [19] Lars George. *HBase: The Definitive Guide*. O'REILLY, 2012.
- [20] Cloudera recommendations on hadoop/hbase cluster capacity planning, 2014. <http://www.cloudera.com/blog/2010/08/hadoophbase-capacity-planning/>.

- [21] Cloudera cdh, 2015. <http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html>.
- [22] Fuzzy clustering, 2015. [http://en.wikipedia.org/wiki/Fuzzy\\_clustering](http://en.wikipedia.org/wiki/Fuzzy_clustering).
- [23] Martin Ester, Hans peter Kriegel, Jörg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.
- [24] RicardoJ.G.B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In Jian Pei, VincentS. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, volume 7819 of *Lecture Notes in Computer Science*, pages 160–172. Springer Berlin Heidelberg, 2013.
- [25] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.
- [26] Gereon Frahling and Christian Sohler. A fast k-means implementation using coresets. In *Proceedings of the 22nd ACM Symposium on Computational Geometry, Sedona, Arizona, USA, June 5-7, 2006*, pages 135–143, 2006.
- [27] Seung-Heon Lee, Byung-Wook Lee, and Young-Kyu Yang. Estimation of link speed using pattern classification of gps probe car data. In MarinaL. Gavrilova, Osvaldo Gervasi, Vipin Kumar, C.J.Kenneth Tan, David Taniar, Antonio Laganá, Youngsong Mun, and Hyunseung Choo, editors, *Computational Science and Its Applications - ICCSA 2006*, volume 3981 of *Lecture Notes in Computer Science*, pages 495–504. Springer Berlin Heidelberg, 2006.
- [28] Yiliang Zeng, Jinhui Lan, Bin Ran, and Yaoliang Jiang. A novel multisensor traffic state assessment system based on incomplete data. *The Scientific World Journal*, 2014:1–13, 2014.

- [29] Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. Performance evaluation of nosql big-data applications using multi-formalism models. *Future Generation Computer Systems*, 37(0):345 – 353, 2014. Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for big data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications.
- [30] Chen Zhang and Xue Liu. Hbasemq: A distributed message queuing system on clouds with hbase. In *INFOCOM, 2013 Proceedings IEEE*, pages 40–44, April 2013.
- [31] Lei Gu and Huan Li. Memory or time: Performance evaluation for iterative operation on hadoop and spark. In *High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC\_EUC), 2013 IEEE 10th International Conference on*, pages 721–727, Nov 2013.
- [32] J. Urbani, A. Margara, C. Jacobs, S. Voulgaris, and H. Bal. Ajira: A lightweight distributed middleware for mapreduce and stream processing. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, pages 545–554, June 2014.
- [33] Mapreduce, 2014. <http://en.wikipedia.org/wiki/MapReduce>.
- [34] Apache zookeeper, 2014. <http://zookeeper.apache.org/>.
- [35] Paolo Atzeni, Francesca Bugiotti, and Luca Rossi. Uniform access to nosql systems. volume 43, pages 117 – 133, 2014.
- [36] Shoji Nishimura, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. MD-hbase: design and implementation of an elastic data infrastructure for cloud-scale location services. *Distributed and Parallel Databases*, 31(2):289–319, 2013.

- [37] Aisling O’ Driscoll, Jurate Daugelaite, and Roy D. Sleator. ‘Big data’, hadoop and cloud computing in genomics. *Journal of Biomedical Informatics*, 46(5): 774 – 781, 2013.
- [38] Karthik Kambatla, Giorgos Kollias, Vipin Kumar, and Ananth Grama. Trends in big data analytics. *Journal of Parallel and Distributed Computing*, 2014.
- [39] Daewoo Lee, Jin-Soo Kim, and Seungryoul Maeng. Large-scale incremental processing with mapreduce. volume 36, pages 66 – 79, 2014. Special Section: Intelligent Big Data Processing Special Section: Behavior Data Security Issues in Network Information Propagation Special Section: Energy-efficiency in Large Distributed Computing Architectures Special Section: eScience Infrastructure and Applications.
- [40] Dbscan spark, 2015.  
[https://github.com/alitouka/spark\\_dbscan](https://github.com/alitouka/spark_dbscan).
- [41] K-means,spark mllib, 2015.  
<https://spark.apache.org/docs/1.1.0/mllib-clustering.html>.
- [42] Scala of fuzzy-c-means clustering, 2015.  
<https://gist.github.com/kralo/8721440>.

# Appendix A

## Hadoop Installation

### I. Modify hosts

```
# sudo vim /etc/hosts
```

### II. Modify hostname

```
# sudo vim /etc/hostname
```

```
# sudo service hostname start
```

### III. Install Java JDK

```
# sudo apt-get -y install openjdk-7-jdk
```

```
# sudo ln -s /usr/lib/jvm/java-7-openjdk-amd64 /usr/lib/jvm/jdk
```

### IV. Add hadoop user

```
# sudo addgroup hadoop
```

```
# sudo adduser --ingroup hadoop hduser
```

```
# sudo adduser hduser sudo
```

### V. Create SSH authentication login

```
# ssh-keygen -t rsa -f ~/.ssh/id_rsa -P ""
```

```
# cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
```

```
# scp -r ~/.ssh hadoopi7-01: /
```

### VI. Download hadoop

```
# cd ~
```

```
# wget http://ftp.twaren.net/Unix/Web/apache/hadoop/common/hadoop-2.2.0/hadoop-2.2.0.tar.gz
```

```
# tar xzf hadoop-2.2.0.tar.gz
# mv hadoop-2.2.0.tar.gz hadoop
```

## VII. Add the environment variable

```
# vim .bashrc
```

```
export JAVA_HOME=/usr/lib/jvm/jdk/
export HADOOP_INSTALL=/home/hduser/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
```

## VIII. Set hadoop config

```
# cd hadoop/etc/hadoop
# vim hadoop-env.sh
```

```
export JAVA_HOME=/usr/lib/jvm/jdk/
```

```
# vim core-site.xml
```

```
<property>
  <name>fs.default.name</name>
  <value>hdfs://hadoopi7-01-master:9000</value>
</property>
```

```
# vim yarn-site.xml
```

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>hadoopi7-01</value>
</property>
```

```
# cp mapred-site.xml.template mapred-site.xml
# vim mapred-site.xml
```

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

```
# mkdir -p /mydata/hdfs/namenode
# mkdir -p /mydata/hdfs/datanode
# vim hdfs-site.xml
```

```
<property>
  <name>dfs.replication</name>
  <value>2</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>/home/hduser/mydata/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>/home/hduser/mydata/hdfs/datanode</value>
</property>
```

```
# vim slaves
```

```
hadoopi7-01
hadoopi7-02
hadoopi7-03
hadoopi7-04
```

## IX. Copy hadoop to all nodes

```
# scp -r /home/hduser/hadoop hadoopi7-01:/home/hduser
# scp -r /home/hduser/hadoop hadoopi7-02:/home/hduser
# scp -r /home/hduser/hadoop hadoopi7-03:/home/hduser
# scp -r /home/hduser/hadoop hadoopi7-04:/home/hduser
```

## X. Format HDFS



```
# hdfs namenode -format
```

XI. Start hadoop

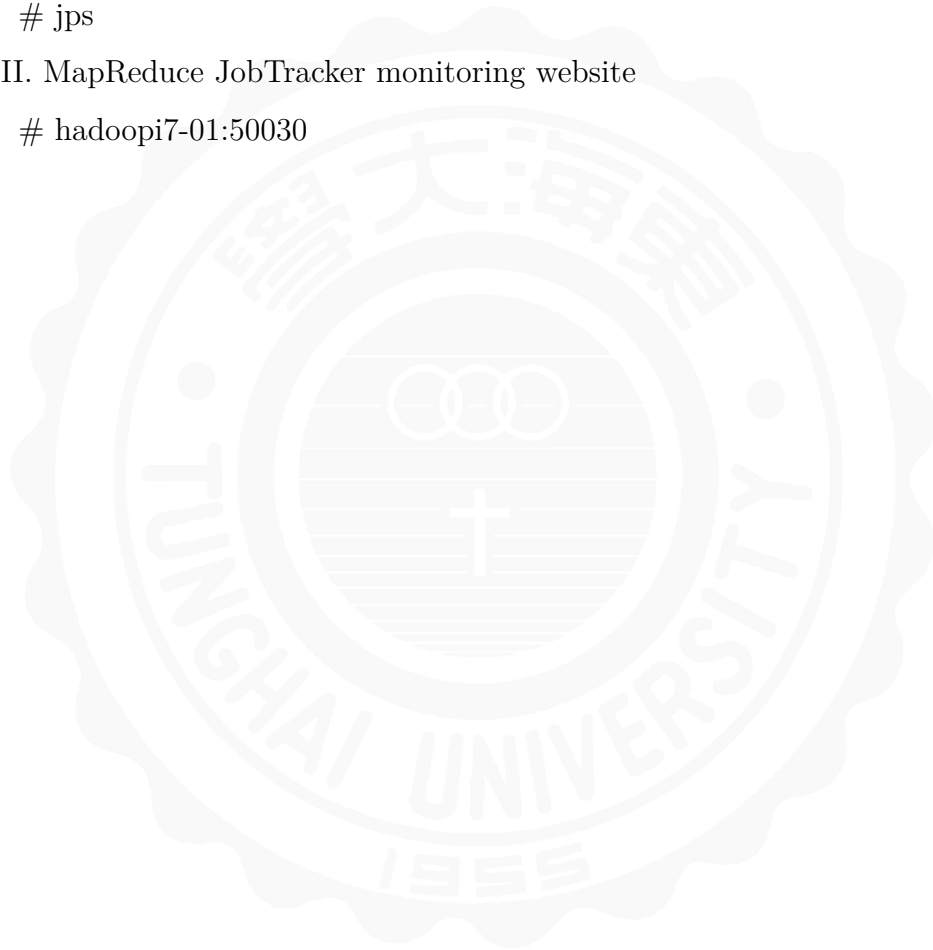
```
# start-all.sh
```

XII. Use jps to see java running program

```
# jps
```

XIII. MapReduce JobTracker monitoring website

```
# hadoopi7-01:50030
```



# Appendix B

## HBase Installation

### I. Download HBase

```
# cd ~  
# wget http://ftp.twaren.net/Unix/Web/apache/hbase/hbase-0.96.0/hbase-  
0.96.0-hadoop2-bin.tar.gz  
# tar xzf hbase-0.96.0-hadoop2-bin.tar.gz  
# mv hbase-0.96.0-hadoop2 hbase
```

### II. Set HBase config

```
# cd hbase  
# vim conf/hbase-env.sh
```

```
export JAVA_HOME=/usr/lib/jvm/jdk  
export HBASE_HOME=/home/hduser/hbase
```

```
# hadoop fs -mkdir /hbase  
# vim conf/hbase-site.xml
```

```
<property>  
  <name>hbase.rootdir</name>  
  <value>hdfs://hadoopi7-01:9000/hbase</value>  
</property>  
<property>  
  <name>hbase.cluster.distributed</name>  
  <value>true</value>
```

```
</property>
<property>
    <name>hbase.zookeeper.quorum</name>
    <value>Test-master</value>
</property>
```

```
# vim conf/regionservers
```

```
hadoopi7-01
hadoopi7-02
hadoopi7-03
hadoopi7-04
```

### III. Copy jar to hbase/lib

```
# rm lib/hadoop-*
# cd /home/hduser/hadoop/share/hadoop
# cp common/hadoop-common-2.2.0.jar common/lib/hadoop-annotations-2.2.0.jar
common/lib/hadoop-auth-2.2.0.jar hdfs/hadoop-hdfs-2.2.0.jar hdfs/hadoop-hdfs-
2.2.0-tests.jar mapreduce/hadoop-mapreduce-client-app-2.2.0.jar mapreduce/hadoop-
mapreduce-client-common-2.2.0.jar mapreduce/hadoop-mapreduce-client-core-2.2.0.jar
mapreduce/hadoop-mapreduce-client-jobclient-2.2.0.jar mapreduce/hadoop-mapreduce-
client-jobclient-2.2.0-tests.jar mapreduce/hadoop-mapreduce-client-shuffle-2.2.0.jar
yarn/hadoop-yarn-api-2.2.0.jar yarn/hadoop-yarn-client-2.2.0.jar yarn/hadoop-yarn-
common-2.2.0.jar yarn/hadoop-yarn-server-common-2.2.0.jar yarn/hadoop-yarn-
server-nodemanager-2.2.0.jar /home/hduser/hbase/lib/
```

### IV. Copy hbase to all nodes

```
# scp -r hbase/hadoop hadoopi7-01:/home/hduser
# scp -r hbase/hadoop hadoopi7-02:/home/hduser
# scp -r hbase/hadoop hadoopi7-03:/home/hduser
# scp -r hbase/hadoop hadoopi7-04:/home/hduser
# bin/start-hbase.sh
```

### V. HBase monitoring website

```
# hadoopi7-01:60010
```

# Appendix C

## Cloudera Manager Installation

### I. Download Cloudera

```
# wget http://archive.cloudera.com/cm4/installer/latest/cloudera-manager-  
installer.bin
```

### II. Install Cloudera

```
# sudo chmod +x cloudera-manager-installer.bin  
# sudo ./cloudera-manager-installer.bin
```

### III. Set Hosts

```
# sudo vim /etc/hosts
```

### IV. Install and set NTP

```
# sudo apt-get install ntp  
# ntpdate -s ntp.ubuntu.com pool.ntp.org
```

# Appendix D

## Experimental-Spark and Hadoop MapReduce Performance Comparison Source Code

Hadoop MapReduce WordCount Source Code [Java]

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
```

```

public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
    }
}

public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                      Context context
                      ) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

## Spark WordCount Source Code [Scala]

```

val textFile = spark.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                    .map(word => (word, 1))

```

```
.reduceByKey(_ + _)  
counts.saveAsTextFile("hdfs://...")
```



# Appendix E

## Experiment-HDFS read and write speed under various replication Data

Experiment Parameter Setting [Number of Map-File Size-Number of Replication-Read(R) or Write(W)]

TABLE E.1: Write 12G in HDFS [Map-Size-Replication-R/W]

Setting	Throughput	Average IO	IO rate	Write Time
12-1000-01-W	581.310856	624.62	162.6187698	26.68
12-1000-02-W	49.89687978	51.80	10.4422035	48.83
12-1000-03-W	30.33673779	30.41	1.554883783	59.03
12-1000-04-W	20.54326669	20.62	1.321273674	76.07
12-1000-05-W	17.17111589	17.19	0.581414858	86.15
12-1000-06-W	14.63100603	14.70	1.000010921	98.21
12-1000-07-W	12.12179984	12.14	0.420148262	110.24
12-1000-08-W	10.99109447	11.01	0.453686448	119.54
12-1000-09-W	10.03082808	10.05	0.484885475	130.04

*Continued on next page*



Table E.1 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
12-1000-10-W	9.495481338	9.52	0.473389491	135.34
12-1000-11-W	8.709227208	8.72	0.302664947	144.71
12-1000-12-W	8.3115962	8.31	0.1456952	146.60
24-500-01-W	80.04429117	334.936554	300.2163623	43.746
24-500-02-W	35.38497377	36.31741333	6.259091493	41.808
24-500-03-W	18.60733408	18.96840477	2.787038416	54.871
24-500-04-W	13.38632832	13.53074646	1.438529511	66.899
24-500-05-W	10.5875004	10.90009975	2.089580233	79.069
24-500-06-W	8.395618327	8.426784515	0.519008839	88.54
24-500-07-W	6.871396096	6.900475025	0.461444768	103.358
24-500-08-W	6.218724996	6.255727768	0.500981892	110.803
24-500-09-W	5.600959631	5.614548206	0.279177047	118.514
24-500-10-W	4.919734531	4.937342167	0.304098938	133.557
24-500-11-W	4.056839019	4.062527657	0.154649932	154.775
24-500-12-W	3.834936656	3.838791847	0.122197337	160.758
36-333-01-W	147.513751	251.3358002	159.718463	32.182
36-333-02-W	25.24097679	26.61917877	6.28261589	41.734
36-333-03-W	14.9046268	15.57662582	3.727489718	50.809
36-333-04-W	8.702646511	9.249137878	2.244672449	75.051
36-333-05-W	7.234228983	7.388176918	1.127956821	80.121
36-333-06-W	5.844959381	5.981119156	0.983059539	91.213
36-333-07-W	5.105076076	5.150288582	0.518104991	95.252
36-333-08-W	4.35569947	4.382982254	0.369705577	107.421
36-333-09-W	3.842558937	3.85191083	0.197094668	116.48
36-333-10-W	3.342273571	3.357328892	0.232648852	130.5
36-333-11-W	3.104327466	3.106856108	0.089809885	135.605
36-333-12-W	2.868586986	2.870315552	0.071868564	146.667
48-250-01-W	75.57769702	238.582016	193.9311365	38.753
48-250-02-W	22.21054112	24.04099464	7.257185815	40.744

*Continued on next page*

Table E.1 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
48-250-03-W	12.11725871	12.70887756	3.122918435	51.93
48-250-04-W	7.235322846	7.632677078	2.238084331	67.019
48-250-05-W	5.733471	5.98286581	1.406185626	78.146
48-250-06-W	4.798662133	4.8323493	0.435738345	83.155
48-250-07-W	3.992633591	4.051212788	0.512243893	96.239
48-250-08-W	3.230157613	3.242685556	0.209554647	107.328
48-250-09-W	2.882245846	2.894417763	0.196868521	118.439
48-250-10-W	2.610244208	2.630482435	0.263859065	127.449
48-250-11-W	2.351893431	2.355366468	0.091837056	136.723
48-250-12-W	2.174949296	2.178193808	0.085369838	146.652
60-200-01-W	112.8710636	186.5179901	140.4396428	33.652
60-200-02-W	18.68742632	19.31556892	3.637570241	39.762
60-200-03-W	9.245134748	9.627094269	2.158185096	52.226
60-200-04-W	6.358124544	6.545865059	1.163472977	64.107
60-200-05-W	5.053382671	5.188427448	0.887556934	73.099
60-200-06-W	4.000772149	4.105906963	0.740164075	84.07
60-200-07-W	3.121406318	3.156785727	0.36078487	97.204
60-200-08-W	2.767286431	2.781476736	0.205413745	104.297
60-200-09-W	2.446113147	2.451550961	0.118338132	114.394
60-200-10-W	2.127490959	2.136369944	0.145476291	125.491
60-200-11-W	1.980887735	1.987024188	0.116903136	131.626
60-200-12-W	1.788849564	1.79017961	0.048834133	141.575

TABLE E.2: Read 12G in HDFS [Map-Size-Replication-R/W]

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
12-1000-01-R	70.95008071	76.37	21.52427863	42.779

*Continued on next page*

Table E.2 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
12-1000-02-R	53.22546306	54.80	10.40072876	47.319
12-1000-03-R	155.0948664	656.93	755.8049565	35.617
12-1000-04-R	96.91018041	272.24	437.998826	39.811
12-1000-05-R	119.8765272	187.73	159.4773912	36.766
12-1000-06-R	177.1270001	558.18	686.6947194	35.812
12-1000-07-R	152.2784666	318.91	346.9360105	35.824
12-1000-08-R	125.1616672	152.16	88.9018017	35.815
12-1000-09-R	223.9558061	266.97	125.9026513	31.992
12-1000-10-R	283.3128718	661.83	664.1442753	29.74
12-1000-11-R	239.2916966	771.92	736.7028048	32.88
12-1000-12-R	375.798572	994.89	783.590854	30.777
24-500-01-R	51.49617855	251.7518311	434.483062	41.742
24-500-02-R	37.11619534	39.50246811	11.53053896	42.796
24-500-03-R	55.85682035	105.6205826	222.1301516	36.671
24-500-04-R	72.29699606	177.8743896	308.7583854	35.641
24-500-05-R	102.3419244	338.9202576	474.1521673	32.707
24-500-06-R	112.8912387	245.1373291	330.3180795	32.834
24-500-07-R	148.0878161	470.6677246	512.6867214	31.764
24-500-08-R	99.95002499	294.8895569	436.1911771	32.723
24-500-09-R	135.0864553	491.5281067	549.5282025	32.302
24-500-10-R	169.1570341	522.6466675	541.0930128	31.213
24-500-11-R	165.3826541	643.2017212	593.137881	30.791
24-500-12-R	170.604794	625.3359985	563.9123504	31.936
36-333-01-R	39.02877347	148.0826111	313.2278793	39.15
36-333-02-R	42.09698319	125.7908554	277.3629387	36.802
36-333-03-R	41.29805705	131.9345398	287.5653931	36.648
36-333-04-R	50.21278022	116.2440338	228.235154	34.733
36-333-05-R	75.79522391	218.6917419	331.8756358	32.757
36-333-06-R	64.42909736	236.1207886	378.5815278	34.746

*Continued on next page*

Table E.2 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
36-333-07-R	78.34526027	260.7580566	382.9924138	32.763
36-333-08-R	91.84869636	372.2658997	412.6304414	32.806
36-333-09-R	106.7535798	474.7655029	463.6453707	33.159
36-333-10-R	96.24122927	474.425293	470.6923599	32.832
36-333-11-R	93.35866924	425.3148499	437.2215868	30.992
36-333-12-R	96.51863064	509.6042786	470.3336693	32.04
48-250-01-R	29.6890811	88.99221039	200.6193919	39.647
48-250-02-R	34.93988883	186.5760651	351.1953206	37.69
48-250-03-R	32.03006555	101.0324707	237.5735791	37.725
48-250-04-R	31.58451622	48.22178268	77.02294267	36.741
48-250-05-R	51.96447362	166.6552429	274.6868223	35.458
48-250-06-R	55.44415388	284.349762	405.7461033	34.851
48-250-07-R	56.86314463	257.5050354	361.8522884	34.734
48-250-08-R	55.67618729	326.2840271	418.8923817	35.306
48-250-09-R	82.74321333	422.4218445	429.3871406	33.914
48-250-10-R	59.17072233	387.1660461	434.9440872	34.189
48-250-11-R	63.57784312	443.8209534	449.7665632	33.787
48-250-12-R	62.77989369	393.0908508	382.8376637	34.911
60-200-01-R	25.11038105	93.96453094	232.1722298	37.665
60-200-02-R	29.63116606	152.7574768	298.0393957	36.917
60-200-03-R	25.96626982	82.733284	202.1670503	38.834
60-200-04-R	31.37312317	99.97427368	223.271127	35.779
60-200-05-R	45.76414684	160.6618958	282.3780112	33.849
60-200-06-R	45.26713267	162.3869934	288.7264254	34.078
60-200-07-R	54.90860006	214.2892151	310.7553667	33.764
60-200-08-R	42.5862638	273.5256653	354.7136492	35.811
60-200-09-R	45.86437141	250.1865845	350.9927899	34.861
60-200-10-R	47.13960788	328.1400452	380.5812801	34.756
60-200-11-R	54.04285598	357.3202209	365.554686	35.759

*Continued on next page*

Table E.2 – *Continued from previous page*

Setting	Throughput	Average IO	IO rate	Write Time
60-200-12-R	38.67524398	212.3633728	273.1053358	38.021

TABLE E.3: Write 60G in HDFS [Map-Size-Replication-R/W]

Setting	Throughput	Average IO	IO rate	Write Time
12-5000-01-W	108.6795078	120.4581375	51.60289325	87.936
12-5000-02-W	52.77620427	53.41374588	5.813535113	135.428
12-5000-03-W	29.57836047	29.63490105	1.316167053	200.97
12-5000-04-W	20.89963161	20.9282074	0.783923315	272.457
12-5000-05-W	16.74795005	16.76549149	0.549844023	334.854
12-5000-06-W	14.35387812	14.36470509	0.397081789	382.157
12-5000-07-W	12.60618346	12.61092472	0.246668578	429.503
12-5000-08-W	10.90177879	10.90343189	0.13460847	488.861
12-5000-09-W	10.02619008	10.03025723	0.202551467	535.079
12-5000-10-W	9.112183278	9.112452507	0.049547077	577.538
12-5000-11-W	8.478975039	8.479894638	0.089173251	621.896
12-5000-12-W	7.840011499	7.840697765	0.073431234	671.128
24-2500-01-W	77.18590482	126.0204773	132.6610587	79.092
24-2500-02-W	34.25811204	34.72610092	4.37067473	105.266
24-2500-03-W	18.60905963	18.72963142	1.647153227	167.698
24-2500-04-W	13.43357467	13.46750164	0.680492094	220.925
24-2500-05-W	10.00799806	10.02917194	0.477745374	284.507
24-2500-06-W	8.386251747	8.397296906	0.31332852	332.775
24-2500-07-W	6.792268542	6.802690029	0.270810223	407.361
24-2500-08-W	6.079985862	6.086941242	0.211722815	448.617
24-2500-09-W	5.372483697	5.377141476	0.162661623	505.096
24-2500-10-W	4.39937045	4.4028759	0.125778397	608.635

*Continued on next page*

Table E.3 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
24-2500-11-W	4.633160269	4.633965492	0.061364078	572.846
24-2500-12-W	4.279154342	4.279790401	0.052704277	616.755
36-1666-01-W	63.45948131	77.02295685	47.4897065	62.037
36-1666-02-W	24.28674689	25.56377983	5.505922469	130.354
36-1666-03-W	13.24784465	13.34307194	1.261572987	159.937
36-1666-04-W	8.992075261	9.081813812	0.967945063	227.184
36-1666-05-W	6.867847638	6.974319458	0.933005418	303.765
36-1666-06-W	5.930964428	5.94067955	0.247250022	315.664
36-1666-07-W	4.989083676	4.997880936	0.217419072	369.992
36-1666-08-W	4.177857647	4.181807995	0.130976709	435.695
36-1666-09-W	3.575787668	3.584031343	0.178793567	512.063
36-1666-10-W	3.321625273	3.324679136	0.103654851	540.917
36-1666-11-W	2.84006023	2.841899633	0.073525375	625.525
36-1666-12-W	2.905792836	2.906923532	0.057587769	610.977
48-1250-01-W	42.32872914	76.62815094	133.8458459	90.065
48-1250-02-W	20.41215544	20.75228691	2.755553709	96.151
48-1250-03-W	11.0141054	11.08403397	0.930636529	147.476
48-1250-04-W	7.45715677	7.489223957	0.506050444	206.096
48-1250-05-W	5.560317865	5.576531887	0.315778479	262.42
48-1250-06-W	4.568043173	4.577654362	0.219647071	311.57
48-1250-07-W	3.401669857	3.441177607	0.410499182	432.329
48-1250-08-W	3.160217164	3.173014641	0.220270284	439.361
48-1250-09-W	2.685763159	2.690415144	0.114930071	509.788
48-1250-10-W	2.566183915	2.567529917	0.059789553	526.227
48-1250-11-W	2.341117158	2.342276812	0.052908061	574.302
48-1250-12-W	2.017604677	2.018187046	0.034314243	661.624
60-1000-01-W	33.97581601	73.4743576	122.5346142	85.138
60-1000-02-W	15.05204621	15.49670315	2.73681037	105.117
60-1000-03-W	8.397047094	8.636459351	1.522077939	169.742

*Continued on next page*

Table E.3 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
60-1000-04-W	5.770140121	5.836211681	0.652776523	224.742
60-1000-05-W	4.484748045	4.559632778	0.608908209	286.69
60-1000-06-W	3.719315646	3.734349728	0.250976413	309.605
60-1000-07-W	2.83354884	2.862062931	0.315688276	413.339
60-1000-08-W	2.242070944	2.267105103	0.264834321	513.707
60-1000-09-W	1.929206358	1.9465698	0.200081045	586.241
60-1000-10-W	1.825006624	1.829602361	0.097241419	594.352
60-1000-11-W	1.791470665	1.793897986	0.068532125	602.291
60-1000-12-W	1.666593198	1.667279482	0.034166441	641.613

TABLE E.4: Read 60G in HDFS [Map-Size-Replication-R/W]

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
12-5000-01-R	64.45548543	72.40193939	29.01877915	120.362
12-5000-02-R	64.79040843	67.44035339	14.1223865	117.32
12-5000-03-R	72.25555344	121.5155563	180.0672206	107.162
12-5000-04-R	68.99065522	70.00814819	8.364283431	105.21
12-5000-05-R	78.07386568	83.45985413	28.35470632	96.61
12-5000-06-R	109.2703111	333.0221863	750.0432727	84.407
12-5000-07-R	86.01164024	88.55670166	16.18616202	90.505
12-5000-08-R	106.5234991	122.2713852	64.07273823	84.148
12-5000-09-R	106.4875782	116.1943283	43.41032359	81.185
12-5000-10-R	169.2505585	174.3617554	34.22027311	60.116
12-5000-11-R	136.8057677	139.3972473	17.21376458	71.193
12-5000-12-R	147.7526817	151.606308	23.5799015	67.153
24-2500-01-R	36.71083184	41.09889603	14.65097714	132.39
24-2500-02-R	43.25901914	43.63182449	4.162791465	89.206

*Continued on next page*

Table E.4 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
24-2500-03-R	47.38849881	82.95085907	135.0640426	87.572
24-2500-04-R	44.74816849	46.57197571	11.61009228	89.168
24-2500-05-R	50.15481118	54.50816345	24.25031229	82.395
24-2500-06-R	47.20436105	49.97365189	13.13708841	93.23
24-2500-07-R	54.95683141	63.91769028	29.94386941	88.223
24-2500-08-R	59.75018448	67.32119751	26.49759384	87.245
24-2500-09-R	77.11091119	94.80745697	52.25029165	80.14
24-2500-10-R	72.99456555	85.23986816	39.95552743	85.265
24-2500-11-R	91.67569418	95.447052	20.15275916	63.297
24-2500-12-R	91.39584301	95.44046783	21.42355391	62.07
36-1666-01-R	34.74172802	142.549118	437.2029337	96.089
36-1666-02-R	31.33409715	33.71170044	12.98911761	86.159
36-1666-03-R	38.69239622	54.54063416	63.19965331	79.206
36-1666-04-R	36.3262604	94.9706955	337.9667672	81.141
36-1666-05-R	31.17467977	33.28878784	9.343693171	95.374
36-1666-06-R	44.60458942	58.90699768	54.55330678	74.053
36-1666-07-R	42.19668145	57.09797668	82.87845701	83.703
36-1666-08-R	44.75844277	50.57923508	24.45288563	75.193
36-1666-09-R	45.46198767	52.28852844	25.08685359	78.157
36-1666-10-R	54.45260047	63.05774689	32.05518695	73.101
36-1666-11-R	49.99237311	54.21111298	15.74734772	80.242
36-1666-12-R	52.49655354	54.99844742	13.52594008	66.165
48-1250-01-R	23.84995516	77.04341888	243.6585708	130.585
48-1250-02-R	28.75341027	116.6693115	347.9150936	77.198
48-1250-03-R	29.01035525	64.14601135	229.9192712	78.064
48-1250-04-R	31.44024624	37.40484238	32.12495187	75.213
48-1250-05-R	31.31037031	32.70367432	7.574166729	78.365
48-1250-06-R	33.98461064	37.13421631	14.43613625	77.097
48-1250-07-R	30.20381535	37.78124237	31.90931579	95.316

*Continued on next page*



Table E.4 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
48-1250-08-R	34.11312138	39.31318283	17.99134852	84.204
48-1250-09-R	32.78996714	39.1232872	22.72456006	85.163
48-1250-10-R	34.13703288	36.5867157	12.12214057	76.482
48-1250-11-R	36.07317082	37.37425232	7.332187034	71.125
48-1250-12-R	34.69463517	36.06271744	8.395726503	72.154
60-1000-01-R	23.07785507	67.84573364	228.0957759	138.842
60-1000-02-R	21.98984655	71.01034546	263.8391725	81.36
60-1000-03-R	23.16908256	52.09474564	202.9073468	86.161
60-1000-04-R	25.86554744	54.90973663	166.2872929	83.471
60-1000-05-R	27.61929926	53.10790634	183.1140448	86.378
60-1000-06-R	27.06126806	30.4484024	15.01428408	78.07
60-1000-07-R	25.90437572	30.51843452	16.83096926	86.234
60-1000-08-R	25.03442233	28.59828377	13.7070189	86.33
60-1000-09-R	23.47774232	26.27509308	10.46570508	89.266
60-1000-10-R	27.74462552	31.18267441	14.32317667	78.1
60-1000-11-R	28.06378149	29.7088604	7.319199864	76.156
60-1000-12-R	27.98525737	29.29996872	6.336384812	77.236

TABLE E.5: Write 120G in HDFS [Map-Size-Replication-R/W]

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
12-10000-01-W	141.3908857	160.6730499	53.97045209	128.426
12-10000-02-W	51.90598787	52.11640549	3.309536677	235.73
12-10000-03-W	29.05262302	29.09033394	1.055974055	381.933
12-10000-04-W	21.63881181	21.65653419	0.618931715	504.45
12-10000-05-W	16.94111508	16.94874954	0.360167319	630.84
12-10000-06-W	14.51457106	14.51840973	0.236264799	726.515

*Continued on next page*

Table E.5 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
12-10000-07-W	12.30835633	12.31240559	0.225934157	851.759
12-10000-08-W	11.12853861	11.13055134	0.150055543	936.235
12-10000-09-W	10.07884343	10.07960796	0.087992157	1026.678
12-10000-10-W	9.336582879	9.338303566	0.128124136	1108.479
12-10000-11-W	8.661998427	8.663191795	0.102216677	1196.472
12-10000-12-W	8.086019071	8.087031364	0.090949404	1280.357
24-5000-01-W	52.0888723	72.22106934	44.49078156	209.861
24-5000-02-W	30.36176806	30.73362732	3.318878211	220.109
24-5000-03-W	18.49744627	18.55735779	1.0493455	318.792
24-5000-04-W	13.0502275	13.08236694	0.664181782	424.45
24-5000-05-W	9.774758608	9.79019165	0.390539741	561.288
24-5000-06-W	7.590583653	7.600771427	0.281829806	715.193
24-5000-07-W	7.015047979	7.019438267	0.177349771	755.356
24-5000-08-W	6.154044661	6.156385899	0.121821815	850.996
24-5000-09-W	5.298122014	5.301908493	0.145490183	989.62
24-5000-10-W	4.862730586	4.863732338	0.070292983	1070.184
24-5000-11-W	4.566310824	4.5669837	0.055755924	1133.058
24-5000-12-W	4.045486915	4.04573679	0.03177255	1271.754
36-3333-01-W	44.45738778	81.17752838	98.32321192	142.536
36-3333-02-W	22.56140842	23.15909958	3.677726821	223.876
36-3333-03-W	13.95452533	14.00731659	0.888613852	277.409
36-3333-04-W	9.443379116	9.473526955	0.552977716	399.192
36-3333-05-W	7.097152403	7.10661459	0.266735997	509.879
36-3333-06-W	5.540251645	5.549935818	0.234750646	663.743
36-3333-07-W	4.402772277	4.415668964	0.245302542	821.841
36-3333-08-W	3.968970528	3.97596693	0.172075618	898.051
36-3333-09-W	3.418864789	3.42083931	0.083335679	1024.695
36-3333-10-W	3.34184447	3.342784882	0.056613454	1041.575
36-3333-11-W	2.996859427	2.997683525	0.050944124	1150.313

*Continued on next page*

Table E.5 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
36-3333-12-W	2.745504011	2.745731592	0.024964893	1257.576
48-2500-01-W	36.18235182	56.10339737	92.5058466	138.456
48-2500-02-W	19.62029497	19.84492683	2.099110627	176.775
48-2500-03-W	10.76563587	10.82552242	0.847078218	274.465
48-2500-04-W	7.477103706	7.501808643	0.448412437	379.053
48-2500-05-W	5.512850109	5.527158737	0.288878632	505.816
48-2500-06-W	4.482977462	4.489147663	0.170532549	603.384
48-2500-07-W	3.664668201	3.674994707	0.204098181	741.26
48-2500-08-W	2.984666129	3.002048492	0.247038659	919.993
48-2500-09-W	2.722244062	2.72567296	0.099480388	975.242
48-2500-10-W	2.156378754	2.159336329	0.082313756	1224.295
48-2500-11-W	1.995232525	1.996933818	0.059495357	1308.935
48-2500-12-W	1.971476805	1.971582055	0.014416835	1304.802
60-2000-01-W	29.87721213	39.67823029	35.26349413	129.432
60-2000-02-W	15.54196785	16.25720978	3.444356495	219.117
60-2000-03-W	8.975885163	9.042650223	0.811452688	276.455
60-2000-04-W	5.659760391	5.728065968	0.636060712	446.362
60-2000-05-W	4.149651367	4.227015495	0.643890655	596.193
60-2000-06-W	2.94731266	2.975856781	0.315659066	770.135
60-2000-07-W	2.505448463	2.537169933	0.305622381	913.893
60-2000-08-W	2.658427255	2.661175489	0.088489856	802.408
60-2000-09-W	2.075150216	2.082937479	0.13488318	1042.543
60-2000-10-W	1.698262318	1.70644784	0.12496641	1263.48
60-2000-11-W	1.643253387	1.645328045	0.060286997	1292.713
60-2000-12-W	1.418990707	1.419016719	0.006000704	1445.447

TABLE E.6: Read 120G in HDFS [Map-Size-Replication-R/W]

Setting	Throughput	Average IO	IO rate	Write Time
12-10000-01-R	76.46711723	83.19487	23.58723695	205.645
12-10000-02-R	71.53907375	72.87535095	11.53261006	176.828
12-10000-03-R	63.85148995	66.56378174	15.81297158	217.973
12-10000-04-R	65.34858569	65.56169891	3.75462772	192.806
12-10000-05-R	89.24793001	100.1324692	40.76054723	168.673
12-10000-06-R	80.75201661	86.68132782	29.30136624	176.816
12-10000-07-R	88.52795713	91.59468842	18.79699754	169.742
12-10000-08-R	100.3731371	107.4131546	29.78143297	150.683
12-10000-09-R	163.7197119	165.0995636	14.73089725	100.41
12-10000-10-R	127.6641921	132.2790375	24.02985599	124.581
12-10000-11-R	131.058593	135.6979065	25.43001366	124.442
12-10000-12-R	139.4340372	144.0866852	26.53987649	118.408
24-5000-01-R	31.52446811	39.69117355	20.46127069	319.695
24-5000-02-R	41.44367666	46.45299149	21.7368731	165.732
24-5000-03-R	50.79345729	81.26520538	132.381837	144.423
24-5000-04-R	43.38492041	45.41969299	12.59962154	158.587
24-5000-05-R	50.81518137	55.64311218	19.4425065	163.735
24-5000-06-R	52.70196381	58.09799194	21.84289032	154.604
24-5000-07-R	60.46945461	66.73223114	26.72769863	131.504
24-5000-08-R	63.49176115	68.47698212	21.82606335	130.399
24-5000-09-R	57.58325939	63.08963394	22.07604805	145.989
24-5000-10-R	74.7476333	78.1186142	17.48764036	114.421
24-5000-11-R	73.07933797	78.09183502	21.94332566	121.549
24-5000-12-R	73.09741144	77.81681061	20.33797783	124.604
36-3333-01-R	28.04336592	157.4050903	523.3120902	201.946
36-3333-02-R	31.89752639	35.1461525	16.37073659	160.496
36-3333-03-R	34.36893471	45.36082458	50.24114299	148.456
36-3333-04-R	36.04617977	45.64379501	37.23226602	144.475

*Continued on next page*

Table E.6 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
36-3333-05-R	34.71547533	36.75190735	9.819535773	152.616
36-3333-06-R	34.97026065	37.62696838	14.09042177	150.678
36-3333-07-R	36.92056607	40.66026306	16.89315328	154.49
36-3333-08-R	41.89952328	46.88611221	20.33490594	139.13
36-3333-09-R	35.97628566	40.04883194	14.37067685	182.862
36-3333-10-R	51.19289235	53.58546829	12.53492893	121.784
36-3333-11-R	50.62703636	53.67988586	13.97608449	115.495
36-3333-12-R	49.28063611	50.57766342	8.049275244	124.542
48-2500-01-R	22.41756222	36.70800781	70.26369622	343.775
48-2500-02-R	26.12694779	77.49364471	342.9304672	140.492
48-2500-03-R	25.70036713	26.5795002	6.754480267	139.481
48-2500-04-R	28.55167353	32.49034882	20.19574407	140.678
48-2500-05-R	25.8058134	29.38248634	14.87194491	174.295
48-2500-06-R	25.58040336	27.83644867	10.15264282	153.556
48-2500-07-R	29.81347201	32.07357788	9.652018256	146.592
48-2500-08-R	32.82054873	36.89367294	17.3284022	140.629
48-2500-09-R	33.336454	34.85076523	7.930251237	133.574
48-2500-10-R	33.06368065	35.81889725	10.38378574	141.151
48-2500-11-R	34.42080118	35.66928864	6.832026032	126.581
48-2500-12-R	37.60718046	38.9940033	7.848257664	125.563
60-2000-01-R	23.64916464	86.97358704	322.8507199	198.832
60-2000-02-R	21.34626253	25.12637329	19.96977902	167.769
60-2000-03-R	24.3198153	41.52109909	117.7956233	155.666
60-2000-04-R	20.45802099	21.72548866	6.19638618	149.039
60-2000-05-R	24.28291539	27.0144577	16.95734093	158.76
60-2000-06-R	22.6294091	24.60138702	9.669112017	167.619
60-2000-07-R	22.30849413	23.78671074	7.412041817	156.655
60-2000-08-R	26.0551579	27.35909653	6.01854968	132.482
60-2000-09-R	23.65222245	25.35221672	7.76954409	143.644

*Continued on next page*

Table E.6 – *Continued from previous page*

<b>Setting</b>	<b>Throughput</b>	<b>Average IO</b>	<b>IO rate</b>	<b>Write Time</b>
60-2000-10-R	23.65399411	24.90903473	6.476827204	144.632
60-2000-11-R	26.77834426	27.8975544	5.89364612	132.065
60-2000-12-R	24.30964163	24.99217033	4.22127426	140.104

