Implementation of Big Data Analysis System for EMS

[1]Junseok Park, [1]Sanghyuck Lee, [1]Hyunil Lee, [*1]Byungchul Lee

Selim TSG Inc., 62-16, Techno 1-ro, Yuseong-gu, Daejeon, 34014, Korea,
junseok@selim.co.kr, leesh@selim.co.kr, *jobbs@selim.co.kr, bclee@selim.co.kr*

Abstract

In order to reduce energy in home appliances (Legacy Devices), we have built a system that performs collection/storage/analytics of the wattage and environmental sensors of legacy devices on cloud-based BigData systems. To ensure the security and scalability of large size data processing, we selected and developed a suitable open software, and based on this have built a system that can analyze the wattage and sensor data of each device from 500 locally distributed households in real-time and non-real-time through a simulator. Such implementation can be utilized as a reference model in developing a system that collects and analyzes large size data that can be generated from future IoT.

Keywords: *Big Data, Cloud, Streaming, Analytics, EMS*

1. Introduction

With the aim of reducing energy in legacy devices, this is a study on developing an autonomic-controlled energy management platform that can control the energy consumption of legacy devices. This study is about a Big Data processing system that collects and analyzes the individual power and environment sensor data generated from multiple households.

The purpose of this study is to periodically collect Individual sensor values such as the power and temperature-humidity of home appliances through central cloud, and process the statistics and analytics on the collected data in real-time and non-real-time, presenting the efficiency of electricity usage through providing the user with the electricity usage pattern and the comparison on households and regions.

Recently in the Big Data field, various studies are conducted on collecting and analyzing streaming data in real time, and with the general computing environment including the Big Data field moving to Cloud, this study has also utilized Cloud in order to flexibly use the entire computing resource.

This paper demonstrates that Cloud resources can be used flexibly through organizing the environment for streaming data processing under the On-premises Cloud environment, and this can be used as a reference model in implementing a system that collects and analyzes large size data including IoT.

2. Big Data Processing Architecture for EMS

2.1. EMS

* Corresponding Author
Received: Oct. 28, 2015, Revised: Nov. 30, 2015, Accepted: Jan.17, 2016

The purpose of an Energy management system is to gather the power and environment sensor data from appliances installed in each households, and perform active control over legacy devices. In order for this, an A-EMS (Automous-EMS) device is installed in individual households and collects required information. The collected data is transmitted periodically to the EMS portal, and the statistics and analytics are performed on the electricity use and environment sensor data from individual home appliances of all households.

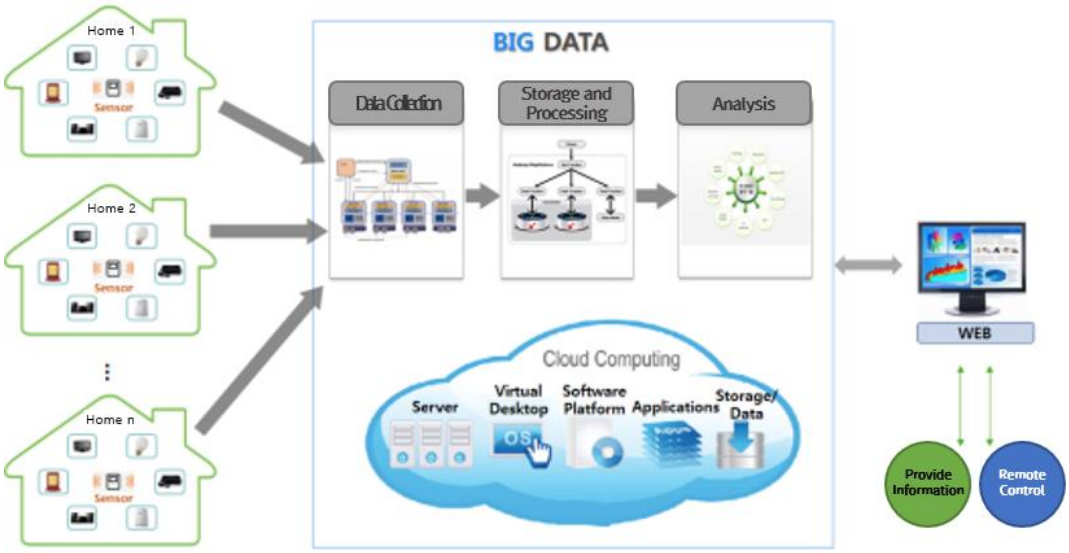

Figure 1. Big Data processing diagram for EMS

2.2. Big Data Processing Architecture for EMS

There are various systems for analyzing Big Data. In this study, we have used OpenStack established from on-premises Cloud, Mesos for data collection, storage, and analytics, and established multiple virtual machines in a cluster. Apache Kafka was used for the secure collection of large-scale data, and the collected data was either stored through HDFS or processed using the Spark Streaming engine. The analyzed results were stored in Redis and MySQL to enable monitoring on the web, and utilized Apache Zeppelin in the analysis process. The software stack for these series of processing is presented in Figure 2.

3. Software Components

The components of the open software used in implementation are OpenStack, Hadoop, Mesos, Marathon, Kafka, Spark, Docker, Zeppelin, Redis etc., and those that are able to collect and store large size data stably, and those that are suitable for advanced analytics and real time streaming processing were selected.

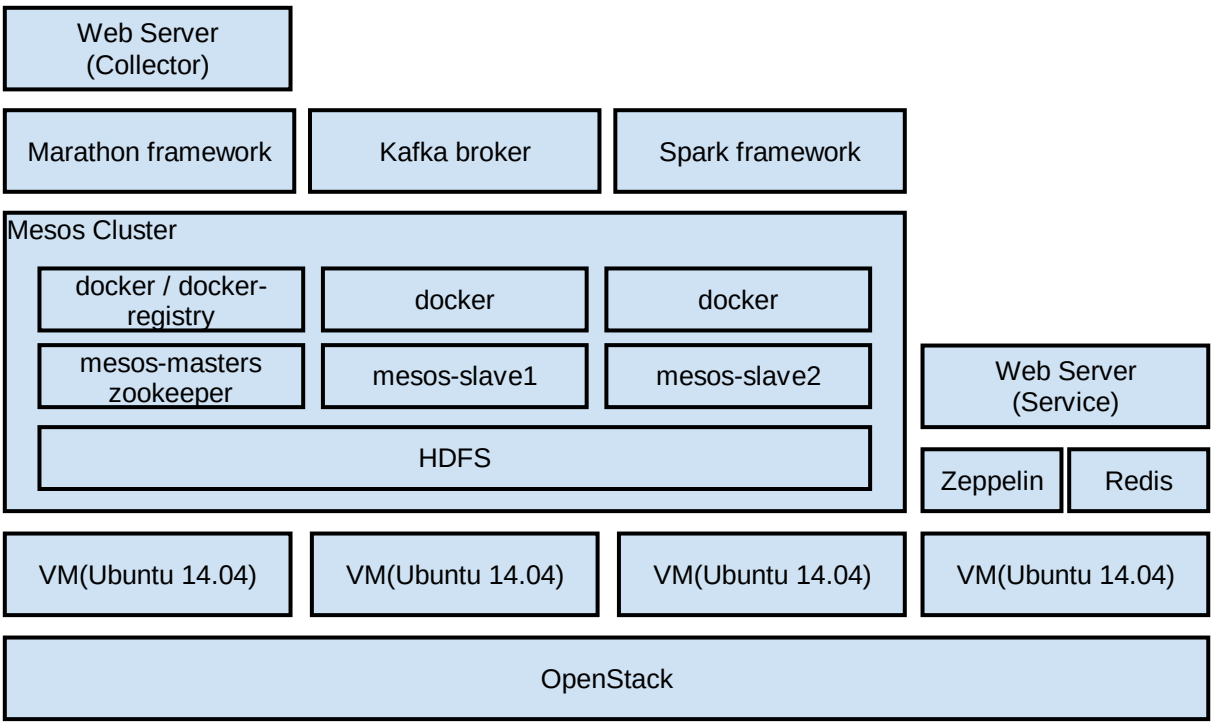

Figure 2. S/W stack of BigData analysis system for EMS

3.1. OpenStack

As a Cloud operating system, OpenStack controls the large-scale pool of, storage resources, network resources, and computing resources from data centers, and manages all of this through the web interface [1]. It is maintained and repaired by the OpenStack Foundation, and is distributed regularly twice a year under the apache license.

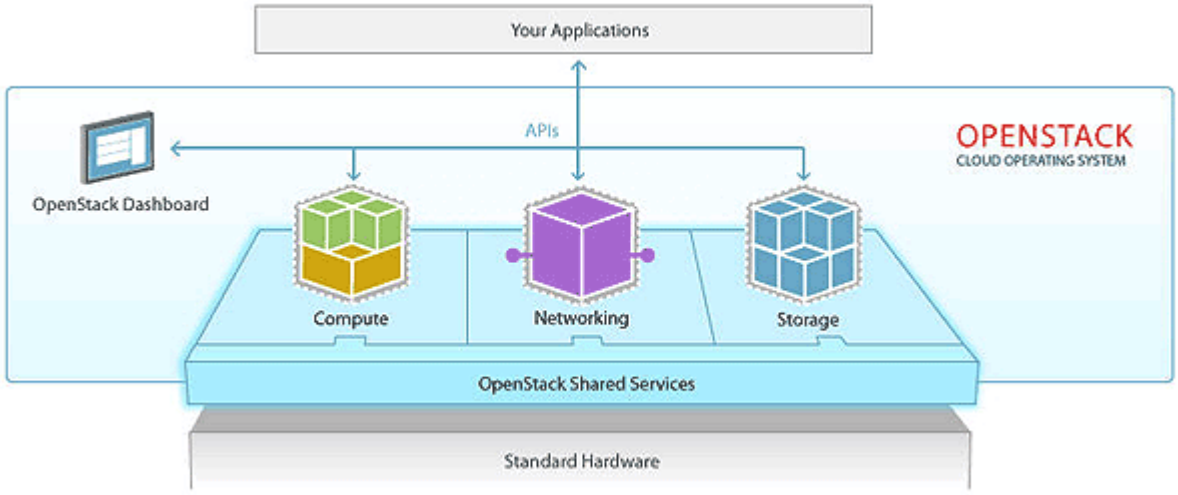

Figure 3. OpenStack Architecture

The basic structure of OpenStack is presented in Figure 3. After virtualizing all the Compute, Network, Storage resources of physical H/Ws, it is provided to users and this is managed in the Dashboard. Table 1. presents the Individual components of OpenStack with the code name, which is also actually a program command.

Table 1. Components of OpenStack

Function Component	***Code Name***	***Function***
Dashboard	horizon	As the web environment for the management of OpenStack, it performs most functions for Cloud by calling the API of other function elements.
Compute	nova	It is in charge of overall functions regarding the virtual machine. It uses hypervisor such as KVM, Xen, Hyper-V to generate and manage a virtual machine.
Network	neutron	It is in charge of the virtual network. It can generate private networks and subnets, and design a complete virtual network environment by creating a virtual router. It also provides the firewall and load balancing function.
Storage	cinder/ swift	Cinder is a block storage similar to a physical hard disk, and the volume generated by cinder can be attached to the virtual machine. Swift is a object storage similar to a web hard, and is generally composed of distributed file systems.
Identity	keystone	It performs an integrated management of functions regarding the user authentication.
Image	glance	It manages OS images for the virtual machine. Most Linux-based OS or Windows server can be used, and generally an OS for OpenStack is additionally provided.
Telemetry	ceilometer	It provides the monitoring and statistics function regarding system resources.
Orchestration	heat	It provides the automating function for automatically powering and composing the virtual machine
Database	trove	It provides the DBaaS(Database As A Service) function.
Data Processing	sahara	It provides Big data processing services such as Hadoop and Spark.

All services of OpenStack provide RESTful API, and various functions operate organizationally through this API, and also new services can be easily integrated.

OpenStack can be established in various ways, but it is generally consisted of Controller, Compute, Network, and Storage Nodes, and organizes the network according to each of their roles. The Management Network connects all these nodes and each Compute and Network nodes, and Compute and Storage nodes independently organize a Tunnel Network and a Storage Network, with Network nodes connecting with the external internet.

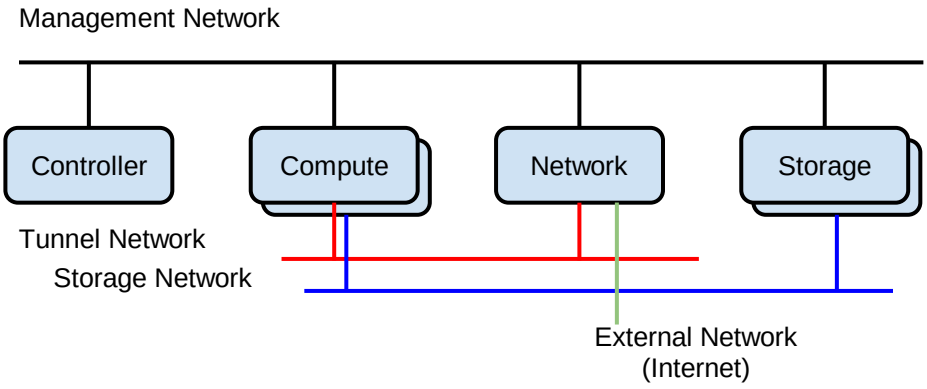

Figure 4. OpenStack Network Structure

3.2. Apache Mesos

Mesos is a distributed linux kurnel that operates distributed applications(ex, Hadoop, Spark, Kafka, ElasticSearch, MPI) by abstracting multiple computing resources(data center) as if into a single computer.[2]

When there is a server consisted of multiple hosts like Figure 4., it is general to use Cloud or Virtualization technologies to allocate VM and operate the necessary application. Mesos, on the other hand, perceives the entire server as a single computer and operates and manages the distributed applications as if it is a single process.

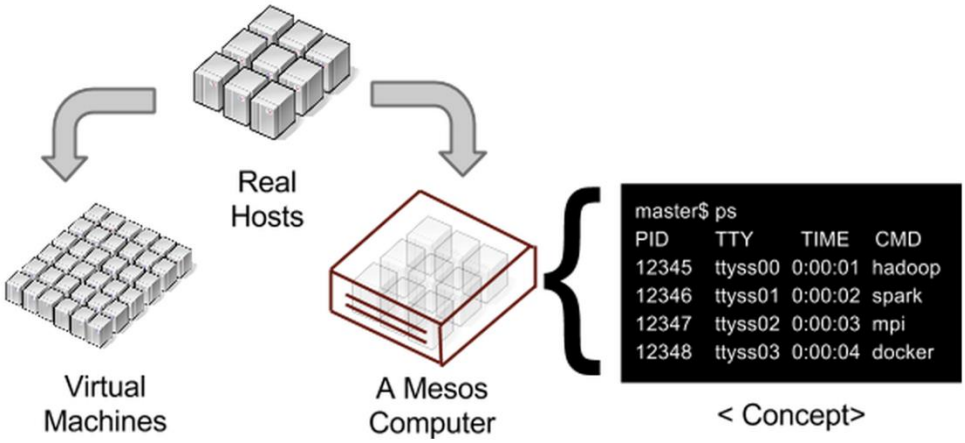

Figure 5. Conceptual operation of Mesos

Figure 6. demonstrates the results of how each application divides and uses the entire CPU, when 4 types of distributed application are operated in 100 nodes.

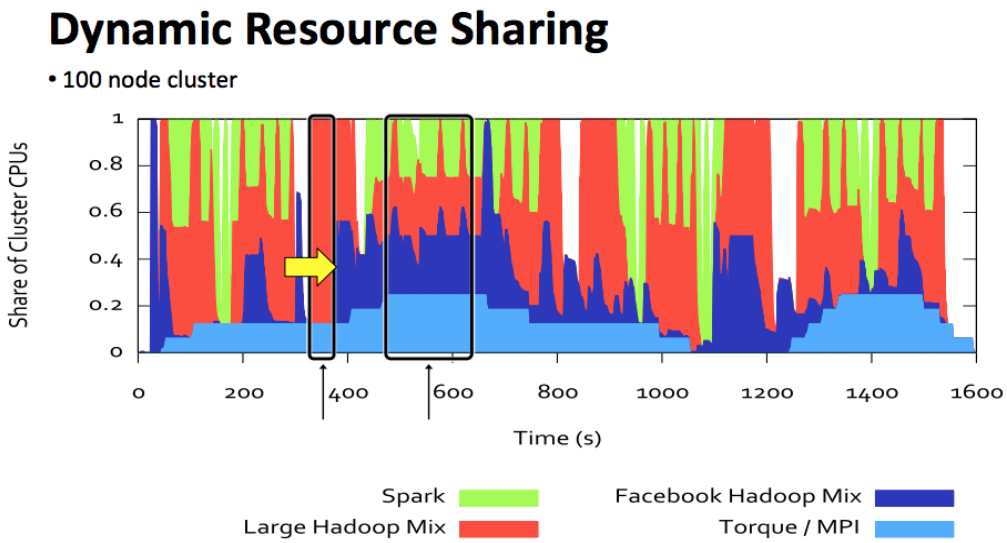

Figure 6. Example of dynamic resource sharing of Mesos

Mesos maintains a master-slave structure, and the master is coordinated by the ZooKeeper for high availability. The actual application operates in the slave nodes, and one slave node activates several scheduled distributed applications.

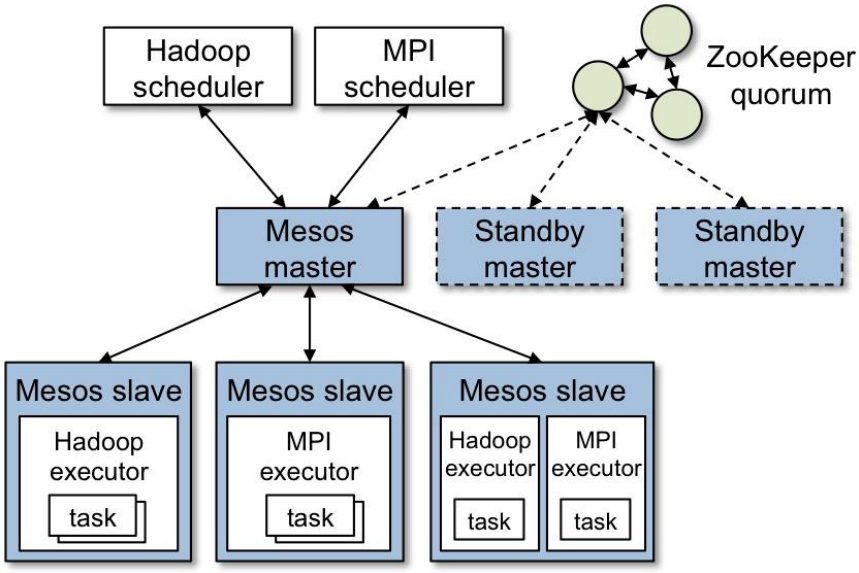

Figure 7. Operation Method of Mesos

For Mesos the actual application is activated by the framework. There are processing frameworks such as Hadoop, Spark, Storm, MPI, or frameworks like distributed data storage such as Cassandra, Hypertable, and these frameworks are utilized in executing the actual application code. For versatility like web services, there is the Marathon framework and since it can execute applications based on the Docker, it is able to perform distributed executions of many applications.

3.3. Apache Kafka

Kafka is a distributed messaging system that supports Publish, Subscribe, and Queue models, and it is in the spotlight as the solution for large-scale data collection [3]. It is an open source software developed by LinkedIn which is now an official Apache project. It is also widely used as a data collection engine for large size log analysis or Hadoop, and as a collection engine for real time streaming.

A messaging system is a system that exchanges information required for the inter-communication and connection of individual softwares in complex work environments where multiple servers and softwares are used together. The Publish/Subscribe model is a model where once a message is generated from a single Topic (keyword), message is transmitted to multiple receivers who have read this topic.

As in Figure 8. multiple producers and multiple consumers can exist in Kafka, and the middle Kafka broker forms a cluster consisted of multiple nodes, ensuring fast and secure mediation of large size messages. Also, a broker within the Kafka cluster is able to perform scaling in real time.

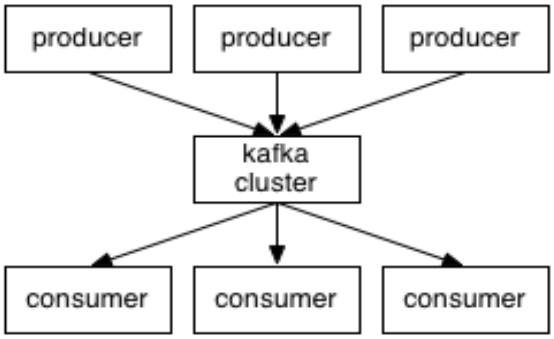

Figure 8. Composition of Kafka

A producer forms the generated data into a message, names a certain Topic, and delivers it to the cluster. The consumer which processes the actual message, takes from the cluster the amount of messages they can process and processes them. Kafka assigns offset numbers to messages on one topic in the message generation order, and it can create several partitions for duplicated storage.

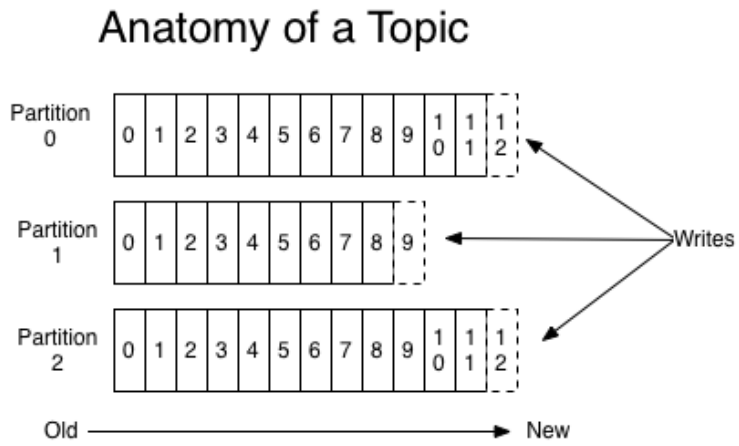

Figure 9. Message Processing Method of Kafka

While the Kafka producer generates messages in [topic name + partition number + content], the consumer which processes the message receives the message in [topic name + consumer group name]. All consumers must be part of a group, and all consumers in the group share the offset number. Therefore, in order to process large size data through multiple consumers, you can operate multiple consumers in one group.

3.4. Apache Spark / Spark Streaming

Apache Spark is a very fast and universal large-scale data processing engine [4]. Although it is often compared with the Hadoop ecosystem, as an in-memory method in some cases it even shows over 100 times the processing speed of Hadoop. Spark was open sourced in early 2010 by the AMPLab in Buckley, and graduated into an Apache top-level project in early 2014.

Spark provides functions such as the SQL (Shark) or machine-learning (MLLib) by integrating them into a library form, which in a Hadoop ecosystem requires the installation of an individual program. Also, provides preexisting functions such as the Spark Streaming function responding to Storm for real time streaming processing, the GraphX for social graph calculations, and the Interface with R (beta). In most cases by using R you can easily write distributed programs without particularly considering the distributed environment. By using the basic provided statistics functions you can easily obtain the average/count/max/dispersion/correlation-coefficient from the files of HDFS (Hadoop File System), and by using the machine-learning library you can easily use functions such as linear regression, classification, grouping, and recommendation.

Like any other distributed processing system, Spark is composed of the master and worker. The master, as the manager of the overall cluster, performs the role of distributing applications to worker nodes and is able to configure HA through the ZooKeeper. The worker represents the server that the actual data processing programs operate on.

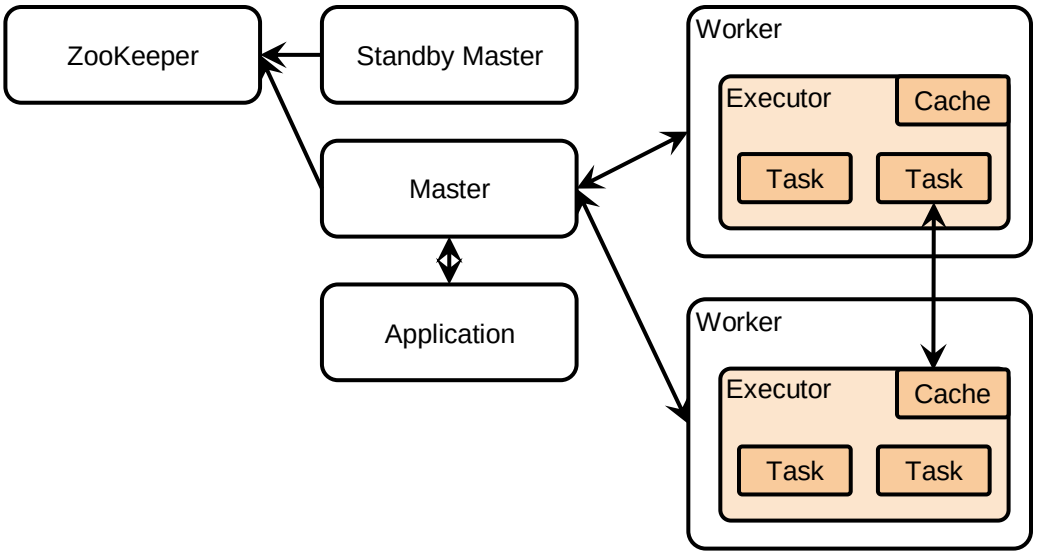

Figure 10. Structure of Spark

4. Implementation and Consideration

4.1. Construction of Infrastructure

Cloud, as presented in Figure 11, was implemented using the OpenStack Juno version. It was organized with one each of the Controller Node and Network Node, five Computer Nodes, two Block Storage Nodes, and three Object Storage Nodes. All nodes were connected through the Management Network (10.0.0.x), and the Network Node and Computer Node were organized in the Tunnel Network (10.0.1.x) with separate L2 switches. Also, the Block Storage Node, Object Storage Node, and Compute Node were organized with the Storage Network (10.0.2.x), and lastly the external internet connection was connected through the Network Node enabling data collection and web service.

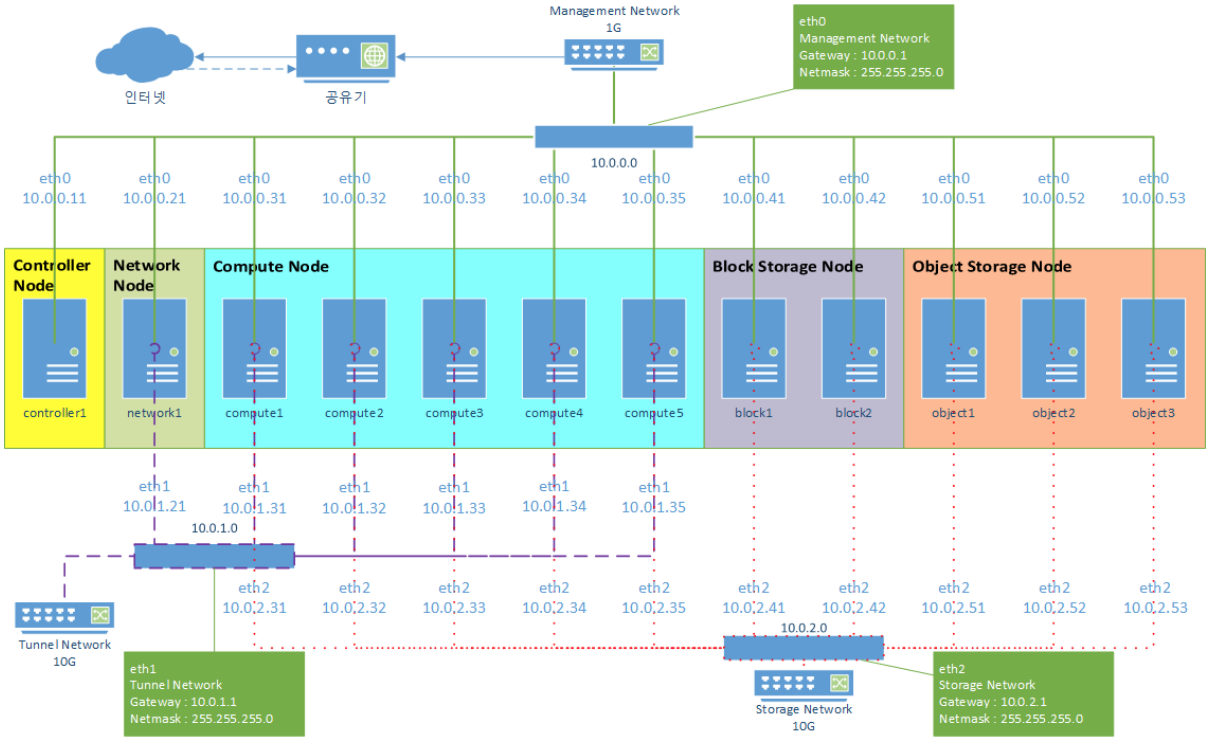

Figure 11. Structure of the server and network infrastructure for Cloud

4.2. System Implementation

4.2.1. Data Collection and Processing Flow

A simulator that generates virtual data with Poisson distribution and transmits data with a period of 15 seconds targeting 500 households was produced. The simulator sends data through the collection server based on the web of cloud, and transmits the Producer API of the Kafka in the form of message through the Kafka Broker. Actual data is composed of modules for processing streaming and saving modules in order to save as HDFS, and is implemented each by utilizing the Kafka Consumer API. This process is shown in Figure 12.

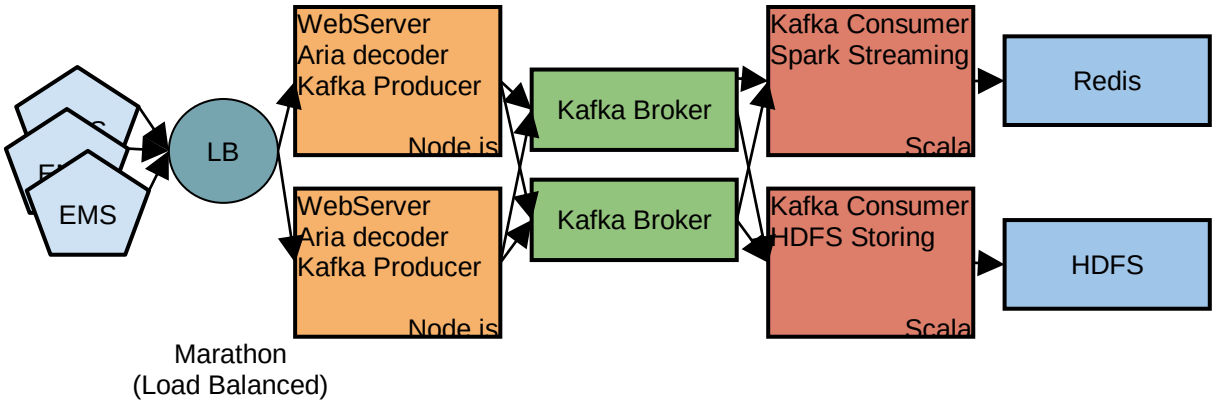

Figure 12. Data collection and processing flow

The data transmitted by the simulator shows the usage of after the precedent transmission of the TV, audio, set-top box, boiler, lights, heater, etc. in the measure of Watt. Also, the measurement of temperature, humidity, illumination and movement-detection is included. An example is shown in Figure 13.

```
<?xml version="1.0" encoding="UTF-8"?>
<ems>
	<emsid>010203</emsid>
	<time>2016.01.30 22:33:10</time>
	<TV-1>123</TV-1>
	<audio-1>123</audio-1>
	<settop-1>123</settop-1>
	<boiler-1>123</boiler-1>
	<lights-1>123</lights-1>
	<heater-1>123</heater-1>
	<total>123</total>
	<tp>20</tp>
	<hmt>30</hmt>
	<itc>40</itc>
	<occu>0</occu>
</ems>
```

Figure 13. Example of Data Format

The web server for data collection operates with the Marathon framework of Mesos. Therefore the data collection web server is containerized, and the docker-registry is organized separately for this [5]. Marathon provides an automatic restoration function on malfunction which helps secure stability, and the Load Balancer is embedded to support horizontal Scale-out, making it possible to efficiently respond to future increases in using households.

The transmission data is encrypted using the Aria algorithm, so the Aria decoder was implemented for decryption. Also it uses the Kafka Producer API to transmit messages to the Kafka broker with the Topic of “EMS”. The web server, Aria decoder and Kafka Producer were all implemented with node.js.

The Kafka Consumer is utilized in two modules. One is for saving in the HDFS and is utilized in the analysis system using the Spark ML(Machine Library), and the other is utilized in the module for real

time streaming processing[4][6]. Here, it was made possible to calculate the estimates and statistics of each household and region, save the data value in Redis, and check through the web browser. In the analysis stage this was realized through Apache Zeppelin, and graphs and maps were expressed utilizing D3 as visualization tools [7-8].

4.2.2. Monitoring Function of Manager

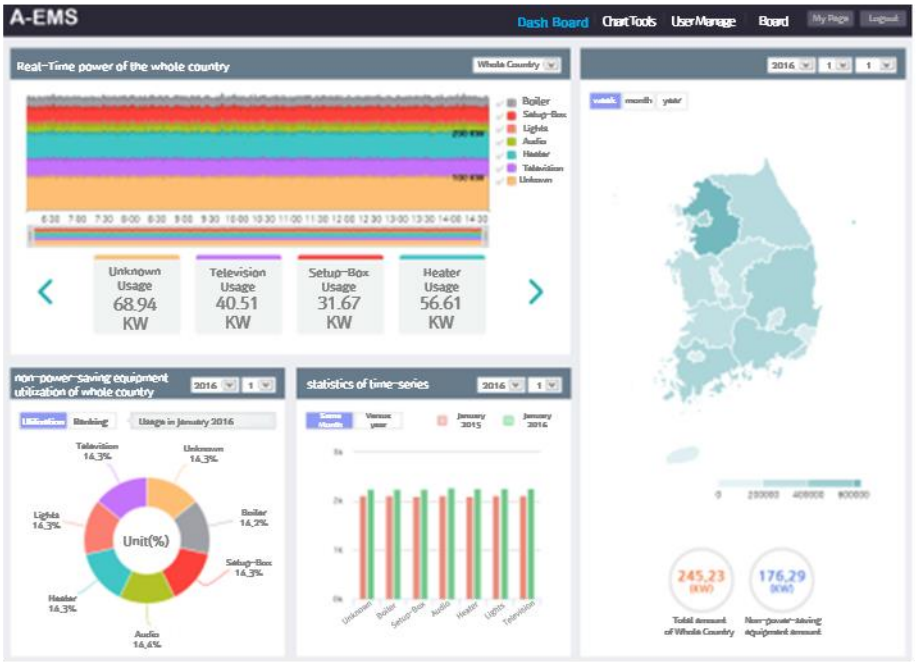

Figure 14. Manager Module Main Screen

A real time monitoring function of the regional/national electricity was implemented. This function makes it possible to see the trend of power by 15 seconds, in combining the real time electricity data coming in from each A-EMS by region and nationwide. Since the Unknown is basically not included in the electricity data, it extracted the Unknown by processing the received electricity data, and visualized them as equally as the legacy device. Because the electricity data of each household transmitted in real time does not actually include regional data, the region/nationwide combined data cannot be obtained by the data itself. Therefore in compensation, this was implemented in a method that combines the region/nationwide electricity data after linking the data serial number (MAC Address) to the user's registration info, after loading on Redis.

The region/nationwide legacy devices use rate function was implemented. This function enables you to see each legacy devices' use rate in percentage and use ranking of the particular month, which is secondly aggregated by region/nation. It uses Ajax as communication with the web screen, and was implemented to be altered in real time, whenever the year/month and use-percentage/use-ranking is changed.

The region/nationwide legacy device time-series function was implemented. This function enables the manager to see the statistics data of the daily-aggregated electricity data of each households, secondly aggregated by region/nationwide then thirdly aggregated in two ways by same month on last year or Year on Year. When each item of legacy device is clicked, depending on the same month on last year / Year on Year option a daily/monthly detail view pops up, and you can also see the national/regional legacy device usage map chart. In addition, a function enabling you to see the current

state of the national electricity usage in a single glance was implemented, which is performed by applying the aggregated data to the map after secondly aggregating by region/nation the daily aggregated electricity data of each households. On the complete map of Korea, the color transparency of each region is controlled by the electricity data volume, with the aggregated data separated by week/month/year. In order to see the area and household information of a particular region, the specific map of the region is shown in a pop-up form when a domain of the map is clicked, and by providing the data source of each area/household it enables you to see the electricity data altogether.

4.2.3. Monitoring Function of User

Figure 15. User Module Main screen

A function to monitor information within households in real time was implemented. It checks the development of electricity usage by visualizing real time electricity data flowing in from users' households' A-EMS per 15 seconds. Each household have a different number of legacy devices, and so a Logic and Chart Library module, which allows the expression of data dynamically, was invented. By expressing sensing data (other than electricity data) such as indoor temperature-humidity, illumination, and whether the user is in or not into a widget form, we made it possible to check immediately. Furthermore, a function that visualizes, on the basis of legacy devices' electricity data aggregated per day, the legacy devices' electricity usage of the period of month/year that the user chose was implemented. This function also provides an estimate of the electricity bill on the basis of an electricity bill calculating algorithm. When you click on the 'each legacy device' item and choose monthly, you will find the electricity usage of weeks 1~4 of the corresponding month expressed in the form of a column chart. If you choose yearly, you will find the electricity usage of month 1~12 of the corresponding year visualized.

A function that allows me to compare my electricity usage with other households was implemented. It makes an average of the electricity usage of other households with the same number of household members in the aggregate data on the basis of the information entered when joining the membership, and compares it to the user's electricity usage and electricity bill. A usage comparison screen of each device pops up when the item is clicked and a Stack Column Chart integrating each devices' usage comparison is put to make it possible to compare from an overall perspective.

4.2.4. Chart Creating Function

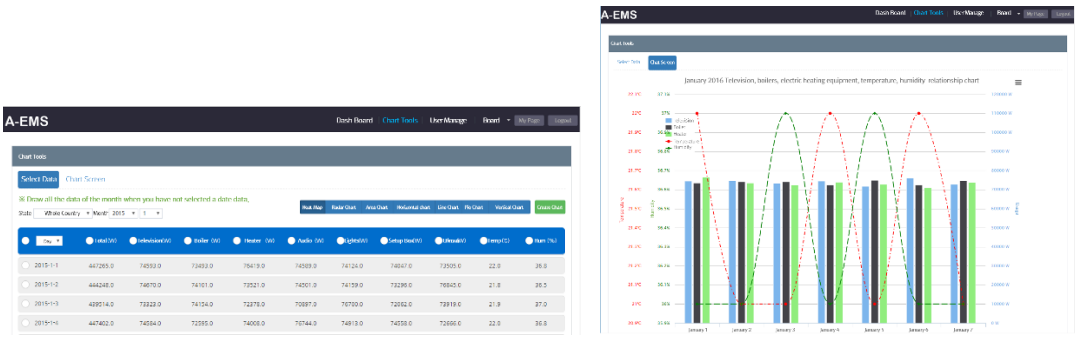

Figure 16. Chart Tool Screen

As shown Fig. 16, Chart tools able to be directly controlled by the user on the web were implemented. Chart tools are a function that loads the statistic data aggregated until the present point into a Grid form and enables manager and users to make a chart of their own by combining the data they want combines the data that the manager/user wants per column/row and to make a chart of his own. The standard data unit is days and years/months can be selected through the Selectbox to load data. The charts that are able to be expressed are Heatmap, Radar, Area, Bar, Line, Pie, Column, etc. and were implemented to be able to combine temperature-humidity that is a different type of data to electricity data, into the chart. Also, a function that allows the created charts to be exported into JPEG, PNG, PDF files is included.

5. Conclusion

By periodically collecting environment sensor values such as the temperature-humidity and the electricity usage generated from legacy devices of numerous households, a big data analysis system that can streaming process and store these information was implemented. Many open softwares including OpenStack was utilized, and it was designed and implemented aiming for a large size, high stability system. A simulation was conducted of 500 households with intervals of 15 seconds, and the individual power, sensor information and regional statistics was expressed using various visualization tools. Such implementation is expected to be utilized as a reference model to implement a system that collects and analyzes large size data including the future IoT in real time.

6. Acknowledgements

This study was conducted with the support from Ministry of trade, industry & energy

7. References

[1] T. Fifield, D. Fleming, A. Gentle, L. Hochstein, J. Proulx, E. Toews & J. Topjian, "OpenStack Operations Guide," O'Reilly Media, 2013

[2] D. Kakadia, "Apache Mesos Essentials," O'Reilly, 2015

[3] N. Garg, "Learning Apache Kafka," 2nd, Packt Publishing, 2015
[4] H. Karau, A. Konwinski, P. Wendell, M. Zaharia, "Learning Spark," O'Reilly Media, 2015
[5] K. Matthias & S. P. Kane, "Docker Up & Running," O'Reilly, 2015
[6] T. White, "Hadoop The Definitive Guide," O'Reilly, 2015
[7] "Apache Zeppelin," https://zeppelin.incubator.apache.org
[8] C. Korner, "Data Visualization with D3 and AngularJS," Packet Publishing, 2015