東海大學

資訊工程研究所

碩士論文

指導教授: 楊朝棟博士

運用 HBase 於醫療雲中資料轉換方法之實作

Implementation of Data Convert Method for HBase in Healthcare Cloud

研究生: 許文鴻

中華民國一零二年七月

摘　要

由於目前醫療體系各科別內部還是有很多使用 Excel 檔案格式來儲存各種量表的統計資料，如臨床常用的自我照顧能力量表 (Functional Independence Measure) 等，將數值存在 Excel 後，透過統計分析軟體 (如:SAS, SPSS, STATA 等) 進行分析。但如此一來各科別之間並無法有效的分享以 Excel 存檔的資料。建議將資料進行格式轉換，存在資料庫內，方便各科別分享，而眾多的 Excel 檔案集合而成的資料庫，就如同一個非關聯性的資料庫，因此使用 Hbase 存放整合後的資料。也因 Hbase 是一非關聯性資料庫，在實作上的可行性也較高。考量到資料使用人員需要快速、準確、高效的查詢各個不同領域所需資料。我們必須提供如同 SQL 語法能夠以語句的方式查詢資料，資料使用人員通過學習、利用和掌握查詢語言，讓 Hbase 裡的資料，能夠得的充分運用，提高統計工作效率。本論文的目的是以 HBase 為基礎，在其周邊建構完整的導入工具與解決方案，為了能夠更方便的使用 HBase，本論文建構一個友善的 HBase 資料庫連線客戶端工具，提供視覺化的管理介面。

Keywords: NoSQL 資料庫, Hadoop, HBase, 鍵值儲存

Abstract

Currently, most digital health care systems used in divisions of medical centers still adopt the Excel file format for a variety of scales statistics, such as the clinical self-care ability scale for Functional Independence Measure. Although people can further analyze excel files using statistical analysis software, such as SAS, SPSS, or STATA, they cannot effectively share the archived data in Excel file format among different divisions. We propose to do format conversion on these data and store them in a database. As the collection of Excel files, a non-relational database, cannot be shared with ease, we plan to use HBase storage to further integrate data. The purpose of this thesis is to construct complete import tools and solutions based on HBase with easy access of data in HBase. Also, a visual interface is used to manage HBase to implement user friendly client connection tools for the HBase database.

Keywords: NoSQL database, Hadoop, HBase, Key-value stores

致謝詞

首先誠摯的感謝指導教授楊朝棟博士，有幸跟隨老師的腳步學習使我由廣到深的學習，從進入雲端運算領域的虛擬化技術到大量資料的處理技術，跟隨著電腦應用趨勢的腳步，使我在短短的兩年中獲益匪淺。老師對學問的嚴謹更是我輩學習的典範。

再來要感謝的是各位協助我論文更加完美的各位口試委員，感謝劉榮春老師，多次與我討論細節，奠定了我的許多基礎知識，以及論文撰稿知識。也感謝陳同孝教授以及大學母校的張志宏教授、盧志偉教授能再次指導我，各位教授不辭辛苦地前來東海當我的口試委員，為我的論文提出了許多的建議，讓我能夠讓論文更為完整且嚴謹。

本論文的完成另外亦得感謝高效能實驗室的各位學長、同學、及學弟妹的大力協助，尤其是陳煒勝學長，從我進實驗室開始便在他的帶領下迅速成長、配合在大學時期就開始學習的 linux 使用技巧以及網路的基礎概念，在各種伺服器架設等等的領域皆越來越得心應手，歐韋伸同學也常常跟我討論一些技術上的重點，一起釐清遇到的問題，以及黃冠龍擔任總管協助處理實驗室的大小事務，幫我們減輕了不少負擔。實驗室的大家，歐韋伸、廖啟瑞、黃冠龍與劉育佐是一起奮鬥兩年的夥伴，雖然負責的專業項目不一樣，但是透過互相學習，大家終於都能夠完成各自的論文，這篇論文的完成也要感謝學弟顏尹臻以及學妹呂欣汶，在他們的協助下能夠順利完成這份論文，另外還有更多更多的人要感謝，他們不只是在論文上，更是生活上的幫助，使我可以順利的完成論文。

兩年的日子裡，實驗室里共同的生活點滴，學術上的討論、言不及義的閒扯、早上四點多的早餐、趕計畫書、日夜顛倒寫論文做實驗的革命情感，感謝眾位學長姐、同學、學弟妹的共同砥礪，你/妳們的陪伴讓兩年半的研究生活變得絢麗多彩。最後，謹以此文獻給一直支持我，我所摯愛的親人。

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In today's national health care system, whether it is up to the competent authority or next to each health care system is potentially a serious problem. it Is dispersed and independent of the medical information system, medical care information system development is highly fragmented, cannot be integrated and interconnected, due to different needs of different ethnic groups, medical care information system is highly fragmented, cannot be easily integrated and interconnected. Moreover, since systems are built at different times, there may be no suitable connection among systems. The causes of this situation include: 1. information used by the personnel, 2. preference of data entry personnel, 3. services on different objects, 4. system build time, 5. change of health regulations, 6. different supporting plans or sources, and 7. varying definition of database field names in different database systems.

Database of medical care does not allow slightest error and makes a mistake. Integration and shutdown of the medical system, under the premise of ensuring the quality of medical care, do not have any change to maintain normal maintenance and operation of the health care system, , is now each medical institution overall conservative approach. Due to database don't have integration lead to patients

re-doing the same checks because of different illness or physical condition. This is not only caused the patient weighed down missed a golden time for treatment, but also caused serious waste of medical resources.

1.2 Contributions

The current health care system within the medical divisions still uses Excel file formats to store a range of scales statistics ,such as commonly used in clinical self-care Ability Scale (Functional Independence Measure). Values stored in Excel can be analyzed via statistical analysis software, such as SAS, SPSS, STATA, but cannot be efficiently shared between divisions. To integrate data, we proposed to converse the format of the Excel data and store them in a database based on HBase.

HBase is a non-relational database with structure similar to Excel, but with higher implementation feasibility. Considering of the data usage characteristic that needs fast, accurate, and efficient query of the required information for people in various areas, we provide a visual graphical interface, instead of query languages, to speed up data query in the HBase database. As a result, people are able to make full use of data in HBase and improve efficiency of statistical work. As opposed to the traditional commercial relational database, HBase is scalable, high-performance, low-cost, and with other advantages, but it does not have a complete and friendly user environment. Therefore, in addition to data conversion in HBase storage, it is important to provide appropriate technical support, friendly HBase user interface, and approachable operation syntax, etc.

The purpose of this thesis is based on HBase to construct complete import tools and solutions in its surrounding. First, we analyzed the characteristics of the source data for conversion into to HBase. Then we recognized patterns of data usage, and constructed HBase Data Model based on user behaviors. To make it easier to access HBase, we also implemented a visual interface to manage HBase as a user friendly database.

1.3 Thesis Organization

The rest of the thesis is organized as follows: In Chapter 2, we describe the used techniques and some background knowledge. Chapter 3 describes the system architecture used in the thesis. Chapter 4 shows experimental results for the proposed system. Chapter 5 provides conclusions and future work of this work.

Chapter 2

Background Review and Related Work

2.1 Bigdata

The Big Data has already become one hot issue; it is mostly encountered in research fields with challenges of analyzing and forecasting huge amount of information, such as in weather forecasting, genetic analysis, biological research, financial or commercial information. To model and predict complex phenomena, we often use high-speed computers combined with distributed or parallel computing techniques to deal with huge amount of data[33]. Moreover, in recent years, more and more enterprises face with challenges of data explosion with unexpectedly rapid growth rates of the amount of data in storage systems [14]. And many companies worry that they soon will encounter the same situation. It is difficult to process big data in most relational database management systems, because it needs to run massive parallel software concurrently on hundreds or thousands of servers.

Big data promises to be transformative. As computing resources have evolved and advanced to be able to handle data of bigger sizes and complexity, companies start to reap many benefits from big data analysis, as shown in Figure 2.1. Little wonder that big data is a hot topic in corporate boardrooms and IT departments,

with many leading firms doing more than talking. According to a recent A.T. Kearney IT innovation study, more than 45 percent of companies have implemented a business-intelligence or big data initiative in the past two years [1].

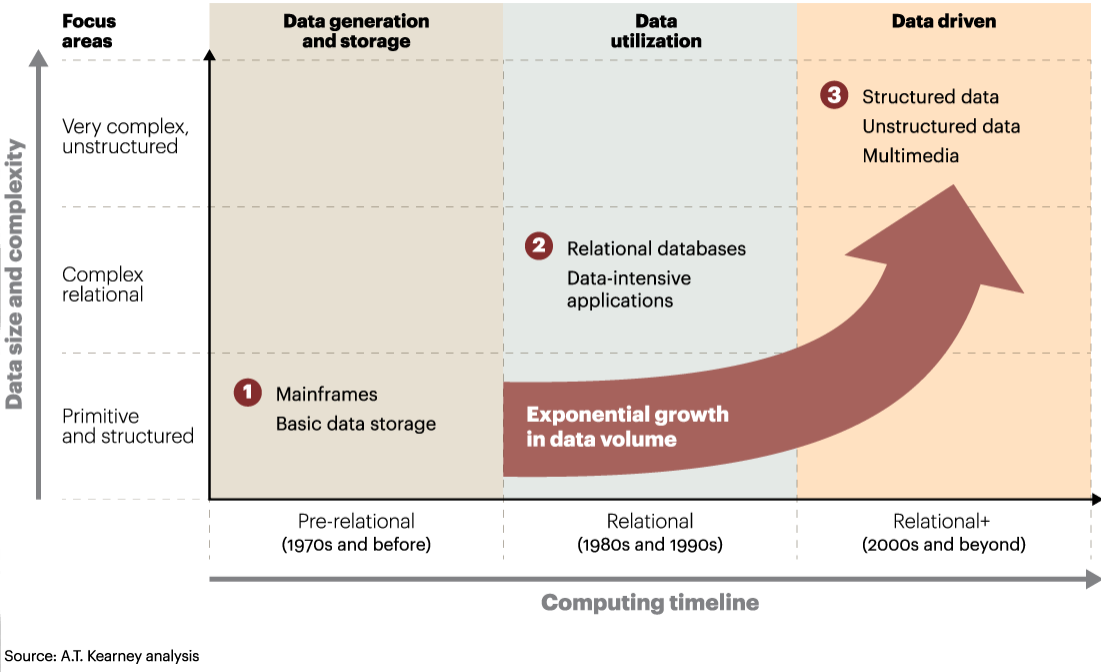

FIGURE 2.1: A.T. Kearney IT analysis the evolution of big data

Common massive amount of data include interactive data information such as images, audios, videos, Internet search indexes, astronomical data, genetic information, medical records, and the website Log records transmitted through sensor networks, social networking, and wireless networks. These raw data present the proliferation of big data [23]. They are mostly non-structured or semi-structured data, not easy to be processed by using the traditional practice in relational database. Through fixed data field architecture, the data can be stored to a relational database for further processing. In addition to the challenges of huge amount of data, the information coming from various structures tends to complicate the situation.

The big data is a large and complex issue. We face with numerous challenges to find useful information from analyzing big data, and use it to reduce enterprise

risks, promote revenues, and improve competitiveness. These challenges include how to obtain, store, search, share, analyze, and visually present the big data.

2.2 NoSQL

Big data also stands for a new and hot issue in cloud computing. Traditionally, structural data are normalized in advanced, stored in databases, and then can be manipulated as the principal resource and headstone to support the enterprise IT systems. On the other hand, the rest of data, which are un-structural/semi-structural and generally in a massive quantity comparing with the structural ones, are hard to be processed and are casted aside or trashed on the corner. However, as new technologies in cloud computing like Hadoop and NoSQL emerging, these "trashes", in a big quantity so are called as the "big data", are now considered as the most valuable resources while the enterprise taking strides into the new market. Thus, issues like gathering, storing, modeling, analyzing, and manipulating big data become hot for cloud computing researches and applications. For big data, what come to mind first are no longer the past system of hegemony of the database market Oracle, or the software giant Microsoft, but instead, the Apache Foundation's open source Hadoop parallel computing and storage architecture, and the HBase NoSQL distributed database[19].

The Hadoop, a parallel computing platform developed by the Apache Software Foundation, is an open source compiler tool and distributed file system. NoSQL database, also called Not Only SQL, is an approach for data management and database design that is useful for very large sets of distributed data. NoSQL, which encompasses a wide range of technologies and architectures, seeks to address the scalability and big data performance issues that relational databases were not designed to solve. NoSQL is especially useful when an enterprise needs to access and analyze massive amounts of unstructured data or data stored remotely on multiple virtual servers in the cloud computing [29]. NoSQL is a general term meaning that the database is not an RDBMS which supports SQL as its primary

access language. There are many types of NoSQL databases: BerkeleyDB is an example of a local NoSQL database, whereas HBase is very much a distributed database.HBase, written in Java, is an open source, non-relational, and distributed database modeled after Google's BigTable [27]. It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of the Hadoop Distributed File system (HDFS), providing BigTable-like capabilities for Hadoop. That is, it provides a fault-tolerant way of storing large quantities of sparse data.

2.3 HDFS

HDFS is an Apache Software Foundation project and a subproject of the Apache Hadoop project. The Hadoop Distributed File System, is a distributed file system designed to hold very large amounts of data (terabytes or even petabytes), and provide high-throughput access to this information. Files are stored in a redundant fashion across multiple machines to ensure their durability to failure and high availability to very parallel applications [31]. HDFS has many similarities with other distributed file systems, but is different in several respects. One noticeable difference is HDFS's write-once-read-many model that relaxes concurrency control requirements, simplifies data coherency, and enables high-throughput access [30]. Another unique attribute of HDFS is the viewpoint that it is usually better to locate processing logic near the data rather than moving the data to the application space. HDFS rigorously restricts data writing to one writer at a time. Bytes are always appended to the end of a stream, and byte streams are guaranteed to be stored in the order written.

2.3.1 HDFS architecture

HDFS is comprised of interconnected clusters of nodes where files and directories reside. An HDFS cluster consists of a single node, known as a NameNode, which manages the file system namespace and regulates client access to files. In addition,

data nodes (DataNodes) store data as blocks within files. Within HDFS, a given name node manages file system namespace operations like opening, closing, and renaming files and directories. A name node also maps data blocks to data nodes, which handle read and write requests from HDFS clients. Data nodes also create, delete, and replicate data blocks according to instructions from the governing name node. The Figure 2.2 shows the architecture of HDFS.

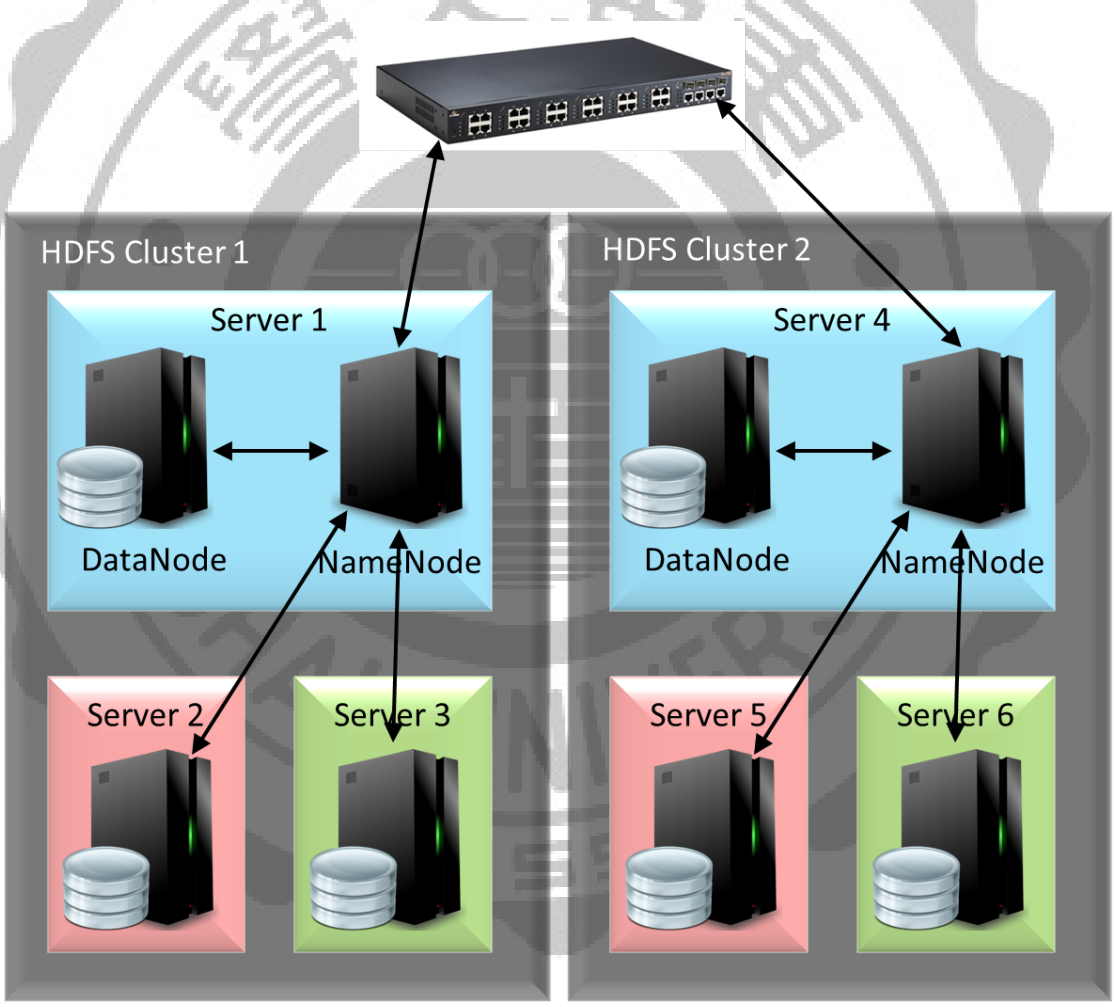

FIGURE 2.2: The HDFS architecture

2.4 HBase

HBase is a column-oriented database management system that runs on top of HDFS. It is well suited for sparse data sets, which are common in many big data. Unlike relational database systems, HBase does not support a structured query

language like SQL; in fact, HBase is not a relational data store at all. HBase applications are written in Java much like a typical MapReduce application, and HBase also supports writing applications in Avro, REST, and Thrift [3, 9].

An HBase system comprises a set of tables. Each table contains rows and columns, much like a traditional database. Each table must have an element defined as a Primary Key, and all access attempts to HBase tables must use this Primary Key. An HBase column represents an attribute of an object; for example, if the table is used to store diagnostic logs from servers, where each row might be a log record, a typical column in such a table would be the timestamp of when the log record was written, or perhaps the server name where the record originated. In fact, HBase allows for many attributes to be grouped together into so called column families, such that the elements of a column family are all stored together. This is different from a row-oriented relational database, where all the columns of a given row are stored together. With HBase you must predefine the table schema and specify the column families. However, it is very flexible in that new columns can be added to families at any time, making the schema flexible and therefore able to adapt to changing application requirements. Just as HDFS has a NameNode and slave nodes, and MapReduce has JobTracker and TaskTracker slaves, HBase is built on similar concepts. In HBase a master node HMaster manages the cluster, and region servers store portions of the tables and perform the work on the data. HMaster is the implementation of the Master Server. The Master server is responsible for monitoring all RegionServer instances in the cluster, and is the interface for all metadata changes. In a distributed cluster, the Master typically executes on the NameNode, and HRegionServer is the RegionServer implementation. It is responsible for serving and managing regions. In a distributed cluster, a RegionServer runs on a DataNode. Through the Zookeeper other machines are selected within the cluster as HMaster in HBase, unlike the HDFS architecture NameNode with single point of availability problems.

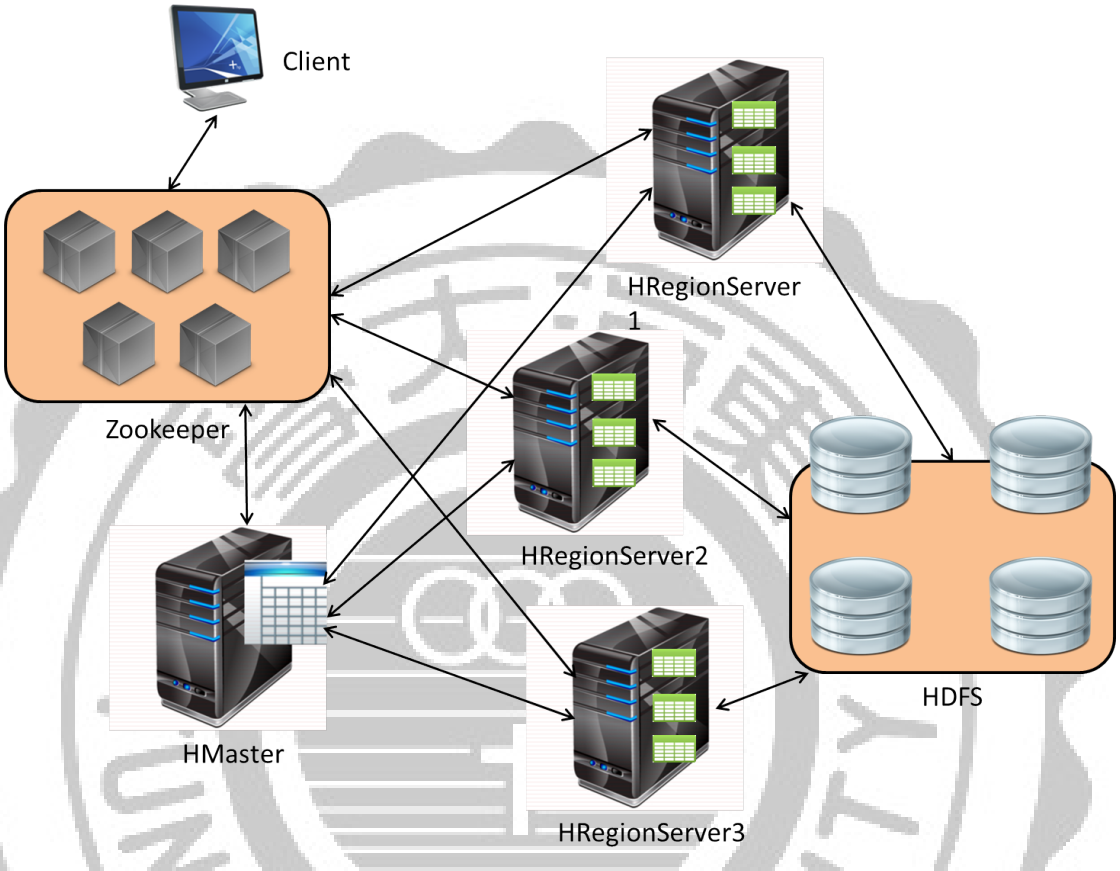

FIGURE 2.3: The roles in an HBase cluster

Figure 2.3 shows the roles in an HBase cluster. HBase is built on top of Apache Hadoop and Apache ZooKeeper. Like the rest of the Hadoop ecosystem components, it is written in Java. HBase can run in three different modes: standalone, pseudo-distributed, and full-distributed. However, HBase has many features which support both linear and modular scaling. HBase clusters can be expanded by adding RegionServers hosted on commodity class servers. For example, when a cluster expands from 10 to 20 RegionServers, it doubles both in terms of storage as well as processing capacity. RDBMS can scale well, but only up to a point - specifically, the size of a single database server - and for the best performance requires specialized hardware and storage devices. HBase features of note are:

- Strongly consistent reads/writes: HBase is not an "eventually consistent" DataStore. This makes it very suitable for tasks such as high-speed counter

aggregation.

- Automatic sharding: HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
- Automatic RegionServer failover.
- Hadoop/HDFS Integration: HBase supports HDFS out of the box as its distributed file system.
- MapReduce: HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink [28].
- Java Client API: HBase supports an easy to use Java API for programmatic access.
- Thrift/REST API: HBase also supports Thrift and REST for non-Java front-ends.
- Block Cache and Bloom Filters: HBase supports a Block Cache and Bloom Filters for high volume query optimization.
- Operational Management: HBase provides build-in web-pages for operational insight as well as JMX metrics.

2.4.1 HBase Region

Regions are the basic element of availability and distribution for tables, and are comprised of a Store per Column Family.Follow Figure shows the heirarchy of objects in HBase.

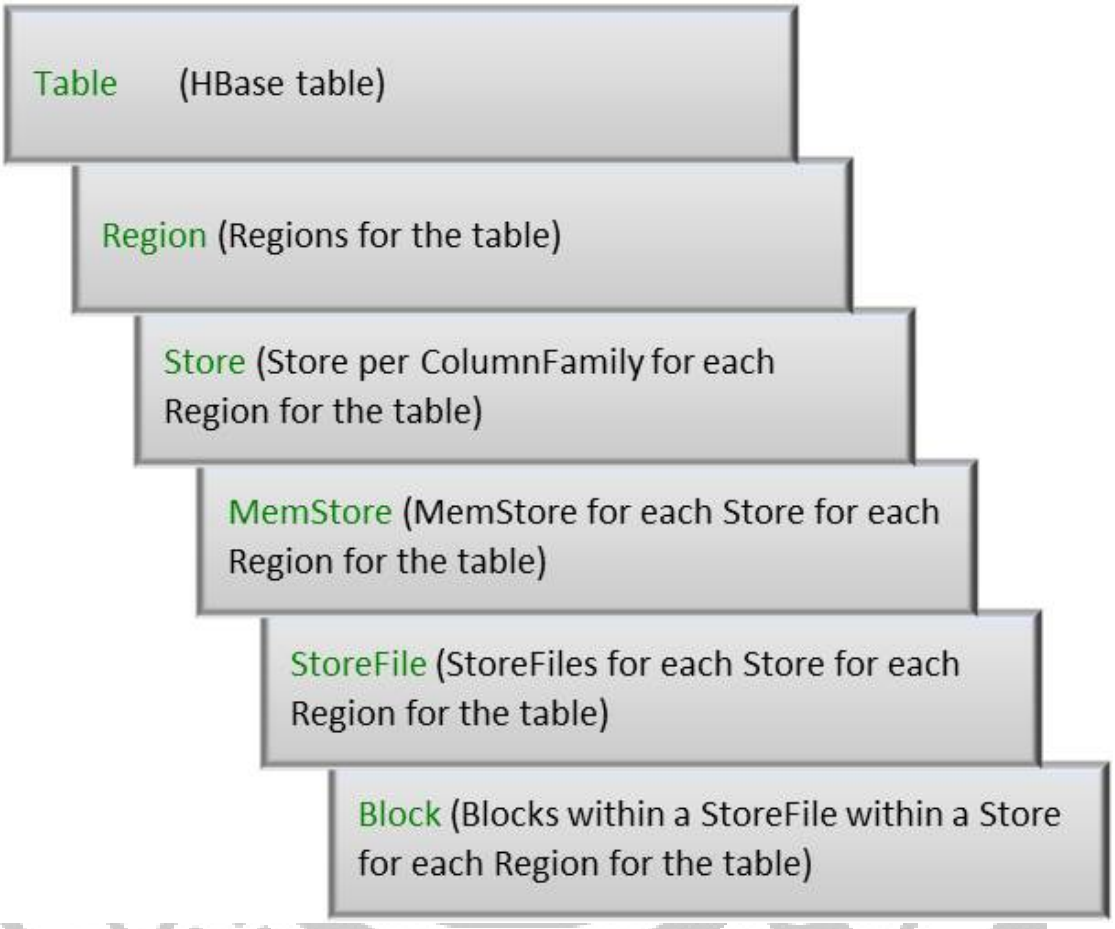

FIGURE 2.4: The heirarchy of objects in HBase

2.4.2 Catalog tables

In HBase, The catalog tables -ROOT- and .META. exist as HBase tables. They are filtered out of the HBase shell's list command, but they are in fact tables just like any other. The following Table shows the Metrics of catalog tables -ROOT- and .META.

TABLE 2.1: The Metrics of catalog tables -ROOT- and .META.

Region Name	Metrics
-ROOT-„0.70236052	numberOfStores=1, numberOfStorefiles=1, storefileUncompressedSizeMB=0, storefileSizeMB=0, memstoreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=538, writeRequestsCount=1, rootIndexSizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=6, currentCompactedKVs=6, compactionProgressPct=1.0, coprocessors=[]
.META.„1.1028785192	numberOfStores=1, numberOfStorefiles=0, storefileUncompressedSizeMB=0, storefileSizeMB=0, memstoreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=8711, writeRequestsCount=70, rootIndexSizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=0, currentCompactedKVs=0, compactionProgressPct=NaN, coprocessors=[]

Region names consist of the containing table's name, a comma, the start key, a comma, and a randomly generated region id. The -ROOT- and .META. tables are internal system tables (or 'catalog' tables). The -ROOT- keeps a list of all regions in the .META. table. The .META. table keeps a list of all regions in the system. The empty key is used to denote the table start and table end. A region with an empty start key is the first region in a table. If region has both an empty start and end keys, it is the only region in the table.

2.4.3 DataModel

The Bigtable data model and therefor the HBase data model too since it's a clone, is particularly well adapted to data-intensive systems. Getting high scalability from your relational database isn't done by simply adding more machines because its data model is based on a single-machine architecture [5]. For example, a JOIN between two tables is done in memory and does not take into account the possibility that the data has to go over the wire. Companies who did propose relational distributed databases had a lot of redesign to do and this why they have high licensing costs. The other option is to use replication and when the slaves are overloaded with writes, the last option is to begin sharding the tables in sub-databases. At that point, data normalization is a thing you only remember seeing in class which is why going with the data model presented in this thesis shouldn't bother you at all.

To put it simply, HBase can be reduced to a Map<byte[], Map<byte[], Map<byte[], Map<Long, byte[]»». The first Map maps row keys to their column families. The second maps column families to their column keys. The third one maps column keys to their timestamps. Finally, the last one maps the timestamps to a single value. The keys are typically strings, the timestamp is a long and the value is an uninterpreted array of bytes. The column key is always preceded by its family and is represented like this: family:key. Since a family maps to another map, this means that a single column family can contain a theoretical infinity of column keys. So, to retrieve a single value, the user has to do a get using three keys:

(Table, Rowkey, Column key, Timestamp)-> Value

2.4.3.1 RowKey

The row key is treated by HBase as an array of bytes but it must have a string representation. A special property of the row key Map is that it keeps them in a lexicographical order. For example, numbers going from 1 to 100 will be ordered like the following Figure 2.5:

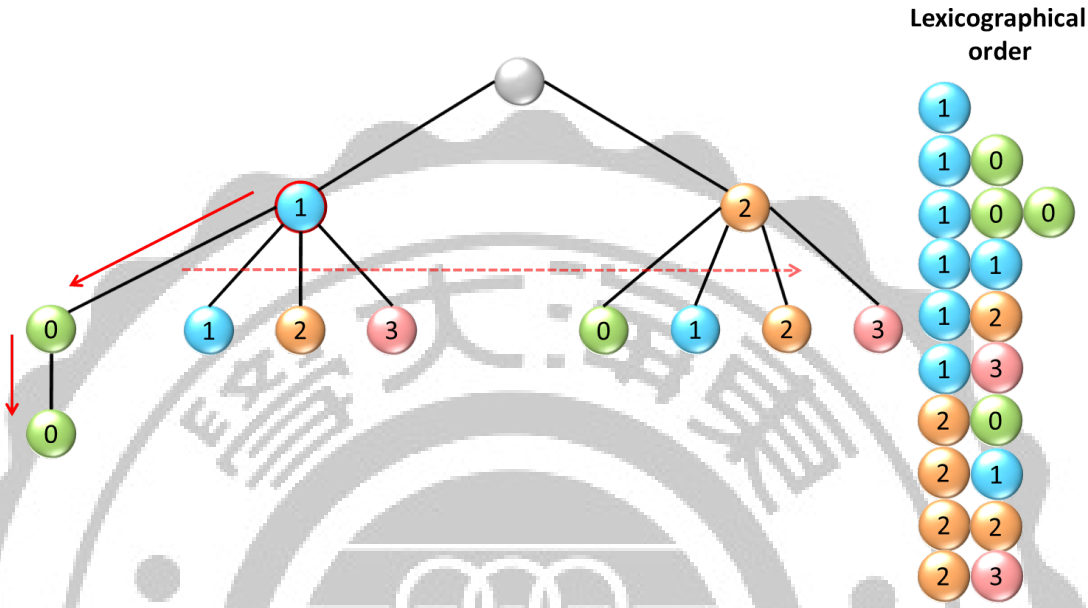

FIGURE 2.5: Lexicographical order

To keep the integers natural ordering, the row keys have to be left-padded with zeros. To take advantage of this, the functionalities of the row key Map are augmented by offering a scanner which takes a start row key (if not specified, the first one in the table) and an stop row key (if not specified, the last one in the table). For example, if the row keys are dates in the format YYYYMMDD, getting the month of July 2008 is a matter of opening a scanner from 20080700 to 20080800. It does not matter if the specified row keys are existing or not, the only thing to keep in mind is that the stop row key will not be returned which is why the first of August is given to the scanner [**?**, 4].

2.4.3.2 Column Families

A column family regroups data of a same nature in HBase and has no constraint on the type [25]. The families are part of the table schema and stay the same for each row; what differs from rows to rows is that the column keys can be very sparse. For example, row "20080702" may have in its "Name:" family the following column keys:

Name : Chinese
Name : Eng
Name : Other

While row ”20080703” only has:

Name : Other

Developers have to be very careful when using column keys since a key with a length of zero is permitted which means that in the previous example data can be inserted in column key ”Name :”. We strongly suggest using empty column keys only when no other keys will be specified. Also, since the data in a family has the same nature, many attributes can be specified regarding performance and timestamps.

2.4.3.3 Timestamps

The values in HBase may have multiple versions kept according to the family configuration. By default, HBase sets the timestamp to each new value to current time in milliseconds and returns the latest version when a cell is retrieved. The developer can also provide its own timestamps when inserting data as he can specify a certain timestamp when fetching it.

2.4.3.4 Family Attributes

The following attributes can be specified or each families:

- Compression
  - Record: means that each exact values found at a rowkey+columnkey +timestamp will be compressed independently.

  - Block: means that blocks in HDFS are compressed. A block may contain multiple records if they are shorter than one HDFS block or may only contain part of a record if the record is longer than a HDFS block.

- Timestamps
  - Max number: the maximum number of different versions a value has.
  - Time to live: versions older than specified time will be garbage collected.

- Block Cache
  - Caches blocks fetched from HDFS in a LRU-style queue. Improves random read performances and is a nice feature while waiting for full in-memory storage.

2.4.4 Real Life Example

A good example on how to demonstrate the HBase data model is a blog because of its simple features and domain. Suppose the following mini-SRS:

- The blog entries, which consist of a title, an under title, a date, an author, a type (or tag), a text, and comments, can be created and updated by logged in users.
- The users, which consist of a username, a password, and a name, can log in and log out.
- The comments, which consist of a title, an author, and text, can be written anonymously by visitors as long as their identity is verified by a captcha.

2.5 Zookeeper

Apache ZooKeeper is a software project of the Apache Software Foundation, ZooKeeper was a sub project of Hadoop but is now a top-level project in its

own right. ZooKeeper's architecture supports high-availability through redundant services. The clients can thus ask another ZooKeeper master if the first fails to answer. ZooKeeper nodes store their data in a hierarchical name space, much like a file system or a trie datastructure. Clients can read and write from/to the nodes and in this way have a shared configuration service. Updates are totally ordered. ZooKeeper is a distributed, open-source coordination service for distributed applications. It exposes a simple set of primitives that distributed applications can build upon to implement higher level services for synchronization, configuration maintenance, and groups and naming. It is designed to be easy to program to, and uses a data model styled after the familiar directory tree structure of file systems. It runs in Java and has bindings for both Java and C.Coordination services are notoriously hard to get right. They are especially prone to errors such as race conditions and deadlock. The motivation behind ZooKeeper is to relieve distributed applications the responsibility of implementing coordination services from scratch. Through the zookeeper selected other machines within the cluster as HMaster in HBase, does not like the HDFS architecture NameNode have single point of availability problems.

2.6 Cloudera CDH

CDH (Cloudera's Distribution, including Apache Hadoop) is Cloudera's 100% open-source Hadoop distribution, and the world's leading Apache Hadoop solution. More enterprises have downloaded CDH than all other distributions combined. Furthermore, CDH is backed by Cloudera's global support organization and its unparalleled team of developers and committers who contribute more to the Apache Hadoop ecosystem than any other company. This combination means that by using Cloudera, the developer can be sure of successfully deploying Hadoop project, and deployed it faster [2]. CDH delivers the core elements of Hadoop – scalable storage and distributed computing – as well as all of the necessary enterprise capabilities such as security, high availability and integration with a broad range of hardware and software solutions. All the integration work is done for the

developer, and the entire solution is thoroughly tested and fully documented. By taking the guesswork out of building out the Hadoop deployment, CDH gives the developer a streamlined path to success in solving real business problems.

2.7 Related Work

Big Data is more talked about than felt in our everyday lives; there will have big changes in industrial and business processes when we have pervasive real-time analytics of sensor data.

The ubiquity of location enabled devices has resulted in a wide proliferation of location based applications and services. To handle the growing scale, database management systems driving such location based services (LBS) must cope with high insert rates for location updates of millions of devices, while supporting efficient real-time analysis on latest location. Shoji Nishimura present the design and implementation of MD -HBase [18], a scalable data management infrastructure for LBSs that bridges this gap between scale and functionality. Their approach leverages a multi-dimensional index structure layered over a key-value store. The underlying key-value store allows the system to sustain high insert throughput and large data volumes, while ensuring fault-tolerance, and high availability. Shoji Nishimura present the design of MD -HBase that demonstrates how two standard index structures—the K-d tree and the Quad tree—can be layered over a range partitioned key-value store to provide scalable multi-dimensional data infrastructure. their prototype implementation using HBase, a standard open-source key-value store, can handle hundreds of thousands of inserts per second using a modest 16 node cluster, while efficiently processing multi-dimensional range queries and nearest neighbor queries in real-time with response times as low as few hundreds of milliseconds.

As the development of telecommunication technology and mobile device technology, geo-location data happened everywhere and every time from humans' real life. Because all of the smart device' s applications are include spatial components

now. When the traditional relational database cannot support the continuously flooded data, researchers developed key-value based NoSQL database system to meet this problem. But spatial data processes are rarely considered until now. In this case client user must have their own spatial data processing component to process the spatial data from NoSQL database. In this thesis, Yan Li and his team proposed a spatial index based on document based NoSQL which can distribute the spatial data by using the geo-hash method and can satisfy the high insert rate by using the b-tree based index method. At last their developed our method on OrientDB which is document based NoSQL[16].

To find a scalable solution to process the large-scale data is a critical issue in either the relational database system or the emerging NoSQL database. With the inherent scalability and fault tolerance, MapReduce is attractive to process the massive data in parallel. Most of previous works focus on the Hadoop distributed file system to support the SQL or SQL-like queries. Wu-Chun Chung with his team propose the JackHare framework with SQL query compiler, JDBC and a systematical method using MapReduce for processing the unstructured data in NoSQL database. While importing the JDBC driver to a SQL client GUI, Wu-Chun Chung et al. provide the corresponding queries to manipulate the data residing in the NoSQL database. To organize the data with less complexity, Wu-Chun Chung with his team further introduce a remapping strategy to translate the data model from relational database to NoSQL database. Experimental results show that their approaches can perform Wu-Chun Chung with his teamll with efficiency and scalability.

Paolo Atzeni et al. propose a common programming interface to NoSQL systems called SOS (Save Our Systems)[7]. Its goal is to support application development by hiding the specific details of the various systems to solve the problem heterogeneity of the languages and of the interfaces they offer to developers and users. They provided a common data model that allows the creation and querying of NoSQL databases defined in MongoDB, HBase and Redis using a common set of simple atomic operation. It is based on a metamodelling approach, in the sense that the specific interfaces of the individual systems are mapped to a common one.

The tool provides interoperability as well, since a single application can interact with several systems at the same time.

Combined the features of data flow management systems and the advantages of Hadoop cloud computing platforms, Zhi QIU et al. design and implement a Hadoop-based platform for distributed data flow management systems, using MapReduce to process the user request, using Hadoop distributed file system (HDFS) to manage the data flow files, and using Hadoop database (HBase) to manage the data flow information[21]. The data flow management system based on Hadoop is improved in efficiency, compared with the traditional system. It also carries on the advantages of Hadoop cloud computing platform, such as reliability and high expandability. Especially when the overall workload is big, this system could expand conveniently, and the administrator could improve the dealing ability of the system through adding more nodes.

Chapter 3

System Design and Implementation

3.1 System Architecture

The main goal is to build an HBase cluster system to store the data which have converted from Excel document file. The sub-system focuses on HBase database services to offer a put/get database service. The second is to offer sample GUI interface to access HBase for general use. In this chapter, we will overview the whole system.

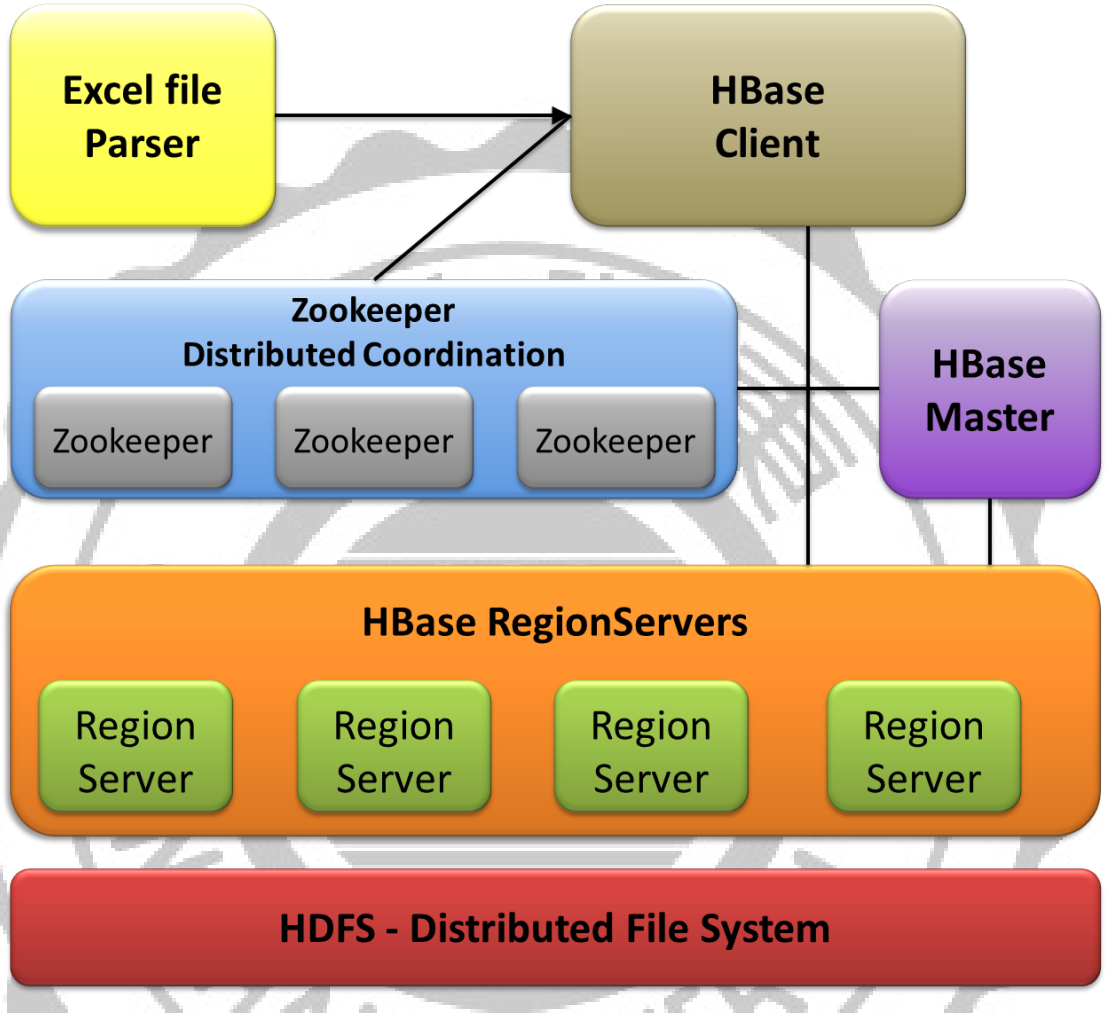

FIGURE 3.1: Structure of the system

The main system shown in Figure 3.1 depicts the structure of the system. The system has ZooKeeper cluster, which provides a coordination service for the entire HBase cluster, and handling master selection, node registration. The Excel file parser can pares text in Excel; it can read data from Excel document file worksheets and convent to string, and put strings into HBase through the Hbase client. The Hbase client provides APIs to access the HBase cluster, and the client communicates with the HBase master to lookup which region server should access to. The Region lookups can find which region server holds a specific row key range by two system tables: ROOT- table and .META. table supported. The -ROOT- table is used to refer to regions in the .META.table, while the .META.table holds references to all user regions.

There are several components in the system:

- Excel file Parser: To Read data from Excel document file and convent them to string.
- HBase Master: Its job includes load balancing, region allocation, failover, and log splitting.
- HBase Client: To communicate with the Zookeeper to lookup which region server should access and put/get/scan to HBase.
- Region server: To hold the actual regions and handle I/O requests, flush the in-memory data store to HDFS, and split or compact regions.
- HDFS: To place data files fromHBase stores and write ahead logs (WAL).

The system of this thesis uses cloudera CDH to build HBase cluster system. HBase cluster plays the role of the database to store medical care information, Figure 3.2 shows two CDH clusters.

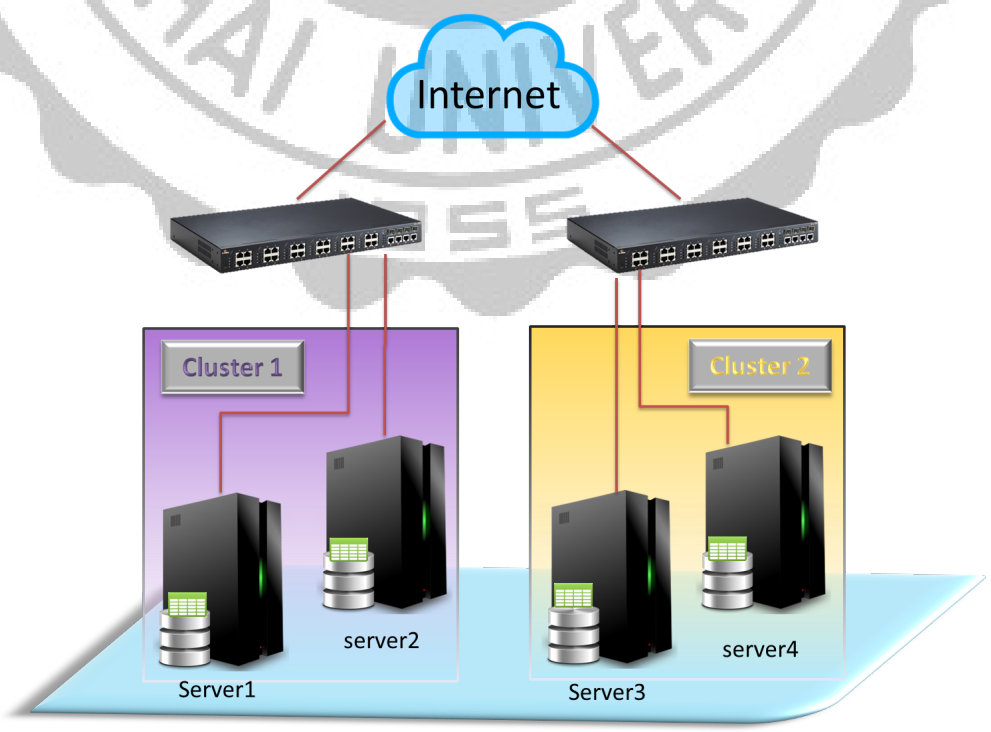

FIGURE 3.2: HBase clusters

CDH cluster combined Hadoop distributions. 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.

Hadoop included hadoop distributed file system (HDFS) and MapReduce. The MapReduce is a programming model for processing large data sets with a parallel. We also implement Zookeeper for maintaining configuration information, naming, providing distributed synchronization, and providing group services on hadoop, as the figure shows the relation of hadoop software.

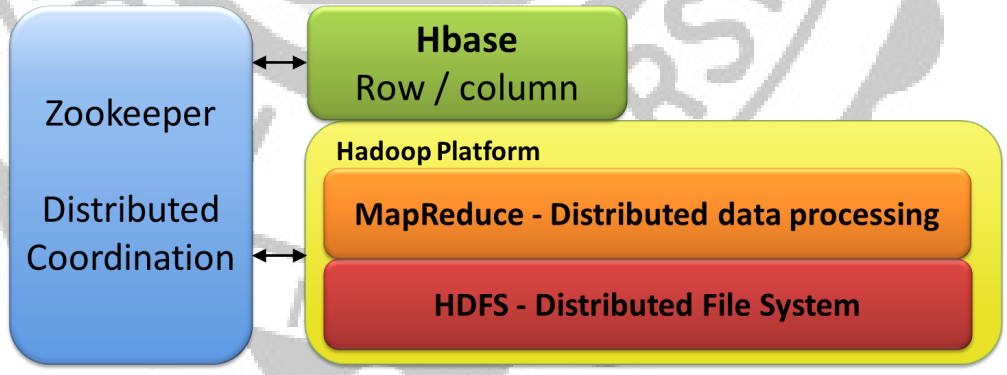

FIGURE 3.3: Hadoop software relation

The main components include:

- Hadoop. Java software framework to support data-intensive distributed applications.
- ZooKeeper. A highly reliable distributed coordination system.
- MapReduce. A flexible parallel data processing framework for large data sets.

- HDFS. Hadoop Distributed File System.
- HBase. Key-value database.

The flowchart in Figure 3.4 describes operations of the data written into hbase.

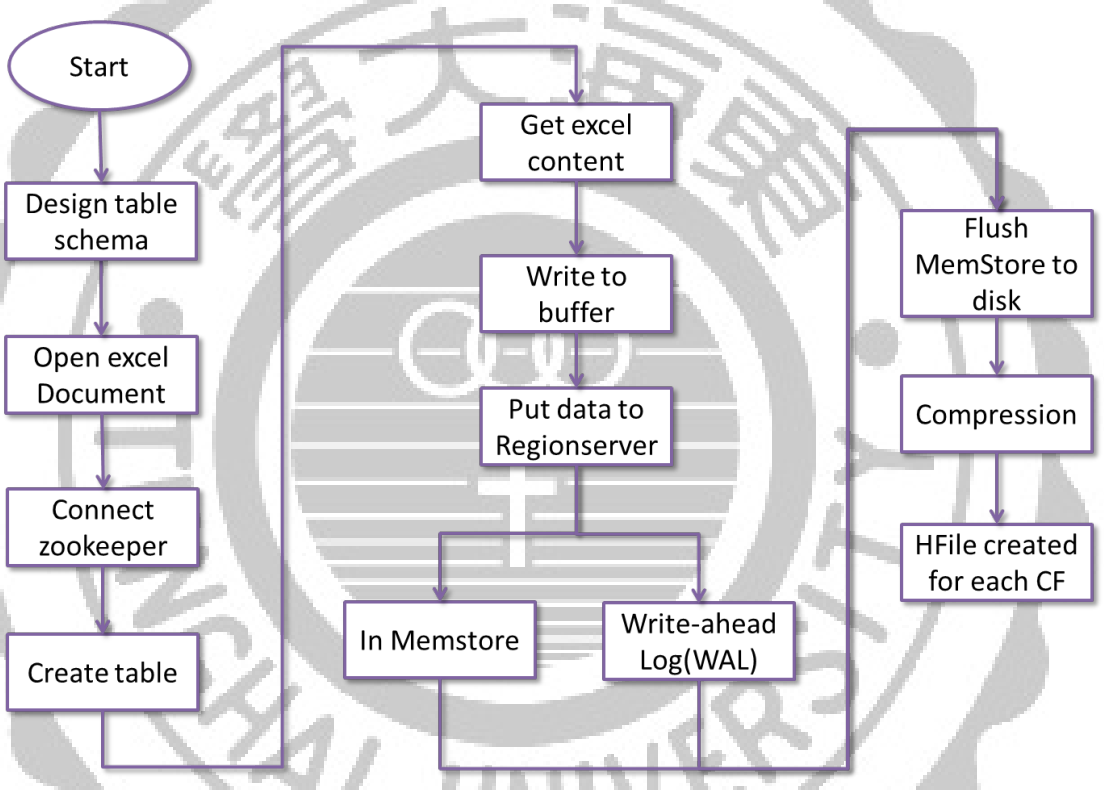

FIGURE 3.4: Flowchart of data write to HBase

The sequence diagrams in Figure 3.5 describes how the main components of the system interact to fulfill the goal of writing data to HBase.

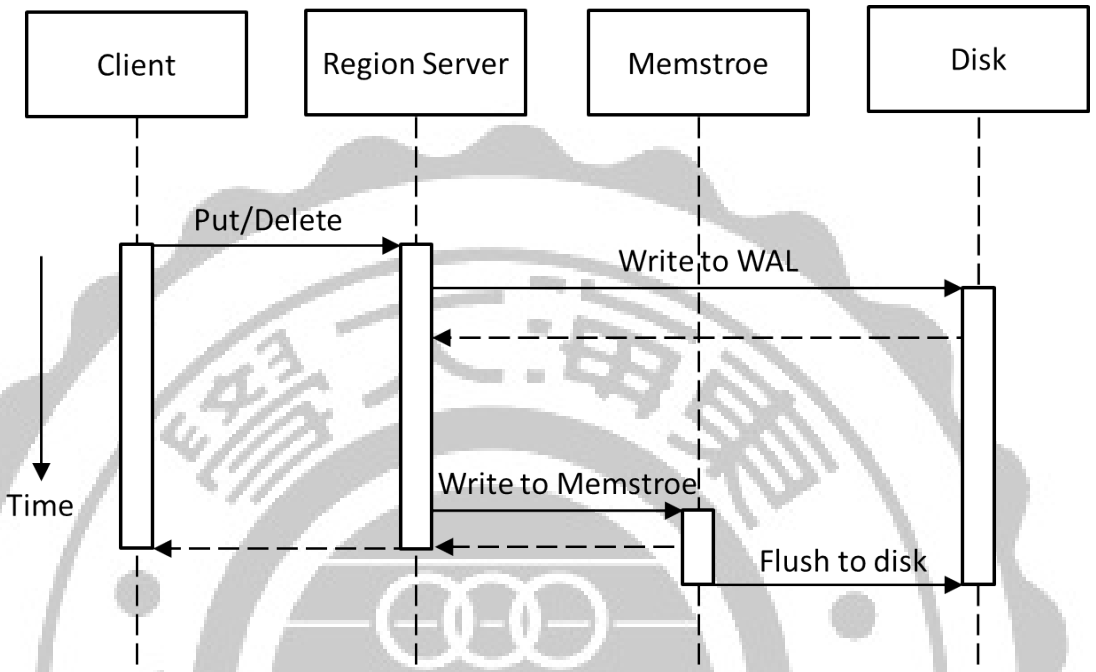

FIGURE 3.5: Sequence diagrams of data write to HBase

3.2 System Setup

3.2.1 HBase Testbed

This section shows our real HBase Testbed. We have three computers and one Gigabit switch. All computers have installed CentOS 6.3 x86_64 version, and already changed firewall to stop status and SELinux to disabled status. It is developed to support Cloudera's distribution including Apache Hadoop. Computer master and 1, 2 act as hosts of the HBase cluster.

3.2.2 CDH installation

First, we should configure /etc/hosts file to know the mapping of some hostnames to IP addresses, then we install OpenSSH: a SSH connectivity tools, since the CDH needs to transfer files between hosts on a network by using the scp command.

```
172.24.12.69    cdh3    regionserver1
172.24.12.66    cdh2    regionserver2
172.24.12.65    cdh1    HMaster
```

FIGURE 3.6: /etc/hosts file mapping hostname to IP address

After we setting IP and SSH service to the computers, we can download the package of CDH to install the Cloudera Manager as shown in Figure 3.7, to control over every part of CDH.

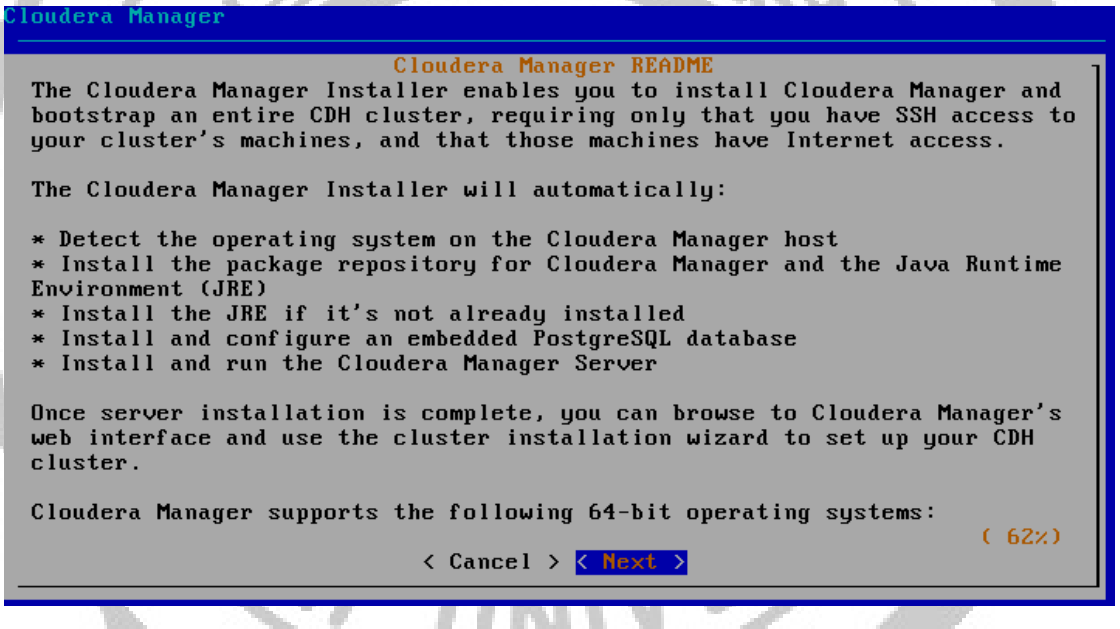

FIGURE 3.7: Install Cloudera Manager

After finishing installing Cloudera Manager Server, we can log into the Cloudera Manager web console on default port 7180. The URL will be like this: http(s)://<Server host>:<port>; then start installing the CDH cluster, shown in Figure 3.8.

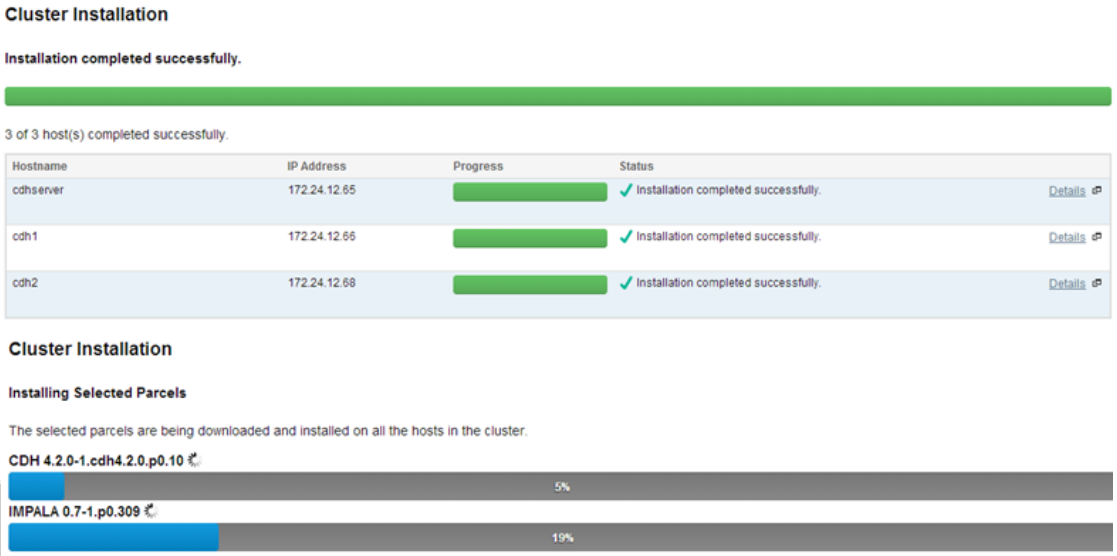

FIGURE 3.8: Cloudera Manager install CDH cluster

The Cloudera Manager web page also provides user to view and modify configuration of Hadoop services. Go to the monitor tab and click status option. Figure **??** shows hosts physical attributes status like memory, disk, CPU, and all service status in the cluster, as shown as Figure 3.10.

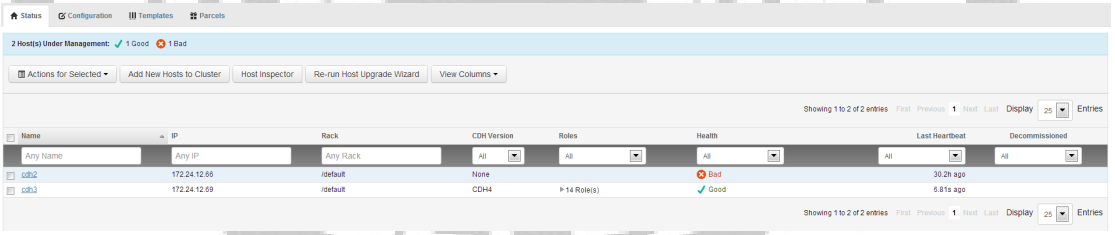

FIGURE 3.9: Cloudera Manager host monitor pag

Processes

Service	Instance	Name	Links	Status	PID	Uptime	Full log file	Stderr	Stdout
None	None	deploy-client-config		Exited		0.00s		Full stderr log	Full stdout log
None	None	host-inspector		Exited		1.00s		Full stderr log	Full stdout log
hbase1	master (cdh1)	hbase-MASTER	HBase Web UI	Running	5776	9.3m	Full log file	Full stderr log	Full stdout log
hbase1	regionserver (cdh1)	hbase-REGIONSERVER	HBase RegionServer Web UI	Running	5727	9.3m	Full log file	Full stderr log	Full stdout log
hdfs1	datanode (cdh1)	hdfs-DATANODE	DataNode Web UI	Running	5231	9.9m	Full log file	Full stderr log	Full stdout log
hdfs1	namenode (cdh1)	hdfs-NAMENODE-createdir		Exited		5.00s	Full log file	Full stderr log	Full stdout log
hdfs1	namenode (cdh1)	hdfs-NAMENODE-createtmp		Exited		5.00s	Full log file	Full stderr log	Full stdout log
hdfs1	namenode (cdh1)	hdfs-NAMENODE	NameNode Web UI	Running	5185	9.9m	Full log file	Full stderr log	Full stdout log
hdfs1	secondarynamenode (cdh1)	hdfs-SECONDARYNAMENODE	SecondaryNameNode Web UI	Running	5135	9.9m	Full log file	Full stderr log	Full stdout log
hive1	hivemetastore (cdh1)	hive-HIVEMETASTORE		Running	6544	8.2m	Full log file	Full stderr log	Full stdout log
hue1	beeswax_server (cdh1)	hue-BEESWAX_SERVER		Running	7389	6.9m	Full log file	Full stderr log	Full stdout log
hue1	hue_server (cdh1)	hue-HUE_SERVER	Hue Web UI	Running	7363	6.9m	Full log file	Full stderr log	Full stdout log
mapreduce1	jobtracker (cdh1)	mapreduce-JOBTRACKER	JobTracker Web UI fairscheduler	Running	6166	9.0m	Full log file	Full stderr log	Full stdout log
mapreduce1	tasktracker (cdh1)	mapreduce-TASKTRACKER	TaskTracker Web UI	Running	6127	9.0m	Full log file	Full stderr log	Full stdout log
oozie1	oozie_server (cdh1)	oozie-OOZIE_SERVER	Oozie Web UI	Running	7311	7.3m	Full log file	Full stderr log	Full stdout log
zookeeper1	server (cdh1)	zookeeper-server		Running	5044	10.3m	Full log file	Full stderr log	Full stdout log

FIGURE 3.10: Service status in CDH cluster

Open HBase web UI by using browser to access master server at port 60000, as shown in Figure 3.11. Figure 3.12 shows RegionServer attributes and status at master host port 60020 web page.

Master: cdh1:60000

APACHE HBASE

Local logs, Thread Dump, Log Level, Debug dump, HBase Configuration

Attributes

Attribute Name	Value	Description
HBase Version	0.94.6-cdh4.3.0, rUnknown	HBase version and revision
HBase Compiled	Mon May 27 20:22:05 PDT 2013, jenkins	When HBase version was compiled and by whom
Hadoop Version	2.0.0-cdh4.3.0, r48a9315b342ca16de92fcc5be95ae3650629155a	Hadoop version and revision
Hadoop Compiled	Mon May 27 19:45:25 PDT 2013, jenkins	When Hadoop version was compiled and by whom
HBase Root Directory	hdfs://cdh1:8020/hbase	Location of HBase home directory
HBase Cluster ID	25204176-de79-4a94-9ea1-50933c598329	Unique identifier generated for each HBase cluster
Load average	3.00	Average number of regions per regionserver. Naive computation.
Zookeeper Quorum	cdh1:2181	Addresses of all registered ZK servers. For more, see zk dump.
Coprocessors	[]	Coprocessors currently loaded loaded by the master
HMaster Start Time	Wed Jul 17 13:27:06 CST 2013	Date stamp of when this HMaster was started
HMaster Active Time	Wed Jul 17 13:27:06 CST 2013	Date stamp of when this HMaster became active

FIGURE 3.11: HBase master status

RegionServer: cdh1,60020,1374038824554

Local logs, Thread Dump, Log Level, Debug dump, HBase Configuration

Attributes

Attribute Name	Value	Description
HBase Version	0.94.6-cdh4.3.0, rUnknown	HBase version and revision
HBase Compiled	Mon May 27 20:22:05 PDT 2013, jenkins	When HBase version was compiled and by whom
Metrics	requestsPerSecond=0, numberOfOnlineRegions=3, numberOfStores=6, numberOfStorefiles=7, storefileIndexSizeMB=0, rootIndexSizeKB=65, totalStaticIndexSizeKB=34, totalStaticBloomSizeKB=0, memstoreSizeMB=0, mbInMemoryWithoutWAL=0, numberOfPutsWithoutWAL=0, readRequestsCount=748, writeRequestsCount=2, compactionQueueSize=0, flushQueueSize=0, usedHeapMB=42, maxHeapMB=125, blockCacheSizeMB=0.26, blockCacheFreeMB=31.22, blockCacheCount=4, blockCacheHitCount=1494, blockCacheMissCount=4, blockCacheEvictedCount=0, blockCacheHitRatio=99%, blockCacheHitCachingRatio=99%, hdfsBlocksLocalityIndex=100, slowHLogAppendCount=0, fsReadLatencyHistogramMean=728582.00, fsReadLatencyHistogramCount=2.00, fsReadLatencyHistogramMedian=728582.00, fsReadLatencyHistogram75th=807932.00, fsReadLatencyHistogram95th=807932.00, fsReadLatencyHistogram99th=807932.00, fsReadLatencyHistogram999th=807932.00, fsPreadLatencyHistogramMean=1919567.00, fsPreadLatencyHistogramCount=2.00, fsPreadLatencyHistogramMedian=1919567.00, fsPreadLatencyHistogram75th=2744544.00, fsPreadLatencyHistogram95th=2744544.00, fsPreadLatencyHistogram99th=2744544.00, fsPreadLatencyHistogram999th=2744544.00, fsWriteLatencyHistogramMean=123771.50, fsWriteLatencyHistogramCount=2.00, fsWriteLatencyHistogramMedian=123771.50, fsWriteLatencyHistogram75th=212984.00, fsWriteLatencyHistogram95th=212984.00, fsWriteLatencyHistogram99th=212984.00, fsWriteLatencyHistogram999th=212984.00	RegionServer Metrics; file and heap sizes are in megabytes

FIGURE 3.12: RegionServer attributes and status

Chapter 4

Experimental Results

4.1 Experimental Environment

This section presents several experiments conducted on one physical machine and one virtual machine. Each nodes contained 1-GE NICs, but had different CPU and memory levels, as shown in Table 2. We used Linux command "dd" to test the disk write performance for each node, and used a network testing tool iperf to measure the throughput of a network by TCP data streams.

Table 4.1: Hardware Specification

Node	CPU	RAM	Disk speed	Network speed	OS version	Java version
Node 1	Intel(R) Core(TM)2 Quad CPU Q9550	4GB	352 MB/s	96.5 Mbits/sec	CentOS x86-64	jre 1.6.0-31 -b04
Node 2	Virtualized 2 cores from Intel i7-2600	1GB	412 MB/s	94 Mbits/sec	CentOS x86-64	jre 1.6.0-31 -b04

In the experiment, we used HBase-0.94.2 API and .hadoop-client-1.0.3 API. We also used JAVA programming language to build up a client. Table 4.2 shows the software specification and setting arguments.

TABLE 4.2: Software specification and setting arguments

	Version	Argument/Option
HBase	0.94.2	Master at Node1:60010
Hadoop	2.0.0	
MapReduce 2	2.0.0	map.tasks.maximum=4 reduce.tasks.maximum=2
HDFS	2.0.0	Block size=64MB Replic=3
Zookeeper	3.4.5	Quorum at port 218

The experimental platform is built on two nodes. Node 1 acts as HMaster. It has 1 Intel Core(TM), 2 Quad Q9550 CPU (12M Cache, 2.83 GHz), 4 GB memory, and 1TB disk. And Node 2 acts as RegionServer. It has 2 Core CPU Virtualized from Intel i7-2600 and 1 GB memory. Since the disk I/O throughput is important for database system and the disk speed of Node 2 is faster than that of Node 1, we used Node 2 to act as RegionServer.

Our experimental data consist of basic information of patients over the age of 65, with 260398 records in total. Four datasets were built, the first three datasets contained 65535 records, and the last dataset contained 63793 records. First, we needed to create HBase schemas by designing Rowkey, ColumnFamily, and qualifier of column. The Rowkey length was kept as short as reasonable such that it can still be useful to access required data. In fact, we should expect tradeoffs when designing Rowkeys: a short key design that is useless for data access is not more valuable than a longer key with better get/scan properties. Table 4.3 shows the schema of HBase used to store patients records.

TABLE 4.3: Schema of HBase which stores patients records

Rowkey	Name	Birth	Address	Sex
	Chinese	Type / Day	Home	Sex
Patients ID				

To convert experimental data from Excel document file to HBase column-oriented table, we design a table schema as shown in previous Table 4.3, but in HBase, the format of data store actually looks like JSON format shown below.

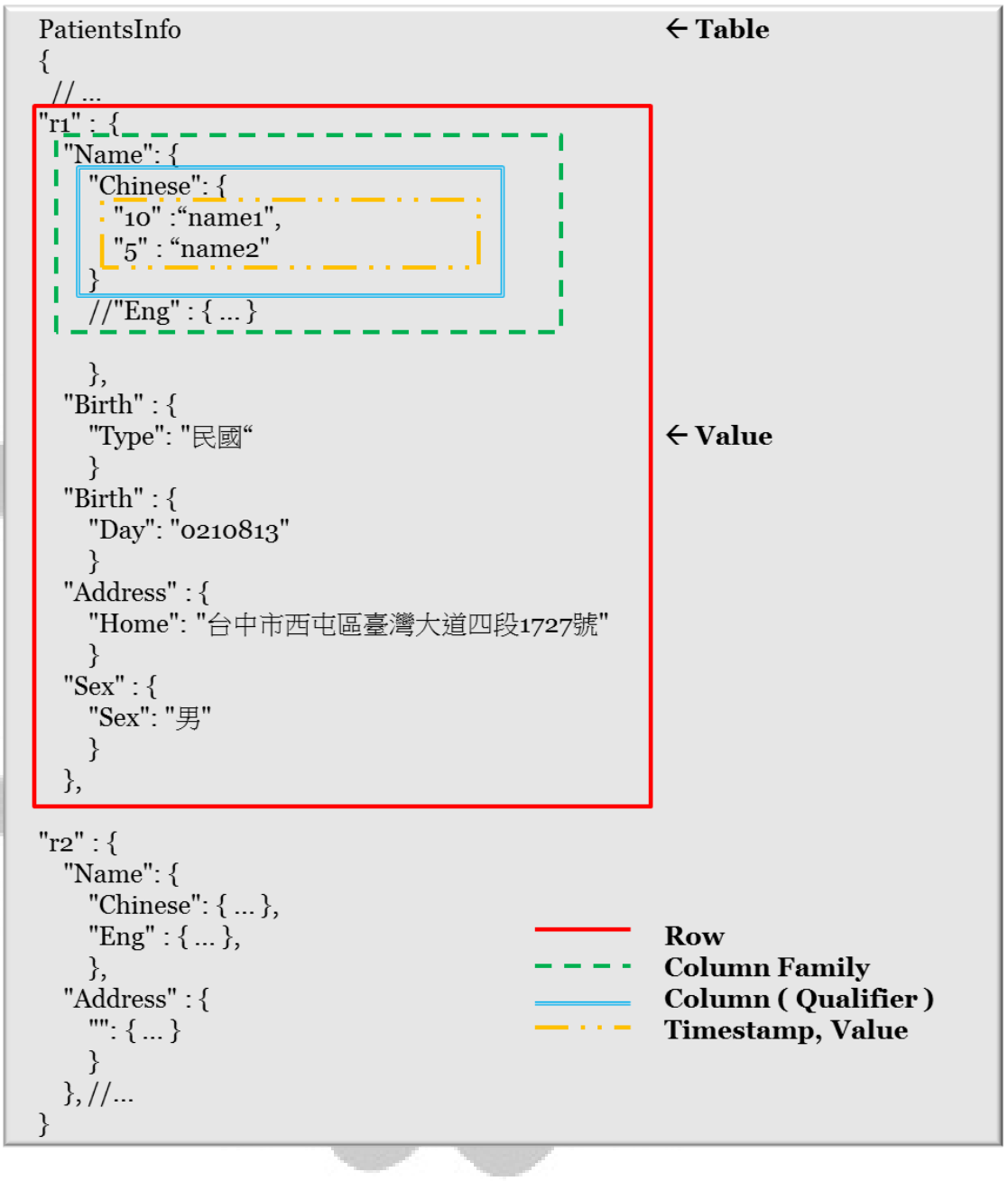

FIGURE 4.1: The data structure of experimental data in HBase

4.2 Experimental Results and Discussion

In the experiments, hbase.hregion.max.filesize is set as 1073741824 (1GB) As shown in Figure 4.2, in which 260398 records were stored in the 4 datasets. Actually all records were put into the table "PatientsInfo" in HBase by generating

monotonically increasing Rowkeys.

FIGURE 4.2: Results of scan the PatientsInfo table

After we put all data into the table "PatientsInfo" of hbase cluster from excel files, we can access RegionServer Web interface to check the table information. Figure 4.3 shows the region details on the RegionServer web GUI interface.

Region Name
PatientsInfo,,1367551721407.af39abae06b7b76deaef702941f5202d.
Metrics
numberOfStores=4, numberOfStorefiles=3, storefileUncompressedSizeMB=53, storefileSizeMB=53, compressionRatio=1.0000, memstoreSizeMB=55, storefileIndexSizeMB=0, readRequestsCount=30199, writeRequestsCount=1279362, rootIndexSizeKB=65, totalStaticIndexSizeKB=34, totalStaticBloomSizeKB=0, totalCompactingKVs=756326, currentCompactedKVs=756326, compactionProgressPct=1.0, coprocessors=[]

FIGURE 4.3: The region detail on RegionServer

- NumberOfStores: This is the number of Stores targeted for compaction in the RegionServer.
- NumberOfStorefiles: Number of StoreFiles opened on the RegionServer. A store may have more than one StoreFile (HFile).

4.2.1 Optimization of HBase properties

We evaluate the cost of putting data into HBase in different configurations on HBase. Table 4.4 shows that the time cost of putting 260398 records into HBase when setting three properties with different value of HBase. Following items describe these three properties.

- setAutoFlush()

  Normally, the puts will be sent one at a time to the RegionServer. If autoFlush set to false, these messages are not sent until the write-buffer is full.

- setWriteToWAL()

  Turning writeToWAL off means that the RegionServer will not write the Put to the Write Ahead Log(WAL), only into the memstore; however the consequence is that if there is a RegionServer failure, there will be data loss.

- setWriteBufferSize(10MB)

  Write buffer size in bytes. A larger buffer requires more memory on both the client and the server because the server instantiates the past write buffer to process it but reduces the number of remote procedure calls (RPC).

TABLE 4.4: Configuring different properties of HBase

HBase Properties/Configuration	A	B	C	D	E
setAutoFlush()	on	on	on	off	off
setWriteBufferSize(10MB)	off	on	on	on	on
setWriteToWAL()	on	on	off	on	off
TIME	950sec	924sec	652sec	63sec	48sec

In the experiments, we convert experimental data from Excel document file to HBase with no limit on the number of times, and no high fault tolerance considerations; the purpose is to find the fastest way. Since errors in the conversion process are few, we choose E configuration that can significantly reduce the time consumed. Finally, we choose E configuration as the optimal setting. If we need to consider a higher stability, or with fault tolerance mechanisms, then need to choose setAWriteToWAL state is ON setting in the different contexts, such as configuration A, B, D, but it has relative increase in time cost. When we configure another property hbase.regionserver.handler.count to 20 on RegionServer, the property setup number of RPC server instances spun up on RegionServer. At configuration A, the time cost reduced 35 seconds, as shown in Table 4.5.

TABLE 4.5: Set HBase RegionServer Handler Count value to 20

HBase Properties/Configuration	A	B	C	D	E
setAutoFlush()	on	on	on	off	off
setWriteBufferSize(10MB)	off	on	on	on	on
setWriteToWAL()	on	on	off	on	off
TIME	915sec	897sec	620sec	63sec	43sec

The following Figure 4.4 to 4.7 collection shows the information when using configuration A to put data into HBASE, including the write requests, memory usage, flush size information. In configuration A, as turning on the setAutoFlush, we can see the figure 4.7 shows memstore in Regionserver be flushed five times during the put status.

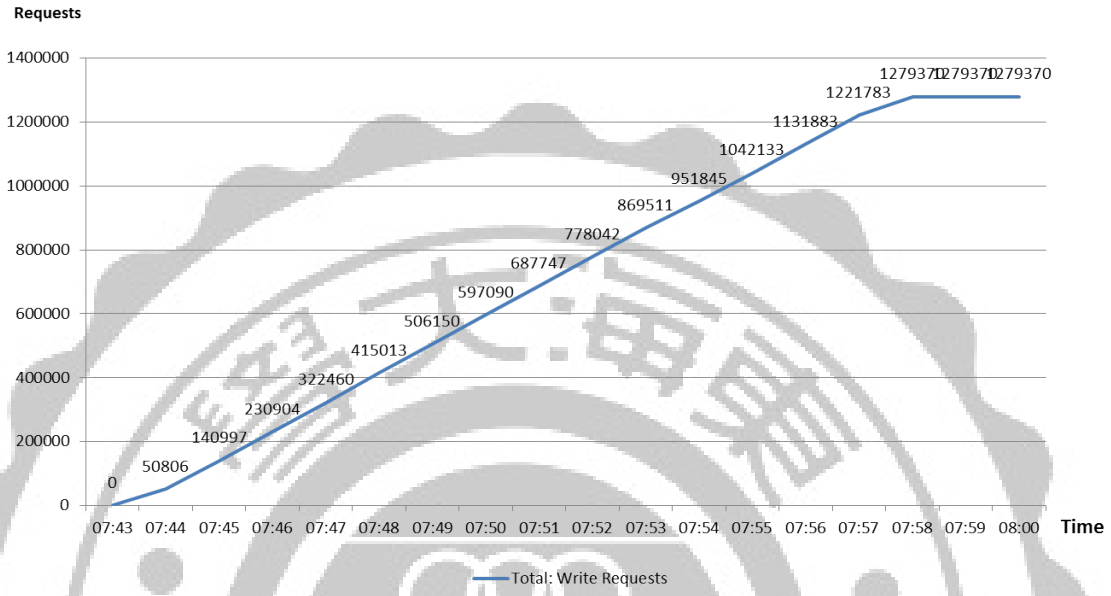

FIGURE 4.4: Write requests on configuration A

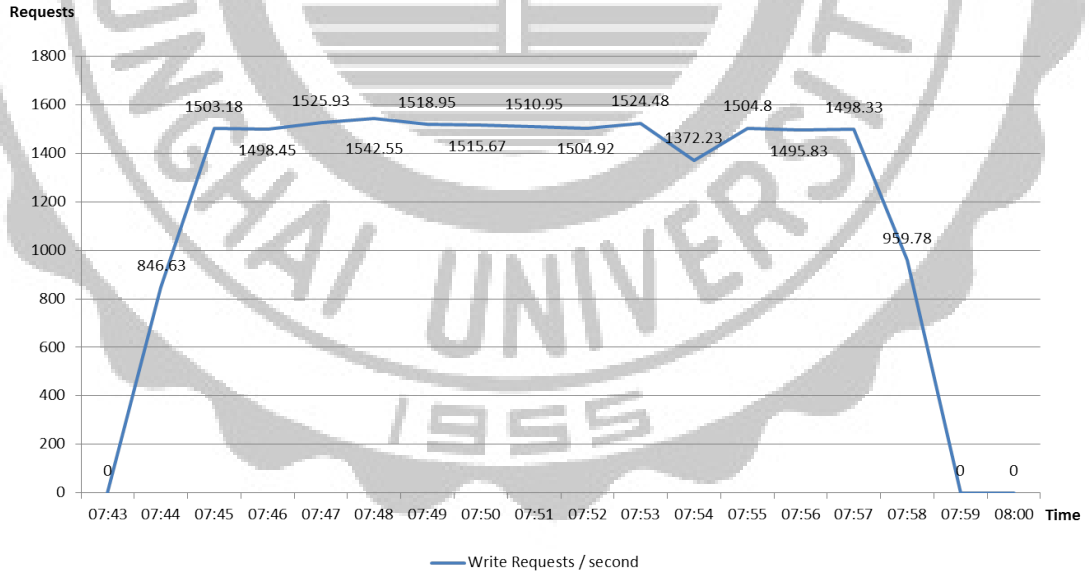

FIGURE 4.5: Write requests per sec on configuration A

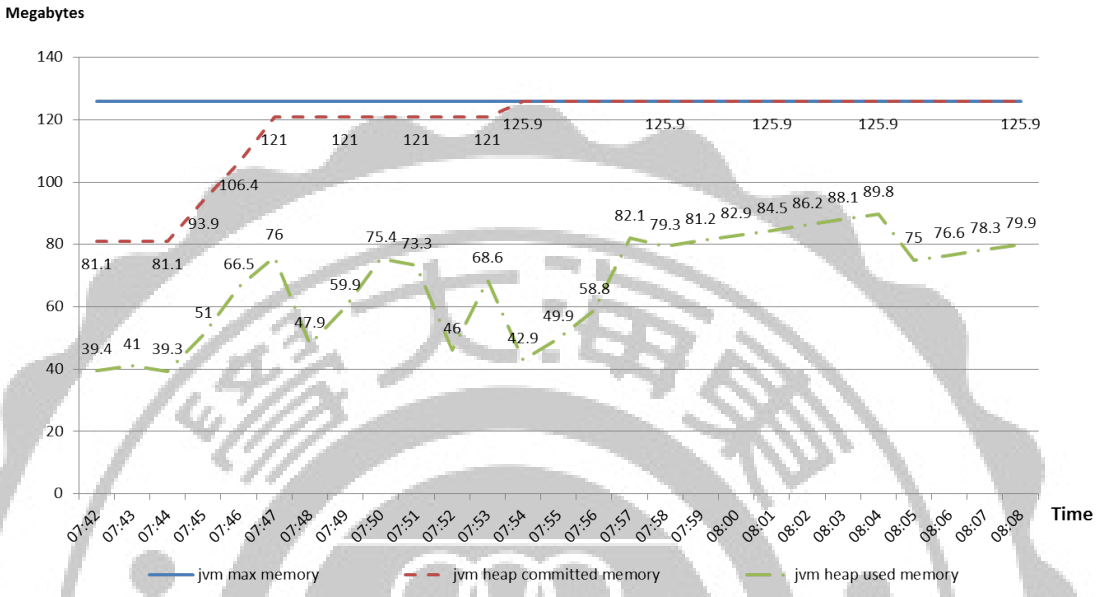

FIGURE 4.6: Memory usage on configuration A

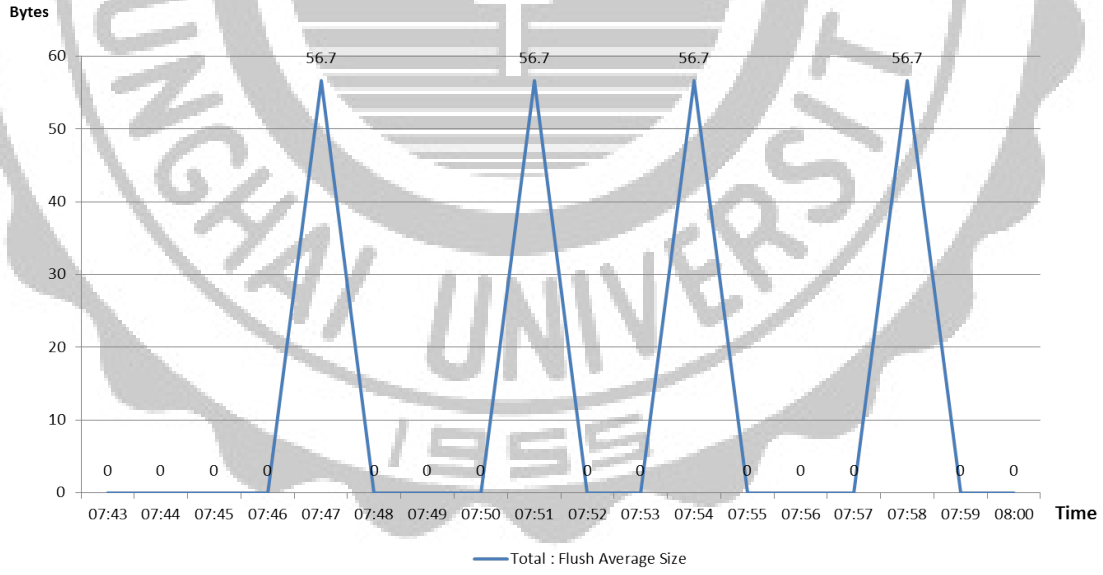

FIGURE 4.7: Flush average size and operations rate on configuration A

The following Figure 4.8 to 4.13 collection shows the information when using configuration E to put data into HBASE. In configuration E, as turning off the setAutoFlush, we can see the figure 4.11 shows memstore in Regionserver be flushed only one time during the put status. And when we turn off the setWriteToWAL, the RegionServer will not write the Put to the Write Ahead Log; so we can see Figure 4.12 recorded 29790 operations without WAL.

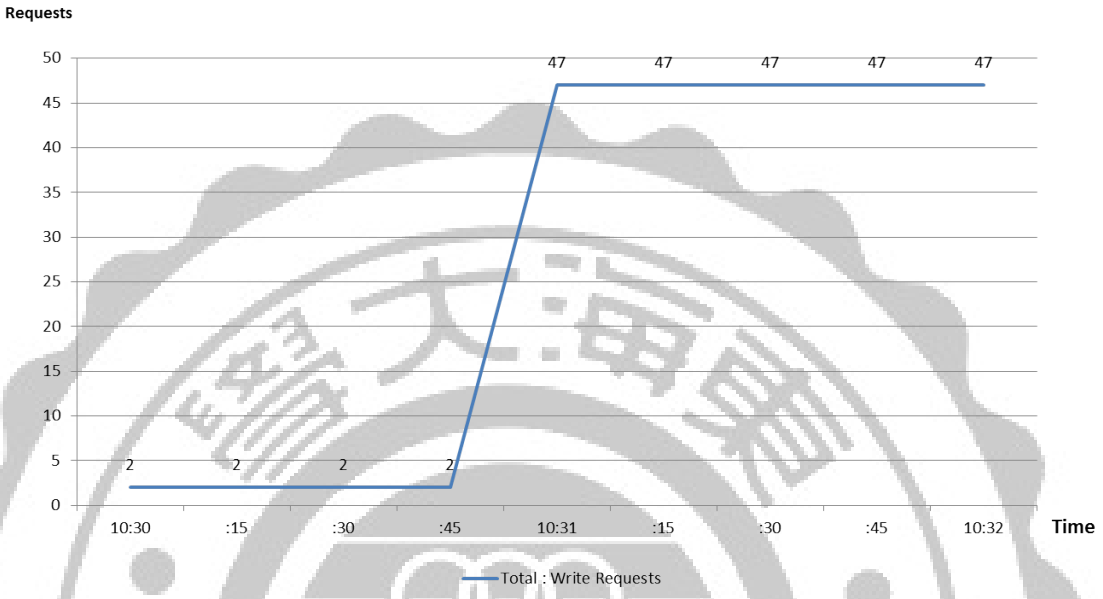

FIGURE 4.8: Write requests on configuration E

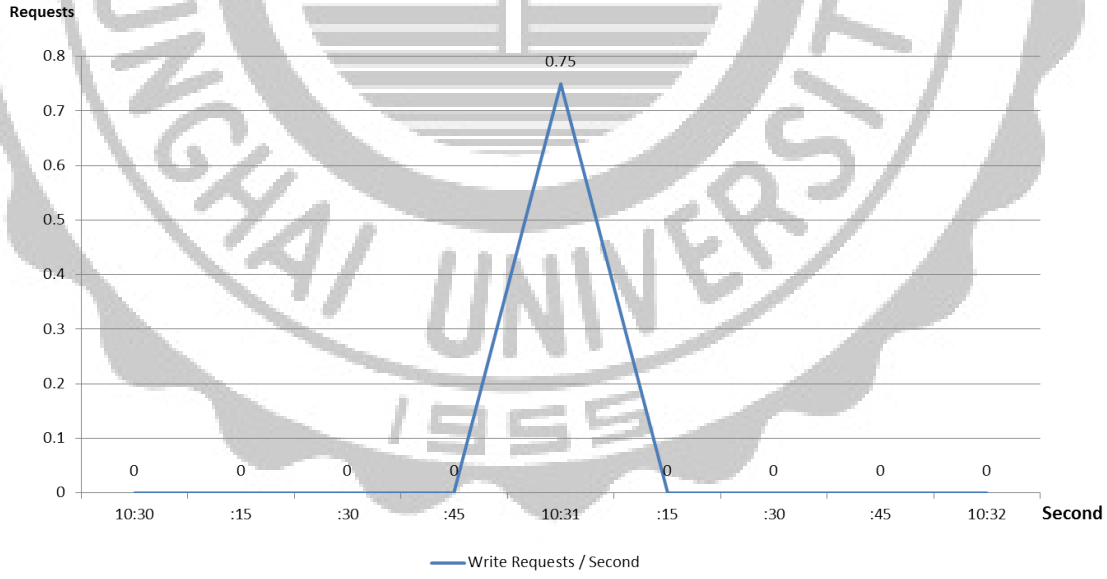

FIGURE 4.9: Write requests per sec on configuration E

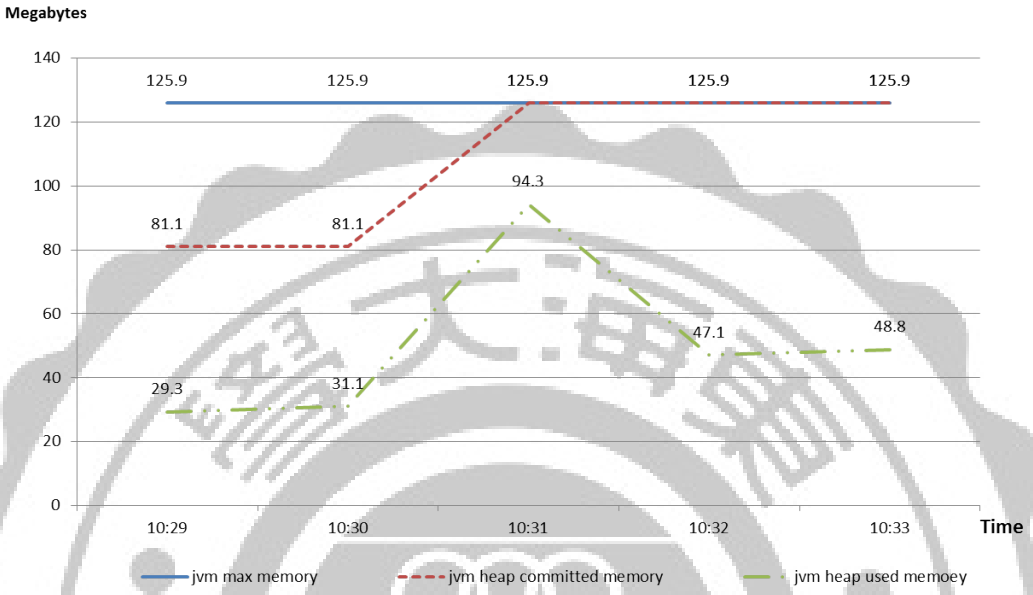

FIGURE 4.10: Memory usage on configuration E

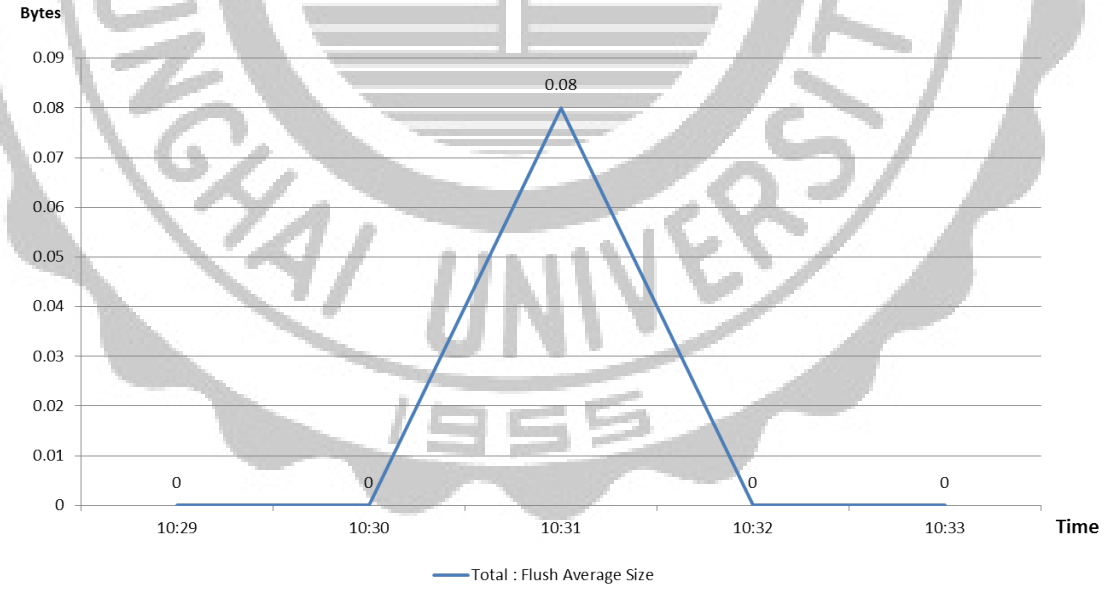

FIGURE 4.11: Flush average Size and operations rate on configuration E

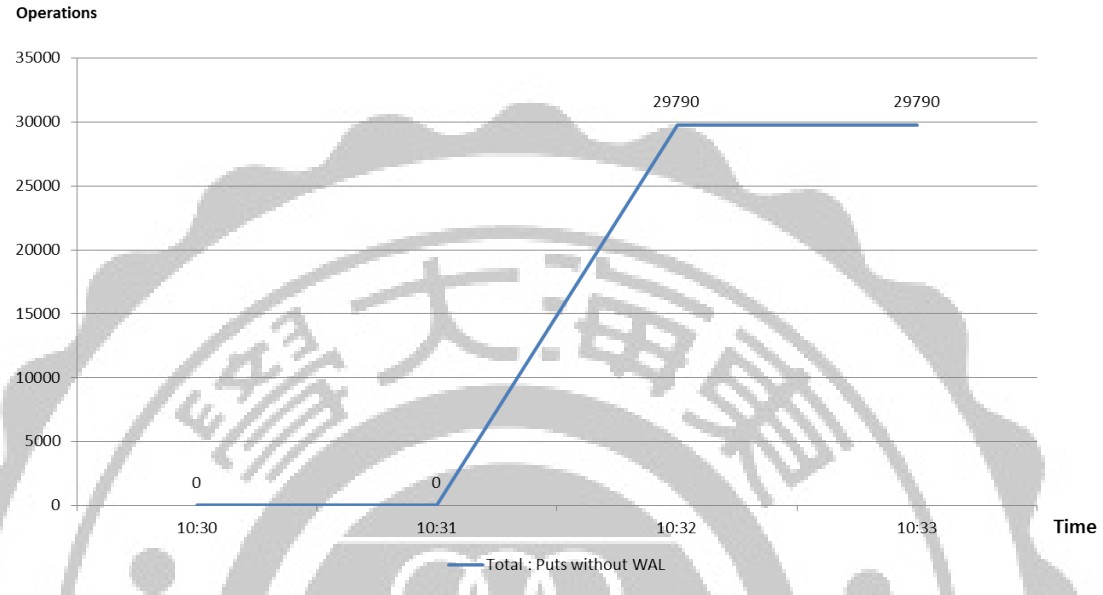

FIGURE 4.12: Total: Puts without WAL on configuration E

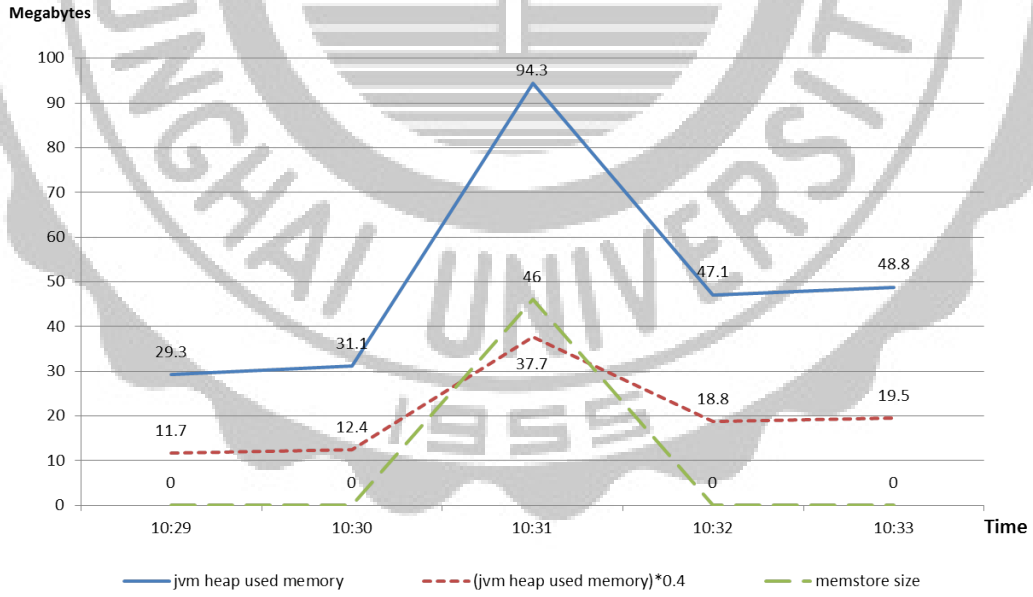

FIGURE 4.13: Memory heap and Memstore size E

4.2.2 Optimization of memory usage

In the Figure 4.10 and 4.6, we found the java process heap committed memory always very close to the maximum memory size for the java process heap even fill all. So we believe that java heap committed size should be required far more

than 125.9MB. Thus, we increase maximum size for the java Process heap to 1GB by java -Xmx command, and as the following Figure 4.14 shows the java heap committed the memory size to 330.2MB and the truly used size reached around 209.9MB.

At the same time, we also modify two properties on hbase.regionserver.global.memstore.upper and hbase.regionserver.global.memstore.lowerLimit in Regionserver to increase memory utilization efficiency. Because the default parameters in HBase are not fully used memory, and without interfering with other used heap size parameters (eg hfile.block.cache.size) situation,
hbase.regionserver.global.memstore.upperLimit upgrade 0.1, also is to increase the heap size 10% occupied space, and will not affect the overall environment. Following two items describe these two properties.

- hbase.regionserver.global.memstore.upperLimit

  This parameter is the order to limit memstores occupied the total memory. We set the value from 0.4 to 0.5, which means when 50% of the heap of the sum of the memory occupied by the all ReigonServer memstore, HBase forced to block all updates and flush of these memstore to release the memory occupied by all memstore.

- hbase.regionserver.global.memstore.lowerLimit

  We set the value from 0.35 to 0.45. With the UpperLimit of only 45% global memstore memory, it does not flush all memstore, and it will find some of the memory footprint of larger memstore; the individual flush, of course, is updated or blocked.

As the memstore size at configuration E reached 46MB over than hbase.regionserver.global.memstore.lowerLimit parameter set at 0.4 times the memory size of heap used memory 37.7MB. The memstore is soon flushed into HFile shown in Figure 4.13.

After the above settings, the hbase.regionserver.global.memstore.upperLimit is set to 0.5; the memstore size is increased from 46MB to 105.45MB shown in Figure 4.15, and the total data put time reduced to 35sec shown in Figure 4.16.

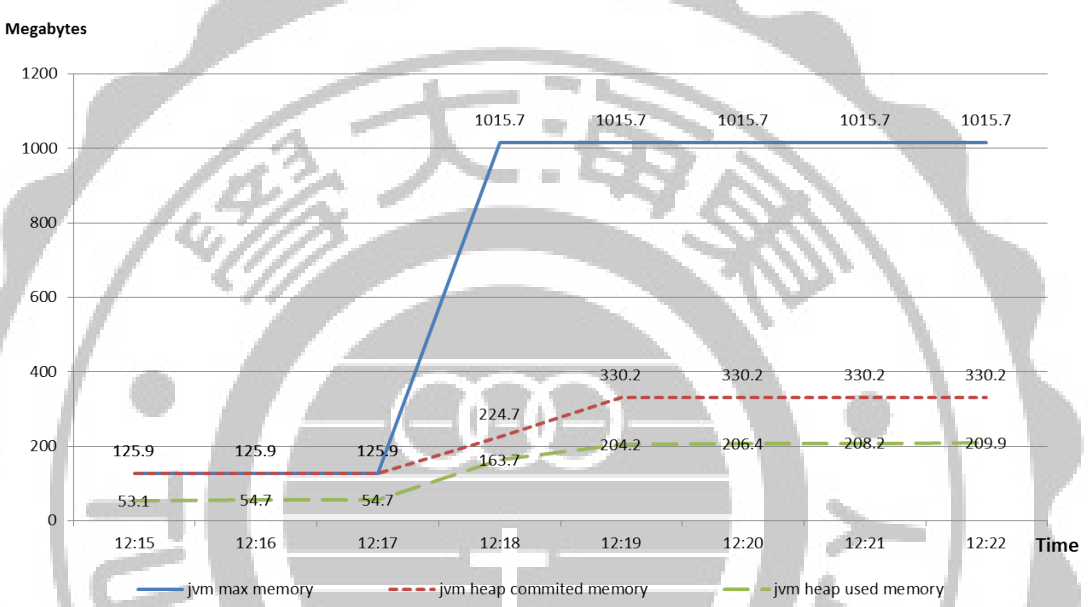

FIGURE 4.14: Memory usage with increased maximum size

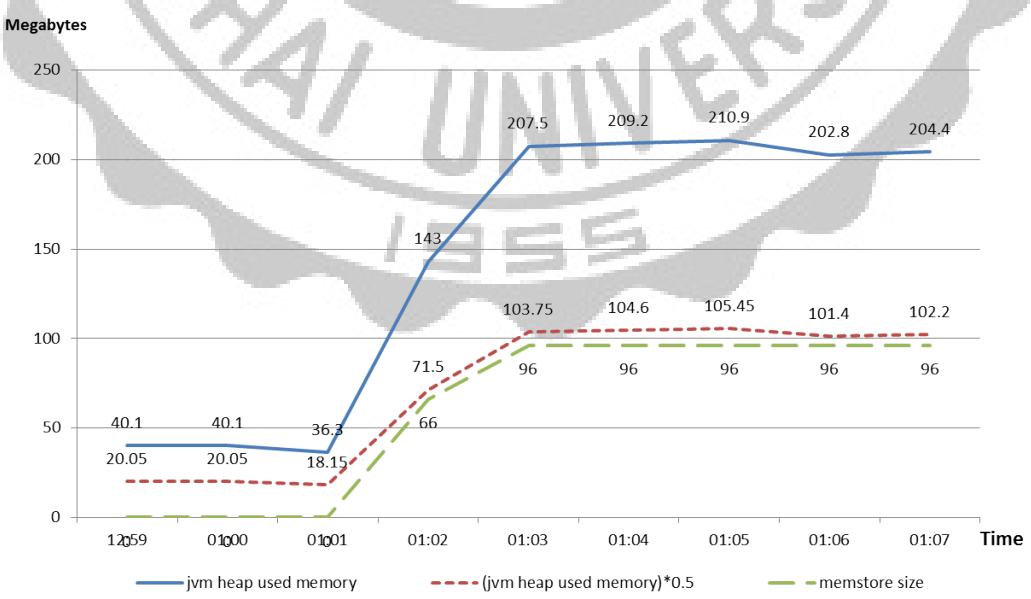

FIGURE 4.15: Memory heap and Memstore size with increased upperLimit

```
六月 26, 2013 11:27:25 上午 org.apache.zookeeper.ClientCnxn$SendThread primeConnection
資訊: Socket connection established to odh1/172.24.12.65:2181, initiating session
六月 26, 2013 11:27:26 上午 org.apache.zookeeper.ClientCnxn$SendThread onConnected
資訊: Session establishment complete on server odh1/172.24.12.65:2181, sessionid = 0x13f7908f6da0c75, negotiated timeout = 60000
cost time:35sec
BUILD SUCCESSFUL (total time: 37 seconds)
```

FIGURE 4.16: Put time optimization

4.2.3 Discussion

Above the experimental, we realize that the RowKey design has some problems, and there are two problems that we should take care of:

- Rowkey Length

  We tried to minimize the row and column sizes. The KeyValue class is the heart of data storage in HBase. When we design Row Key, ColumnFamily, and column in HBase, the names must be as short as possible, because all of them are embedded within the KeyValue instance. The longer these identifiers are, the bigger the KeyValue is. As the Table 4.6 shows the KeyValue of data storage in HBase [20].

TABLE 4.6: KeyValue of data storage in HBase.

Rowkey=r1, cf:attr1= v1
rowlength ———— 2
row ———————— r1
columnfamilylength — 2
columnfamily ——— cf
columnqualifier —— attr1
timestamp ———— server time of Put
keytype ————- Put

- Rowkey relation

  While creating corresponding Rowkeys in HBase, problems appeared when we saved monotonically increasing values in the alphabetical order. The new writes were not evenly distributed [32], and the last of the original region become a new high hit rate region in need of a split. Monotonically increasing keys in a single region can reduce the randomly distributed consumption, but overall, it is best to avoid using the sequence number or timestamp as the row-key.

Chapter 5

Conclusions and Future work

As stated, goal of this thesis is to provide an integrated data parallel processing service environment, to ensure various demands of data services from different medical divisions, and to offer services required for the information combination and computing resources. Therefore, the architecture of the system must be modular to support data services of customized, reusable, and scalable characteristics. Based on the needs of different applications, the cluster resources are timely adjusted to serve demands for data services of each application. Through the automatic information integration mechanism, not only data consistency and integrity requirements can be met in different data services applications, but also a unified data access information system can be built to ensure the individual as well as overall service needs.

In the future, we plan to use virtualization technology to construct a dynamic increase or decrease RegionServer in the HBase cluster for detection of load written by RegionServer, and evaluation of the total load of the physical machine; and then move data from regions in high access rates during off-peak hours to achieve load balancing in the RegionServers. On the other hand HBase can effectively store large number of sensor data like environmental monitoring data. Because the amount of data collected in a day is greater than 18,000, it is a very large load for a single stage sensor. We also plan to implement HBase block caches and Bloom filters for real-time queries to provide a more complete cloud service.

Bibliography

[1] A.t. kearney it analysis. http://www.atkearney.com/home.

[2] Cloudera recommendations on hadoop/ hbase cluster capacity planning. http://www.cloudera.com/blog/2010/08/hadoophbase-capacity-planning/.

[3] Hadoop wiki - hbase. http://wiki.apache.org/hadoop/Hbase.

[4] Nchc cloud computing research group. http://trac.nchc.org.tw/cloud.

[5] E. Abstract). Database scalability, elasticity, and autonomy in the cloud. In J. Yu, M. Kim, and R. Unland, editors, *DASFAA 2011*, volume I of *Part*, page 6587. Springer, pp.

[6] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 53–64, New York, NY, USA, 2012. ACM.

[7] P. Atzeni, F. Bugiotti, and L. Rossi. Uniform access to nosql systems. *Information Systems*, (0):–, 2013.

[8] C.-R. Chang, M.-J. Hsieh, J.-J. Wu, P.-Y. Wu, and P. Liu. Hsql: A highly scalable cloud database for multi-user query processing. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 943–944, 2012.

[9] L. Ding, G. Wang, J. Xin, X. Wang, S. Huang, and R. Zhang. Commapreduce: An improvement of mapreduce with lightweight communication mechanisms. *Data & Knowledge Engineering*, (0):–, 2013.

[10] J. Dittrich and J.-A. Quiané-Ruiz. Efficient big data processing in hadoop mapreduce. *Proc. VLDB Endow.*, 5(12):2014–2015, Aug. 2012.

[11] B. Dong, Q. Zheng, F. Tian, K.-M. Chao, R. Ma, and R. Anane. An optimized approach for storing and accessing small files on cloud storage. *Journal of Network and Computer Applications*, 35(6):1847 – 1862, 2012.

[12] A. Floratou, J. M. Patel, E. J. Shekita, and S. Tata. Column-oriented storage techniques for mapreduce. *Proc. VLDB Endow.*, 4(7):419–429, Apr. 2011.

[13] Y. Hu and W. Qu. Efficiently extracting change data from column oriented nosql databases. In J.-S. Pan, C.-N. Yang, and C.-C. Lin, editors, *Advances in Intelligent Systems and Applications - Volume 2*, volume 21 of *Smart Innovation, Systems and Technologies*, pages 587–598. Springer Berlin Heidelberg, 2013.

[14] W. Jiang, H. Li, H. Jin, L. Zhang, and Y. Peng. *VESS: An Unstructured Data-Oriented Storage System for Multi-Disciplined Virtual Experiment Platform*. Springer-Verlag, Jul 2011.

[15] K. K.-Y. Lee, W.-C. Tang, and K.-S. Choi. Alternatives to relational database: Comparison of nosql and {XML} approaches for clinical data storage. *Computer Methods and Programs in Biomedicine*, 110(1):99 – 109, 2013.

[16] Y. Li, G. Kim, L. Wen, and H. Bae. Mhb-tree: A distributed spatial index method for document based nosql database system. In Y.-H. Han, D.-S. Park, W. Jia, and S.-S. Yeo, editors, *Ubiquitous Information Technologies and Applications*, volume 214 of *Lecture Notes in Electrical Engineering*, pages 489–497. Springer Netherlands, 2013.

[17] Y. Luo, S. Luo, J. Guan, and S. Zhou. A {RAMCloud} storage system based on hdfs: Architecture, implementation and evaluation. *Journal of Systems and Software*, 86(3):744 – 750, 2013.

[18] S. Nishimura, S. Das, D. Agrawal, and A. El Abbadi. $\mathcal{MD}$ -hbase: design and implementation of an elastic data infrastructure for cloud-scale location services. *Distributed and Parallel Databases*, 31(2):289–319, Jun 2013.

[19] A. Path and B. Mappings. Data integration over nosql stores using. In A. Hameurlain et al., editors, *DEXA 2011*, volume I of *Part*, page 6860. Springer, pp.

[20] P. Pirzadeh, J. Tatemura, and O. Po. Performance evaluation of range queries in key value stores. *Journal of Grid Computing*, 10(1):109–132, 2012.

[21] Z. QIU, Z. wen LIN, and Y. MA. Research of hadoop-based data flow management system. *The Journal of China Universities of Posts and Telecommunications*, 18, Supplement 2(0):164 – 168, 2011.

[22] K. Slagter, C.-H. Hsu, Y.-C. Chung, and D. Zhang. An improved partitioning mechanism for optimizing massive data analysis using mapreduce. *The Journal of Supercomputing*, pages 1–17, 2013.

[23] J. Sun, Q. Jin, D. of Computer, S. Department, of Computer, and Science. Scalable rdf store based on hbase and mapreduce. In *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*. IEEE, 2010.

[24] R. Taylor. An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. *BMC Bioinformatics*, 11(Suppl 12):1–6, 2010.

[25] Y. University, C. HP, Labs, N. Haven, CT, U. Amsterdam, T. N. P. Alto, CA, and USA. Column-oriented database systems daniel j. abadi peter a. boncz stavros harizopoulos. 2009.

[26] R. F. van der Lans. Chapter 3 - data virtualization server: The building blocks. In *Data Virtualization for Business Intelligence Systems*, pages 59 – 107. Morgan Kaufmann, Boston, 2012.

[27] via an Extended and C. Framework. Enhancing query support in hbase. In W. Abramowicz et al., editors, *ServiceWave 2011*, volume 6994 of *LNCS*, page 75–87. Springer, 2011.

[28] H. Wang, X. Qin, Y. Zhang, S. Wang, and Z. Wang. Lineardb: A relational approach to make data warehouse scale like mapreduce. In J. Yu, M. Kim, and R. Unland, editors, *DASFAA 2011*, volume II of *Part*, page 6588. Springer, pp.

[29] C.-T. Yang, C.-T. Kuo, W.-H. Hsu, and W.-C. Shih. A medical image file accessing system with virtualization fault tolerance on cloud. In R. Li, J. Cao, and J. Bourgeois, editors, *GPC 2012*, volume 7296 of *LNCS*, page 338–349. Springer, 2012.

[30] C.-T. Yang, W.-C. Shih, G.-H. Chen, and S.-C. Yu. *Implementation of a Cloud Computing Environment for Hiding Huge Amounts of Data*, pages 1–7. Institute of Electrical and Electronics Engineers, Sep 2010.

[31] C.-T. Yang, W.-C. Shih, and C.-L. Huang. Implementation of a distributed data storage system with resource monitoring on cloud computing. In R. Li, J. Cao, and J. Bourgeois, editors, *GPC 2012*, volume 7296 of *LNCS*, page 64–73. Springer, 2012.

[32] C. Zhang, H. De, Sterck, D. R. Cheriton, S. of Computer, S. Department, of Applied, and Mathematics. Supporting multi-row distributed transactions with global snapshot isolation using bare-bones hbase. IEEE, 4244.

[33] F. Zhu, J. Liu, L. Xu, and T. C. of Software Engineering. A fast and high throughput sql query system for big data. In X. Wang et al., editors, *WISE 2012*, volume 7651 of *LNCS*, page 783–788. Springer, 2012.

Appendix A

CentOS System Settings

I. Make sure that SELinux is disabled or permissive

#selinux:setenforce 0

```
#vi /etc/selinux/config
```

Replace the SELINUX option enable to disable

```
#SELINUX=disabled
```

II. Disable iptables

```
#echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
```

III. Disable iptables

```
#service iptables stop
#chkconfig iptables off
```

IV. Specify TCP network information

```
#vi /etc/sysconfig/network-scripts/ifcfg-eth0 BOOTPROTO=none
ONBOOT=yes
IPADDR=192.168.100.1
NETMASK=255.255.255.0
```

```
GATEWAY=192.168.100.254
```

V. Lists hosts to be resolved locally

```
#vi /etc/hosts hbase-mserver0 192.168.100.1
hbase-rserver1 192.168.100.2
hbase-rserver2 192.168.100.3
hbase-rserver3 192.168.100.4
hbase-rserver4 192.168.100.5
```

VI. List DNS servers for internet domain name resolution

```
#vi /etc/resolve.conf
#nameserver 8.8.8.8
```

VII. Restart network

```
#service network restart
```

Appendix B

Cloudera Manager installation

I. Install CDH related packages

```
#yum -y install wget openssh-clients perl ntp
```

II. Time Correction

```
#ntpdate time.windows.com && hwclock -w
#date -R
#cp /usr/share/zoneinfo/Asia/Taipei /etc/localtime
```

III. Download Cloudera CM4

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

Open browser [HostIP]:7180 Default account and password {admin, admin}

IV. Cluster Installation

Specify hosts for CDH cluster installation.

Appendix C

Pseudo Code

```
/* HBase connection setup
*/
HBaseConfiguration config;
        config = new HBaseConfiguration();
        config.set("hbase.zookeeper.quorum", "hostname");
        config.set("hbase.zookeeper.property.clientPort", "port");
                HTable table = new HTable(config, "TableName");
        String row;
        String family[] = {"FamilyColumn1", "FamilyColumn2", "FamilyColumn3",
        String column[] = {"Column1", "Column2", "Column3", "Column4", "Column5"};
        String value;
        Put p;

/* Optimization properties
table.setWriteBufferSize(10 * 1024 * 1024);
table.setAutoFlush(false);
p.setWriteToWAL(false);
*/

/*
Open Excel file and get value
Build put object
*/
try {
            Workbook book = Workbook.getWorkbook();
            int PagsOfTotalSheet
            int RowsOfNowSheet

     for (int columnsOfNowSheet = 0; columnsOfNowSheet <= non-empty columns; columnsOfNowSheet++) {
                for (int count = 0; count <= PagsOfTotalSheet; count++) {
                    for (int i = 0; i < RowsOfNowSheet; i++) {
```

```
                    Sheet sheet = book.getSheet(count);
                    Cell cella = sheet.getCell(columnsOfNowSheet, i);
                    value = cella.getContents();

                    p = new Put(Bytes.toBytes(row));
                    totalheapsize += p.heapSize();
                    p.add(Bytes.toBytes(family[columnsOfNowSheet]),
                    table.put(p);
                }
        }
}
book.close();
```