東海大學

資訊工程研究所

碩士論文

指導教授：江輔政博士、楊朝棟博士

以用電巨量資料庫為主體，探討有效數據存取平台的設計及實作

Design and Implementation on Data-Accessing Platform Built from Big Data Warehouse of Electric Loads

研究生：周聖滄

中華民國一零六年七月

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 周 聖 滄 所提之論文

以用電巨量資料庫為主體，探討有效數據存取平台的設計及實作

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 [signature] 簽章

委 員 [signature]

[signature]

指 導 教 授 [signature] 簽章

指 導 教 授 [signature] 簽章

中華民國 106 年 6 月 30 日

摘 要

隨著物聯網科技的蓬勃發展，現今無所不在的電能資料可以透過網路被存取並提供監測需求作即時的資料分析。隨著時間的增長，如此大量的電能資料可以累積到將近 Terabytes 的大小。為了達到更即時的電能監測平台，在這篇論文中設計開發了一個有效率且新穎的資料處理技術，並成為本論文中的研究核心。基於多個巨量資料處理軟體的階層整合方法，本論文由下至上提出一個由 Hadoop Ecosystem、Spark 以及 Cloudera Impala 建置的電能巨量資料處理平台架構。本論文採用實務的電能資料為數據基礎，這些大數據源自實際配置於企業內部工廠中的「智慧電表」，接著以 Spark 作為電能資料的資料擷取、轉換和載入工具，Hive 用以建置巨量資料倉儲系統，Impala 則作為前端巨量資料搜尋引擎。在系統測試方面，本論文以 Hive、Spark 和 Impala 三種軟體來實作巨量資料叢集之資料搜尋及資料 ETL (Extract-Transform-Load)的效能測試，這兩種實驗皆使用了同樣的模組來進行。本論文的核心貢獻著重在兩個部分: 第一，採用多層軟體模組化(Software Modules)架構，設計建置了一個高效率的即時電能監測平台，具有高可行性及低成本的特性;第二，各實驗測試項目，經過實務實驗的測試，驗證了本系統的資料處理回應效率，確實頗具有改善實證，是有效率的，低成本的兼具可行性的，可以作為日後相關研究的參考依據。

關鍵字: 物聯網，巨量資料倉儲系統，智慧電表，資料 ETL，即時資料處理。

Abstract

With the flourishing of Internet of Things (IoT) technology, ubiquitous power data can be linked to the Internet and be analyzed for real-time monitoring requirement. Numerous power data would be accumulated to even Terabyte level as the time goes. To approach a real-time power monitoring platform on them, an efficient and novel implementation techniques has been developed and formed to be the kernel material of this thesis. Based on the integration of multiple software subsystems in a layered manner, the proposed data-accessing platform has been established and is composed of Apache Hadoop (as storage subsystem), Apahe Spark (as data ETL tool), Apache Hive (as big data warehouse), and Cloudera Impala (as big data real-time search engine) from bottom to top. The generic power-data source is provided by the so-called smart meters equipped inside factories located in an enterprise practically. The data collection and storage are handled by the Hadoop subsystem and the data ingestion to Hive data warehouse is conducted by the Spark unit. On the aspect of system verification, under single-record query, these software modules: Apache Hive, Apache Spark, and Impala had been tested in terms of query-response efficiency. And for the performance exploration on the statistical query function and data ETL processing. The relevant experiments have been conducted on the same three software modules as well. The kernel contributions of this research work can be highlighted by two parts: (1) Multi-layer software modules are adopted to design and implement the real-time power-monitoring platform embedded with some excellent characteristics of high efficiency, high feasibility and low cost. (2) The rudimental experiments are conducted to verify the query-response efficiency, and performance evaluations for the proposed real-time power-monitoring platform, which reveals the high feasibility for the target research goals.

Keywords: Internet of Things, Big data warehouse, Smart meter, Data ETL, Real-time processing.

致謝詞

這兩年就讀研究所的時間，改變了我很多的想法，不再像大學時期對於未來感到迷惘。進入了高效能計算實驗室之後，在楊朝棟教授以及學長們的帶領之下，不僅僅在學業研究方面有更多學習的機會，與此同時，對於未來科技趨勢也有了更深入的了解。

在碩士班的這兩年，接觸了雲端計算以及巨量資料等領域，並依循著實驗室學長姐們的腳步，逐漸的找到研究的定位，最後順利完成本論文的系統。當然這階段當中最感謝的莫過於我的指導教授楊朝棟教授以及江輔政教授，楊朝棟教授透過其豐富的研究經驗，引導我們正確的研究觀念、思考方向和研究方法，並時常囑咐我們做人處事的道理以及做事態度，鼓勵我們突破侷限住自己的框架，引領我們發現每件事情都是具有多種可能性的，並非只有單一解。另外要特別感謝的是我的指導教授江輔政教授，除了給予我許多做研究以及日後進入職場的建議，在這篇論文撰寫的過程中也協助我修正論文格式以及許多的英文文法，使本篇論文的內容更加精實、完整。

特別感謝口試委員張玉山教授、江輔政教授、伍朝欽教授以及詹毓偉教授特地撥空前來參加我的論文口試，在論文口試時提出很多論文的盲點和許多寶貴的意見，使本篇論文架構更加完整，學生在此由衷感謝。當然我也要感謝實驗室的學長、同學和學弟妹們，我將會十分懷念這兩年來我們不分日夜在實驗室共同學習、互相指導的時光。

再來我要感謝我的父母，讓我在求學期間能夠不虞匱乏地專心在學業上，即使在碩士班期間因為做研究沒有辦法時常回去，仍然會時常透過電話關心我，回家也一定準備了一桌好料，讓我回到家能夠放鬆身心再出發，你們是我遇到困境時堅持下去的最大動力來源。

東海大學資訊工程學系 高效能計算實驗室 周聖滄 106 年 07 月

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviation

- Apache Software Foundation (AFS): AFS is an American non-profit corporation to support Apache software projects.
- CDH (Cloudera Distribution including Apache Hadoop): CDH is the world's most popular Hadoop distribution. It includes all the leading Hadoop ecosystem components to store, process, discover, model, and serve unlimited data, and it's engineered to meet the highest enterprise standards for stability and reliability.
- ETL (Extract-Transform-Load): ETL is a process in data warehousing responsible for pulling data out of the source systems and placing it into a data warehouse.
- HDFS (Hadoop Distributed File System): HDFS is a distributed file system of Hadoop which is designed to run on commodity hardware.
- HBase: HBase is distributed database on HDFS architecture, and is non-relational database. It is modelled with reference of Google's BigTable, programmed in Java, and fault-tolerant for storing massive sparse data.
- HiveQL (Hive Query Language): HiveQL is a primary query language for Hive data warehouse to process data.
- IP Cam (IP Camera): IP Cam is a type of digital video camera commonly employed for surveillance, and which can send and receive data via a computer network and the Internet.
- JDBC (Java Database Connectivity): JDBC is an application programming interface (API) for the programming language Java, which defines how a client may access a database.
- MySQL: MySQL is an open-source relational database management system (RDBMS).
- MEMS (Microelectromechanical Systems): MEMS is the technology of microscopic devices, particularly those with moving parts.
- M2M (Machine to Machine): M2M refers to direct communication between devices using any communications channel which including wired and wireless.
- NoSQL (Not only SQL): NoSQL database provides a mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases.
- ODBC (Open Database Connectivity): ODBC is a standard application programming interface (API) for accessing database management systems (DBMS).

- OLAP (On-line Analytical Processing): OLAP is part of the broader category of business intelligence, which also encompasses relational database and data mining.
- P2P (Peer to Peer): P2P is a computing or networking distributed application architecture that partitions tasks or workloads among peers.
- RDBMS (Relational Database Management System): RDBMS is a database management system (DBMS) that is based on the relational model.
- RFID (Radio Frequency IDentification): RFID is a wireless communication technology that uses electromagnetic fields to automatically identify and track tags attached to objects.
- Solr: Solr is an open source enterprise search platform, written in Java. Its major features include full-text search, faceted search, real-time indexing, dynamic clustering, database integration, NoSQL features and rich document handling.
- SQL (Structured Query Language): SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS).
- UID (Unique Identifiers): UID is any identifier which is guaranteed to be unique among all identifiers used for those objects and for a specific purpose.
- WiMAX (Worldwide Interoperability for Microwave Access): WiMAX is a family of wireless communication standards based on the IEEE 802.16 set of standards.
- WSN (Wireless Sensor Network): WSN is spatially distributed autonomous sensors to monitor physical or environmental conditions.
- 2G/3G: 2G and 3G are short for wireless telephone technology.

Chapter 1

Introduction

With the widespread tides of Internet of Things and smart-electricity environment, the need to explore more efficient management on the electrical equipment and energy consumption attracts more attention. Basically, the records (or log files) of electrical loads are everyday data and the growth patterns on such day-by-day data source would be accommodated so steadily that the data volumes may be quite huge, even up to Terabytes levels. As the amount of data is expanding rapidly, the effectiveness of the traditional data storage system appears to be hard to handle them efficiently. In order to store raw data, long-term historical data, and processed data to facilitate future analysis of electricity consumption behavior, we used the concept of big data warehouse system to enhance system scalability, data integrity, and query speed. In terms of data querying, we choose Impala as SQL (Sequence Query Language) engine for processing the data that is sitting inside of Hive and then compare query efficiency among Apache HiveQL, Apache Spark SQL, and Impala SQL in different parameter.

1.1 Motivation

By deploying a large amount of smart meters and environmental sensors on campus, it already has accumulated a lot of historical log data. Those sensors deployed on campus which returns log data in every two seconds, thus, the volume of processed data is growing quickly. It already causes a burden to the traditional relational database and previous storage system. The performance of original relational database gradually cannot meet the system requirement because the data volume would be growing to hundreds of gigabytes as the time pass by. In order to avoid the drawbacks of the traditional relational database, and to effectively process the status monitoring data of the electrical equipment, this research proposed a multi-layer software architecture for the real-time power-monitoring platform embedded with some excellent characteristics of high efficiency, high feasibility, and low cost.

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. It has been built to scale up from single servers to thousands of machines, each offering local

computation and storage.

Hadoop has been the mainstream framework of the cloud computing technology, which HDFS and MapReduce is the two core technologies. Hadoop manages the data by breaking up the files into blocks and distributes them to the nodes of the Hadoop cluster. In general, Hadoop has better scalability, reliability, and usage efficiency of equipment than traditional data processing. Hive is a data warehouse system tool built on the top of Hadoop that can be used to handle structured and semi-structured data. After changing the overall storage architecture, the data stored in HDFS can be managed more easily.

Apache Hive is a component of Hortonworks Data Platform (HDP). Hive provides a SQL-like interface to data stored in HDP. In the previous tutorial, we used Pig, which is a scripting language with a focus on data flows. Hive provides a database query interface to Apache Hadoop. Moreover, Apache Hive is data warehouse infrastructure built on top of Apache Hadoop for providing data summarization, ad-hoc query, and analysis of large datasets. It provides a mechanism to project structure onto the data in Hadoop and to query that data using a SQL-like language called HiveQL (HQL).

In order to extract the meaningful content, we have to process the raw data and aggregate them, find out the correlation among them, and then store into data warehouse. In spite of Hive provided a SQL-like data manipulation language called HiveQL to process data, yet it gets Hive query converted to MapReduce program whenever it carries out query work. It will spend a lot of time on querying, because it follows the process model of Hadoop needs to read/write from the disk very frequently. The process interval will relatively spend more. On the contrary, in this work we use Spark distributed computing framework to deal with real-time power data. Spark is a general-purpose data processing engine, suitable for use in a wide range of circumstances. Its in-memory data processing engine can minimize the read times while accessing to the data on disk. Spark also provides high-level processing tools, such as SQL queries, spark machine learning library and streaming processing, and so on.

In terms of SQL-on-Hadoop, HiveQL and Spark SQL have provided a great efficiency on querying big data yet still far from Cloudera Impala. Cloudera Impala [1] is an massively

parallel processing SQL query engine for processing the data stored in HDFS, Hive, and HBase. Impala can interoperate with data stored in Hive, and tracks metadata about schema objects such as tables and columns. Impala does not work without the metastore database. Unlike Hive, Impala does not translate the queries into MapReduce jobs but executes them natively. However, Impala is memory intensive and does not run effectively for heavy data operations like joins because it is not possible to push in everything into the memory. That is why choose Impala as our specialized back-end query engine.

Above all, monitoring the whole status of the system includes hardware, software, and clusters, is also an essential factor to enterprises. Cloudera Manager [2] provides a fast way to deploy cluster, no matter what the scale or the deployment environment, complete with intelligent default settings based on your system. Not only can it monitor all components across all clusters (including Cloudera Manager itself), it can also easily monitor jobs and query performance. Cloudera Manager has the industry's only customizable dashboard, with the ability to create advanced charts for historical monitoring and custom triggers and thresholds for the environment.

1.2 Contributions

The kernel contributions of this research work can be highlighted in two parts: (1) Multi-layer software modules are adopted to design and implement the real-time power-monitoring platform embedded with some excellent characteristics of high efficiency, high feasibility and low cost. A big data warehouse and ETL process of cleansing, customization, reformatting, integration, and insertion into our data warehouse. (2) The rudimental experiments are conducted to verify the query-response efficiency, and performance evaluations for the proposed real-time power-monitoring platform, which reveals the high feasibility for the target research goals.

1.3 Thesis Organization

The remainder of this thesis is stated as follows. Chapter 2 describes background materials and relevant research work, including Internet of Things, Cloud Computing, Big Data, Hadoop

Ecosystem, and In-memory processing frameworks like Apache Spark and Cloudera Impala. Chapter 3 demonstrates the proposed system architecture which includes rudimental functions like extraction feature, data transformation, and load data processing function. The related performance evaluation and experimental results with analysis are presented in Chapter 4. Chapter 5 gives a discussion and summary to the proposed system and future works.

Chapter 2

Background and Related Works

2.1 IoT and Big Data

2.1.1 The Internet of Things (IoT)

Internet of Things (IoT) [3] is about every object, including the general items, animals and even people are equipped with a UID (Unique Identifiers). The data and information of objects hooked up on any computer networks, like wired-LAN or Wireless LAN, can be shared directly through the Internet. It is no longer to rely on the interaction between people or people and machines. The future will be the world of machine-to-machine (M2M), directly by the machine to complete a variety of work on the machine. From a practical point of view, the concept on IOT can be divided into three-tier architecture, from the bottom to the upper layer are respectively sensing layer, network layer, and application layer as shown in Figure 2.1.

- Perception (Sensing) layer: Perception layer is composed of devices that can sense the signal and monitor physical or environmental conditions of the field.

- Network layer: Network layer includes wireless or cable Internet and cloud technology to provide reliable network transmission so that each object can connect to Internet by specific connections and IP address.

- Application layer: Application layer is the kernel of IoT. It receives information from the Middleware layer and provides global management of the application presenting that information.

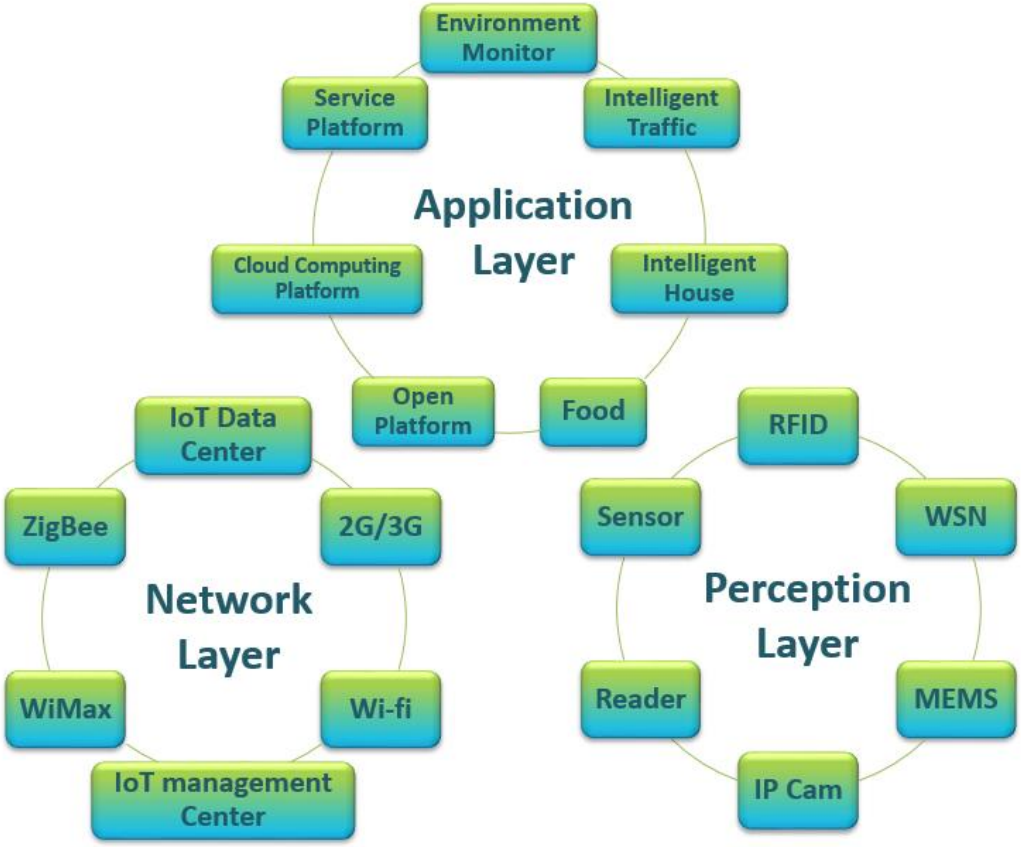

Figure 2.1: IoT 3-tiers

2.1.2 Cloud Computing

The term Cloud Computing comes from Google CEO Eric Schmidt who presented the idea for the first time on August 9, 2006 at the SES San Jose. According to the definition from National Institute of Standards and Technology (NIST) on Cloud Computing in May 2012: cloud computing is a model that provides ubiquitous, convenience, on-demand, and share resource that can be rapidly provisioned and released with minimal management effort or service provider interaction. It is composed of five essential characteristics, three service models and four deployment models as shown in Figure 2.2.

Five essential characteristics:

- On-demand self-service.
- Anytime, anywhere access by any network device.

- Resource pooling.
- Quick redeployment.
- Can be monitored and measured.

Three service models:

- Infrastructure as a Service (IaaS): IaaS is the way that users can use the computing resources, such as processor, storage capacity, and network through renting to cloud service providers but not buy hardware and build their own infrastructure.
- Platform as a Service (PaaS): PaaS is a cloud computing service to deliver hardware and software tools for those customers are needed for application development.
- Software as a Service (SaaS): Consumers use software deployed or data stored in the cloud but without managing cloud infrastructure and programming execution environment. No longer do customers need to install software on their computer, therefore reducing maintenance works and software support issues.

Four deployment models:

- Public Cloud: Public cloud services are available to users through the Internet and third-party service providers. Public cloud providers typically have some access control mechanisms for users.
- Private Cloud: Private cloud has the advantages of many public cloud environments, such as flexibility and appropriate for providing services. The difference between private cloud and public cloud is that the private cloud manages data and programs by themselves and is not affected by network bandwidth, security concerns, and regulatory restrictions.
- Hybrid Cloud: The hybrid cloud combines the benefit of public and private cloud, in which users typically outsource unimportant enterprise information and handle it on the public cloud, at the same time, still control the main enterprise services and information.
- Community Cloud: Community cloud is shared by several organizations to support a particular community that having common concerns.

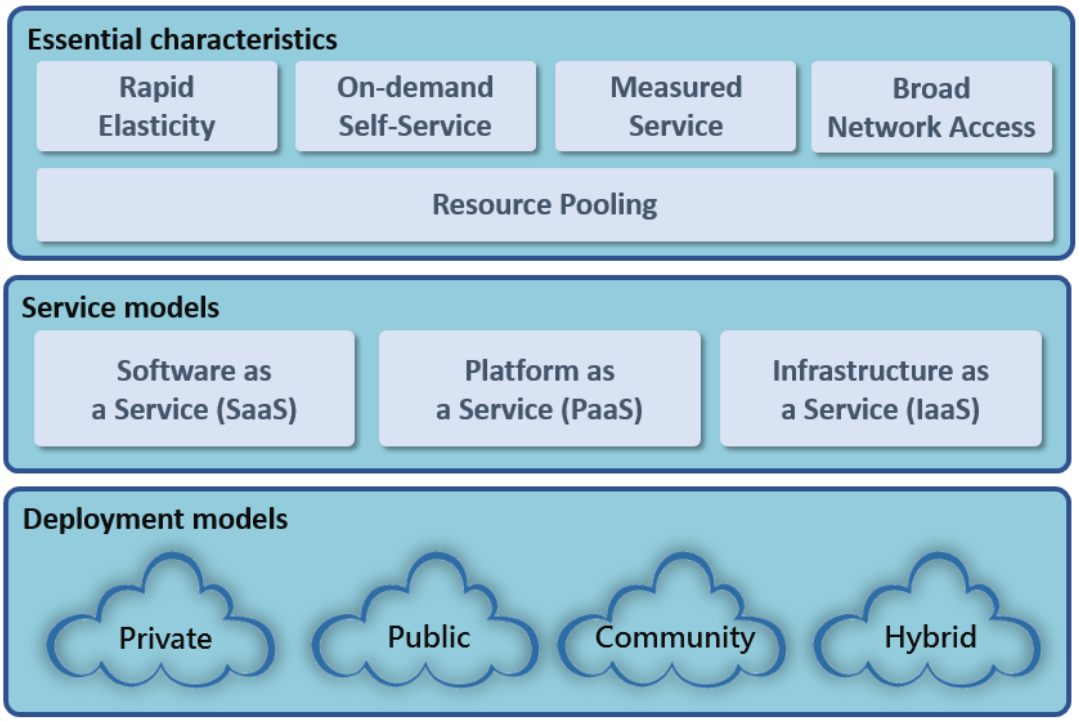

Figure 2.2: Definition of Cloud Computing

2.1.3 Big Data

Big data [4][5] is a term that has been in use since 1990s. As the data volume grows explosively and ubiquitously, the traditional techniques on data processing applications are inadequate to deal with ever-growing data volumes. Almost 90 percent of the data in the world was generated during the past two years. According to International Data Corporation (IDC), in 2013 there are 4.4 zettabytes data in the world, and they predict it will soon reach 44 zettabytes in 2020. Big Data is also a way to deal with a great volume of the structured, semi-structured and unstructured data. The development of big data has four directions as shown in Figure 2.3 [6].

- Volume: A large amount of data would be generated, processed, and stored.
- Velocity: The speed of data in and out.
- Variety: Variability refers to as the source of information is extraordinary inclusive and diverse of data type, including text, video, pictures, web index, data stream, astronomical data, and other unstructured data. Both of them are difficult to be processed in the structure of traditional relational data field.
- Veracity: The less being mentioned characteristic of big data, it means the uncertainty

of data. By analyzing and filtering the data which are deviation, forgery and abnormality to prevent these "dirty data" from compromising the integrity of the data system and then affecting decision making.

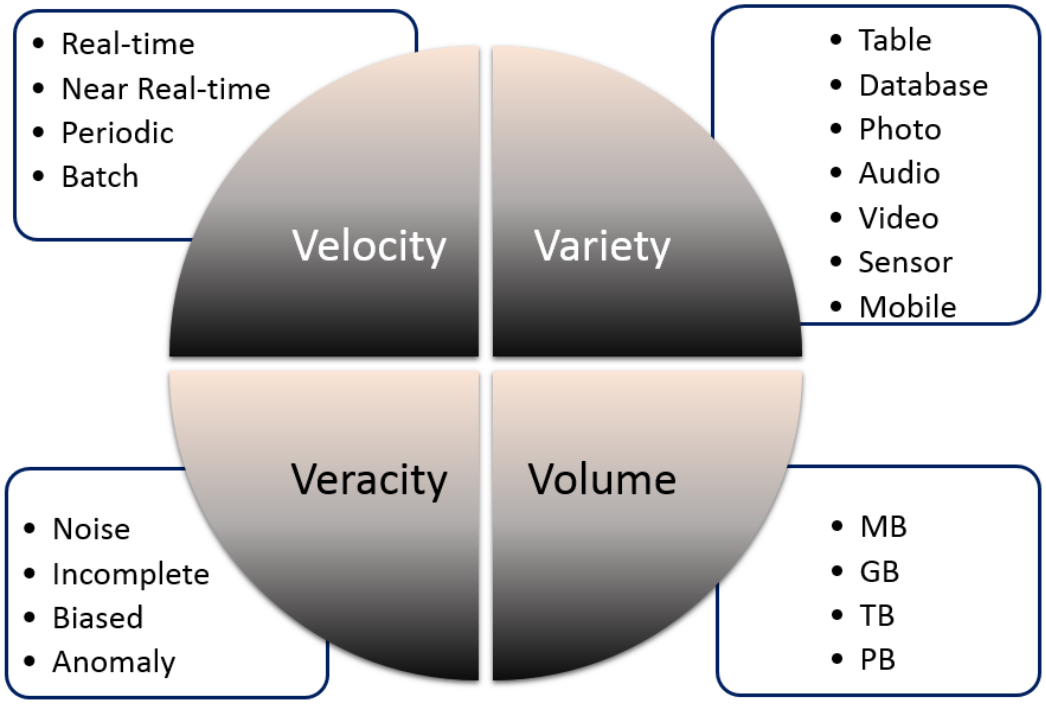

Figure 2.3: The 4V's of Big Data

While the term "Big Data" is relatively new, as we can see the act of gathering and storing large amounts of data for eventual analysis is ages old.

2.2 Hadoop Ecosystem

2.2.1 Apache Hadoop

Apache Hadoop [7] is an open-source software framework that being broadly used for big data processing nowadays. It came from the Google File System paper which was published in October 2003 and the paper of MapReduce. The Apache Hadoop framework is built on the top of Hadoop Distributed File System (HDFS), which supports a stable and automatic distributed processing system. Hadoop implements MapReduce [8] [9] programming framework which divided file into the same block size. Data fragments can be executed in parallel on any node in

the cluster. Hadoop is designed to provide parallel computing and scale up the processing ability from single server to thousands of machines. As Figure 2.4 can see, NameNode is responsible for the assignment of tasks to Task Trackers and assign data to DataNodes.

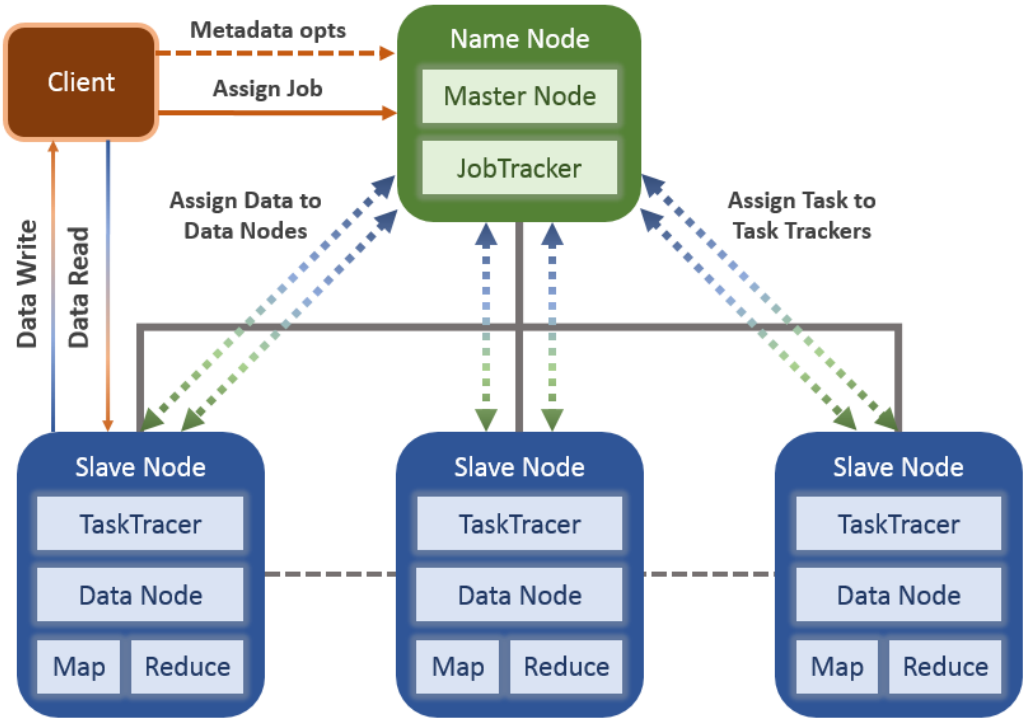

Figure 2.4: Master/slave Architecture in Hadoop

2.2.2 HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. The detection of faults and automated recovery is an important architectural goal of HDFS. HDFS has master-slave architecture with a single Name Node as the master server which manage the file system. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of Data Nodes. The Name Node executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The Data Nodes are responsible for serving read and write requests from the file system clients. HDFS ensures

input distribution and provides the user with an interface whose role is to provide chunks of data files to cluster nodes. Among its chief advantages, the Hadoop Distributed File System provides input locality by enabling nodes hosting input shards to apply their processing on such chunks, rather than on remotely stored data. Figure 2.5 shows the architecture of HDFS [10] [11].

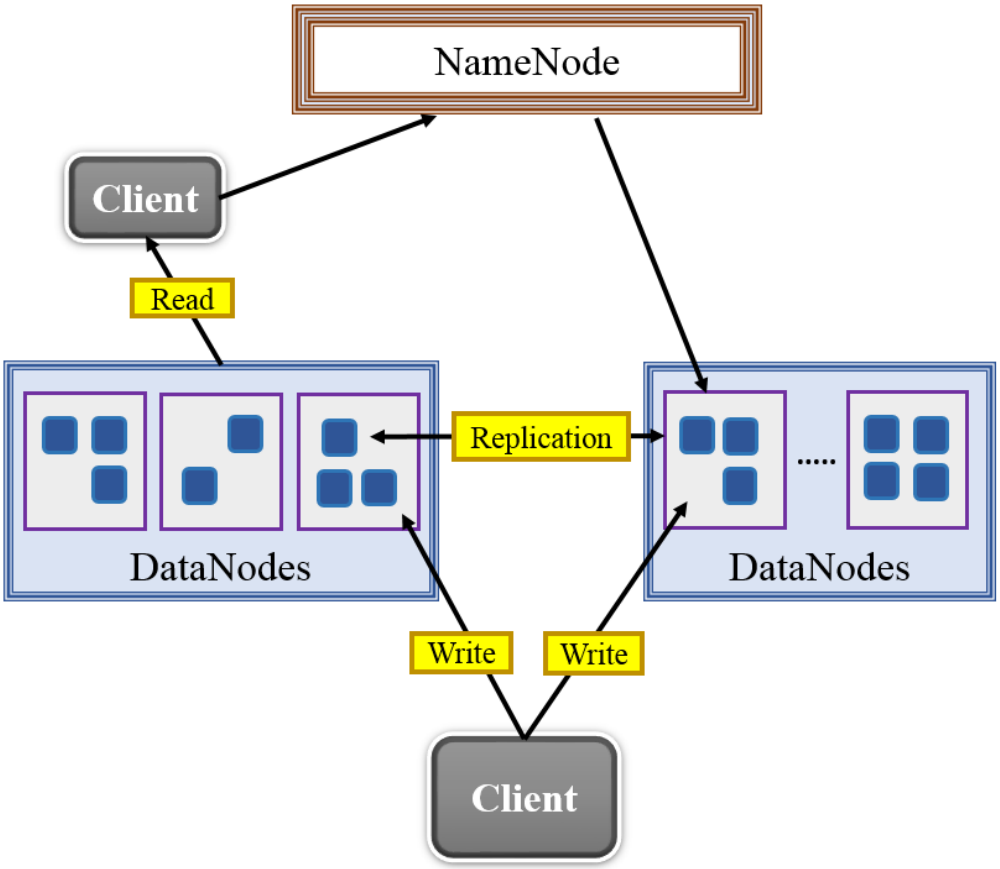

Figure 2.5: The Architecture of HDFS

2.2.3 Apache Hive

Apache Hive [12] is a data warehouse solution that has been developed by Apache software foundation to integrate data storage and querying and managing large datasets. Hive as a data warehouse application on top of Hadoop MapReduce, and it allows users to handle the data stored in it as if it was stored in a regular database. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. Hive enables user who have experience using traditional RDBMS to fun familiar queries on

MapReduce [13]. Hive's advantages are as follows:

- Very powerful at big data storing
- Easy to learn and understand
- Portable, Multiple data views
- Used with and DBMS system with vendor
- Well defined standards exist and used relational databases
- High Speed, Integrates with Java

Figure 2.6 shows the relation between Hadoop and Hive.

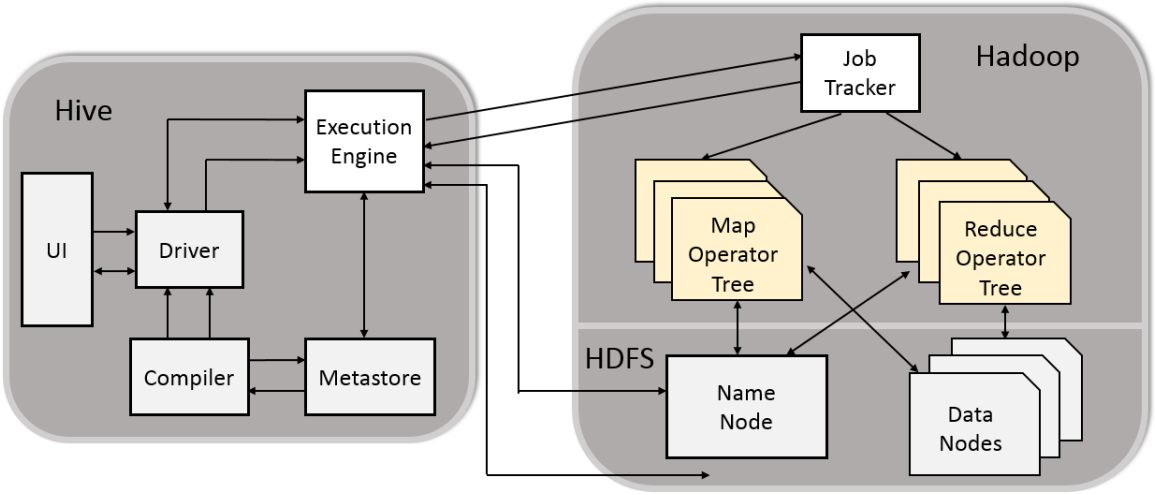

Figure 2.6: Relation between Hadoop and Hive

2.2.4 Apache Sqoop

Sqoop [14] [15] is a tool for SQL to Hadoop. Sqoop is a convenient tool that moves data between traditional relational database and NoSQL. Sqoop takes advantage of Hadoop MapReduce parallel feature that accelerates data migration by batch processing. Sqoop is an import tool that supports data migration from relation database to Hive, HDFS, and HBase; it also supports full table import and incremental table import. Figure 2.7 shows the basic workflow of Sqoop. When Sqoop imports table data from RDB, it depends on different split-by values to split data; next it lets segmented blocks assigned in different map, and each map will process its block data. Finally, it stores data in the Hadoop distributed storage system. There are three features of Sqoop below:

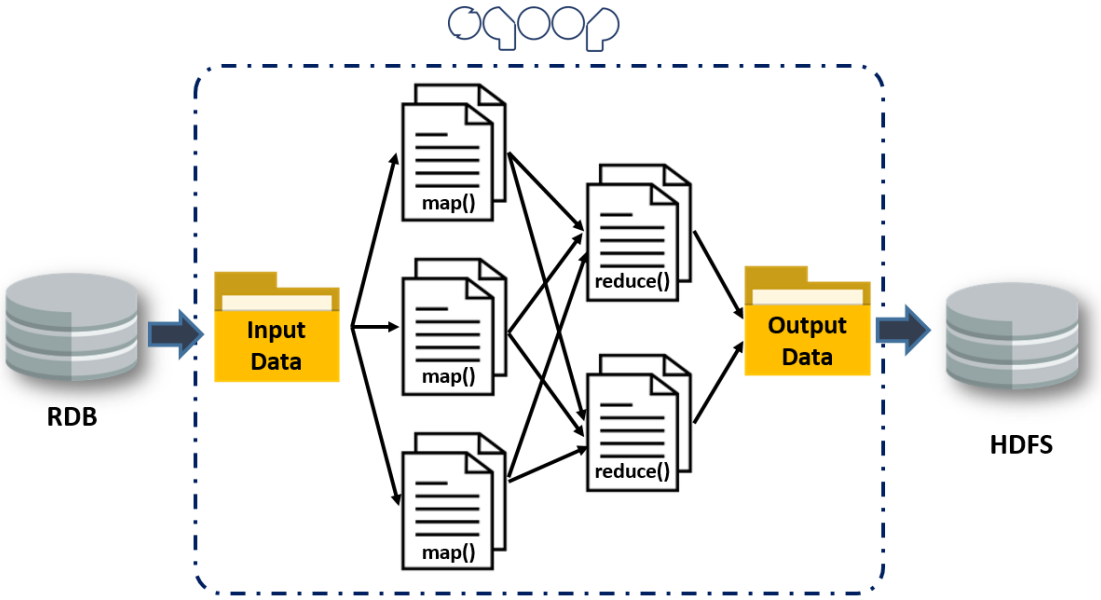

Figure 2.7: Sqoop Workflow

2.3 In-memory processing framework

2.3.1 Apache Spark

Apache Spark [16] is an open-source cluster computing framework originally developed in the AMPLab at UC Berkeley. In contrast to the two-stage disk-based MapReduce paradigm of Hadoop, Spark in-memory primitives provide performance up to 100 times faster for certain applications. By allowing user programs to load data into a memory of cluster and query it repeatedly, Spark is well suited to machine learning algorithms. Spark requires a cluster manager and a distributed storage system. For cluster management, Spark supports standalone (native Spark cluster), Hadoop YARN, or Apache Mesos. For distributed storage, Spark can interface with a wide variety, including HDFS, Cassandra, OpenStack Swift, and Amazon S3. Spark also supports a pseudo distributed local mode, usually used only for development or testing purposes, where distributed storage is not required and the local file system can be used instead; in this scenario, Spark is running on a single machine with one executor per CPU core. Spark has over 465 contributors in 2014, making it the most active project in the Apache Software Foundation and among big data open source projects.

2.3.2 Spark SQL

Like Apache Spark in general, Spark SQL [17] in particular is all about distributed in-memory computations. One use of Spark SQL is to execute SQL queries. Spark SQL can also be used to reading data from an existing Hive installation. For more on how to configure this feature, please refer to the Hive tables section. When running SQL statement from within another programming language the results will be returned as a Dataset/DataFrame. You can also interact with the SQL interface using the command-line or over JDBC/ODBC.

2.3.3 Impala

Impala [18] is a real-time SQL query engine that brings scalable parallel database technology for the Hadoop ecosystem. It allows user use SQL to query Petabytes of data stored in HDFS and HBase without data movement or transformation. Impala uses Hive metastore, and it can be used to querying data from Hive tables directly. Unlike Hive, Impala SQL does not translate the queries into MapReduce jobs but executes them natively. However, Impala is memory intensive and does not run effectively for heavy data operations like joins because it is not possible to push in everything into the memory. The role of Impala played in Cloudera environment as shown in Figure 2.8.

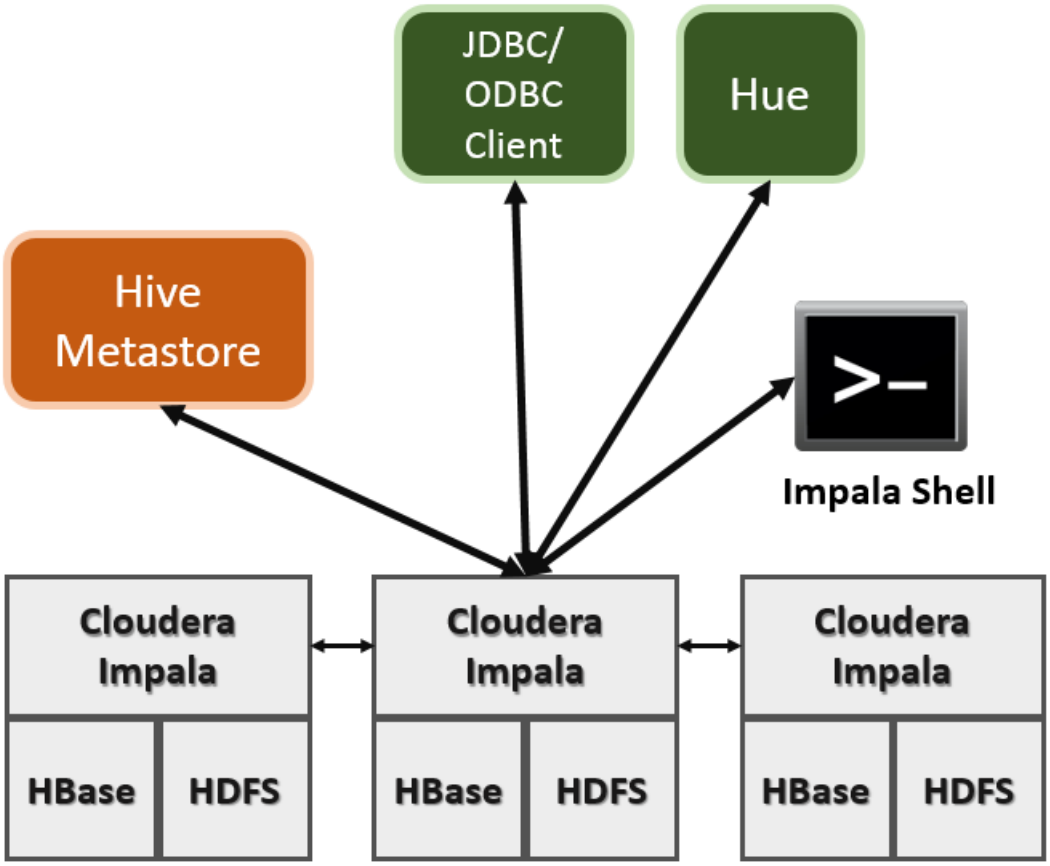

Figure 2.8: Impala in Cloudera Environment

2.4 Related Works

Smart meter data are typically bundled with social economic data in analytics, such as meter geographic locations, weather conditions and user information, which makes the data sets very sizable and the analytics complex. In Xiufeng Liu et al. [19] 2016, they proposed a solution to offer an information integration pipeline for ingesting data from smart meters, a scalable platform for processing and mining big data sets, and a web portal for visualizing analytics results. The implemented system has a hybrid architecture of using Spark or Hive for big data processing, and using the machine learning toolkit, MADlib, for doing in-database data analytics in PostgreSQL database.

Extract-Transform-Load (ETL) tools are pieces of software responsible for the extraction of data from several sources, its cleansing, customization, reformatting, integration, and

insertion into a data warehouse. Building the ETL process is potentially one of the biggest tasks of building a warehouse; In Shaker H. Ali El-Sappagh et al. [20], they proposed a model which can be used to designing ETL scenarios, and document, customize, and simplify the tracing of the mapping between the data source attributes and its corresponding in data warehouse.

Beyond the benefits of Spark is compatible with Hadoop HDFS, and using in-memory distributed memory technology, it allows data repeatedly calculated by cache data in memory since Spark in-memory primitives provide performance up to 100 times faster for certain applications. According to the thesis that, Spark has a higher speed than Hadoop processing capability, and access to data on the YARN. Chao-Tung Yang et al, [21] 2015., the experimental results of this paper show the Spark processing speed is faster than Hadoop, so we use Spark as our process tool.

Through the analysis of OLAP technology in big data environment, a kind of analytical platform of status monitoring big data of electric power equipment was designed. This platform includes relational on-line analysis base on Hive, relational on-line analysis base on Impala, and multi-dimensional on-line analysis based on HBase. Aiming to solve the problems of large cost of connection operation and low query speed of distributed relational analysis data model, Wang, Dewen, and Zhou. [22], presented a kind of data schema of state monitoring of power equipment which was based on not-join level-encoding technologies. In order to reduce the number of connection options to optimize performance, encoded the level information of dimension table, compressed to the fact table.

SQL is a special purpose programming language that have been used for many years to manage data in relational databases. Although SQL is not suitable for every data issue and it cannot be applied to a complicated analysis. It has been applied to many enterprise developers and business analysts because its availability. In Ilias Mavridis and Helen Karatza [23], they have investigated the distributed SQL-type querying with Apache Hive and Spark SQL in real Apache Web Server log files. After several experiments, they concluded that Spark SQL is much faster than Hive. That happens because Spark SQL has a set of techniques to prevent reads and writes to disk storage, caching of tables in memory and optimizing efficiency.

For the final objective of a big data warehouse, a platform to present the results of Business Intelligent and data mining are quite essential. In Ren-Hao Liu and Chao-Tung Yang [24], they had built a cloud intelligent campus energy monitoring system and used big data technology to test the best processing and storage frameworks, including planning of hardware, build application platform, and big data processing (Apache Hadoop and Apache Spark) and big data storage (MySQL, Apache Hive, and Apache HBase) for power data. It also be the reason we built this data warehouse system: to make query processing more efficient, even the data volume is growing day by day. In the experiment of this paper, they measured the data process time of HDFS and MySQL in one of experiments, the subsection compared the data search speed of MySQL and Hive using Spark. They noticed that the response time of Hive through Spark is less than MySQL through Spark. The difference of response time is not very obvious when the test data is getting smaller. But when the data is 2GB, Hive is 100% faster than MySQL, so the response time difference of both will be increased by data size.

Yin-Zhen Yan and Chao-Tung Yang et al [25] 2016, they proposed a cloud green energy management system to settle the problem of oversize data and the computational efficiency of data analysis, they added the big data technology and cloud computing to upgrade the system performance. By building cloud infrastructure and distributed storage cluster, they adopt the open source framework, Hadoop, to implement the two main functions: storage and computation. Based on these two functions, the system they proposed speed up the analysis and processing of big data by using Hadoop MapReduce to access HBase.

The Smart grid can be made more intelligent by processing and deriving new information from these data in real time. In Bharathi Ganesh HB, Sachin Kumar S, and Shyam R et al. [26], this paper presented Apache spark as a unified cluster computing platform which is suitable for storing and performing Big Data analytics on smart grid data for applications like automatic demand response and real time pricing. In data science, the term data analysis, data mining and textmining refers to the same technique of deriving hidden information using various machine learning algorithms from the data acquired.

With the complexities and challenges involved in big data computing, the need for large computational infrastructure, expensive software, and effort are raised as well. In Hameeza

Ahmed and Muhammad Ali Ismail et al. [27] 2016, this paper deployed an Apache Spark cluster as a cloud service (SAAS) on OpenStack cloud. There are several benefits of providing Apache Spark as SAAS namely scalability, backup and restore facility, ease of use, high speed, increased throughput, lower cost and many others. The work being presented in this paper makes an in-depth analysis of the performance of Spark cluster as a SAAS. It does so by comparing the results of a Spark cluster configured as cloud service with the conventional one. The analytical query involves three benchmarks namely Hive Join, Scan and Hive Aggregate respectively. The final results clearly depict how apache Spark cluster deployed on OpenStack dominates the conventional cluster both in terms of speed and throughput.

Smart city is very important issue for future development. M. Mazhar Rathore et al, [28] 2016. In order to solve the growing data generated by IoT, it must be resolved through big data processing architecture. They use the Hadoop ecosystem to assist in data processing and storage. The intelligent system must provide two data types, one is real-time data, the other is the historical data. Real-time data provided decision-makers to make decisions in a short time, the analysis of the historical data can provide the foundation of city or system planning. For data processing, in order to real-time data processing, it must through the MapReduce architecture. MapReduce is composed by the map and reduce, and the input is divided into a plurality of block, and then executed on each node. For data storage, they use HBase and Hive for managing Database.

Chapter 3

System Design and Implementation

In this section, we present the architecture of our proposed electric power data warehouse and the ETL service of data warehousing. In Subsection 3.1, the proposed system architecture would be introduced. In Subsection 3.2, we talk about the design of electric power data warehouse, and the transfer of old data from traditional database to Hadoop big data warehouse. Finally, the implementation of electric power data warehouse and data transferring application in sub-section 3.3.

3.1 System Architecture

The proposed system needs to receive and process power data per second from sensors spreaded all over the campus for energy monitoring, early warning, analysis and other functions, thus the scalability and flexibility of the system are very important. Accordingly, the proposed system has multi-tier architecture, i.e., data generation and collection, data processing, and data analysis. This multi-tier architecture, as shown in Figure 3.1, can efficiently process and analyze the huge amount of power data, and its architecture introduced as follows:

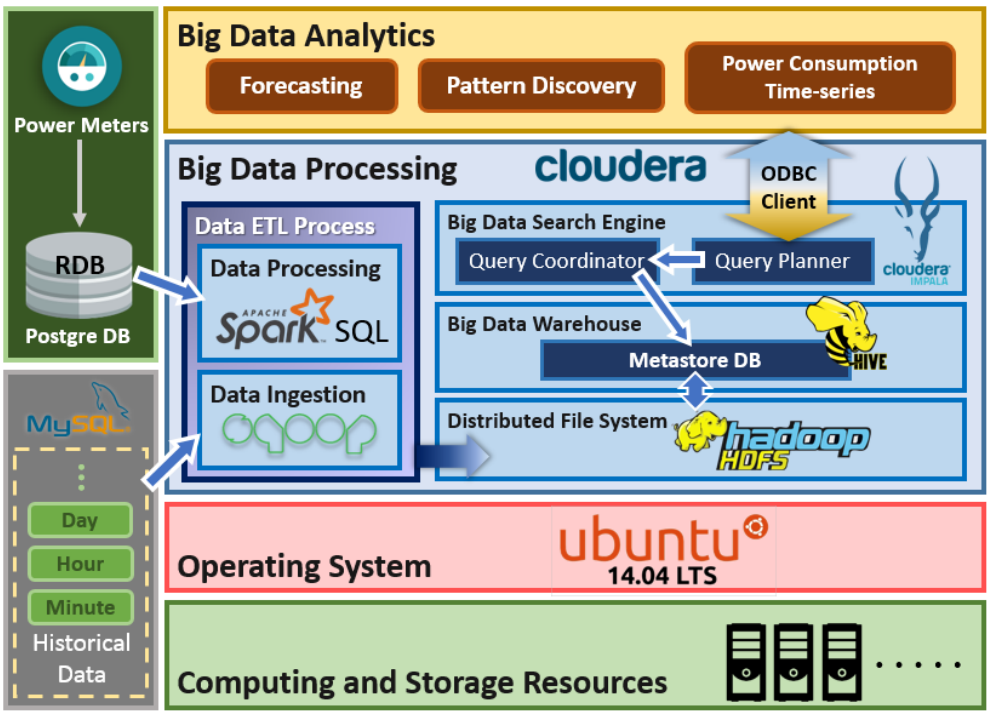

Figure 3.1: The Overview of System Architecture

The bottom layer of Figure 3.1 is the computing and storage resources, including one master node with 10 processors and 128GB (Gigabyte) memory, and four slave nodes with 8 processors and 16GB memory. Chapter 4 will explain the hardware specification in detail. The detail of hardware and network specification as shown in Figure 3.2.

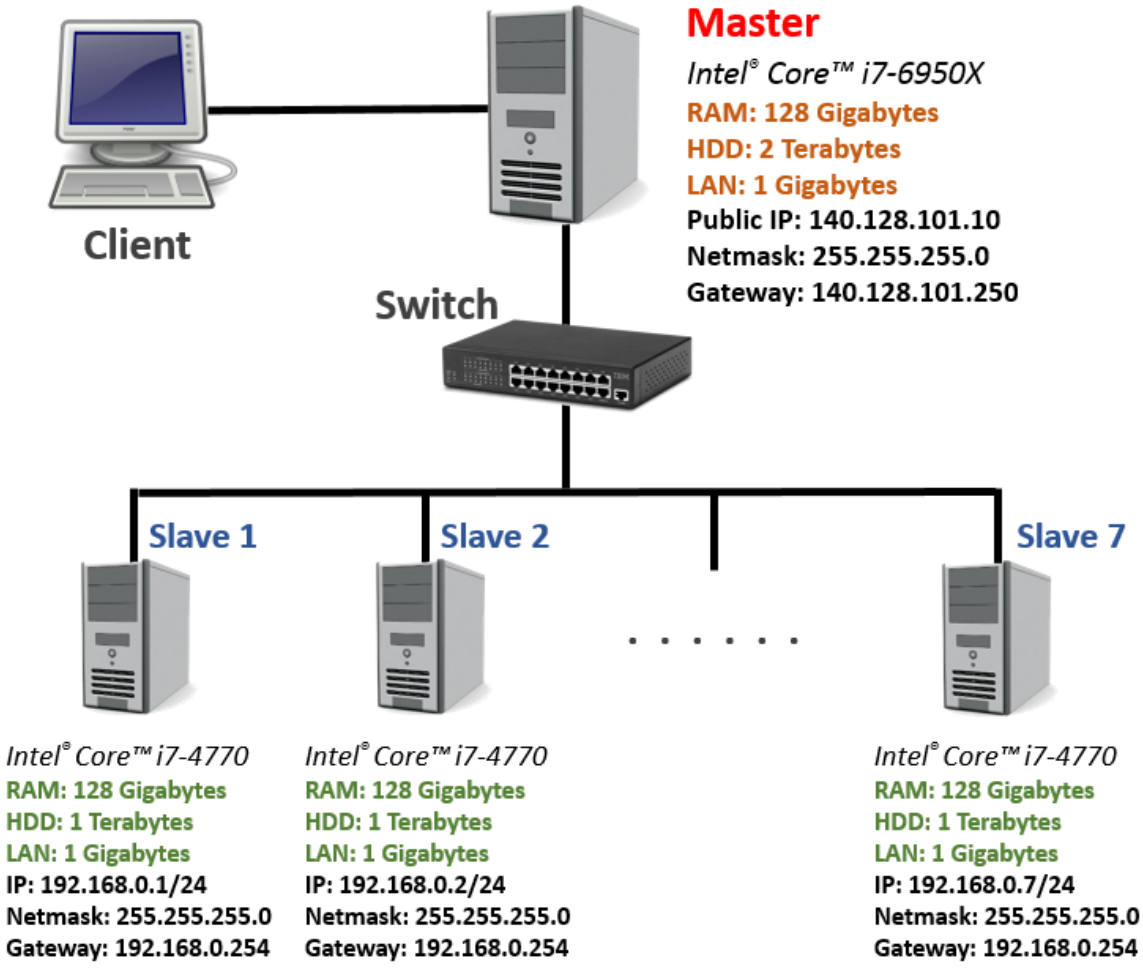

Figure 3.2: The Detail of Computing and Storage Resources Layer

In the second layer, Ubuntu Linux is our first choice operating system, and the version 14.04 LTS which is more stable one.The third layer is the main core of the system architecture, which is composed of data ETL process, big data storage, and data search engine. The data source of ETL process includes old data stored in MySQL relational database and raw streaming data from smart meters. On the part of data ingestion, we adopted Spark SQL as task scheduling module to collect raw data on a regular basis; Apache Sqoop as transfer tool to directly ingest old data stored in operational databases from MySQL to Hive data warehouse. Basically, Hive

only defined the table schema and partition in the Hive Metastore, yet it does not store or process data. Hive is totally relying on HDFS and MapReduce to do data manipulation. Therefore, the action of ingesting data into Hive, in fact, it was being ingested into HDFS.

In the sight of big data search engine, Cloudera Impala integrates with the Hive metastore database, to share databases and tables between both components. Impala provides high-performance, low-latency SQL queries on data stored in Hadoop file formats, like Hive or HBase. The fast response for queries enables interactive exploration and fine-tuning of analytic queries, rather than batch jobs traditionally associated with SQL-on-Hadoop technologies. Besides, Impala also supports JDBC client to interact with it. So as to do further data analytics, like foresting, pattern discovery, power consumption time-series through high-performance, Impala become the principle choice for an OLAP service.

3.2 Design of Data Warehouse and ETL Service

As seen from the middle layer of Figure 3.1, Hive becomes a big data warehouse of the power data, and the data source of the system is conducted from the operational MySQL database and real-time smart meter data. The ETL processing of data warehousing had been done with Sqoop as a tool to extract operational databases from MySQL to Hive data warehouse and Spark as real-time data processing module to transform raw data to useful data. Finally, executing the Spark application periodically to make sure power data can be imported in a stable condition.

3.2.1 Power Utilization Assessment

In the section of power-sensing we used the WPM-100 Wireless Multifunction Power Meter to collect data, including: voltage, current, power, power factor, frequency, etc. In addition to this, it can be transmitted through the wireless means of sensing data for a broad range of energy data collection to reduce the work of wiring. These data will be returned to the electric power data warehouse immediately.

Figure 3.3: WPM-100 Wireless Multifunction Power Meter

Figure 3.4: WPM-100 Wireless Multifunction Power Sensor

The power meter we used in the proposed system is developed by EverComm Opto Ltd. (ECO). ECO is a manufacturer of motor and generator, and vendor of system integration located in New Taipei City, Taiwan. It began offering the service of photoelectric, wireless communication and information system since March 26, 2015. ECO has products of energy management system, green energy management, street light management and landscape illumination. Besides, ECO has provided a cycle diagram for energy management system as shown below in Figure 3.5.

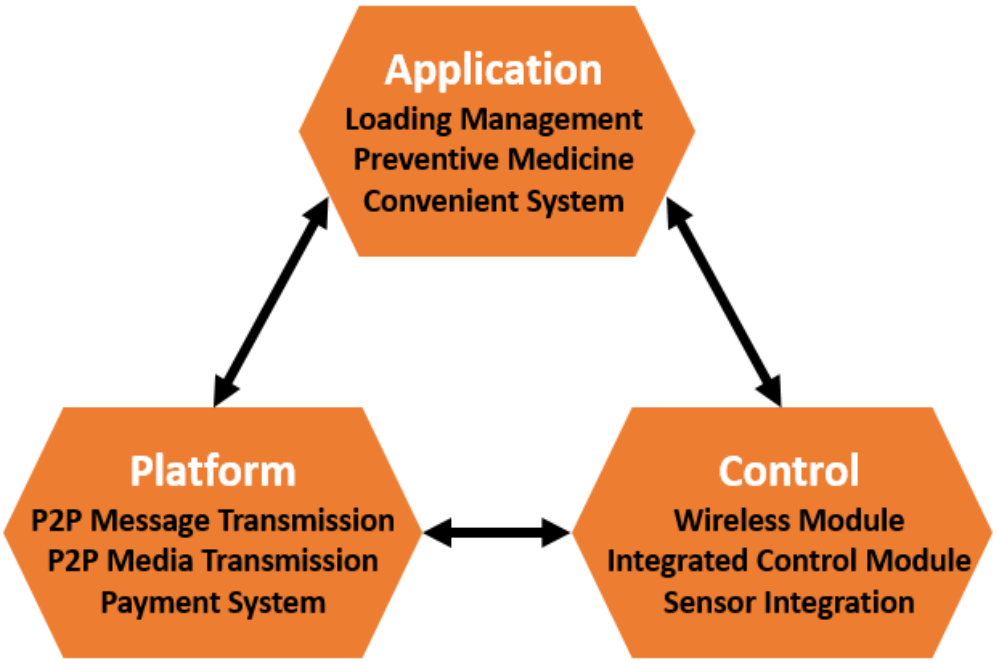

Figure 3.5: Energy Management System Diagram Proposed by ECO

The three portions of energy management system that proposed by ECO.

- Application: Supply an energy saving and management service for client.
- Control: Develop the application of wireless module, integrated control and sensor in green energy industry.
- Platform: Building a neural network platform for information transmission on the P2P.

3.2.2 Power Utilization Assessment

For the sake of assessing data volume more accurately. The data format of smart meter as shown in Table 3.1 below. The return data is presented in JSON format with 26 fields.

Table 3.1: Smart Meter Data Format

Field	Description	Example
location	Meter ID	LIB-4
time_stamp	Datetime	1491818534115
KW	Current power of the meter	228
total_KWH	Accumulated power this hour	1026109
ch1_pf	CH1-PF	0.807
ch1_voltage	CH1-Voltage	121.6
ch1_current	CH1-Current	10.343
ch1_hz	CH1-Hz	1015
ch2_pf	CH2-PF	0.878
ch2_voltage	CH2-Voltage	122.2
ch2_current	CH2-Current	9.786
ch2_hz	CH2-Hz	1050
ch3_pf	CH3-PF	0.88
ch3_voltage	CH3-Voltage	121.8
ch3_current	CH3-Current	11.466
ch3_hz	CH3-Hz	1229
voltage12		211.2
voltage23		211.4
voltage31		210.9
ch1_THDi		1.60
ch2_THDi		9.10
ch3_THDi		2.00
ch1_THDv		1.60
ch2_THDv		1.70
ch3_THDv		1.90
total_pf	Total PF	0.98

The data collected per minute for each meter which means that each sensor will receive about 128 Bytes of data every two seconds. In the case of TungHai University, there are about 40 dormitories, and administrative and academic building has equipped smart meter, based on one building with one meter. The density of the sensor in campus is about to reach 120, after deducing, one year will produce the following amount of data:

$$128\ (\text{Bytes}) \times 43200\ (\text{Sec}) \times 365\ (\text{Days}) \times 120\ (\text{Sensors}) \fallingdotseq 230976\ \text{MB} \fallingdotseq 225.6\ \text{GB}$$

In addition to smart meter sensing data, the environment of sensing area also causes an impact of electric power consumption. In environmental sensing, each environment has 20 sensors detecting once in a second. The estimated volume of data will be as follow:

$$100\ (\text{Bytes}) \times 86400\ (\text{Sec}) \times 365\ (\text{Days}) \times 20\ (\text{Sensors}) \fallingdotseq 60150\ \text{MB} \fallingdotseq 58.7\ \text{GB}$$

According to the above calculation we can simulate the data volume would reach almost 300 GB a year, besides, after built our campus energy management system, the volume of processed data is growing quickly as well. It already causes a burden to the traditional relational database and previous storage system.

3.2.3 Transferring Operational Data from MySQL to Hive Database

The transfer of historical meter data from the relational database to HDFS can be done through Apache Sqoop. Sqoop is a data transfer tool that can transfer data from a traditional relational database to a Hadoop storage system by using Hadoop MapReduce parallelism to speed up the process of data migration. It supports not only transfer data from MySQL to HDFS but also Hive and HBase. Figure 3.6 shows the workflow for operational data transfer.

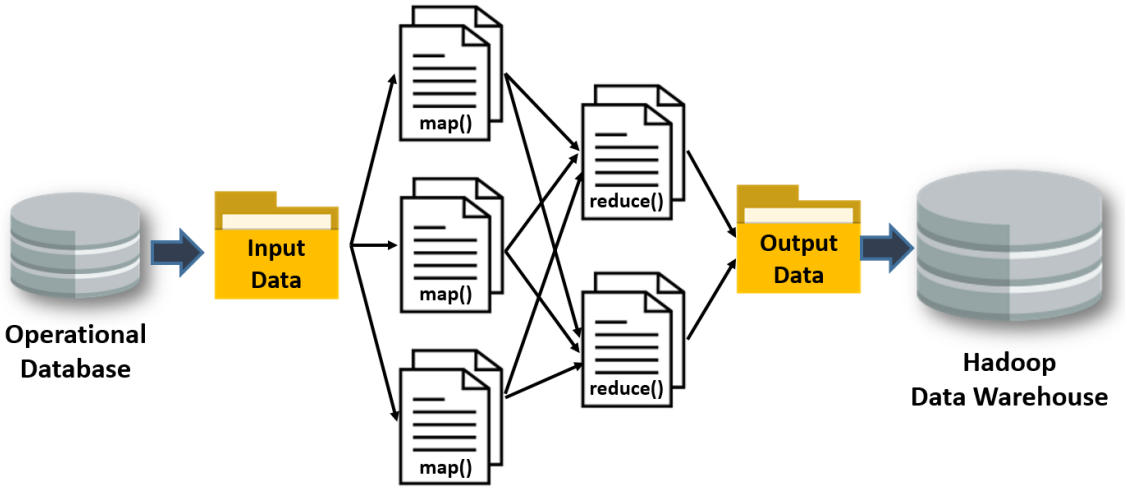

Figure 3.6: Sqoop Workflow for Operational Data Transfer

3.2.4 Data ETL Service

Data ETL Service can be used to transferring raw data to the data warehouse via the ETL process. Raw data includes real-time data in data center and data of campus buildings and stable data writing is done by periodically executing Data ETL Service. Figure 3.8 shows the use case diagram of the data ETL service. As shown in Figure 3.7, we choose Spark SQL DataFrames API as data processing module to develop data ETL application.

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.SparkSession

object sparksqlETL {
  def main(args: Array[String]): Unit = {
    val rootLogger = Logger.getRootLogger()
    rootLogger.setLevel(Level.ERROR)
    val spark = SparkSession
      .builder()
      .appName("Spark Hive Example")
      .config("spark.sql.warehouse.dir", "hdfs://master:9000/user/hive/warehouse")
      .enableHiveSupport()
      .getOrCreate()

    import spark.implicits._
    import spark.sql

    sql("select * from powerdata")
  }
}
```

Figure 3.7: Spark SQL API

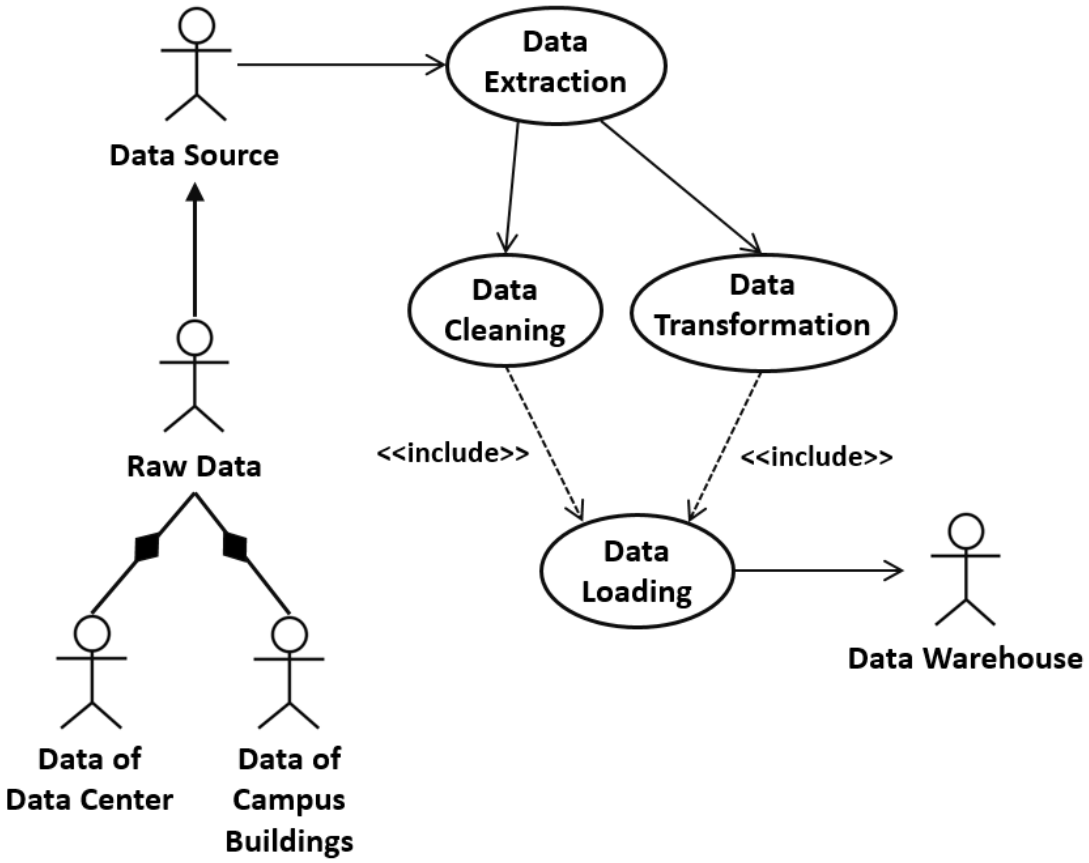

Figure 3.8: Use Case Diagram for Data ETL Service

3.2.5 Periodic Statistical Service

To achieve real-time presentation on front-end web user-interface, the process of calculating raw data to different kinds of processed data periodically is required. By the reason of the efficiency of Spark, Scala is our first choose to program the periodic processing application. There are several types of data that is needed to generate, such as accumulated power data in each minute, hour, day, month, and year. Another benefit of adopting spark as back-end processing is to reduce the burden of front-end.

3.3 System Implementation

In this work, we have established the big data clusters through four physical machines, one node as master, three nodes as the computing node to set up Cloudera big data platform that including CDH (Cloudera Distribution Including Apache Hadoop), Apache Spark, Apache Sqoop2, Apache Hive, and Cloudera Impala. Table 3.2 shows the software specification of five cluster nodes.

Table 3.2: Software Specification

	Version
Cloudera Manager	5.10.1
Hadoop	2.6.0-cdh5.10.1
HDFS	2.6.0-cdh5.10.1
Hive	1.1.0-cdh5.10.1
YARN	2.6.0-cdh5.10.1
Spark	2.1.0-cdh5.10.1
Sqoop2	1.99.5-cdh5.10.1
Impala	2.7.0-cdh5.10.1
Hue	3.9.0-cdh5.10.1

In the big data processing platform construction, we used the newest version of CDH (Cloudera Distribution Including Apache Hadoop) as big data service platform. The reason we did not use native Hadoop is that CDH not only provides better stability than native Hadoop ecosystem but convenient platform monitoring and management, each node in the cluster can be monitored instantly via the web user interface in Cloudera Manager. In addition, Cloudera Manager allows developers add new cluster or host manually and choose custom services to deploy on it.

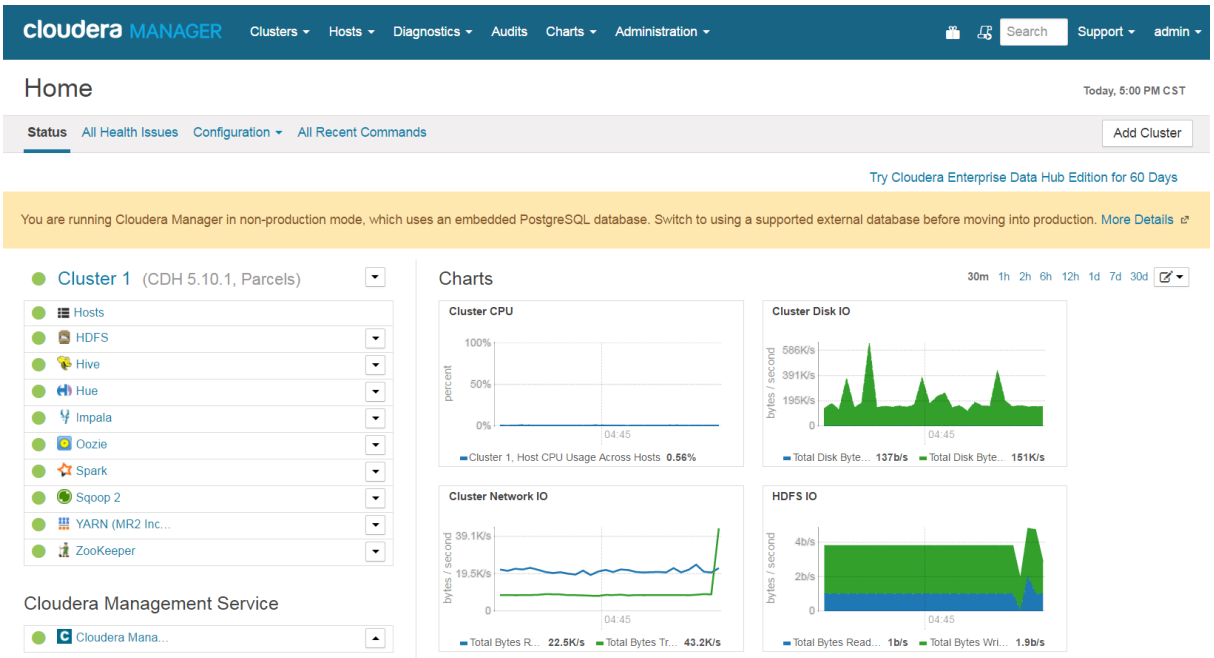

Figure 3.9: Cloudera Manager Web User Interface

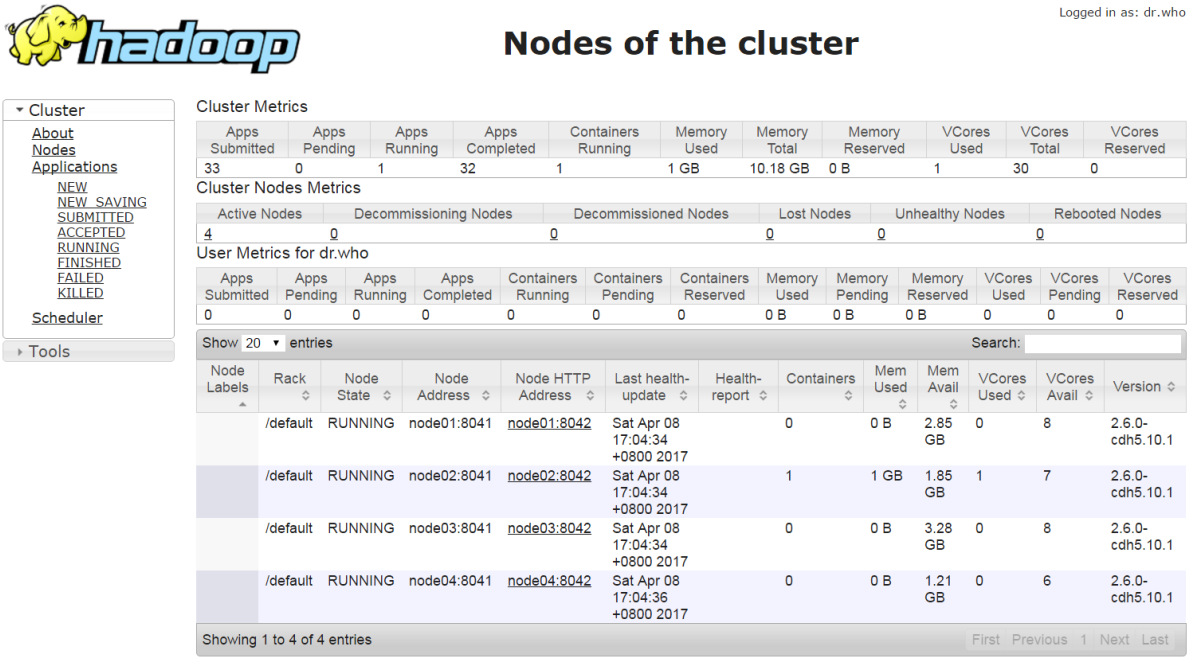

Figure 3.10: Nodes of Hadoop Cluster

By the visualization Hue web user interface, the thing of querying records inside the Hive table become much more intuitive. Hue not only supports to manipulate data in Apache Hadoop ecosystem, but also provides corresponding dynamical search dashboard with Solr. The most important is that it support interactive query of HiveQL and Impala. In Figure 3.11 and Figure 3.12, we can see the visualization interface of Hue.

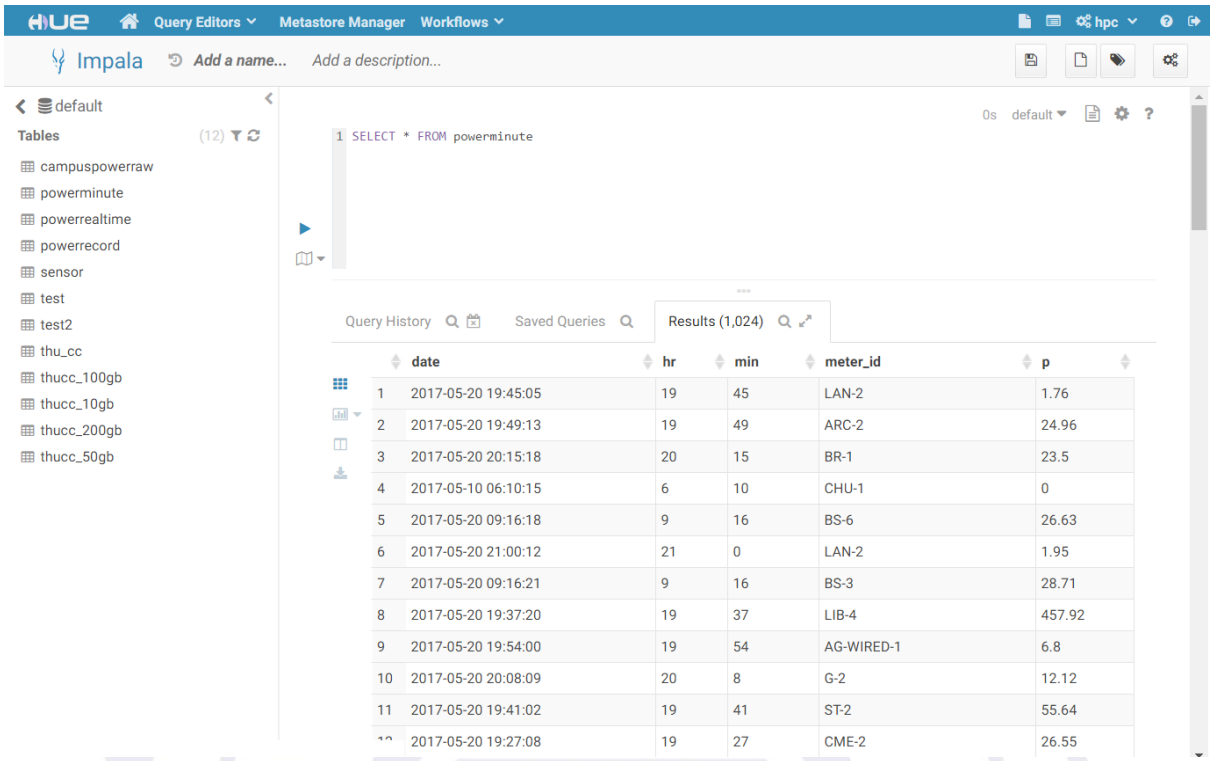

Figure 3.11: Hue Web User Interface

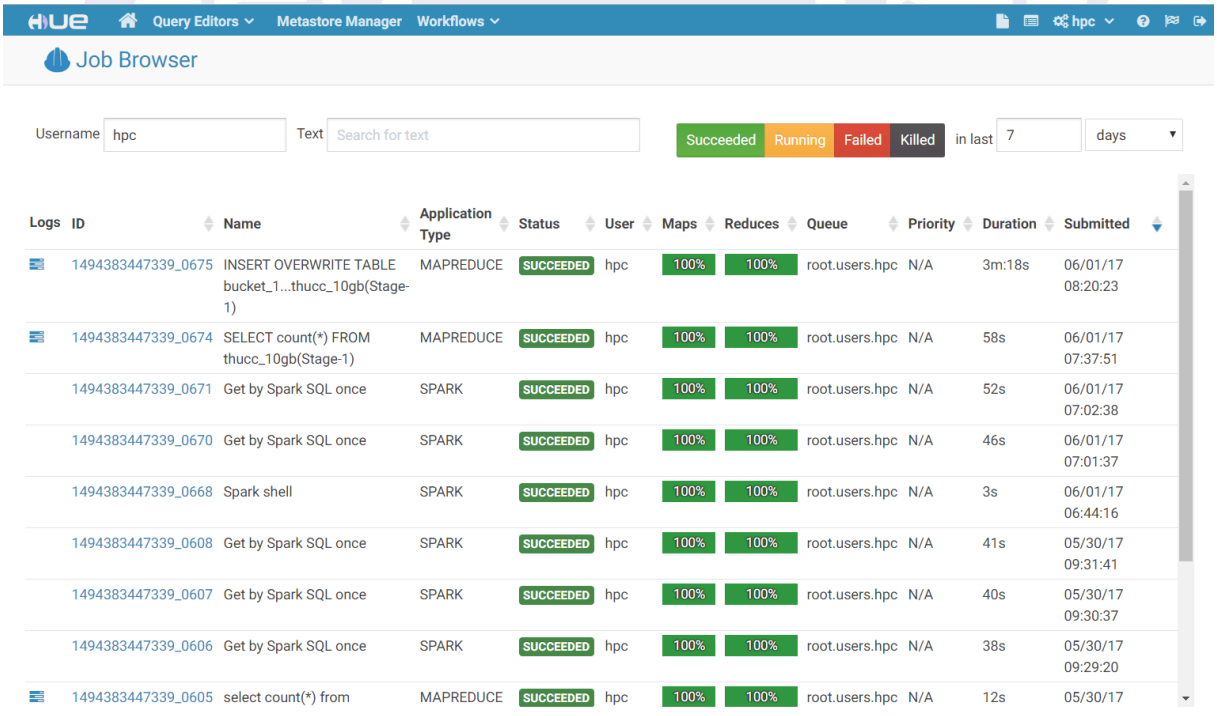

Figure 3.12: Hue Job Browser

Chapter 4

Evaluation and Experimental Results

In this section, the experimental environment and results of the proposed cloud intelligent campus energy monitoring system are described. After building the proposed system, we have collected about 400GB of data. In order to make the evaluation and experiments easy to be measured, the data size has been divided into 10GB, 50GB, 100GB, and 200GB. In section 4.1, we introduce the experimental environment and implementation of the proposed system. Sections 4.2 to 4.5 show the performance tests for verifying the efficiency of the system.

4.1 Experimental Environment

This section presents our hardware and software environmental environment. The proposed system is implemented with eight physical servers connected by Gigabit Ethernet to build a computing cluster. Each physical server consists of Intel Core i7 CPU with 16 GB Memory and 1TB HD. Besides, Ubuntu 14.04 LTS is adopted as our operating system. Also, the newest version of Hadoop 2.6.0-cdh5.10.1, Spark 2.1.0-cdh5.10.1, Sqoop 1.4.6-cdh5.10.1, Hive 1.1.0-cdh5.10.1, and Impala 2.7.0-cdh5.10.1 in Cloudera Manager 5.10.1 are installed, as shown in Table 4.1 and Figure 4.1.

Table 4.1: Experimental Environment

ID	CPU	RAM	HDD	Num of Cores
1	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	8
2	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	8
3	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	8
4	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	8
5	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	8
6	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	8
7	Intel® Core™ i7-4770@3.40GHz	16GB DDR3	1TB	8
8	Intel® Core™ i7-6950X@3.00GHz	128GB DDR3	2TB	10

Figure 4.1: CDH Computing Cluster

4.2 Performance Evaluation of Record Counting Speed between Table Created by Hive and Impala

In the first experiment, we used two kinds of framework including Hive and Impala to figure out how performance-intensive a record counting would be, when table is created in Hive and Impala respectively. The difference of table being created in Hive and Impala is that Impala can do the file compression with a better way. For the certification perspective, we tested it with Impala SQL COUNT Function in tables with different number of record rows. The range of testing data number is from 56 to 1120 million.

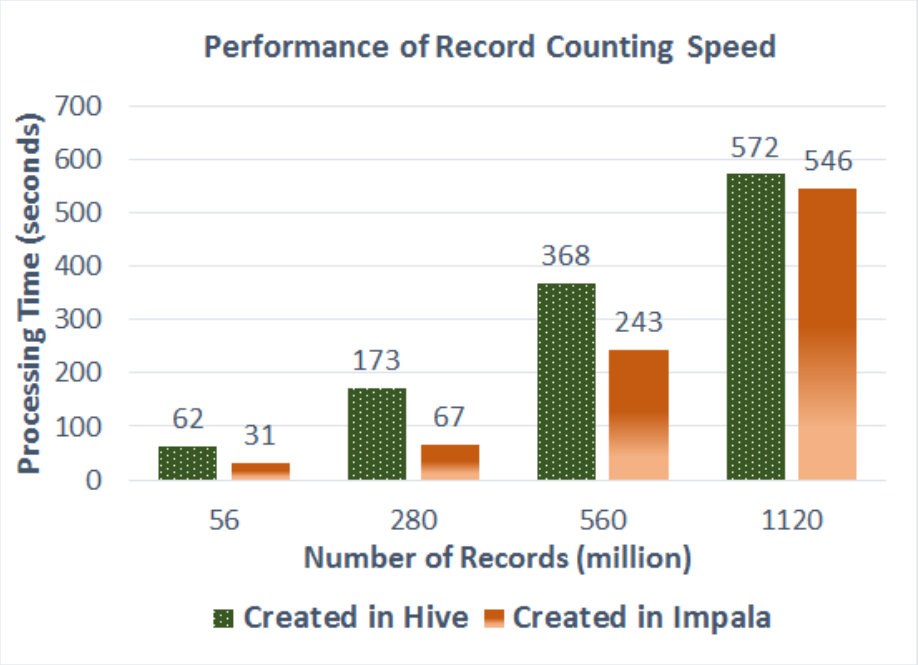

Figure 4.2: Comparing Record Counting Speed of Table Being Created in Hive and Impala Respectively

After the evaluation, Figure 4.2 shows the result of record counting speed. As our expected, the processing time of table being created in Impala shows a better efficiency than table being created in Hive. Table which is created in Hive has spent more time than Impala in any number of records, due to it is using MapReduce as the hood of Hive. In contrast, Impala daemon processes are started at boot time, and each Impala node caches all of its metadata to reuse for future queries against the same table. Thus, Impala is always ready to execute a query.

4.3 Performance Evaluation of Querying in a Single Condition among HiveQL, Spark SQL, and Impala SQL

The response time of data search plays an important role on real-time power management system. Therefore, the efficiency of querying records in different conditions should be evaluated. In order to verify the performance among three modules in detail, a single condition query experiment was being derived from different scales of data. The comparison of HiveQL, Spark SQL and Impala SQL searching speed in a single condition query as shown in Figure 4.3.

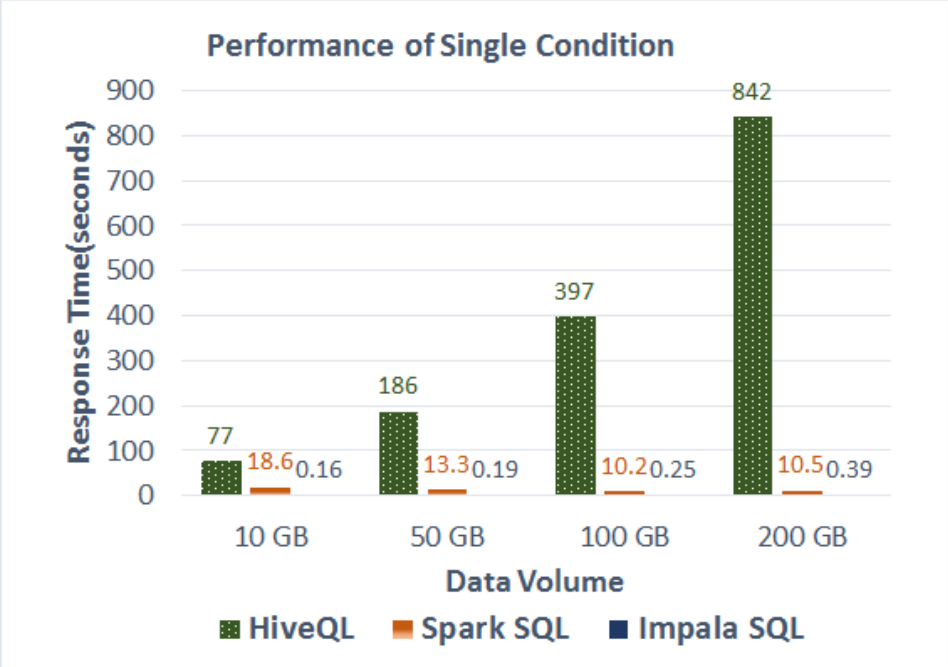

Figure 4.3: Comparison of HiveQL, Spark SQL, and Impala SQL Searching Speed in a Single Condition Query

The result as shown in Figure 4.3, Impala SQL still have a numerous gap corresponding to HiveQL and Spark SQL. The essential difference of Impala SQL with HiveQL and Spark SQL is that Impala SQL generates less segment files than HiveQL and Spark SQL while running a query task in the same cluster. As a result, the process time of shuffle is going to spend less than the other modules.

4.4 Execution Time of ETL Application

For the efficiency of ETL process in data warehousing, the execution time of ETL application is crucial as well. The speed of data ETL will immediately affect the speed of data visualization. For the following experiment shown in Figure 4.4, we developed two ETL applications in Hive and Spark to extract data from data resource, transfer format, and load into Hive table. By intuitively comparing the execution time of each application to know the best efficient ETL tool is.

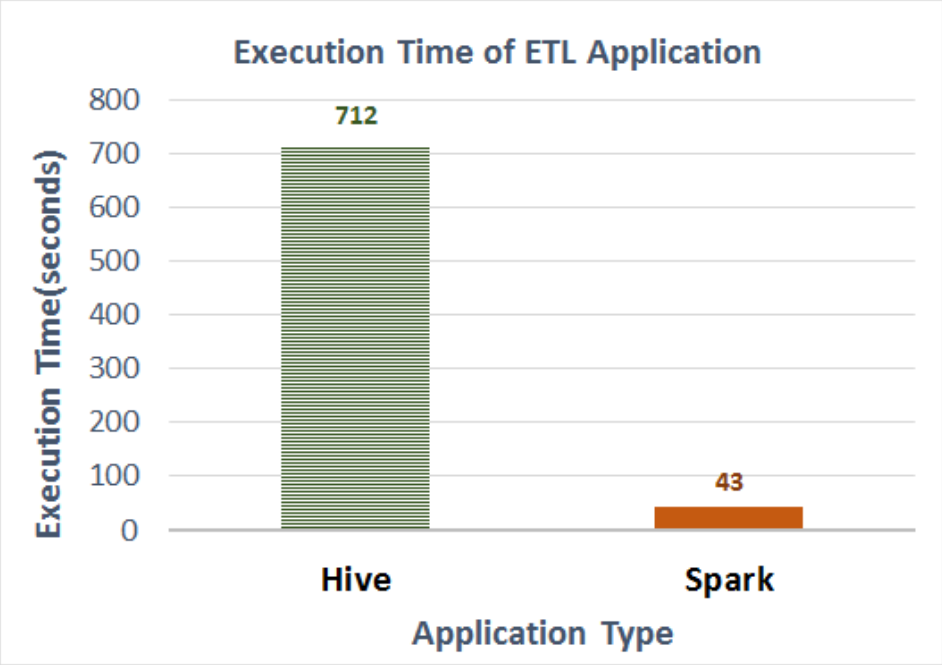

Figure 4.4: Execution Time of ETL Application between Hive and Spark

As Figure 4.4 can see, we noticed that Spark is quick more than sixteen times for ETL application. As a result, the latency while ingesting data would decrease to one minutes inside.

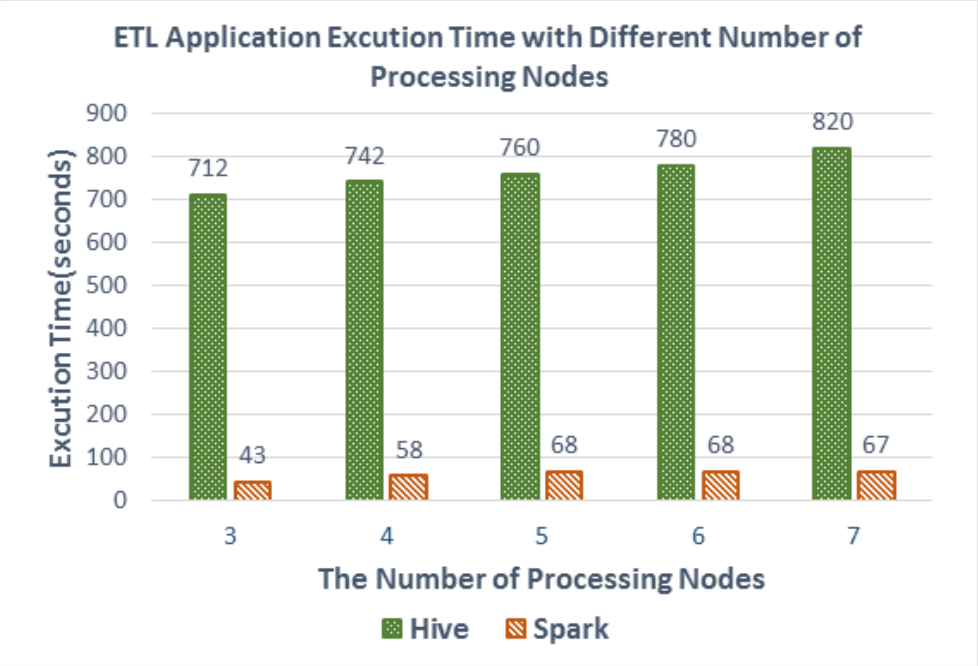

Figure 4.5: Execution Time of ETL Application with Different Number of Processing Nodes

In order to highlight how the slave node number would cause processing time difference, we executed the same ETL application in the same cluster with different number of slave nodes. In Figure 4.5 can observe that with the number of processing node increasing, the more

execution time would be consumed. When Hadoop is getting data from external data resources, it used to take three copies in default before storing them into HDFS. It means Hadoop must split the data into more nodes to execute pipeline mission. The more slave node in cluster, the more time latency while connecting.

4.5 Processing Time of Statistical Procedure

In general, most of website would put calculating or statistical procedures at the website server and use several script languages to compute the information they need. As the purpose to achieve a rapid presentation on front-end website, those statistical procedures must be compute quickly in the background cluster. In the front-end website of the proposed system as shown in Figure 4.6, a history data collection service need to calculate power data in each minute, hour, day, month, and year. Thus, we compared the statistical procedure processing time of Hive and Spark on background cluster. In addition, we have measured the processing time in different numbers of computing node.

Figure 4.6: Front-end Website of the Proposed System

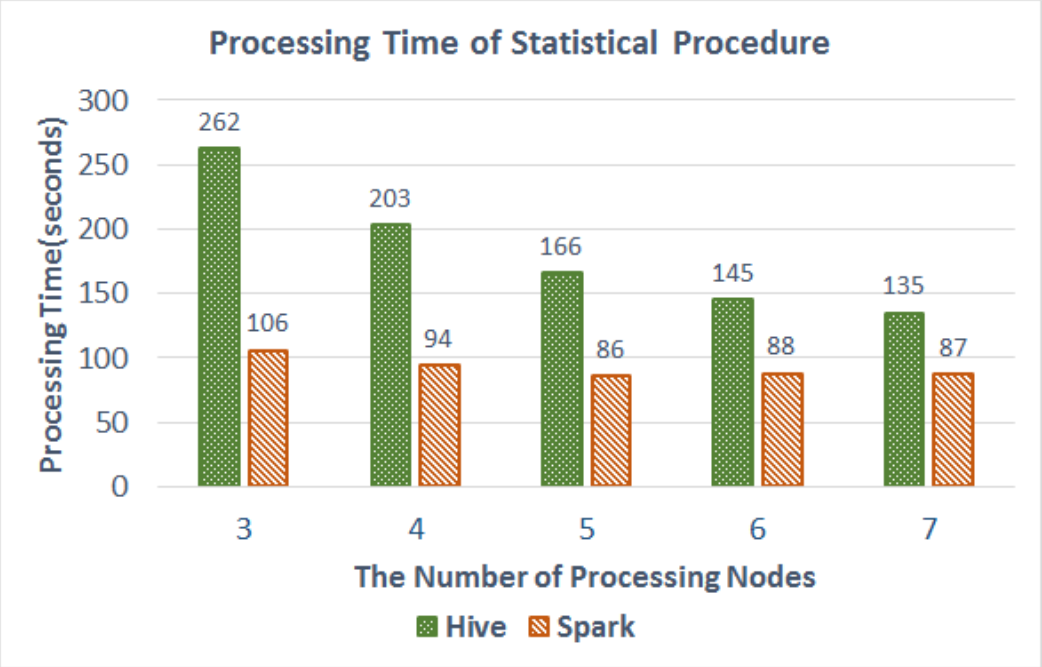

Figure 4.7: Processing Time of Statistical Procedure in Different Numbers of Processing Nodes

Figure 4.7 shows the processing time of statistical procedure using different numbers of computing hosts. We noticed that processing time of Spark is gradually approaching a stable status in three computing resource, however, Hive is still dropping down and spent far more time than Spark.

Finally, after the previous experiments we observed that the number of processing node will directly affect the processing of data ETL and statistical query. Thus, we also compared the performance trend between read/write manipulation on Hive and Spark, the result is as shown in Figure 4.8 below.

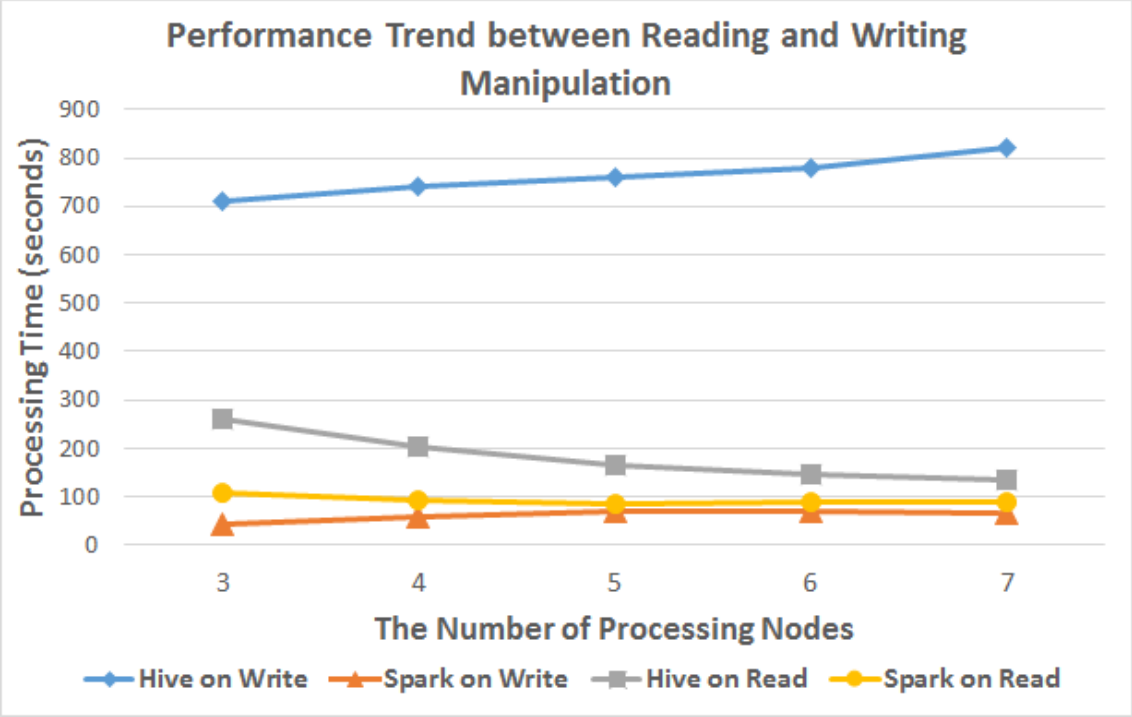

Figure 4.8: Performance Trend between Read/Write manipulation on Hive and Spark

In Figure 4.8, we can figure that the processing time of write manipulation is keeping on rising as the number of processing node increasing. That is because the more processing node will cause the requirement to build pipeline to each DataNode. In contrast, read manipulation do not have this problem, the more computing resource cluster has, the less processing time spend.

4.6 Resource Utilization

In the second experiment, we tested the query speed in a single condition among HiveQL, Spark SQL, and Impala SQL. In terms of resource utilization, the processing time of multiple condition query is long enough for presenting the resource utilization of cluster. Besides, Spark and Impala are both memory intensive framework. Thus, we compared the memory usage of these two frameworks, by presenting the experimental utilization results for the same manipulation statement in both frameworks.

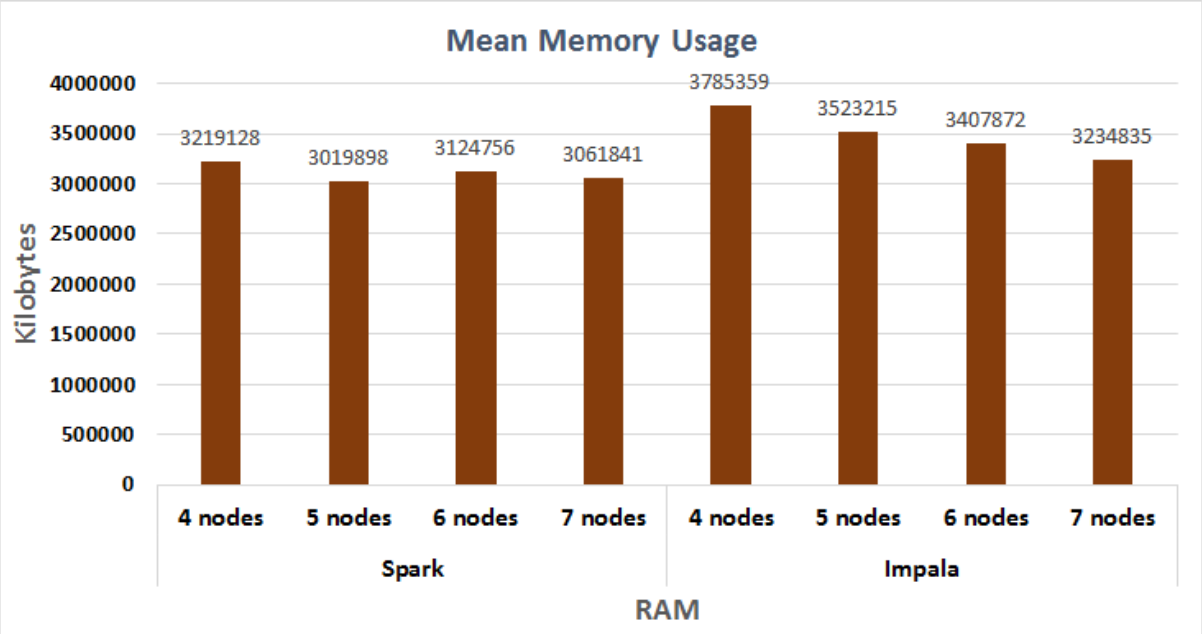

Figure 4.9: Memory Utilization

In Figure 4.9, we see the KiloBytes of memory needed for current workload in relation to the total amount of memory. In this way, we can clearly see that Impala does better memory usage than Spark does. As we mentioned before Impala uses more effectively the main memory and achieves better performance.

Chapter 5

Conclusions and Future Works

5.1 Concluding Remarks

In this work, we built a big data warehouse of power data and ETL processing for data warehousing with several big data modules. The proposed system includes Hadoop (as storage subsystem), Hive (as data warehouse), Spark (as data ETL tool), and Impala (as big data search engine) from bottom to top. The generic power-data source is provided by the so-called smart meters equipped in real field. The data collection and storage are handled by the Hadoop subsystem and the data ingestion to Hive data warehouse is conducted by the Spark unit. In order to evaluate the query-response performance of our data warehouse, several tests had been done in different data volume.

The major contributions of this research can be concluded by two aspects: (1) The open and non-proprietary multi-layer software modules are selected systematically. The design goal had been approached by constituting the real-time power-monitoring platform embedded with some excellent characteristics of high efficiency, high feasibility and low cost. Moreover, a big data warehouse and ETL process of cleansing, customization, reformatting, integration, and are inserted into the proposed data warehouse. (2) The requisite experiments had been conducted to verify the query-response efficiency, and performance evaluations for the proposed power data processing platform, which reveals the high feasibility for the target research goals.

5.2 Future Works

In the future work, we are going to finish the ODBC connection between front-end website and the data warehouse that we had built in this work. In that way, an integral real-time power data analytic platform we proposed can be fully achieved. Besides, we will also improve the system by adding different categories of data, such as semi-structure data (like system logs or service logs). Finally, to confirm the system's scalability we will add more hosts and observe the condition of each hosts to prevent additional incident happened.

To enrich the proposed design work, it is expected that several system parameters like the crowd movement, temperature, and humidity of the target building can be sensed and collected to proceed similar processing. After integrating and analyzing these system parameters together,

it is anticipated that more practical and accurate results would be approached other than just using only parameter, the power(load).

References

[1] Cloudera, “Impala,” Cloudera Company, 2014. [Online]. Available: http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html.

[2] Making Hadoop Easy with Cloudera Manager, 2017. https://www.cloudera. com/products/product-components/cloudera-manager.html.

[3] Diane J Skiba. The Internet of Things (IoT). Nursing Education Perspectives, 34(5):63 – 64, 2015.

[4] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. Mobile Networks and Applications, 19(2):171–209, 2014.

[5] Tom White. Hadoop: The definitive guide. “O’Reilly Media, Inc.”, 2012

[6] The Four V’s of Big Data, 2017. http://www.ibmbigdatahub.com/ infographic/four-vs-big-data.

[7] Apache Hadoop, 2017. http://hadoop.apache.org/.

[8] Jens Dittrich and Jorge-arnulfo Quian. “Efficient Big Data Processing in Hadoop MapReduce,” Proceedings of the VLDB Endowment, 5(12):2014– 2015, 2012.

[9] Mapreduce, 2014. https://hadoop.apache.org/docs/r2.6. 0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/ MapReduceTutorial.html.

[10] Dhruba Borthakur. The Hadoop Distributed File System: Architecture and Design, 2014. http://hadoop.apache.org/docs/r2.6.0/ hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[11] Farag Azzedin. Towards a Scalable HDFS Architecture. In Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013, pages 155–161, 2013.

[12] Apache Hive, 2014. https://hive.apache.org.

[13] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy.” Hive - A Petabyte Scale Data Warehouse Using Hadoop”. Proceedings - International Conference on Data Engineering, pages 996–1005, 2010.

[14] Apache Sqoop, 2017. http://sqoop.apache.org/.

[15] Sahithi Tummalapalli and Venkata rao Machavarapu. Managing MySQL Cluster Data Using Cloudera Impala. Procedia Computer Science, 85:463–474, 2016.

[16] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. HotCloud, 10:10–10, 2010.

[17] Spark SQL, Dataframes and Datasets Guide, 2016. http://spark.apache.org/docs/2.0.1/sql-programming-guide.html.

[18] K. Li, F. Su, X. Cheng, W. Chen, and K. Meng. “The Research of Performance Optimization Methods Based on Impala Cluster”. p. 336–341, 2016.

[19] Xiufeng Liu and Per Sieverts Nielsen. “A Hybrid ICT-solution for Smart Meter Data Analytics”. Energy, 115, Part 3:1710 – 1722, 2016. Sustainable Development of Energy, Water and Environment Systems.

[20] Shaker H. Ali El-Sappagh, Abdeltawab M. Ahmed Hendawi, and Ali Hamed El Bastawissy. “A Proposed Model for Data Warehouse ETL Processes”. Journal of King Saud University - Computer and Information Sciences, 23(2):91 – 104, 2011.

[21] Chao-Tung Yang ; Yin-Zhen Yan ; Ren-Hao Liu ; Shuo-Tsung Chen. Cloud City Traffic State Assessment System Using a Novel Architecture of Big Data. 2015 International Conference on Cloud Computing and Big Data (CCBD), 2015.

[22] D. Wang and Q. Zhou. “A method of Distributed On-line Analytical Processing of Status Monitoring Big Data of Electric Power Equipment.” Zhongguo Dianji Gongcheng

Xuebao/Proceedings of the Chinese Society of Electrical Engineering, 36(19):5111–5121, 2016.

[23] I. Mavridis and H. Karatza. “Performance Evaluation of Cloud-based Log File Analysis with Apache Hadoop and Apache Spark”. Journal of Systems and Software, 125:133–151, 2016.

[24] Ren-Hao Liu and Chao-Tung Yang. “On Construction of an Energy Monitoring Service Using Big Data Technology for Smart Campus”. Master’s thesis, Dept. of Computer Science, Tunghai University, 2016.

[25] Chao-Tung Yang, Yin-Zhen Yan, Shuo-Tsung Chen, Ren-Hao Liu, Jean-Huei Ou, Kun-Liang Chen: iGEMS: A Cloud Green Energy Management System in Data Center. GPC 2016: 82-98.

[26] Shyam R., Bharathi Ganesh H.B., Sachin Kumar S., Prabaharan Poornachandran, Soman K.P., Apache Spark a Big Data Analytics Platform for Smart Grid, Procedia Technology, Volume 21, 2015, Pages 171-178, ISSN 2212-0173, http://dx.doi.org/10.1016/j.protcy.2015.10.085.

[27] Hameeza Ahmed, Muhammad Ali Ismail, Muhammad Faraz Hyder, Syed Muhammad Sheraz, Nida Fouq, Performance Comparison of Spark Clusters Configured Conventionally and a Cloud Service, Procedia Computer Science, Volume 82, 2016, Pages 99-106, ISSN 1877-0509, http://dx.doi.org/10.1016/j.procs.2016.04.014.

[28] Awais Ahmad Suengmin Rho M. Mazhar, Anand Paul. Urban planning and building smart cities based on the Internet of Things using Big Data analytics. Computer Networks, 2016.

Appendix A

Cloudera Manager Installation and Configuration of CDH Environment

I. Download the newest version of Cloudera Manager from official website.

wget http://archive.cloudera.com/cm5/installer/latest/cloudera-manager-installer.bin

II. Modify the authority of Installation file, and start to install.

sudo chmod 775 cloudera-manager-installer.bin

sudo ./cloudera-manager-installer.bin

III. Set up NTP server on Linux

sudo apt-get install –y ntp

ntpdate –s ntp.ubuntu.com pool.ntp.org

IV. Open up any explore and type http://masterIP:7180 in address bar to go to Cloudera Manager User Interface Website(Login with default username and password)

V. Specify hosts(with hostname or IP address) for CDH cluster

VI. Make sure it have got the situation of hosts

VII. Choose CDH version (Basically recommend default setting) and additional parcels

VIII. Providing SSH login credentials with sudoer user of master node

sudo vim /etc/sudoer

IX. Choose the CDH services that you want to install on cluster

X. After finishing installation process, the status of each cluster will be monitored on the Cloudera Manager website by logging in with the same username, password, and port.

Appendix B

Transferring Data from Relational Database to Hadoop by Using Apache Sqoop

As Figure 2.7 shows the data flow of Apache Sqoop, an application for transferring data between relational databases and Hadoop.

XI. Deploy mysql JDBC connector.

Download mysql-connector from the following address.

wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.41.tar.gz

Uzip it into sqoop directory.

tar -zxf mysql-connector-java-5.1.41.tar.gz

sudo cp mysql-connector-java-5.1.41/mysql-connector-java-5.1.41-bin.jar /opt/cloudera/parcels/CDH-5.10.1-1.cdh5.10.1.p0.10/lib/sqoop/lib/

XII. Try to print a table with Sqoop command.

Table A.1 Sqoop Arguments

Command	Function
--connect [jdbc-uri]	Jdbc connect URL
--username [mysql-username]	MySQL login username
--P	MySQL password
--table [mysql-table-name]	MySQL table
--hive-import	Import to Hive
--hive-table [table-name]	Choose hive table to import to
--target-dir	Target Hive diretory
--split-by [column-name]	Split by column name
-m	Split into how many Map processes

sqoop list-databases --connect jdbc:mysql://MySQL_DB_IP:3306/power --username hpc –P

```
hduser@master:~$ sqoop list-databases --connect jdbc:mysql://120.109.150.175:3306/power --username hpc --P
Warning: /home/hduser/sqoop/../hcatalog does not exist! HCatalog jobs will fail.
Please set $HCAT_HOME to the root of your HCatalog installation.
Warning: /home/hduser/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
Warning: /home/hduser/sqoop/../zookeeper does not exist! Accumulo imports will fail.
Please set $ZOOKEEPER_HOME to the root of your Zookeeper installation.
17/03/25 15:25:05 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6
Enter password:
17/03/25 15:25:08 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
information_schema
ServerMonitor
environment
mysql
performance_schema
phpmyadmin
power
testRH
testSC
wordpress
```

Figure A.1: List All Database in MySQL Database Using Sqoop

XIII. Transferring table from relational database in MySQL to Hive data warehouse, and store as another table with same content

sqoop import --connect jdbc:mysql://MySQL_DB_IP:3306/power --username hpc -P --table PowerDaily --hive-import --hive-table PowerDaily --target-dir /user/hive/warehouse/PowerDaily --split-by no -m 1

```
17/03/25 20:25:14 INFO mapreduce.Job: The url to track the job: http://master:80
88/proxy/application_1490438530741_0001/
17/03/25 20:25:14 INFO mapreduce.Job: Running job: job_1490438530741_0001
17/03/25 20:25:22 INFO mapreduce.Job: Job job_1490438530741_0001 running in uber
 mode : false
17/03/25 20:25:22 INFO mapreduce.Job:  map 0% reduce 0%
17/03/25 20:25:28 INFO mapreduce.Job:  map 100% reduce 0%
17/03/25 20:25:28 INFO mapreduce.Job: Job job_1490438530741_0001 completed succe
ssfully
17/03/25 20:25:28 INFO mapreduce.Job: Counters: 30
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=133021
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=87
                HDFS: Number of bytes written=5297
```

Figure A.2: Importing Data from MySQL to Hive Database Using Sqoop MapReduce Process

```
17/03/25 20:25:32 INFO hive.HiveImport: SLF4J: Found binding in [jar:file:/home/
hduser/hive/lib/log4j-slf4j-impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.cl
ass]
17/03/25 20:25:32 INFO hive.HiveImport: SLF4J: Found binding in [jar:file:/home/
hduser/hbase/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.clas
s]
17/03/25 20:25:32 INFO hive.HiveImport: SLF4J: Found binding in [jar:file:/home/
hduser/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/S
taticLoggerBinder.class]
17/03/25 20:25:32 INFO hive.HiveImport: SLF4J: See http://www.slf4j.org/codes.ht
ml#multiple_bindings for an explanation.
17/03/25 20:25:32 INFO hive.HiveImport:
17/03/25 20:25:32 INFO hive.HiveImport: Logging initialized using configuration
in jar:file:/home/hduser/hive/lib/hive-common-2.1.1.jar!/hive-log4j2.properties
Async: true
17/03/25 20:25:35 INFO hive.HiveImport: OK
17/03/25 20:25:36 INFO hive.HiveImport: Time taken: 2.3 seconds
17/03/25 20:25:36 INFO hive.HiveImport: Loading data to table default.powerdaily
17/03/25 20:25:36 INFO hive.HiveImport: OK
17/03/25 20:25:36 INFO hive.HiveImport: Time taken: 0.65 seconds
17/03/25 20:25:37 INFO hive.HiveImport: Hive import complete.
17/03/25 20:25:37 INFO hive.HiveImport: Export directory is contains the _SUCCES
S file only, removing the directory.
hduser@master:~$
```

Figure A.3: Successfully Importing Data to Hive DB

XIV. Getting into Hive to check if the data has been imported.

hive

SELECT * FROM table;

```
210     2017-02-27      19329.8
211     2017-02-28      19538.3
212     2017-03-01      22825.8
213     2017-03-02      22951.4
214     2017-03-03      25333.6
215     2017-03-04      21565.8
216     2017-03-05      22156.5
217     2017-03-06      27038.8
218     2017-03-07      25577.1
219     2017-03-08      25689.6
220     2017-03-09      26787.1
221     2017-03-10      25918.3
222     2017-03-11      23244.8
223     2017-03-12      23826.7
224     2017-03-13      27833.8
225     2017-03-14      20496.5
226     2017-03-15      26482.7
227     2017-03-16      27085.1
228     2017-03-17      27281.6
229     2017-03-18      24337.6
230     2017-03-19      24515.9
231     2017-03-20      29756.8
232     2017-03-21      28903.7
233     2017-03-22      28472.8
234     2017-03-23      28990.6
235     2017-03-24      28869.4
Time taken: 1.701 seconds, Fetched: 235 row(s)
hive> select * from powerdaily;
```

Figure A.4: Full Table Scan Test

Appendix C

Programming Code

I. Power Data ETL Processing Code

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.SparkSession
import java.util._
import java.text.SimpleDateFormat
import org.json.JSONArray
import org.json.JSONObject
import scala.io.Source
import org.apache.log4j.{Level, Logger}

object CampusMinute {
  case class UnderAgeException(message: String) extends Exception(message)
  val rootLogger = Logger.getRootLogger()
  rootLogger.setLevel(Level.ERROR)
  val spark = SparkSession
    .builder()
    .appName("Campus power data in Minute")
    .config("spark.sql.warehouse.dir", "hdfs://master:9000/user/hive/warehouse")
    .enableHiveSupport()
    .getOrCreate()
```

```
import spark.implicits._
import spark.sql


implicit def everyMin(f: () => Unit): TimerTask = {
   return new TimerTask {
      def run() = f()
   }
}


def main(args: Array[String]): Unit = {
   val timer = new Timer()
   timer.schedule(everyMin(getPower), 1000L, 20000L)
}


def getPower() = {
   try {
      val url = Source.fromURL("Data Resource URL")
      val urlString = url.mkString
      val jArray = new JSONArray(urlString)
      for (i <- 0 until jArray.length()) {
         val jdata = jArray.getJSONObject(i)
         val pid = jdata.get("location")
         val P = jdata.get("KW").toString
         val totalP_H = jdata.get("totalKWH").toString
         val PF = jdata.get("total_pf").toString
```

```
val ch1_current = jdata.get("ch1_current").toString().toFloat
val ch2_current = jdata.get("ch2_current").toString().toFloat
val ch3_current = jdata.get("ch3_current").toString().toFloat
val I = ch1_current + ch2_current + ch3_current
val ch1_voltage = jdata.get("ch1_voltage").toString().toFloat
val ch2_voltage = jdata.get("ch2_voltage").toString().toFloat
val ch3_voltage = jdata.get("ch3_voltage").toString().toFloat
val V = ch1_voltage + ch2_voltage + ch3_voltage
val ch1_hz = jdata.get("ch1_hz").toString().toFloat
val ch2_hz = jdata.get("ch2_hz").toString().toFloat
val ch3_hz = jdata.get("ch3_hz").toString().toFloat
val ch1_pf = jdata.get("ch1_pf").toString().toFloat
val ch2_pf = jdata.get("ch2_pf").toString().toFloat
val ch3_pf = jdata.get("ch3_pf").toString().toFloat
val v12 = jdata.get("voltage12").toString().toFloat
val v23 = jdata.get("voltage23").toString().toFloat
val v31 = jdata.get("voltage31").toString().toFloat
val ch1THDi = jdata.get("ch1_THDi").toString().toFloat
val ch2THDi = jdata.get("ch2_THDi").toString().toFloat
val ch3THDi = jdata.get("ch3_THDi").toString().toFloat
val ch1THDv = jdata.get("ch1_THDv").toString().toFloat
val ch2THDv = jdata.get("ch2_THDv").toString().toFloat
val ch3THDv = jdata.get("ch3_THDv").toString().toFloat

val timeStamp = jdata.get("time_stamp").toString
```

```
            val time = timeFormat(timeStamp)
            val timeSplit = timeFormat(timeStamp).split(" |:|-")
            val YMD = timeSplit(0)+"-"+timeSplit(1)+"-"+timeSplit(2)
            val hour = timeSplit(3)
            val min = timeSplit(4)
            sql(s"INSERT INTO powerminute VALUES ('$YMD','$hour', '$min', '$pid', '$P')")
            sql(s"INSERT INTO campuspowerraw VALUES ('$pid','$time', '$P', '$totalP_H',
'$ch1_pf', '$ch1_voltage', '$ch1_current', '$ch1_hz', '$ch2_pf', '$ch2_voltage', '$ch2_current',
'$ch2_hz', '$ch3_pf', '$ch3_voltage', '$ch3_current', '$ch3_hz', '$v12', '$v23', '$v31', '$ch1THDi',
'$ch2THDi', '$ch3THDi', '$ch1THDv', '$ch2THDv', '$ch3THDv', '$PF')")
          }
          val today = Calendar.getInstance().getTime()
          println("done! " + today)
        } catch {
          case UnderAgeException(msg) => msg
        }
    }

    def timeFormat(time: String): String = {
      var sdf: SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
      var date: String = sdf.format(new Date(time.toLong))
      date
    }
}
```

II. Periodic Statistical Service Code(Power Load in Every Hour and Day)

```
def getHourPower() = {
    try {
      val df = spark.read.table("powerminute")
      val today = Calendar.getInstance().getTime() // Current date
      var sdf_d: SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd") //date format
      val day: String = sdf_d.format(today)
      val sdf_h: SimpleDateFormat = new SimpleDateFormat("hh")
      val hour: String = sdf_h.format(today)
      val url =
Source.fromURL("http://140.128.197.129:8080/rest/buildingMeter/powerUsage/") // data
resource
      val urlString = url.mkString
      val jArray = new JSONArray(urlString)
      for (i <- 0 until jArray.length()) {
        val jdata = jArray.getJSONObject(i)
        val pid = jdata.get("location")
        val avg = sql(s"SELECT AVG(p) FROM powerminute WHERE `date` = '$day'
AND `hour` = '$hour' AND `meter_id` = '$pid'").head().getDouble(0)
        sql(s"INSERT INTO powerhour VALUES ('$day','$hour', '$pid', round('$avg',2))")
      }
      println("done! " + day + " " + hour)
    } catch {
```

```
        case UnderAgeException(msg) => msg
      }
    }

def getDayPower() = {
      try {
        var sdf_d: SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd")          //date
format
        val day_start = Calendar.getInstance()       // Start date
        day_start.add(Calendar.DATE, -1)
        val yesterday: String = sdf_d.format(day_start.getTime())
        val day_stop = Calendar.getInstance()       // Start date
        val today: String = sdf_d.format(day_stop.getTime())
        val url =
Source.fromURL("http://140.128.197.129:8080/rest/buildingMeter/powerUsage/")
        val urlString = url.mkString
        val jArray = new JSONArray(urlString)
        for (i <- 0 until jArray.length()) {
          val jdata = jArray.getJSONObject(i)
          val pid = jdata.get("location")
          val avg = sql(s"SELECT date, p/1000 as P FROM powerhour WHERE `date` >=
'$yesterday' AND `date` = '$today'").head().getDouble(0)

          sql(s"INSERT INTO powerhour VALUES ('$today', round('$avg',2))")
        }
```

```
        println("done! " + today)
    } catch {
        case UnderAgeException(msg) => msg
    }
}
```