

東海大學

資訊工程研究所

碩士論文

指導教授：楊朝棟博士

混合型資料庫轉換器應用於空氣汙染監測服務

The Implementation of Air Pollution Monitoring Service

Using Hybrid Database Converter

研究生：孫培倫

中華民國一零五年六月

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 孫培倫 所提之論文

混合型資料庫轉換器應用於空氣汙染監測服務

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

伍朝欽

簽章

委

員

時文中

劉榮春

呂芳心

指導教授

楊朝棟

簽章

中華民國 105 年 6 月 27 日

## 摘要

隨著空氣污染日益的嚴重，污染程度已經開始為害到人體的健康，人們開始重視空污因子的即時濃度變化量並著手對空污因子作監控與紀錄分析。由於不斷的從空污監測站收集資料，資料庫儲存的資料量越來越大，舊有的關聯式資料庫已經無法處理過於龐大的資料，為了保持監控的流暢，必須刪除未整理過的歷史資料。然而，在作空污分析時，歷史資料是一個非常重要的指標。因此在不變動原本資料庫架構的情況下，如何將大部分的資料庫資料轉存到分散式資料庫變成一個相當重要的事情。為了實現此目標，我們空氣污染監測系統結合 Hadoop 叢集將舊有的資料庫資料轉移至分散式資料庫並且備份。這樣不但降低舊有資料庫的負擔且提升了服務的品質。資料轉存必須在不影響空氣污染的即時監控服務前提下進行，在這部分我們特別強調網頁服務不中斷的部分。透過最佳化轉存的模型提高約 60% 的效能，並且舊有資料庫受損時備份的資料讓受損的服務可以透過分散式資料庫和 MapReduce 的方式讓服務可以快速的重新啟動。最後，空氣污染監測系統提供一個關於空氣中污染因子變化量的訊息，供相關單位作為環境檢測和分析的重要依據，讓人們生活在一個更為舒適的環境。

關鍵字: NoSQL, 環境監控, 雲端運算, 巨量資料, 混合資料庫系統

# Abstract

As air pollution becomes more and more serious, pollution hurts human health, people start to pay attention on real time value of air pollution factors monitoring and recording analysis. Because our system need to get data from air pollution monitoring stations usually. Among of data is growing faster, RDB (Relational database) is hard to process huge data. In order to maintain smooth monitoring, we must remove the historical data is not consolidated. But when we analysis air pollution data, historical data is an important target. So, how to dump data to NoSQL without change RDB system become an important things. In order to achieve our goal, this paper proposed an air pollution monitoring system combines Hadoop cluster to dump data from RDB to NoSQL and backup. This will not only reduce the loading of RDB and also keep the service performance. Dump data to NoSQL need to processing without affecting the real time monitoring on air pollution monitoring system. In this part we focus on without interruption web service. Improve 60% efficiency up, through optimization of dump method and data backup service let service quickly restart by MapReduce and distributed databases when RDB impaired. And through three different types of conversion mode get the best data conversion to be our system. At last, air pollution monitoring service provide message about air pollution factors variation, as an important basis of environment detection and analysis, let people live in a more comfortable environment.

**Keywords:** NoSQL, Sensors, Cloud Computing, Big Data, Hybrid database system

## 致謝詞

研究所兩年的生活，歷經許多挑戰，從中不斷的克服困難解決問題，也增加許多寶貴的經驗，也獲得許多貴人的幫忙。首先要感謝我的指導教授楊朝棟博士，於碩士班兩年內各項能力指導上所花費的精力與時間，讓我不只將學術中學到的能力與知識妥善應用在這篇論文中，讓本論文得以完成，同時教導我們待人處事方法。謝謝您孜孜不倦的教導，讓我接觸各項新的事務，擴展我的視野和國際觀，也給予我很多鼓勵，使我有動力繼續往前邁進，至此之後，也會繼續努力的遵循和實踐教授所給予的教誨。另外我還要謝謝劉榮春老師與陳碩聰博士對我的研究提供了很多的建議、指導和鼓勵，讓我的論文研究可以更加完整。

特別感謝口試委員時文中教授、伍朝欽教授、呂芳懌教授與劉榮春教授特地撥空於論文口試時給予指正與建議，在修正之後讓我的論文能更加嚴謹完善，同時也給我許多啟發與收穫，學生於此衷心感謝。

我也要謝謝實驗室的學長姊、同學、學弟以及好朋友們，謝謝你們陪我一起度過這兩年的時光，當我有困難的時候、需要幫忙的時候，彼此互相鼓勵打氣，開心的時候，一起出去玩，經歷了各種的風風浪浪，因為有大家的陪伴在我研究所生活，增添了許多快樂，讓我不只學會了需多技術，同時也增廣了各方面的見聞。

最後要特別感謝我的家人，在求學的這段日子中，使我可以專心地在學術研究中，因為你們對我的關心，讓我堅定自己的人生道路，也因為有你們，碩士兩年不時回家充電，才得以充滿活力，在度過的每一天才得以不孤單，由衷感謝一路陪伴的所有人。最後，僅將此論文與各位分享。

東海大學資訊工程學系 高效能實驗室 孫培倫 105 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 . . . . .	1
1.2 Contributions . . . . .	2
1.3 Thesis Organization . . . . .	2
<b>2 Background and Related Work</b>	<b>3</b>
2.1 Cloud Computing and Big Data . . . . .	3
2.1.1 Cloud Computing . . . . .	3
2.1.2 Big Data . . . . .	5
2.1.3 NoSQL . . . . .	7
2.2 Internet of Things and Wireless Sensor Network . . . . .	8
2.2.1 Internet of Things (IoT) . . . . .	8
2.3 Hadoop Ecosystem . . . . .	10
2.3.1 Hadoop . . . . .	10
2.3.2 HDFS . . . . .	11
2.3.3 HBase . . . . .	13
2.3.4 Hive . . . . .	16
2.3.5 Sqoop . . . . .	18
2.4 Related Work . . . . .	19
<b>3 System Design and Implementation</b>	<b>21</b>
3.1 The Proposed System Architecture . . . . .	21
3.2 Design Detail of Hybrid Database Converter . . . . .	23
3.2.1 Database Controller . . . . .	23

3.2.2	Data Dumping . . . . .	24
3.3	Hybrid Database Converter Implementation . . . . .	27
3.3.1	Cluster Deployment . . . . .	28
3.3.2	Data Converter . . . . .	30
3.3.3	Data Converter . . . . .	31
<b>4</b>	<b>Experimental Results</b>	<b>33</b>
4.1	Experimental Environment . . . . .	33
4.1.1	Compare massive storage methods . . . . .	35
4.2	Data Dumping Experiment . . . . .	35
4.2.1	First dump . . . . .	37
4.2.2	Batch of Dump . . . . .	38
4.2.3	Database Disaster Recovery . . . . .	38
4.2.4	The Three Models Compare . . . . .	39
4.3	Platform Implementation . . . . .	40
<b>5</b>	<b>Conclusions and Future Work</b>	<b>46</b>
5.1	Concluding Remarks . . . . .	47
5.2	Future Work . . . . .	47
	<b>References</b>	<b>48</b>
	<b>Appendix</b>	<b>52</b>
<b>A</b>	<b>Hadoop Installation</b>	<b>52</b>
<b>B</b>	<b>HBase Installation</b>	<b>56</b>
<b>C</b>	<b>Hive Installation</b>	<b>58</b>
<b>D</b>	<b>Sqoop Installation</b>	<b>61</b>

# List of Figures

2.1	Service Mode	4
2.2	Big data 4V	6
2.3	Compare with NoSQL and RDBMS	8
2.4	Three classes of IoT	9
2.5	Apache Hadoop Ecosystem	11
2.6	HDFS Architecture	13
2.7	HBase Service Architecture	14
2.8	Data Model of HBase	15
2.9	Hive Architecture	17
2.10	Sqoop basic workflow	19
3.1	Air pollution monitoring system data accesss	22
3.2	Join hybrid database converter	23
3.3	Hybrid Database Converter	24
3.4	Transformation types between RDB and NoSQL databases	25
3.5	Three kinds of data conversion model	26
3.6	Hadoop NameNode information	28
3.7	Hadoop cluser information	29
3.8	Air pollution monitoring platform architecture	32
4.1	Spark and Hadoop Computing cluster	34
4.2	Data Transform Time	35
4.3	Variation of Three Kinds of Data	36
4.4	Data Growth of Three Kind of Data	36
4.5	First Dump Time	37
4.6	Batch of Dump Time	38
4.7	DB Disaster Recovery	39
4.8	Experiment Results	40
4.9	Air Quality	41
4.10	Daily Stats	41
4.11	Daily Values Record	42
4.12	Monthly Values Record	42
4.13	Annual Values Record	43
4.14	Regional Numerical Comparison	43
4.15	Value Record of Each Years	44
4.16	Numerical Comparison of Each Years	44

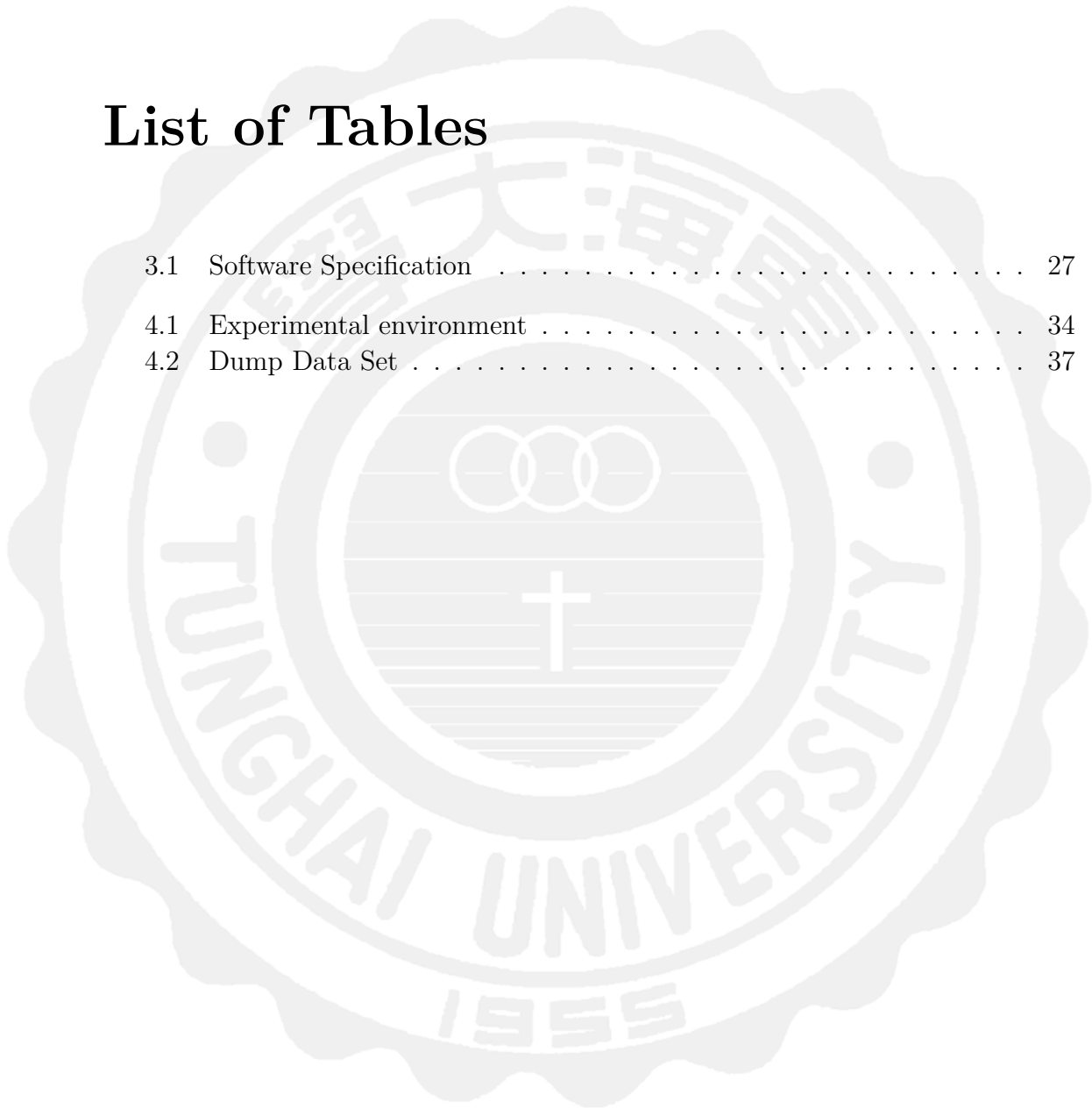


4.17 DB Controller . . . . . 45



# List of Tables

3.1	Software Specification . . . . .	27
4.1	Experimental environment . . . . .	34
4.2	Dump Data Set . . . . .	37



# Chapter 1

## Introduction

Big data application become increasingly important, cloud computing and cloud computing also become more popular. Many existing systems will face growing data. Big data analysis system and cloud computing can solve the problems on big data analysis and storage. However, not all systems have adequate resources to build system on a new big data analysis system. So, many systems based on RDB which cannot support huge data store and data processing. How to integrate RDB and distributed apply on data analysis and system service is an important issue. This section describes the motivation and contribution of the proposed system.

### 1.1 Motivation

In recent years, the rise of Big Data transform database model to NoSQL. But not all institution have resource to change database immediately. Therefore, most of the existing systems are still store data by RDB. With the growing data volumes, distribute for analysis or want to get faster access on big data. Thus, governments and enterprises start to research how to add NoSQL on the exist systems, and try to dump data to NoSQL without affecting the quality of service. RDB system advantage is real time analysis database data, when application query database can using SQL function receive data.

## 1.2 Contributions

Air pollution monitoring system is a good experiment example. In recent years, air quality of Taiwan getting worse. In order to provide the users to view the real-time air quality data. Air pollution monitoring system must continually store the air data from local stations and return information to database. However, growing data let the services of air pollution monitoring system worse. So, we propose a hybrid database converter mechanism to do transforming and processing data without effected system services. Reduce data size of RDB and upgrade service quality, also analyzing huge pollution factor data result on NoSQL by using MapReduce then dump the result to RDB. We propose a hybrid database converter mechanism to integrate RDB and NoSQL on air pollution monitoring system.

## 1.3 Thesis Organization

The rest of the thesis is organized as follows. In Section 2, we review background and related work. In Section 3, we introduce the proposed system design and implementation. Section 4 shows our experiments and results. In Section 5, conclusions and future work are given.

## Chapter 2

# Background and Related Work

### 2.1 Cloud Computing and Big Data

In this chapter, we present background information relevant to our work. First, we discuss the Cloud Computing, OpenStack and Internet of Things. Then we discuss Big Data, NoSQL and Hadoop about Big data analysis and processing applications. Third, we discuss HDFS, HBase Hive and Sqoop about Big data storage and conversion tools on Hadoop . Finally, we also survey related work and point out their relationship to our work.

#### 2.1.1 Cloud Computing

Cloud computing [1–5] is an operation mode based on the Internet. In this way, the resources of hardware and software can be provided to computers and other devices on demand. Users no longer need to know the details of the cloud infrastructure, or have the appropriate expertise and direct control. Cloud computing typically involves with the Internet providing dynamic and easy to expand functionality, and usually consists of virtualized resources. Cloud computing was first proposed by Google, but this concept is not originated by it. Now, cloud computing is well-known and it is gradually evolved by a series of technologies such as grid

computing, utility computing and others. Due to the ever increasing capabilities of computers and decreasing of prices of them in the recent years, more and more companies step into this field.

The main features of cloud computing include on-demand self-service, rapid redeployment of flexibility, shared resource pools to achieve economies of scale, and the service can be calculated. Service providers integrate a large number of resources to multiple users. Users can easily request or rent more resources and adjust the use at any time and release unwanted resources to the whole structure. Therefore, users do not need to purchase a lot of resources for short-term demand, they just need to increase rent amount, and decrease it after completion of tasks. As shown in 2.1, service models of cloud computing are introduced as follows.



FIGURE 2.1: Service Mode

- Infrastructure as a Service (IaaS) [6]: Consumers use the underlying computing resources such as processing units, storage spaces, network elements or intermediary software. Consumers can control the operating system, storage, deployed applications and network elements (such as firewalls, and load balancers), but does not control the cloud infrastructure. For example: Cisco, Hewlett-Packard (HP), EMC, VMware, Intel and IBM.

- Platform as a Service (PaaS) [7]: Consumers use the host to operate applications which just provide hardware-related environments. PaaS is mainly hosted client application execution platforms. Programmers do not have to install application development software, they only need to use the Internet and complete program system development. Users control the operation of the application environment, but do not control the network infrastructure, operating systems, hardware or its operations. The application platform is usually infrastructure, for example: Google App Engine, Salesforce's Force.com and the like.
- Software as a Service (SaaS) [8,9]: Consumer use applications, but not control the operating system, network infrastructure, hardware or its operations. SaaS is the basis of a service concept and is usually combined with commercial software. It is generally regarded as enterprises to obtain the same low cost commercial license. This concept is not only different from the traditional software packaging, deployment, and license, but also it changes the development framework and charging methods, and maintenance mode of software. Its concept is leasing customer service, rather than selling it. A common pattern is to provide a set of account passwords. Examples of SaaS include Microsoft CRM and Salesforce.com.

### 2.1.2 Big Data

Big data [10–12] refers to data in such a huge scale that, within a reasonable time, cannot be manually captured, managed, processed, and organized to become information comprehensible by human. Compared with the individual analysis of small independent data sets with the same total amount of data, after combining the various small data sets as a big data set, additional information and data relevance can be retrieved and used to detect trends, determine product quality

and real-time messages, etc. Such use is the major reason for the prevalence of big data.

IDC defines big data as a new generation of technologies and architectures designed to economically extract value from very large volumes of a wide variety of data by enabling high velocity capture, discovery and analysis. Big data definitions are summarized into four “V”, as show in Figure 2.2.

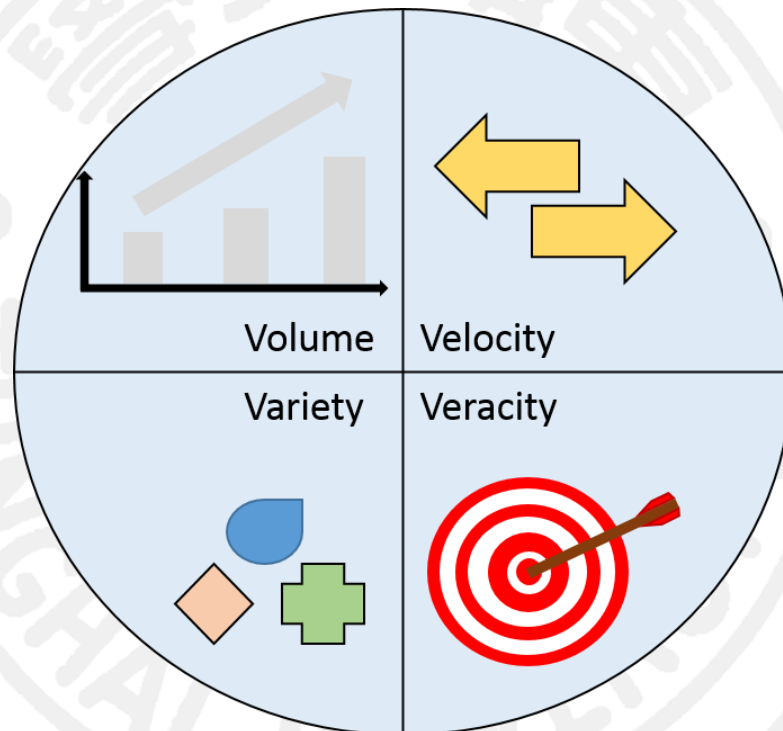


FIGURE 2.2: Big data 4V

That combination of greater capacity (volume), higher diversity (variety), faster formation rate (velocity), and the first three “V” promoted the fourth one - the value (value) as described below:

- Volume: A large amount of data generated, processed, and stored, and it's the literal meaning of big data with massive amount of information.
- Velocity: For example: market forecasting is aging fast and a prediction will be meaningless if not made in time; so timeliness of big data is very critical.
- Variety: It refers to the form of information, including text, audio, video, web, and streaming data that are structural or non-structural.



- Veracity: If the information is from diverse sources, we need to discuss the reliability and quality of the information. If the data have problem, the results of analysis cannot be trusted.

Nowadays the big data processing and analysis application is becoming a new merging point of information technology applications. The mobile Internet networking, social networking, digital home, IoT, and e-commerce comprise the next generation IT applications, and these applications will continue to generate a huge amount of data.

### 2.1.3 NoSQL

NoSQL [13–15] appeared in 1998. It is developed by Carlo Strozzi as a lightweight, open source, relationship database without SQL function. In 2009, Eric Evans from Rackspace's put forward the concept of NoSQL once again. In this time, NoSQL mainly refers to non-relational, distributed, and not provide ACID . [16] repository model. The slogan of NoSQL East conference held in Atlanta is “select fun, profit from real world where relational= false.”Therefore, the most common explanation is “non-associated type”, which emphasizes the advantages of key-value stores and document repository, rather than simply opposes Relational Database Management System (RDBMS). The full name of NoSQL is Not Only SQL. It is different from relational database management system design, as shown in Figure 2.3.

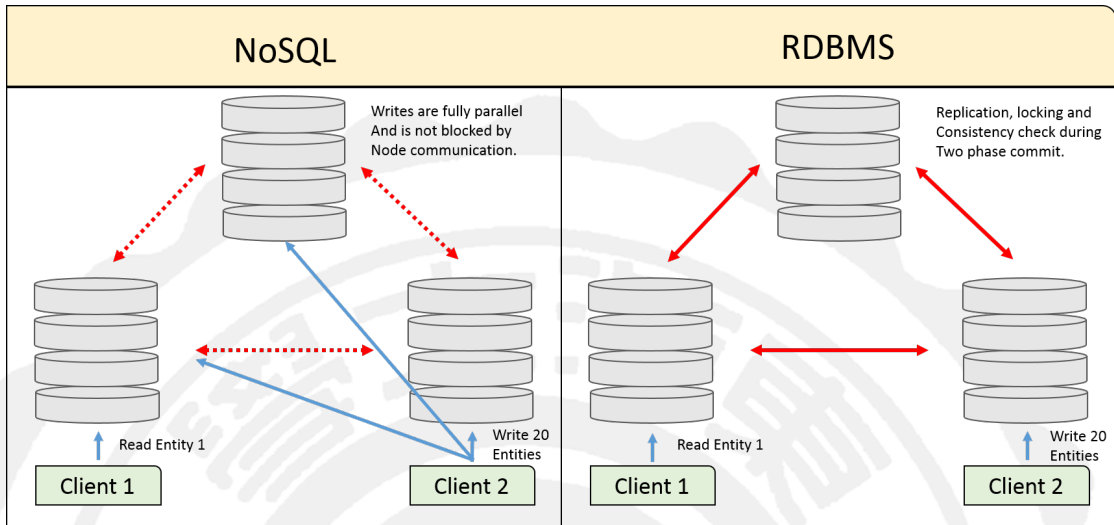


FIGURE 2.3: Compare with NoSQL and RDBMS

Implementation of NoSQL can use either hard disk or RAM for storage. Relational database are not very efficient for mechanisms with frequent reading and writing or mechanisms of writing few but huge data. NoSQL structures usually provide weak consistency guarantees, such as eventual consistency or transactions limited to single data items. It use decentralized structure with data redundantly stored in multiple servers. It often uses a distributed hash table; in this way, the system can be easily scaled to add more servers and to implement fault tolerance for servers. The famous applications of NoSQL include BigTable, which is independently developed by Google, and Dynamo by Amazon; besides, in the open source projects, there are HBase and Apache Cassandra.

## 2.2 Internet of Things and Wireless Sensor Network

### 2.2.1 Internet of Things (IoT)

The IoT [17, 18] is based on the Internet, traditional telecommunication network and other information carriers to enable all ordinary physical objects, which can be independently addressed, achieve interoperability of networks. IoT is machine

to machine (MTM) with the Internet. It covers everything in the world by using RFID and wireless data communication technology. IoT generally uses a wireless network; since the number of devices around everyone can reach 1000 to 5000, so IoT might include more than 500 Trillion objects. BY the IoT, everyone can use electronic tags to find real objects on the Internet and find out their specific locations. Users can use a central computer to manage and control machines, equipment and personnel; they can even remote control house devices and cars, and search locations to prevent goods from stealing. By IoT, systems with GPS can communicate with each other and share information. Figure 2.4 shows three classes of IoT.

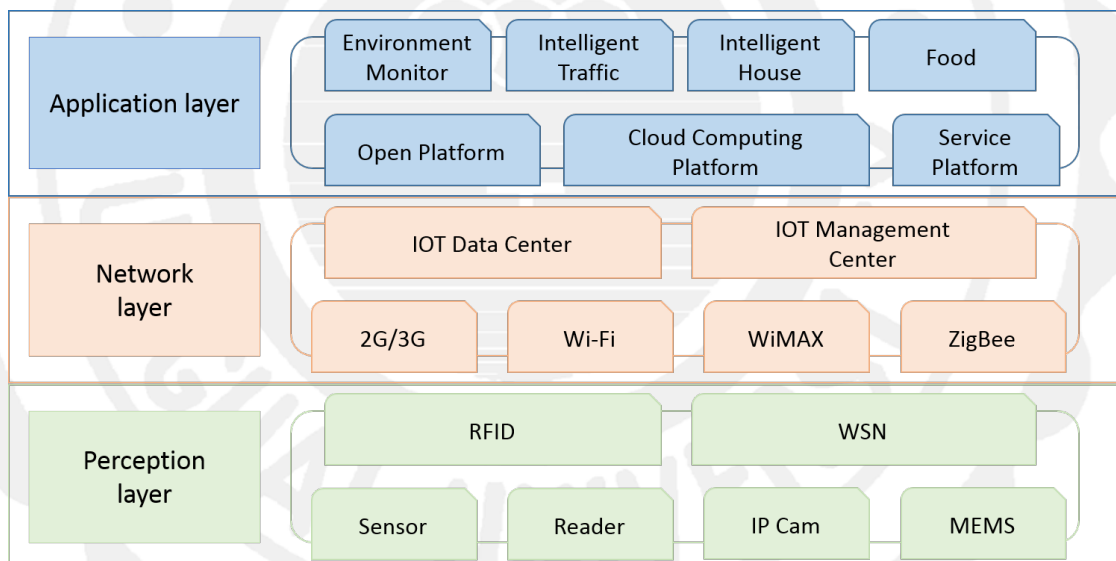


FIGURE 2.4: Three classes of IoT

IoT can be divided into three classes:

- Perception layer: Perception and monitoring carried out for different scenarios; it has sensing, identification, and communication capabilities.
- Network layer: It transmits the data collected from the perception layer to the Internet.
- Application layer: According to different requirement, experts of IoT and industry work together to develop the appropriate integral application software.

Currently, many governments have already announced that the IoT will be raised from general business practices to national strategic industry. In China, people are optimistic about the potential of IoT development, and it has been listed as a key project of the 12th five-year plan. Meanwhile, Japan and South Korea and other countries also actively commit to the development of IoT: US intelligence Earth program, Europe's i2010 policy, Japan's I-japan plan, the new network in South Korea, and Perception mainland in China, all of them are trajectory toward IoT. The business sector also develops new products for this trend. So IoT will have a revolutionary impact on the future of human life. Among them, industries relevant with the safety of people are facing with a major revolution.

## 2.3 Hadoop Ecosystem

### 2.3.1 Hadoop

Hadoop [19, 20] is an open source project under the Apache Software Foundation. The initial prototype of Hadoop-Nutch was developed for web searching by Doug Cutting and Mike Cafarella. In 2006, Doug Cutting joined Yahoo and set up a professional team to continue research and development of this technology, officially named as Hadoop. Hadoop is written in java; it can provide a distributed computing environment for huge data. The concept of Hadoop architecture is based on the BigTable and Google File System papers published by Google. Currently, Yahoo! and other companies have teams for Hadoop development; and more and more companies and organizations publicly express the intention to use Hadoop as cloud computing platform.

Hadoop includes a number of sub-projects. Hadoop MapReduce provides a distributed computing environment; Hadoop Distributed File System provides a lot of storage space; and HBase provides a BigTable-like distributed database. There are other parts that can be used to link together these three main parts,

providing easy integration of cloud services, as shown in Figure 2.5. The following section will introduce Hadoop Distributed File System (HDFS) and HBase.

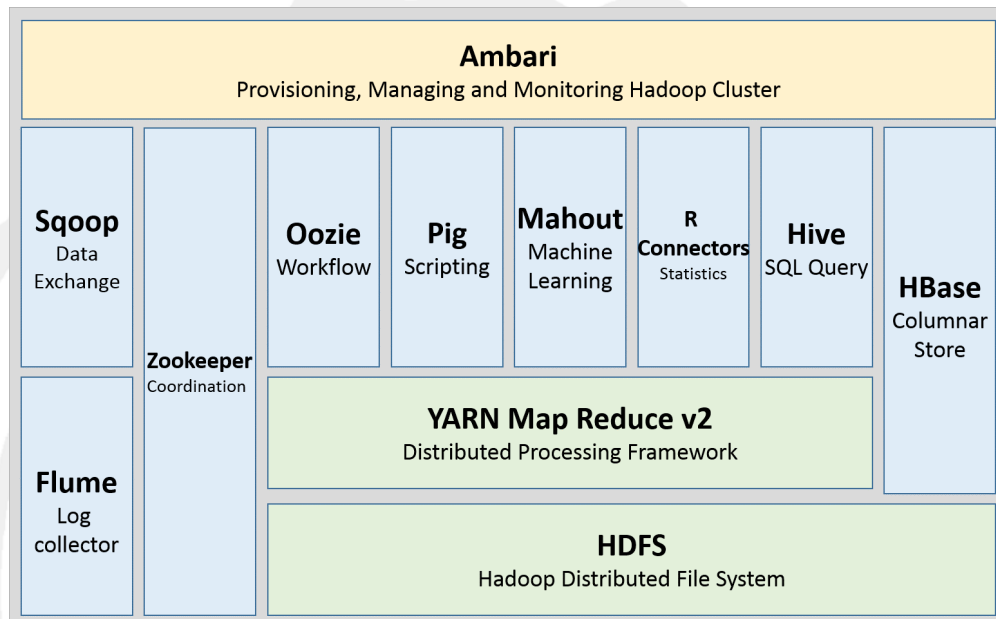


FIGURE 2.5: Apache Hadoop Ecosystem

### 2.3.2 HDFS

Hadoop is a cluster system, which is an integrated super computer expanded from a single server to thousands of machines. In this cluster the information is stored in HDFS, which integrates dispersed storage resources into a fault-tolerant, high efficiency, large capacity, and remote backup storage environment. In Hadoop systems, the large amount of data and temporary files generated during computation are stored on this distributed file system.

- **NameNode:** It is responsible for maintaining the HDFS File System Namespace. It records the mapping relations of a file and its blocks. It also records blocks and data nodes in the blocks for Hadoop cluster configuration management and backup management of the file blocks. Metadata of NameNode is stored in the memory without any paging operation of the virtual memory.
- **DataNode:** It is the server where file blocks are actually stored. It is used for recording metadata of the blocks. The most important information it

provides is the mapping relation for the location of data in the local file system. It provides metadata to clients, and it will periodically transmit the status of existing blocks to NameNode.

Through HDFS, Hadoop can store tera bytes (TB) or peta bytes (PB) of big data. It does not need to worry about the size of a single file exceeding the size of a disk sector, or data lost caused by damaged machines. HDFS has not been integrated into the Linux kernel, and it only can operate files via `dfs` shell command of Hadoop, or use FUSE to be treated as a file system under the user space. All systems under Hadoop are integrated with HDFS as a data storage, backup, and sharing medium. As mentioned earlier, when the system is assigning computing tasks, MapReduce will assign computing task to the nodes stored with the data for operation, thus reducing the time to transmit the large amount of data via networks.

- **HDFS Architecture**

HDFS is master/slave architecture [21, 22], composed of two roles, i.e., the name node and data nodes. Name node is responsible for managing and storing permissions for each file attribute information (such as metadata and namespace) in the file system. The data node usually consists of hundreds of nodes. A file is divided as several smaller blocks and stored in different data nodes; each block has several replicas of data stored in different nodes. When one of the node is damaged, data stored the file system still can be intact. The NameNode needs to record the locations for every file. When there are needs to access files, it coordinates the DataNode for responses. When a node is damaged, NameNode will automatically move and copy data. Figure 2.6 shows HDFS architecture.

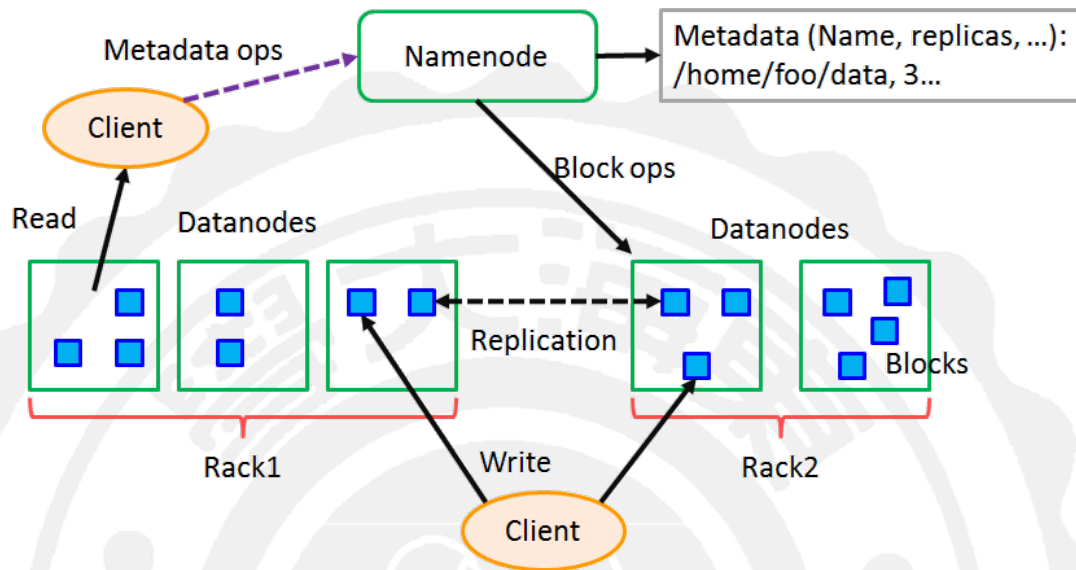


FIGURE 2.6: HDFS Architecture

A file can be thought as a treasure map. Inside the machine a MasterNode is used to manage other slave/worker nodes. In order to securely store the treasure map, it is divided into several pieces (blocks), typically 128 MB a piece; and each piece is copied into three copies (data replication) and these pieces are distributed to slave nodes for storage. The slaves use “DataNode” program to store the treasure map, while the master uses “NameNode” program to monitor the status of the treasure map store in slaves. If the master’s program, NameNode, finds a piece of the treasure map in some DataNode is missing or damaged, it will find the other piece of it on the other DataNode and duplicate it to keep three replicas of every piece of the treasure map in the whole system.

### 2.3.3 HBase

Apache HBase [23–25] is a project undertaken by Powerset to deal with the huge amount of data generated by natural language searching. But now it is already a top-level project of the Apache Foundation. HBase runs on HDFS and has attracted widespread attention. Facebook chose HBase to implement its messaging





HBase uses row and column as index to access data values. It is more like using map container when querying. Another feature of HBase is that each piece of data has a timestamp, so that in a same field there are multiple sets of data of different time. An HBase data table is composed of a number of rows and columns families; each column has a row key as index. A column family is a set of column label, which may have many groups of labels. These labels can be added as needed any time without having to reset the entire data table. When access data in data table, one usually uses a combination of ('row key', 'family: label') or ('row key', 'family: label', 'timestamp', 'value') to retrieve the required fields. Next, we will introduce the Data Model in HBase, which is shown in Figure 2.8.

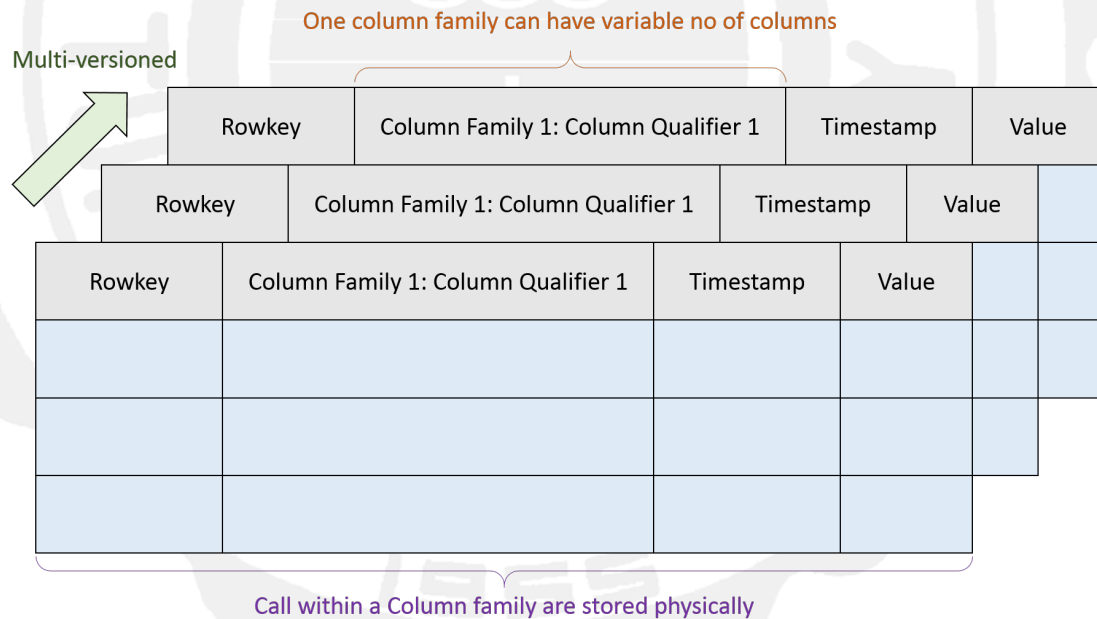


FIGURE 2.8: Data Model of HBase

- Table: It is composed of a number of rows, which are decided at the first time when the table is constructed.
- Row: One row contains a row key and one or more columns; similar to HDFS, rows are ordered in the alphabetical order.
- Column: One column contains a column family and a qualifier, and is separated by “:”.

- **Column Family:** It is a set of columns and its corresponding values. Each column family will have a bunch of property values associated with the store. For example, whether the value needs to be stored in the cache, how the data is compressed, how row keys are encoded. Each row has the same column family, but it is possible that the family is empty.
- **Column Qualifier:** Column qualifier can be understood as the index to the column family. The column family is determined when the table is created, but the qualifier can be added when needed and thus each column qualifier can be very different. Because HBase does not provide meta table that records information of columns, users must manage and remember information of the used columns.
- **Cell:** A complete cell consists of row, column family, column qualifier, value and timestamp.
- **Timestamp:** The timestamp is added when a value is written. It is unique by default, but it can also be set artificially.
- **Namespace:** It can be understood as to manage table by groups. This is a new concept for subsequent version with new features and prepared specifically for use in HBase Shell.

### 2.3.4 Hive

Apache Hive [26–28] is a data warehouse based on Hadoop open source tools for storing and processing massive amounts of structured data. As show in Figure 2.9. MapReduce is the foundation of hive architecture. Hive architecture includes the follow components: CLI (Command Line Interface)、JDBC/ODBC、Thrift Server、WEB GUI、Metastore and Driver (Compiler、Optimizer and Executor), these components are divided into two categories: Service module and client module.

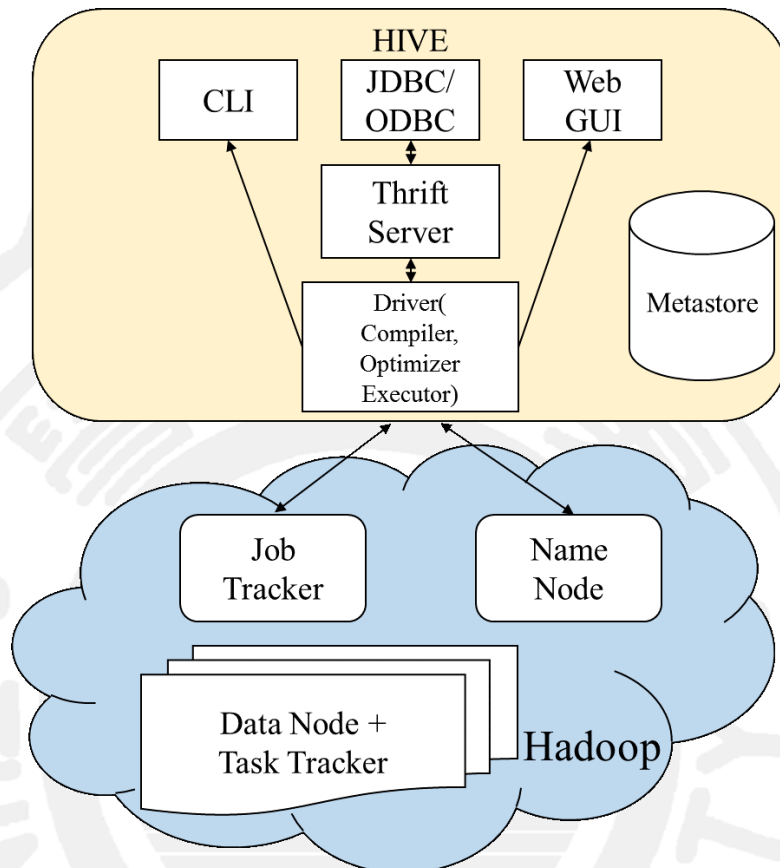


FIGURE 2.9: Hive Architecture

- Service module:
  1. Driver: This module contain compiler, optimizer and executor, this module use HiveQL (like SQL) statements to parsing and optimization compiler, execution task, then invoke MapReduce computing framework.
  2. Metastore: This module store hive metadata, hive store metadata in RDB. Metadata is important to hive, so hive support independent Metastore, install on a remote server cluster, Decouple Hive services and service Metastore, to make sure hive operation robustness.
  3. Thrift service: Thrift is a software framework developed by Facebook, it used to developed scalable and cross-language services.
- Client module:

1. CLI: Command Line Interface.
2. Thrift client: Many client interface of Hive architecture are built on this module, consist of JDBC and ODBC interfaces.
3. WEBGUI: Hive client provides a way to get hive services by web interface, this interface corresponds to the HWI (Hive Web Interface) module.

### 2.3.5 Sqoop

Sqoop [29–31] is SQL to Hadoop, Sqoop is a convenient tool that migration data between traditional relational database and NoSQL. Sqoop take advantage of Hadoop MapReduce parallel feature that accelerate migration data by batch processing.

Sqoop is a import tool that support data migrate from relation database to Hive、HDFS and Hbase, also support full table import and incremental table import. As show in Figure 2.10 is Sqoop basic workflow, when Sqoop import table data from RDB, depend on different split-by value to split the data, next let segmented blocks assigned in different map, each map will process block data. Finally, store data in the Hadoop distributed storage system.

- Sqoop feature:
  1. High efficiency control resources, task parallel processing to save program execution time.
  2. Data type mapping and transforming can be automatically, users can also define their own.
  3. Supports multiple relational databases, MySQL, Oracle, SQL Server, DB2 etc.

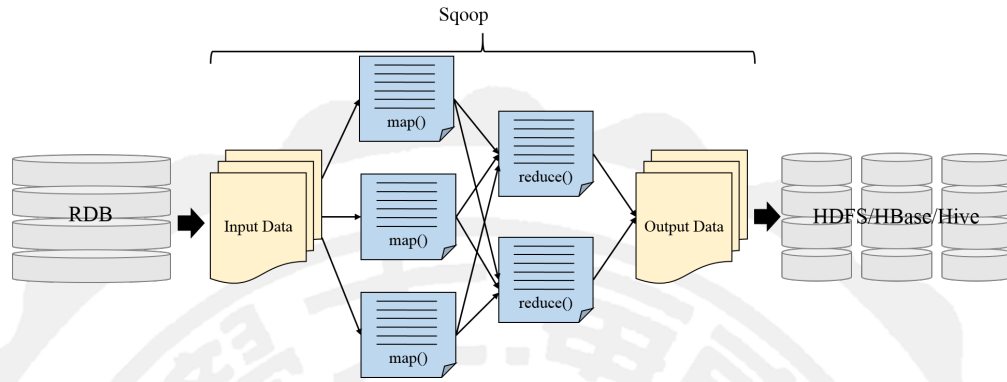


FIGURE 2.10: Sqoop basic workflow

## 2.4 Related Work

Our entire system is divided into several sections to study and find out what we can optimize and reference information. With the data growing getting bigger and increasingly complex, many research are focus on combine RDB and NoSQL. Ying-Ti Liao et al. [32] proposed a data adapter, application can easily make access to the database and dump all the data in RDB to NoSQL database by data adapter. This paper does raise a very good mixed type library data dump modules can be used for our reference. The experiment make we had some doubts, when a large amount of data to a certain extent, all the data stored in two databases will not affect the system service efficiency.

In the data access part, Fan Zhang et al. [33] proposed a task-level adaptive MapReduce framework for real-time streaming data in healthcare applications. Real-time data processing system estimate the cluster performance when processing real-time data into the map on the first time, after the second, when the real-time data entering, application will analysis of the amount of data size then Determine whether to turn to the other map and testing the effectiveness of the newly opened map at the same time. Thus improving real-time processing of massive data. This paper provide large real-time data processing architecture for our reference. After evaluation and experiments, we found when data set is not much

larger. Large real-time data processing architecture will spent to many time on system sets up and cleans up the job, also not conducive to crawling data.

In the data dump part, IKrishna Karthik Gadiraju et al. [34] proposed a Benchmarking performance for migrating a relational application to a parallel implementation. They compare the performance of loading data and querying for SQL and Hive Query Language (HiveQL) on a relational database installation (MySQL) and on a Hive cluster, measure the speedup for query execution for three dataset sizes resulting from the scale up. Hive loads the large datasets faster than MySQL, while it is marginally slower than MySQL when loading the smaller datasets. Query execution in Hive is also faster.

The last part is the air pollution monitoring system, Yun-Ting Wang proposed the Implementation of Sensor Data Accessing on HBase for Intelligent Indoor Environmental Monitoring Cloud Service. the proposed Intelligent Indoor Environment Monitoring System in Cloud (iDEMS) combined environmental sensors with ZigBee wireless sensor network technology to store and process environmental data in HBase. The environmental data collected by sensors will be stored and processed cloudy in HBase which support large amounts of data to store in, free to increase storage space easily. It also can compute through Hadoop MapReduce for HBase database to do distributed computing or cloud computing to process environments records. We refer part of the iDEMS Architecture to make our service consummate.

## Chapter 3

# System Design and Implementation

This section presents the proposed air pollution monitoring system with hybrid database converter and its implementation. In sub-section 3.1, we introduce the proposed system architecture. Next, we introduce the design detail of hybrid database converter in sub-section 3.2. Finally, hybrid database converter implementation and three way to dump data in sub-section 3.3.

### 3.1 The Proposed System Architecture

The air pollution monitoring system is a system which can save and analysis the concentration of air pollution factor by catching the real-time data of monitoring station around to achieve real-time monitoring and provide the historical data for experts and most people by graphical interface as shown in Figure 3.1. In terms of data storage, air pollution monitoring system adopts RDB to store data after catching data of air pollution factor from monitoring station and then requests data from RDB through the application. However, this concept is not efficient for big data today. So we propose a hybrid database converter architecture that can save data quickly and conform to Big Data analysis system architecture without

affecting service. As show in Figure 3.2, air pollution factor transforms data between RDB and NoSQL through hybrid database converter to reduce the amount of data of RDB and then avoid lower performance by the growing data. Moreover, we use NoSQL to analysis big data quickly to achieve the combination of two different database.

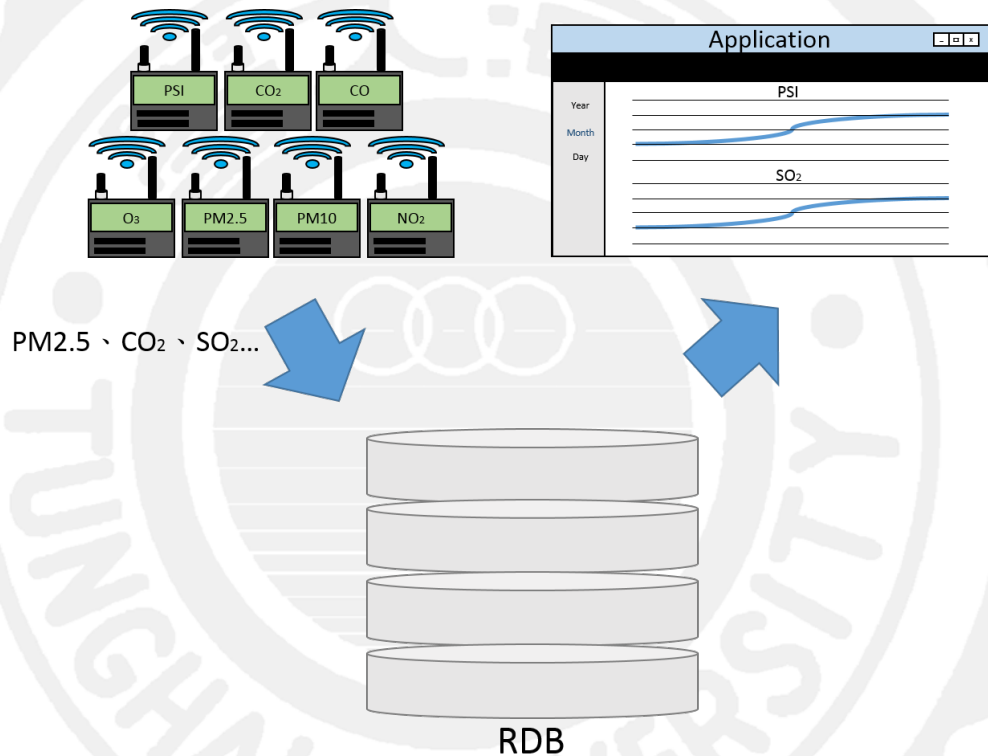


FIGURE 3.1: Air pollution monitoring system data access

In terms of database design, we will divide the data into two categories: the original data that catch from monitoring station and the resulting data after analysis. For convenient analysis by users, the caught data are classified into day data, month data, and year data. In order to prevent the original data used during analyzing, we must arrange the data and show on application by graphical. Two kinds of data is growing continuously, but the magnitude of growth varies greatly.



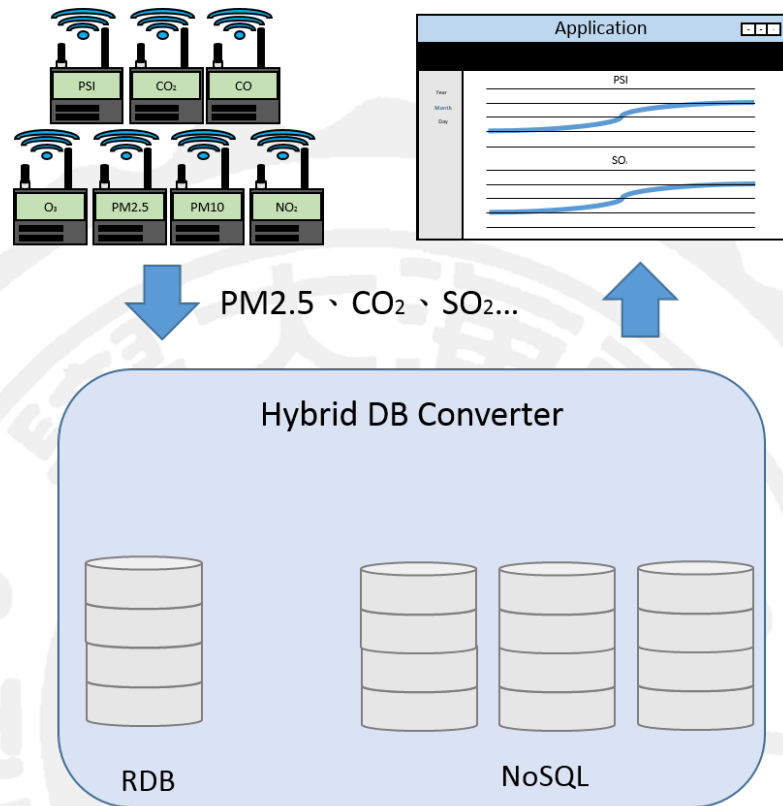


FIGURE 3.2: Join hybrid database converter

## 3.2 Design Detail of Hybrid Database Converter

In this section, we first introduce the design detail of hybrid database converter by two parts: first is database controller, and the other is data dumping. Finally, hybrid database converter implementation, cluster deployment, and data converter are present.

### 3.2.1 Database Controller

Database controller has four models, data collection, data processing, data calculation, and controller. They are shown in Figure 3.3

- **Data Collection** is mainly catch data and store into RDB, through api to catch information of air pollution factor from monitoring station then store into RDB and catch the record to send to controller.

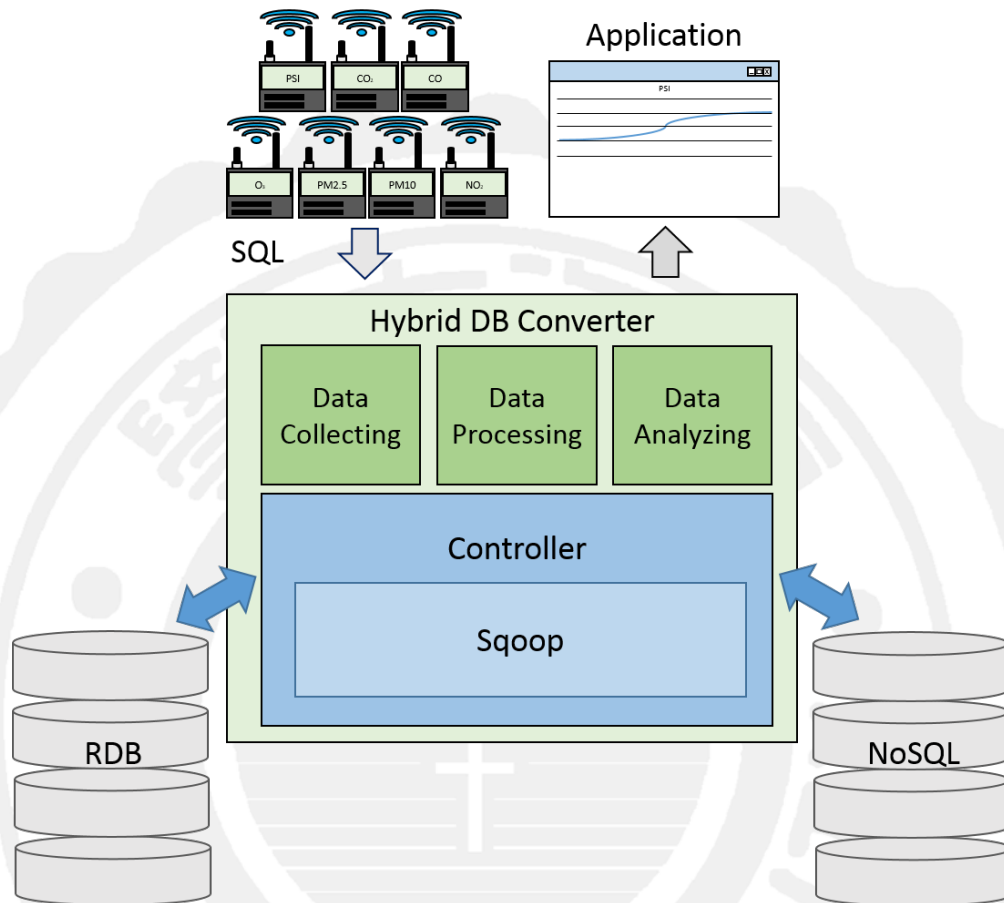


FIGURE 3.3: Hybrid Database Converter

- **Data Processing** is mainly to do first process before convert after data store in RDB, to reduce the amount of data of RDB and data searching time, and also provide graphical data pre-processing.
- **Data Analyzing** is mainly use NoSQL to analysis and process the big data, then return result to the controller after analyzing and processing.
- **Controller** is mainly to execute dump data, also according dump requirement to insert or delete tables data.

### 3.2.2 Data Dumping

The ways of dumping can be divide into two different directions: RDB to NoSQL or NoSQL to RDB, and four types of dump: both of NoSQL and RDB have all data, NoSQL has all data and RDB has part of data, NoSQL and RDB have part

of data that each other do not have and part of data that both have, and NoSQL and RDB have part of data but do not have common data. As show in Figure 3.4.

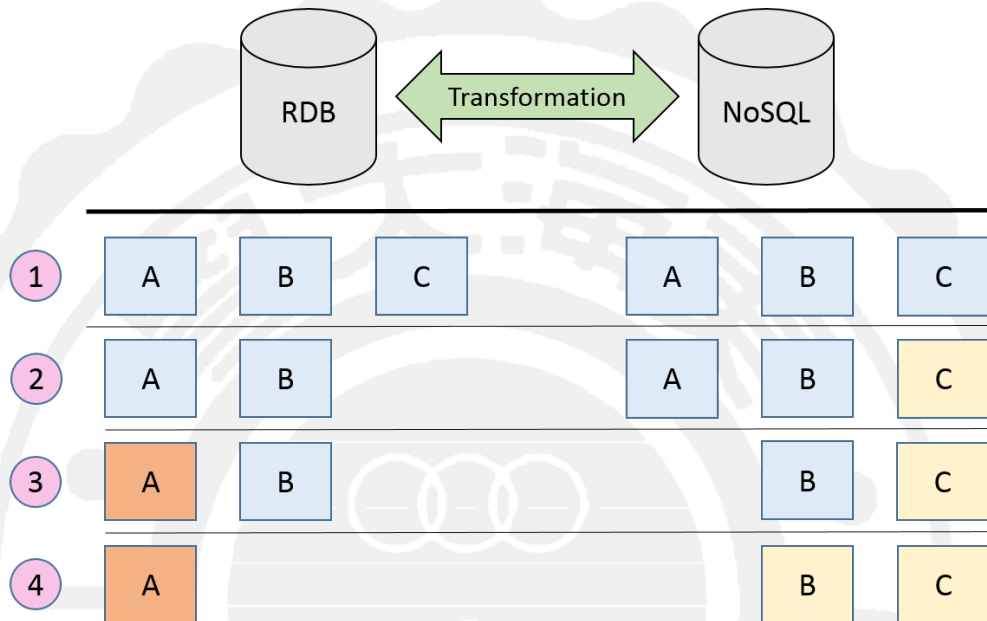


FIGURE 3.4: Transformation types between RDB and NoSQL databases

We choose three different types of data store to compare and get the type that most conform with our air pollution monitoring system. To compare the three different types of data dumping, we divide three classification of data: original data, data after arranging, and others data; original data have the original information that catch from all monitoring station, the data after arranging have day data, month data, and year data from monitoring station after processing and calculating, others data have the record of user and information from each monitoring station.

Synchronize:

This model will store all data into NoSQL and RDB, and the new data will be stored in RDB first, then dump into NoSQL through controller in Non-access time, so in the case of backup, when there has data loss, it can recover data through the backup in resource pool. But when the data of data table reaches a certain amount, the speed of store data into database from air pollution monitoring system will be slowly.

All data in NoSQL and RDB has part of data:

This model will store all data into NoSQL, RDB only stored the data which air pollution monitoring system needs. So we metastasis all the original data into NoSQL, and clear the original data in RDB to reduce the amount of data in database. It's not only can lengthen the usable time of database but also backup data in the resource pool. Reducing load of database through metastasis data in regular time for make sure that the service of air pollution monitoring system be perfectly.

Only part of the same data:

This model will store data into RDB and NoSQL separately, in addition to the information is the same, the data is owned by individually. RDB in this model is mainly store the data and information from monitoring station which is usable for air pollution monitoring system, and the original data and information that is not often used is stored into NoSQL. It's can lengthen the usable time of database and reduce load of database but when it do a backup, it may cause the original system failure and loss data because the data is not completely..As show in Figure 3.5.

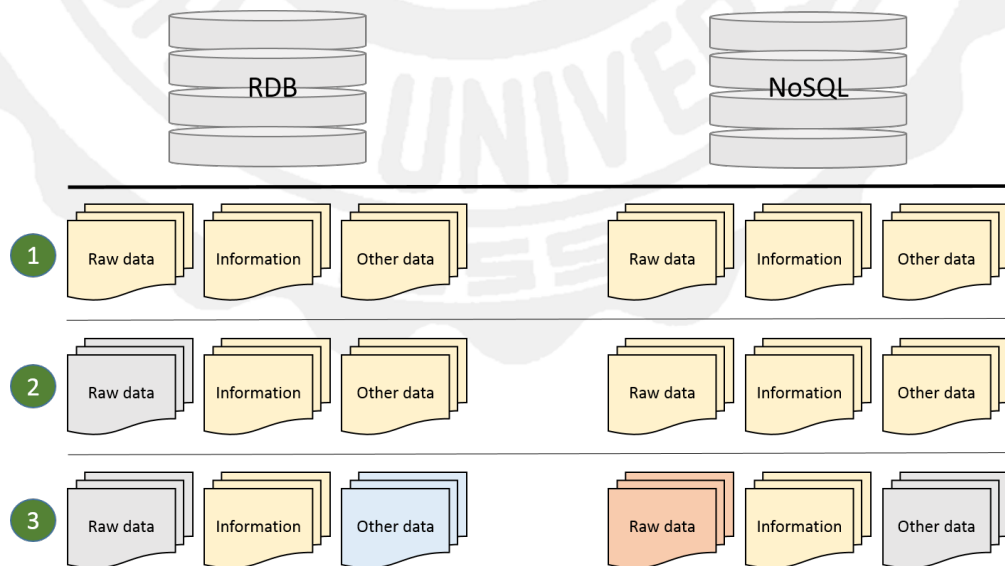


FIGURE 3.5: Three kinds of data conversion model

### 3.3 Hybrid Database Converter Implementation

In this work, we have established the cloud big data clusters through thirteen physical machines, one node as master, twelve node as the computing node to set up Apache Hadoop, Apache Spark, Apache Sqoop, Apache Hive, Apache HBase. Table 3.1 shows the software specification.

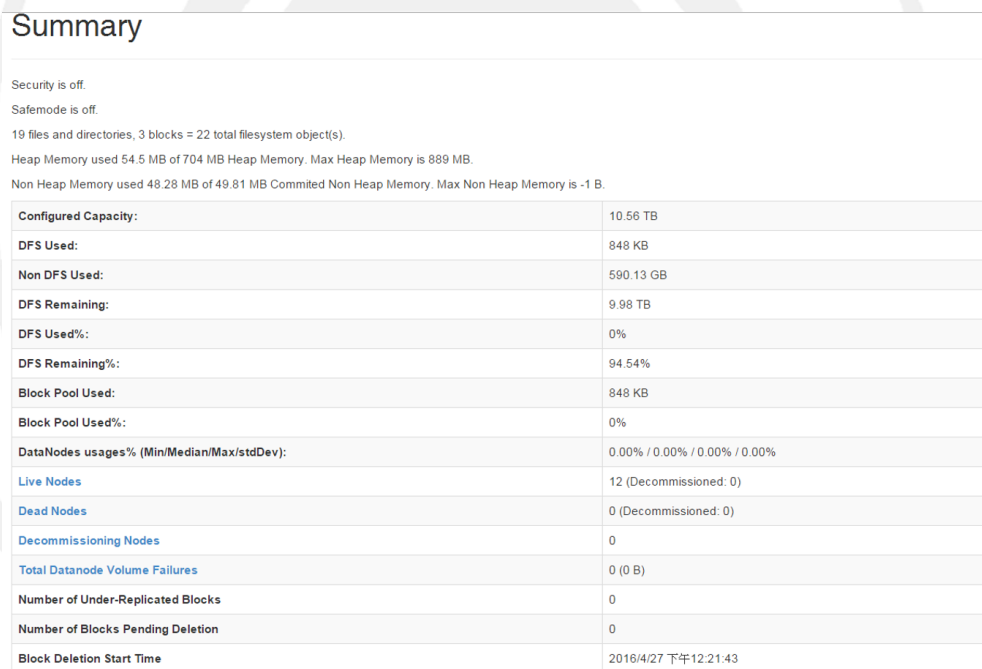
TABLE 3.1: Software Specification

	Version
Hadoop	2.6.0-cdh5.4.5
HDFS	2.6.0
YARN	2.6.0
Spark	1.3.0
Sqoop	1.4.5
Hive	1.2.1
HBase	1.0.0

### 3.3.1 Cluster Deployment

On the deployment, platform environment using one server as master, and computing nodes using 1 Gigabit Ethernet, each node as DataNode, NodeManage, and RegionServer, where three computing nodes as ZooKeeper.

Through the Thirteen hosts, i.e., the one NameNode and Twelve DataNodes, the Hadoop HDFS NameNode Web Interface shows that the cluster provides 10.56 TB of big data storage space. This information also shows how many live DataNodes are functioning shown in Figure 3.6.



The screenshot shows the 'Summary' page of the Hadoop NameNode web interface. It displays various cluster statistics, including capacity, usage, and node health. A table at the bottom provides a detailed breakdown of these metrics.

Summary	
Security is off.	
Safemode is off.	
19 files and directories, 3 blocks = 22 total filesystem object(s).	
Heap Memory used 54.5 MB of 704 MB Heap Memory. Max Heap Memory is 889 MB.	
Non Heap Memory used 48.28 MB of 49.81 MB Committed Non Heap Memory. Max Non Heap Memory is -1 B.	
Configured Capacity:	10.56 TB
DFS Used:	848 KB
Non DFS Used:	590.13 GB
DFS Remaining:	9.98 TB
DFS Used%:	0%
DFS Remaining%:	94.54%
Block Pool Used:	848 KB
Block Pool Used%:	0%
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	12 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	2016/4/27 下午12:21:43

FIGURE 3.6: Hadoop NameNode information

Figure 3.7 shows thirteen nodes are currently in operation and the Applications running status or history.

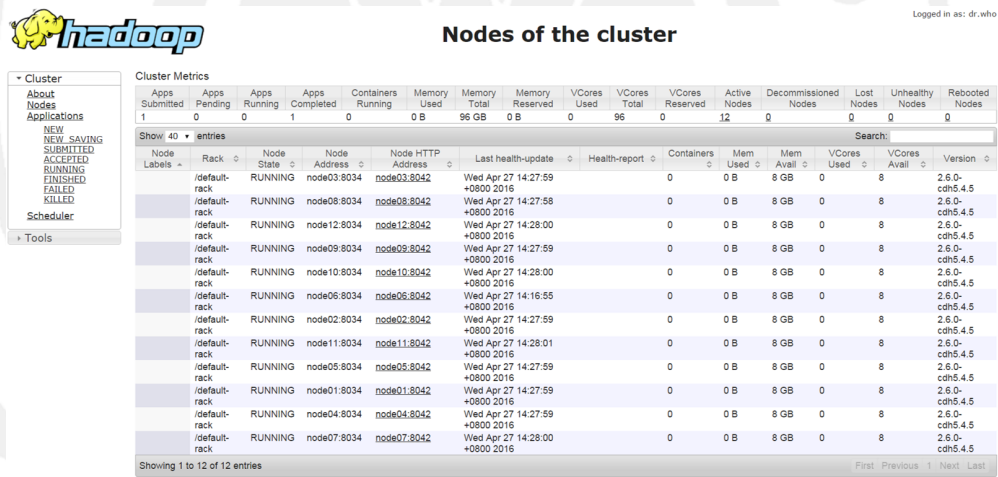


FIGURE 3.7: Hadoop cluster information

### 3.3.2 Data Converter

In data converter, we split dump data into three parts, first dump, batch of dump, and time of disaster recovery. First dump is mainly to save data table in Hive sequential through in accordance with different modules, batch of dump will dump new data in a period of time, disaster recovery will revert RDB which has broken data, and we will simulate by different size of data.

#### First Dump:

In this step, the system will dump all the data table to NoSQL directly, when system is connected to the hybrid database converter system, hybrid database will fetch all the data from RDB, and put the data to NoSQL by the Sqoop. If it is not the first time connected to the hybrid database, the system will dump the data after it store the information to a certain extent.

#### Batch of Dump:

In this step, system will examine RDB tables in certain period of time, if table has update data, system will dump the update data to the NoSQL. Show as Algorithm 1. In order to achieve the batch dump, we when dump to time  $t$ , we will check whether a table data  $T_n$  have updated information or not, if there is updated information we will dumping  $T_i$  data to NoSQL. Because some data table needs to be read and can't be clear the data table, so  $T_i$  must to determine it's necessary to clear the data or not. If it needed, it'll clean up the  $T_i$  data and make the next batch dump time.

#### DB Disaster Recovery:

In this step, for our system can do DB disaster recovery in database, we use two dump methods to store all data in NoSQL, and use algorithm to recover data in RDB in the most short time and rebuild service. Show as Algorithm 1. In order to reply the database, we check database whether destroyed or not, when the database destroyed, check all tables  $T_n$  in NoSQL, as different module, judge the



table  $T_i$  whether need to back to RDB or not. If  $T_i$  need reply, copy the  $T_i$  data to HDFS list  $L$ , then put  $L$  to RDB.

---

**Algorithm 1** Batch of Dump Algorithm
 

---

```

1: while (time equal to  $t$ ) do
2:   for ( $i=1; i \leq n; i++$ ) do
3:     if ( $T_i$  has the update data) then
4:       Dump  $T_i$  data to  $T'_i$ 
5:     end if
6:     if ( $T_i$  need to be clear) then
7:       Clean the table
8:     end if
9:   end for
10: end while

```

---



---

**Algorithm 2** DB Disaster Recovery Algorithm
 

---

```

1: while (Determine whether the need to reply) do
2:   for ( $i=1; i \leq n; i++$ ) do
3:     if ( $T_i$  need to be reply to RDB) then
4:       if ( $T_i$  need to be processing) then
5:         Processing  $T_i$  data to  $L$ 
6:       else
7:         Copy  $T_i$  data to  $L$ 
8:       end if
9:     end if
10:   end for
11:   Put the data  $L$  to the RDB
12: end while

```

---

### 3.3.3 Data Converter

We take an air pollution monitoring platform as an example, we implement a service platform, mainly is used to monitor changes in air pollution factor, it will fetch 76 monitoring stations and data will be stored in our hybrid database system. Each station can fetch an amount of about 1MB of data per day, so every stations can fetch to the amount of data of about 76MB a day. Through data collection and collation of air pollution in nearly seven years, the current amount of data about 2GB. As shown in Figure 3.8, the system will fetch the

data back to the database then sent to the server through the wireless network and the existing data dump into the cloud platform quickly. The cloud system is based on 12 physical machine, using the features of Hadoop to link the physical machine effectively, and then made a huge historical data through the MySQL connect with the Hive, the results are presented in the monitoring system. When each data dump to the cloud-based platform, the information will be placed in distributed database systems, every fixed time, data will also be stored in the cloud. After stored in cloud storage, data will be processed and operation. After the data processing, users can handle the necessary information to do data dumping through the management platform. Finally, the system will be presents in web pages by using HTML 5, JavaScript and CSS 3, in order to achieve the dynamic web pages, our paper through the implementation a UI by jQuery, users can see the information about air pollution and instant information via the user interface. In this service, we focus on air pollution monitoring, the detection of air quality by sensors and the data convert to the visual information displayed on the user interface, and saved to the database at the same time. In the system interface, we configured the data converter, providing periodic information dumping, the managers can dumping data whatever they want, we also had automatic control system, if the data in the RDBMS reaches a certain level, it will dump into the cloud system, it mainly to maintain application performance when fetching data in database, and warning to air pollution monitoring.

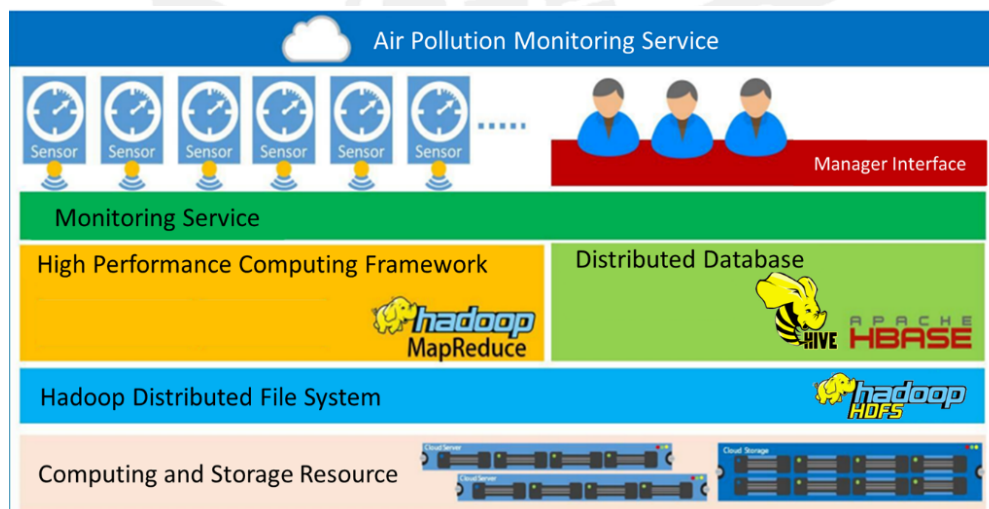


FIGURE 3.8: Air pollution monitoring platform architecture

## Chapter 4

# Experimental Results

This chapter introduces experiments in detail. First, both hardware and software used in experimental environment are listed. Then, the experimental results are shown and analyzed. Finally, discussions are made.

### 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 14.04.2 is adopted as our operating systems. Also, Hadoop 2.6.0-cdh5.4.5、Spark 1.3.0、Sqoop 1.4.5、Hive 1.2.1、HBase 1.0.0 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	1Gb Ethernet



FIGURE 4.1: Spark and Hadoop Computing cluster

### 4.1.1 Compare massive storage methods

In terms of data storage, we compare three different types of storage: HDFS, HBase, and Hive. Compare speed and performance of data dumping from RDB, to find the best way of storage for NoSQL to improve the overall performance. We split several different sizes data from the air pollution database to assess the transmission speed impact of different storage methods.

As show in Figure 4.2. when the amount of data is growing bigger and bigger, the transform time of HBase is much larger than the other two dump mode, Hive and HDFS efficacy is almost the same. Therefore, in the data dump part Hbase performance is lower than the other two.

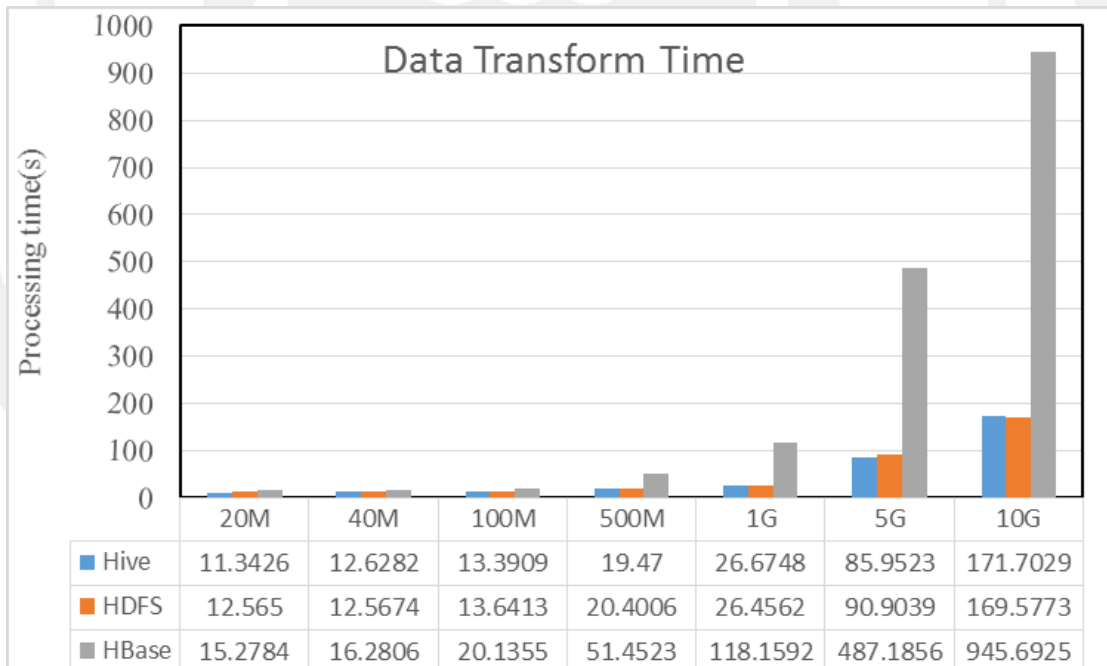


FIGURE 4.2: Data Transform Time

## 4.2 Data Dumping Experiment

First we analyze the change of data of three categories. With the increase of time, the amount of data also become more and more big. Therefore, we must observed the change of original data, arranging data, and other data.

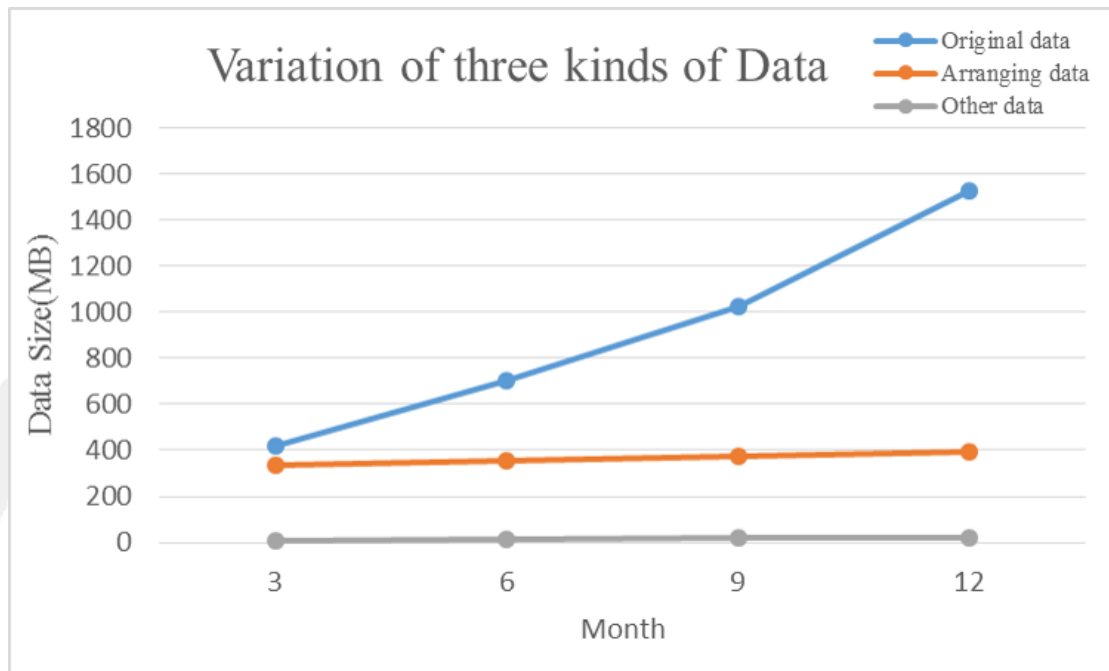


FIGURE 4.3: Variation of Three Kinds of Data

Show as Figure 4.3 and Figure 4.4, we can find the original data collect by air pollution station growing very fast. Arranged data and other data cumulative compare to original data is almost not change.

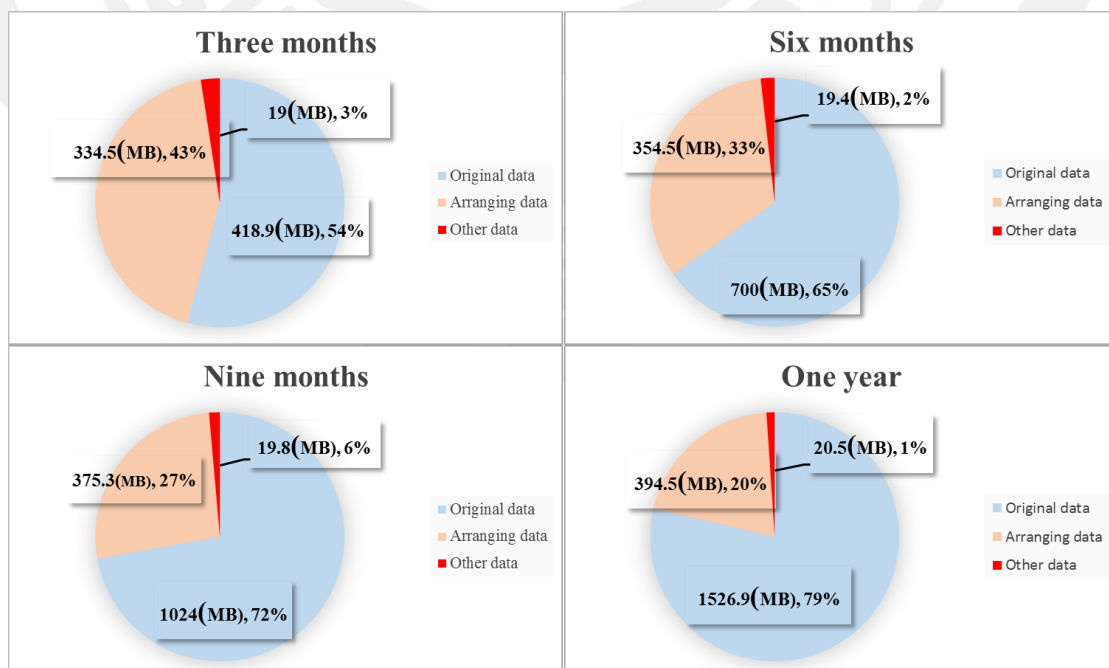


FIGURE 4.4: Data Growth of Three Kind of Data

### 4.2.1 First dump

We arrange three different size of data, one year, six years, and twelve years of data to put in our module to do data dumping experiment, as shown in Table 4.2.

TABLE 4.2: Dump Data Set

	Original data	Arranging data	Other data
1 year	1526.9(MB)	394.5(MB)	20.5(MB)
6 years	9161.4(MB)	719.5(MB)	25.5(MB)
12 years	18322.8(MB)	1174.5(MB)	31.5(MB)

We put three different size of data in three different modules, show as figure, synchronize represent synchronize data, All in NoSQL & part in RDB represent all data that NoSQL has, Part of same represent part of same data in database. Show as Figure 4.5, we find that best performance of three modules in first dump is Part of same module, almost fifty seconds faster average in each data sets, there has one reason that the amount of dumping is smaller than other two modules.

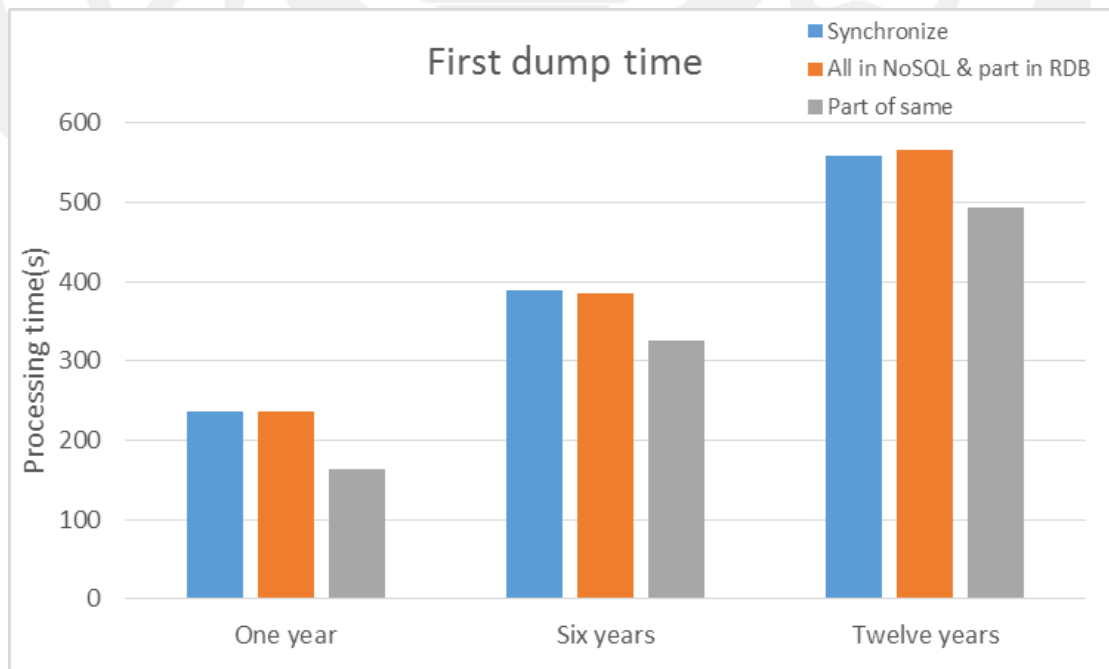


FIGURE 4.5: First Dump Time

### 4.2.2 Batch of Dump

In this experiment will process the data that increase every day in RDB and dump in Hive. We will use three different phases' data to do the experiment, to experiment that the degree of data accumulation will affect the speed of dump on module or not.

The speed of dumping in three modules, different years, and different size of data is show as Figure 4.6. The process time of synchronize module and part of same module is increase by time, but all in NoSQL & part in RDB module is not decrease its' dump time by time. It knows that the best module is all in NoSQL & part in RDB module, although all in NoSQL & part in RDB module is slow than third module in first dump, but the speed of batch of dump is faster than other two modules.

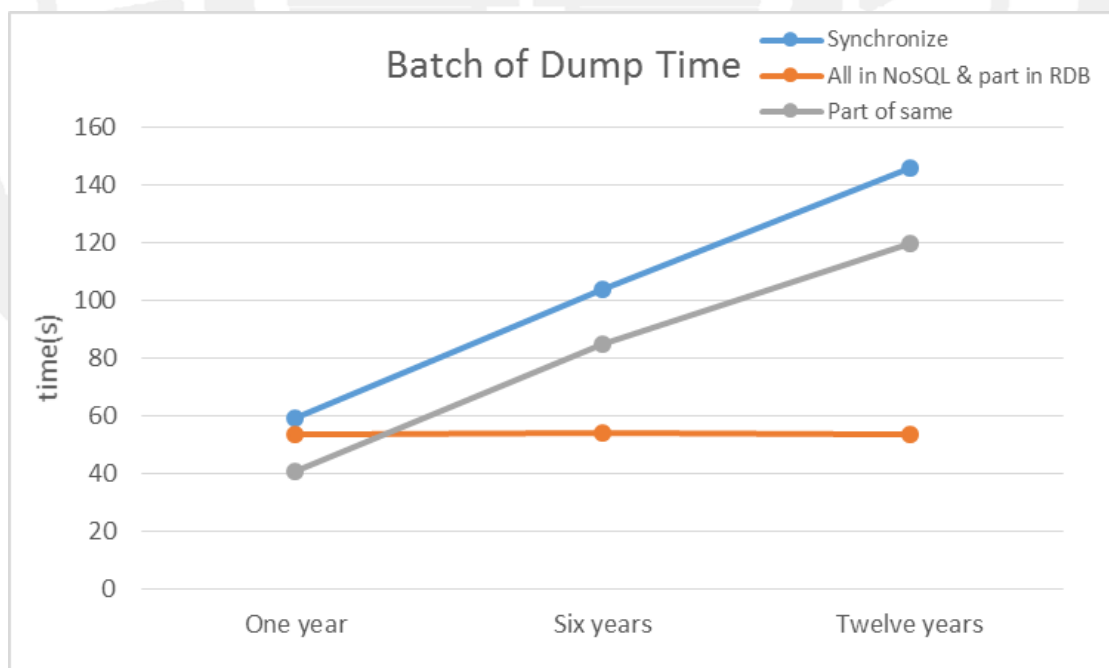


FIGURE 4.6: Batch of Dump Time

### 4.2.3 Database Disaster Recovery

This experiment will assume RDB without any data to do experiment of data recovering, to compare the time of three different data dump with three different



type of data. The experiment of data recovering in different module and different size of data is shown as Figure 4.7. The recovery time of synchronize and all in NoSQL & part in RDB module is nearly, but the part of same module need a lot of time to recover database. During the part of same module recovering, it' s not only rewind data in RDB, but also process the original data to the data after arrange to save in RDB to provide service save and take.

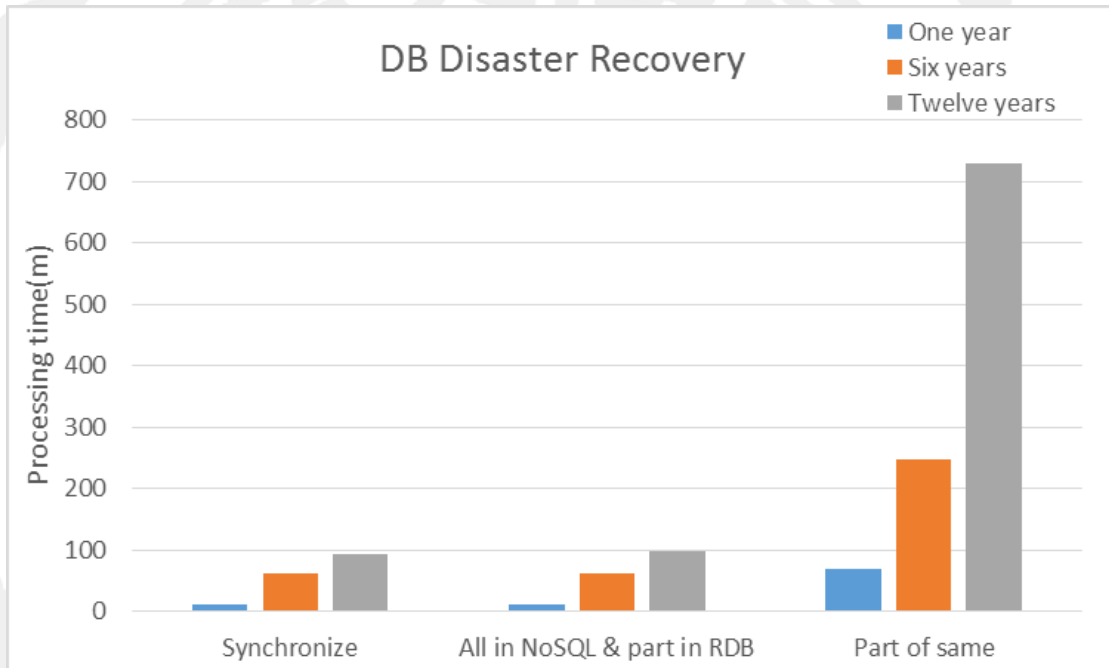


FIGURE 4.7: DB Disaster Recovery

#### 4.2.4 The Three Models Compare

Through the three experiment knows that the best module is all in NoSQL & part in RDB module. The performance of the three modules in three different experiment, the performance of second module is the best. Show as the Figure 4.8, we use 10 to 1 to rate the performance of three models experimental data which is high or low. When the time was increased to six years, three experiments are biased in favor of the all in NoSQL & part in RDB model. Therefore, we use the all in NoSQL & part in RDB model for the entire hybrid library service.

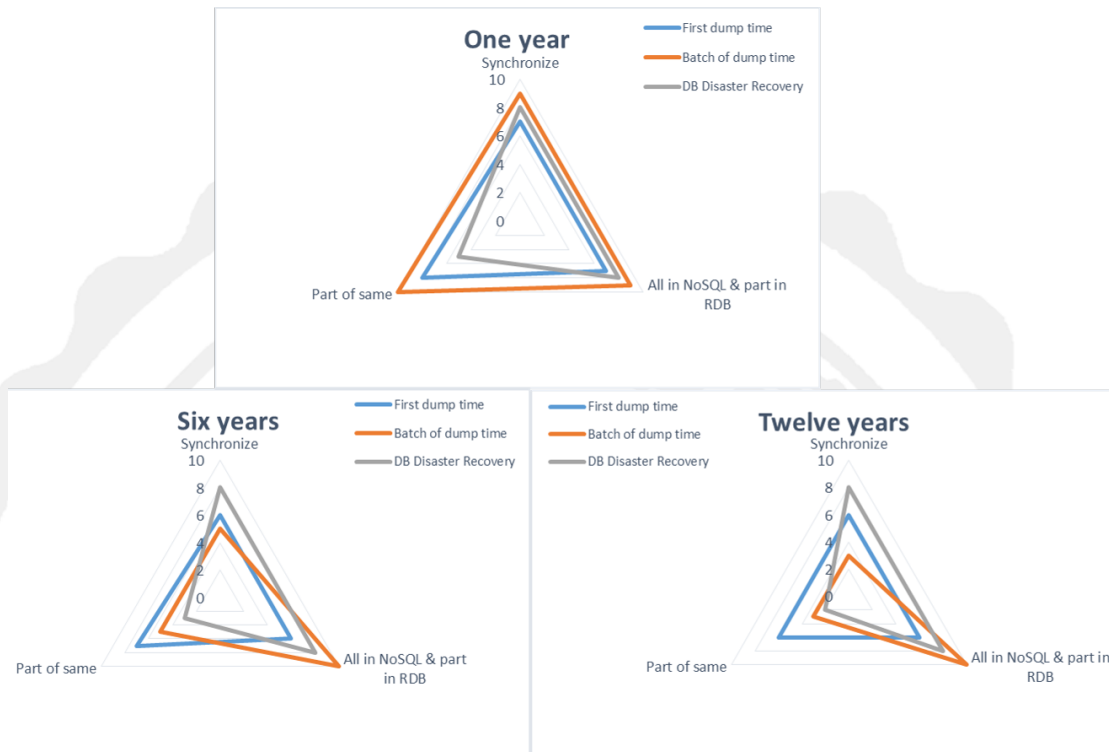


FIGURE 4.8: Experiment Results

### 4.3 Platform Implementation

We implement an air pollution monitoring platform, collect data from 76 station and store in our hybrid database system. Each station can collect 1MB data a day, we have 76 station, so we can collect 76MB data every day. Through collect and arrange 7 years air pollution data, we have about 2GB data. On the web site, we show our data by two part, one is real-time data and another is historical data. In the case of real-time data, it will show the PSI and air quality of each station. In the case of historical data, it' s will be more detail, every day, every month, every year and each year air pollution data and it can compare change with air pollution with other station. Show as Figure 4.9, we catch air pollution real-time data from each station and show to user, to provide air quality and PSI value now for user. In the system, we build a backstage service for user that can provide user to watch the status of data in database, and it can dump by batch and recover data.

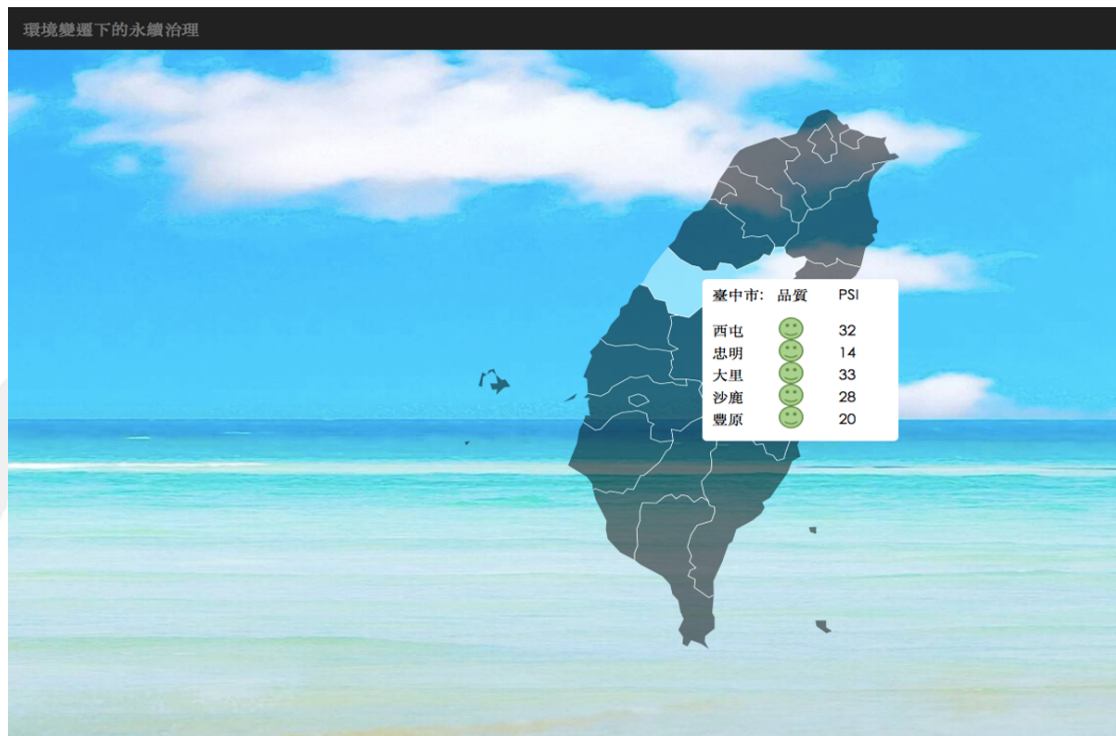


FIGURE 4.9: Air Quality

Show as Figure 4.10, we show all data with a graphic from station, to provide the change of air pollution factor for user.

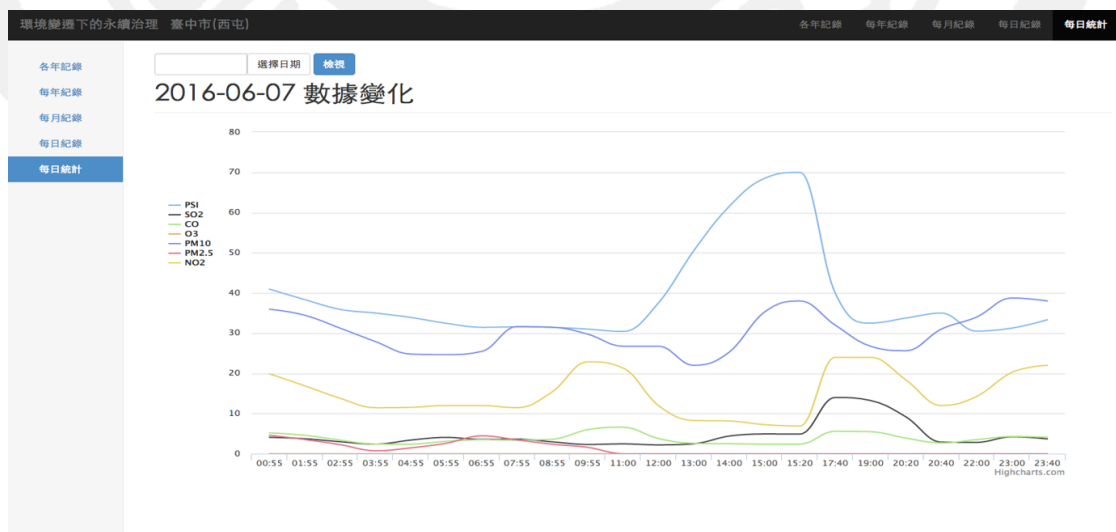


FIGURE 4.10: Daily Stats

Show as Figure 4.11, Figure 4.12, and Figure 4.13, we show air pollution factor from each station, and through observed value of every day, every month, every year to achieve monitoring.



FIGURE 4.11: Daily Values Record

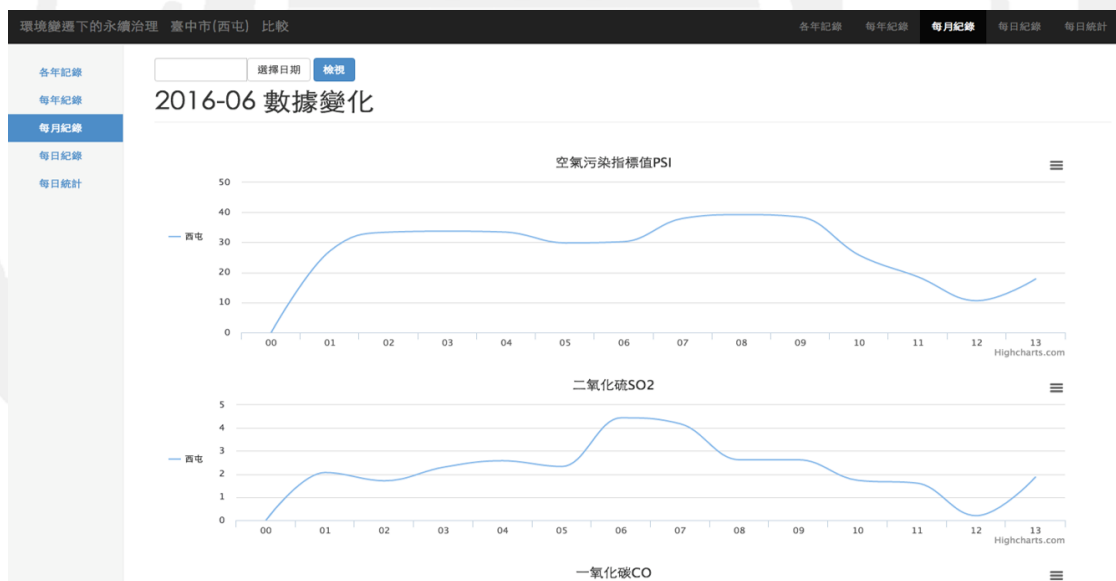


FIGURE 4.12: Monthly Values Record

Show as Figure 4.14, Figure 4.15 and Figure 4.16. against data of every year to monitoring further, to compare air pollution factor with different station, and for more comprehensive to observed the change of air pollution factor, we add the change of air pollution factor of every year, to provide more comprehensive monitoring data.

Show as Figure 4.17, we make functions on the web site that can provide user batch dump and disaster recovery in database. For make disaster recovery

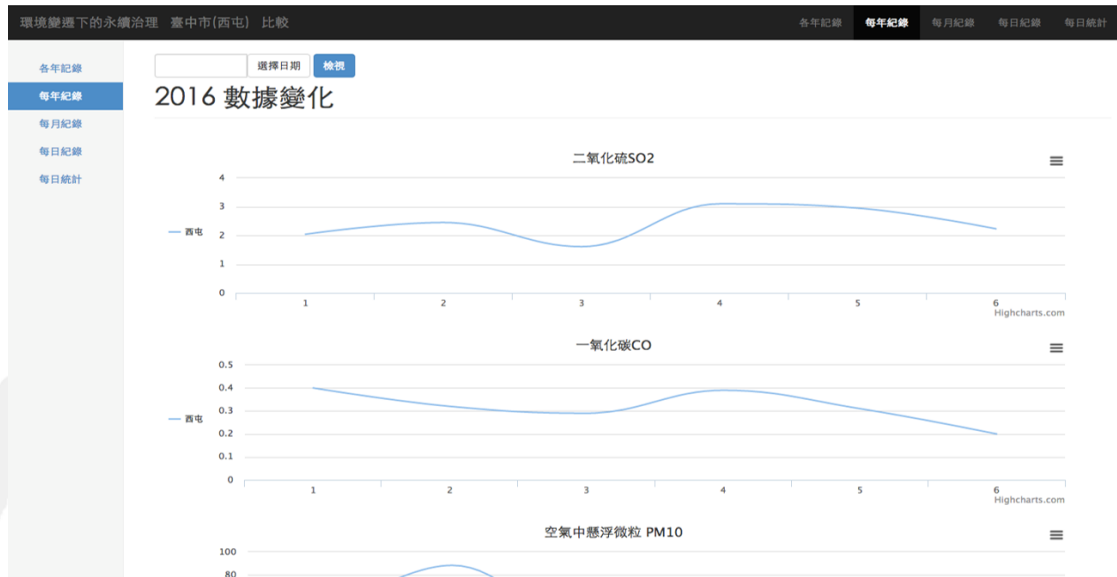


FIGURE 4.13: Annual Values Record

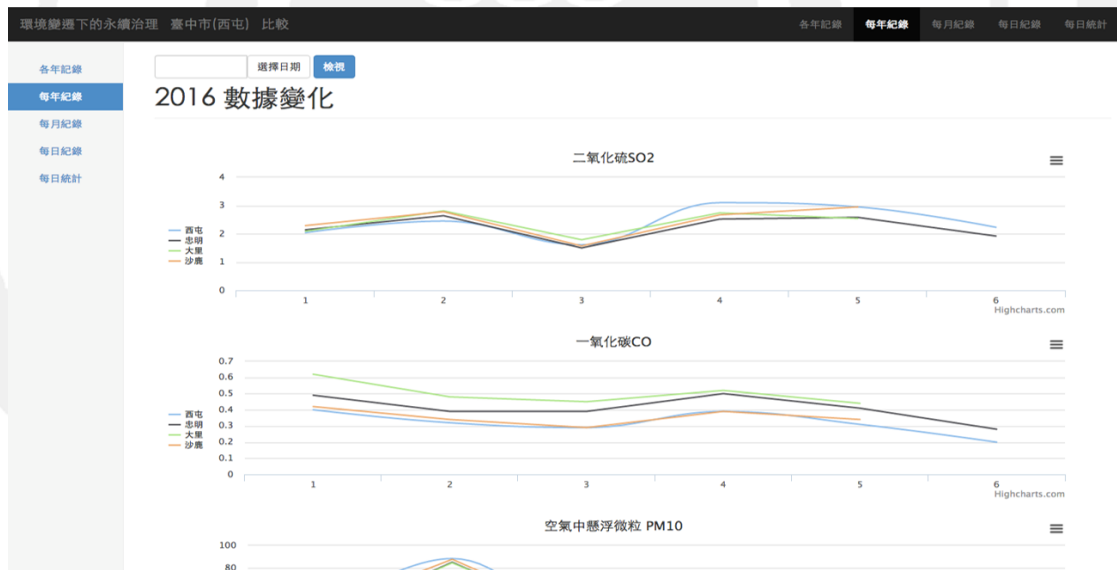


FIGURE 4.14: Regional Numerical Comparison

experiment can more smoothly, we also provide to clear database that user can do the function of disaster recovery.



FIGURE 4.15: Value Record of Each Years

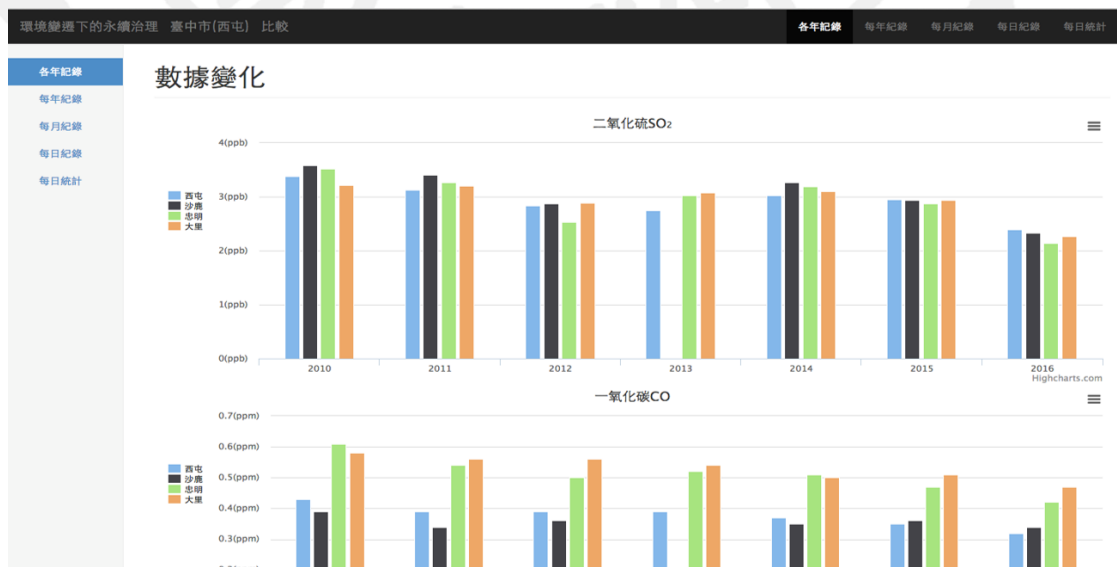
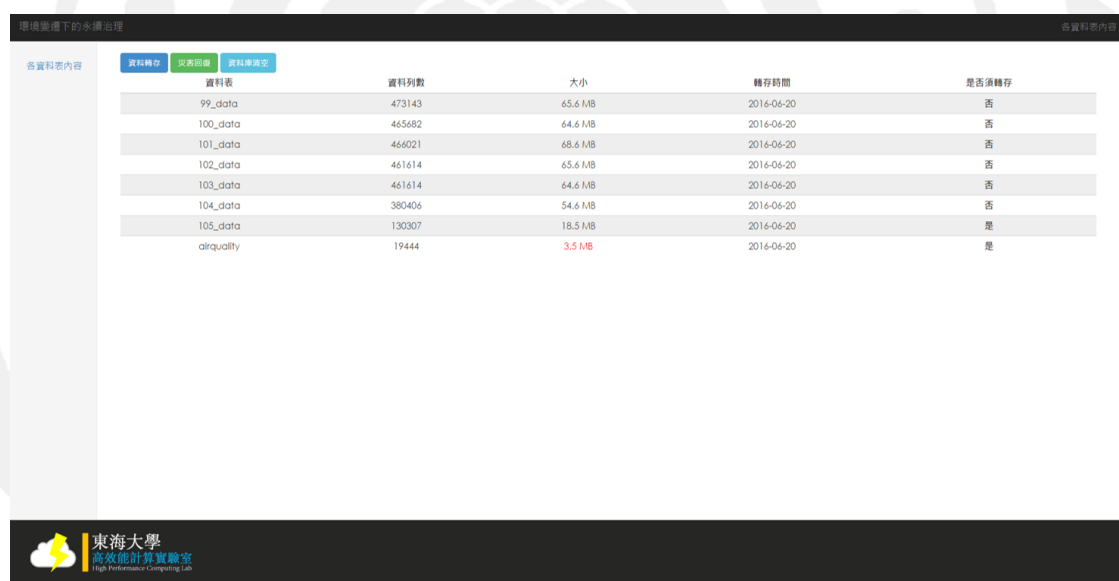


FIGURE 4.16: Numerical Comparison of Each Years



The screenshot displays a web interface for a DB Controller. At the top, there are navigation tabs: '資料轉存' (Data Migration), '資源目錄' (Resource Directory), and '資料庫清空' (Clear Database). Below these is a table with the following columns: '資料表' (Table Name), '資料列數' (Number of Rows), '大小' (Size), '轉存時間' (Migration Time), and '是否須轉存' (Whether to Migrate). The table lists several data tables, with 'airquality' highlighted in red for its size of 3.5 MB.

資料表	資料列數	大小	轉存時間	是否須轉存
99_data	473143	65.6 MB	2016-06-20	否
100_data	465682	64.6 MB	2016-06-20	否
101_data	466021	68.6 MB	2016-06-20	否
102_data	461614	65.6 MB	2016-06-20	否
103_data	461614	64.6 MB	2016-06-20	否
104_data	380406	54.6 MB	2016-06-20	否
105_data	130307	18.5 MB	2016-06-20	是
airquality	19444	3.5 MB	2016-06-20	是

東海大學  
高效能計算實驗室  
High Performance Computing Lab

FIGURE 4.17: DB Controller

## Chapter 5

# Conclusions and Future Work

In order to solve that the growing information cause the computing capacity of system decrease, and affecting the overall performance of system services; so we propose a hybrid database converter to achieve the integration of original system and big data analysis platform. So that the original system has big data analysis capability, and do data backup at the same time. Compare three different types of converting data and get the most suitable converter for air pollution system, to improve the data analysis capability of the original system. In the part of converter, in the case of not effect service, we decrease 60% of time in batch processing. User not only can observed real-time information of air pollution, but also can observed every day, every month, every year, and all change of air pollution factor through historical data. To provide user complete data search service from slight to comprehensive air pollution data.



## 5.1 Concluding Remarks

In this study, we consider most of the existing systems do not have enough resources to direct re-build the system on big data processing architecture. So, we proposed hybrid database converter to solve the problem of resources insufficient. It's not only provide Big Data analyze, but also decrease the load of original database, to prove the life of database. At the time, it's can keep database complete through database disaster recovery. Solve the difficulties of big data processing and storage for our system, provide a more flexible Big Data system.

## 5.2 Future Work

In this paper, we propose an efficient hybrid database converter. In the future, we hope to further let hybrid database converter erected on Docker Container, by Docker rapid deployment and the feature of virtual machines can be monitored, Strengthen the capacity of data analysis, store and data backup on all cluster. In the case of data collect, only the data of air pollution cannot analyze accurate, therefore we will add the data of water, electricity, speed of wind, and even oil liquefaction, to provide user can analyze more kinds of data. Finally we hope we can add Big Data analysis and data mining, not only for data monitoring, but also achieve decision making for service.

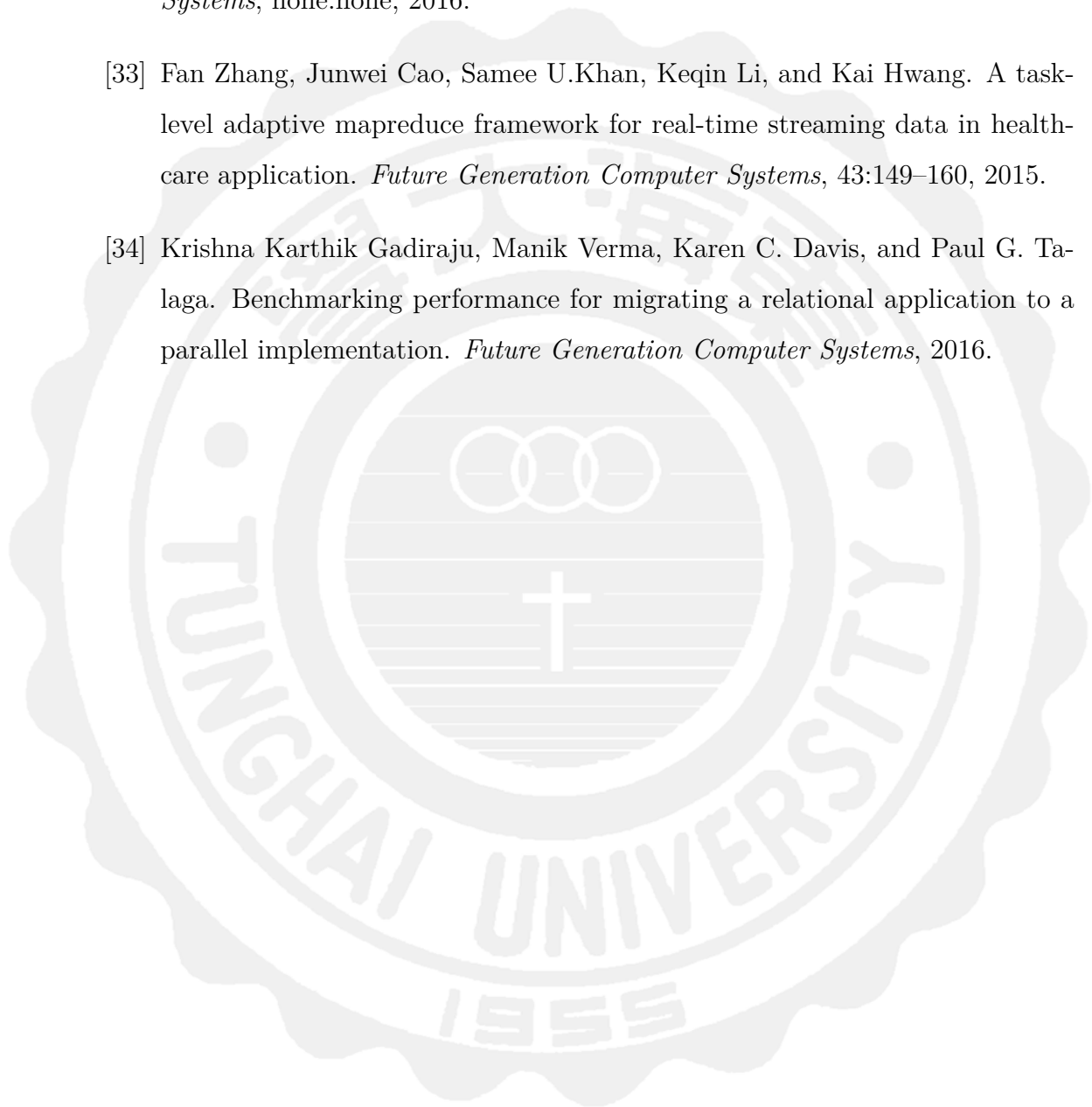
## References

- [1] Sunilkumar S. Manvi and Gopal Krishna Shyam. Resource management for infrastructure as a service (iaas) in cloud computing: A survey. *In Journal of Network and Computer Applications*, 41:424–440, May 2014.
- [2] Cloud computing, 2015. [http://en.wikipedia.org/wiki/Cloud\\_computing](http://en.wikipedia.org/wiki/Cloud_computing).
- [3] FarrukhShahzad. State-of-the-art survey on cloud computing security challenges. *Procedia Computer Science*, 37:357–362, 2014.
- [4] Nabil Sultan. Discovering the potential of cloud computing in accelerating the search for curing serious illnesses. *International Journal of Information Management*, 34:221–225, April 2014.
- [5] Jianghua Liu, Xinyi Huang, and Joseph K. Liu. Secure sharing of personal health records in cloud computing: Ciphertext policy attribute based sign-cryption. *Future Generation Computer Systems*, October 2014.
- [6] A. Meera and S. Swamynathan. Agent based resource monitoring system in iaas cloud environment. *Procedia Technology*, 10:200–207, 2013.
- [7] Geetha Manjunath and Dinkar Sitaram. *Moving To The Cloud - Chapter 3 - Platform as a Service*. Syngress, 2012.
- [8] Ying-Dar Lin, Minh-Tuan Thai, Chih-Chiang Wang, and Yuan-Cheng Lai. Two-tier project and job scheduling for saas cloud service providers. *Journal of Network and Computer Applications*, 52:26–36, 2015.

- [9] Chandan Banerjee, AnirbanKundu, and RanaDattagupta. Saas oriented generic cloud compiler. *Procedia Technology*, 10:253–261, 2013.
- [10] Hai Wang, Zeshui Xua, Hamido Fujita, and Shousheng Liud. Towards felicitous decision making: An overview on challenges and trends of big data. *Information Sciences*, 367-368:747–765, 2016.
- [11] David Gil and Il-Yeol Song. Modeling and management of big data: Challenges and opportunities. *Future Generation Computer Systems*, 63:96–99, 2016.
- [12] Jorge Merino, Ismael Caballero, Bibiano Rivas, Manuel Serrano, and Mario Piattini. A data quality in use model for big data. *Future Generation Computer Systems*, 63:123–130, 2016.
- [13] Grolinger. Data management in cloud environments: Nosql and newsql data stores. *Journal of Cloud Computing*, pages 2–22, 2013.
- [14] R. Rees. No problem: An intro to nosql databases,. 2010.
- [15] E. Gallagher. Nosql benchmark study release. 2014.
- [16] Sarwar Kamal, Shamim Hasnat Ripon, Nilanjan Dey, Amira S. Ashour, and V. Santhid. A mapreduce approach to diminish imbalance parameters for big deoxyribonucleic acid dataset. *Computer Methods and Programs in Biomedicine*, 131:191–206, 2016.
- [17] QaziMamoon Ashraf and Mohamed HadiHabaeb. Autonomic schemes for threat mitigation in internet of things. *Journal of Network and Computer Applications*, 49:112–127, March 2015.
- [18] Heiko Niedermayer, Ralph Holz, Marc-Oliver Pahl, and Georg Carle. On using home networks and cloud computing for a future internet of things. *Future Internet*, 6152:70–80, March 2009.
- [19] Nitesh Maheshwari, Radheshyam Nanduri, and Vasudeva Varma. Dynamic energy efficient data placement and cluster reconfiguration algorithm for

- mapreduce framework. *Future Generation Computer Systems*, 28:119–127, January 2012.
- [20] Apache hadoop, 2015. <http://hadoop.apache.org/>.
- [21] Hdfs architecture guide, 2015. [https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html).
- [22] Yifeng Luo, Siqiang Luo, Jihong Guan, and Shuigeng Zhou. A ramcloud storage system based on hdfs: Architecture implementation and evaluation. *Journal of Systems and Software*, 86:744–750, March 2013.
- [23] Apache hbase, 2015. <http://hbase.apache.org/>.
- [24] Ho Lee, Bin Shao, and U. Kang. Fast graph mining with hbase. *Information Sciences*, 315:56–66, September 2015.
- [25] C. Li. Transforming relational database into hbase: A case study. *International Conference on Software Engineering and Service Sciences*, pages 683–687, 2010.
- [26] Apache hive vs mysql-what are the key differences?, 2015. <http://blog.matthewrathbone.com/2015/12/08/hive-vs-mysql.html/>.
- [27] Hive vs. rdbms, 2014. <http://hadooptutorial.info/hive-vs-rdbms/>.
- [28] Apache hive, 2014. <http://hive.apache.org/>.
- [29] Apache sqoop, 2016. <http://sqoop.apache.org/>.
- [30] Arushi Jain and Vishal Bhatnagar. Crime data analysis using pig with hadoop. *Procedia Computer Science*, 78:571–578, 2015.
- [31] Sahithi Tummalapalli and Venkata rao Machavarapu. Managing mysql cluster data using cloudera impala. *Procedia Computer Science*, 85:463–474, 2016.
- [32] Ying-Ti Liao, Jiazheng Zhou, Shih-Chang Chen, Ching-Hsien Hsu, Wenguang Chen, Mon-Fong Jiang, and Yeh-Ching Chung. Data adapter for querying and

- transformation between sql and nosql database. *Future Generation Computer Systems*, none:none, 2016.
- [33] Fan Zhang, Junwei Cao, Samee U.Khan, Keqin Li, and Kai Hwang. A task-level adaptive mapreduce framework for real-time streaming data in health-care application. *Future Generation Computer Systems*, 43:149–160, 2015.
- [34] Krishna Karthik Gadiraju, Manik Verma, Karen C. Davis, and Paul G. Tagala. Benchmarking performance for migrating a relational application to a parallel implementation. *Future Generation Computer Systems*, 2016.



# 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. Creat SSH authentication login

```
# ssh-keygen -t rsa -f \~{}/.ssh/id\_{}rsa -P ""
# cp \~{}/.ssh/id\_{}rsa.pub \~{}/.ssh/authorized\_{}keys
# scp -r \~{}/.ssh hduser:~/
```

## VI. Download hadoop

```
# cd ~
# wget http://ftp.twaren.net/Unix/Web/apache/hadoop/common \\\
/hadoop-2.6.0/hadoop-2.6.0.tar.gz
# tar zxf hadoop-2.6.0.tar.gz
# mv hadoop-2.6.0.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://hduser-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>hduser</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

hduser
hduser-02
hduser-03
hduser-04
```

## IX. Copy hadoop to all nodes

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

```
# hduser:50030
```

# Appendix B

## HBase Installation

### I. Download HBase

```
# cd ~
# wget http://ftp.twaren.net/Unix/Web/apache/hbase\
  /hbase-1.0.0/hbase-1.0.0-hadoop2-bin.tar.gz
```

### II. Unzip hbase-1.0.0-hadoop2-bin.tar.gz

```
# tar xzf hbase-1.0.0-hadoop2-bin.tar.gz
```

### III. Move the File of HBase

```
# mv hbase-1.0.0-hadoop2 hbase
```

### IV. 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://hduser: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/regionserver

hduser          hduser-02
hduser-03      hduser-04
```

### III. Copy jar to hbase/lib

```
# rm lib/hadoop-*
# cd /home/hduser/hadoop/share/hadoop
# cp *.jar /home/hduser/hbase/lib/
```

### IV. Copy hbase to all nodes

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

### V. HBase monitoring website

```
# hduser:60010
```

# Appendix C

## Hive Installation

### I. Download Apache Hive

```
# sudo wget http://apache.stu.edu.tw/hive/hive-1.2.1/apache-hive-1.2.1-bin.tar.gz
```

### II. Unzip hive-1.2.1.bin.tar.gz

```
# tar -zxvf apache-hive-1.2.1-bin.tar.gz  
# mv apache-hive-1.2.1-bin/ hive
```

### III. Setting .bashrc

```
# vim .bashrc  
# export HIVE_HOME=/home/hduser/hive  
# export PATH=$HIVE_HOME/bin:$HIVE_HOME/conf:$PATH  
# source .bashrc
```

### IV. Create Hive file on HDFS

```
# hadoop fs -mkdir /tmp  
# hadoop fs -mkdir /user/warehouse  
# hadoop fs -chmod 777 /tmp  
# hadoop fs -chown 777 /user/warehouse
```

## V. Installation libmysql-java

```
# sudo apt-get install libmysql-java
# sudo cp /usr/share/java/mysql-connector-java-5.1.28.jar ~/hive/lib
```

## VI. Create a hive user on MySQL

```
# mysql -u root -p
# mysql > create database hive;
# mysql> grant all on *.* to 'hive'@'%' identified by 'hive';
# mysql> flush privileges;
# mysql> exit;
# sudo vim /etc/mysql/my.cnf
# Annotate bind-address = 127.0.0.1
```

## VII. Create a file in hive file

```
# mkdir -p /home/hduser/hive/iotmp
# chmod 777 /home/hduser/hive/iotmp
# cp hive-default.xml.template hive-site.xml
# vim hive-site.xml

<configuration>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://master:3306/hive?createDatabaseIfNotExist=true</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hive</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hive</value>
  </property>
  <property>
    <name>hive.metastore.uris</name>
    <value>thrift://master:9083</value>
  </property>
```

```
<property>
  <name>hive.exec.local.scratchdir</name>
  <value>/home/hduser/hive/iotmp</value>
  <description>Local scratch space for Hive jobs</description>
</property>
<property>
  <name>hive.downloaded.resources.dir</name>
  <value>/home/hduser/hive/iotmp</value>
  <description>Temporary local directory for added resources in the remote
    file system.</description>
</property>
<property>
  <name>hive.querylog.location</name>
  <value>/home/hduser/hive/iotmp</value>
  <description>Location of Hive run time structured log file</description>
</property>
<property>
  <name>hive.server2.logging.operation.log.location</name>
  <value>/home/hduser/hive/iotmp</value>
  <description>Top level directory where operation logs are stored if logging
    functionality is enabled</description>
</property>
</configuration>

# cp hive-env.sh.template hive-env.sh
# sudo vim hive-env.sh
  export HADOOP_HEAPSIZE=1024
  export HADOOP_HOME=/home/hduser/hadoop
  export HIVE_CONF_DIR=/home/hduser/hive/conf
  export HIVE_AUX_JARS_PATH=/home/hduser/hive/lib
```

# Appendix D

## Sqoop Installation

### I. Download Apache Sqoop

```
# sudo wget http://apache.stu.edu.tw/sqoop/1.4.6/sqoop-1.4.6.tar.gz
```

### II. Unzip sqoop-1.4.6.tar.gz

```
# tar -zxvf sqoop-1.4.6.tar.gz  
# mv sqoop-1.4.6/ sqoop
```

### III. Setting .bashrc

```
# sudo vim ~/.bashrc  
export SQOOP_HOME=/home/hduser/sqoop  
export SQOOP_CONF_DIR="$SQOOP_HOME/conf"  
export SQOOP_CLASSPATH="$SQOOP_CONF_DIR"  
export PATH=$PATH:$SQOOP_HOME/bin  
# source ~/.bashrc
```

### IV. Create empty accumulo directory

```
# sudo mkdir accumulo  
# sudo chown hadoop:hduser accumulo
```

## V. Create and edit sqoop-env.sh

```
# sudo cp sqoop/conf/sqoop-env-template.sh sqoop/conf/sqoop-env.sh
# sudo vim sqoop/conf/sqoop-env.sh
export HADOOP_COMMON_HOME=/home/hduser/hadoop
export HADOOP_MAPRED_HOME=/home/hduser/hadoop
export HCAT_HOME=/home/hduser/hive/hcatalog
export HBASE_HOME=/home/hduser/hbase
export HIVE_HOME=/home/hduser/hive
export ACCUMULO_HOME=/home/hduser/accumulo
```

## VI. Download MySQL connector

```
# wget http://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.36.zip
# sudo mv mysql-connector-java-5.1.36-bin.jar sqoop/lib
```