Journal of Engineering and Applied Sciences 13 (5): 1064-1072, 2018
ISSN: 1816-949X

Big Data-based Log Collection and Analysis in IoT Environments

Dong Jin Shin, Jong Min Eun, Ho Geun Lee, Myoung Gyun Lee,
Jeong Min Park and Jeong Joon Kim
Department of Computer Science and Engineering, Korea Polytechnic University,
Gyeonggi-do, 15073 Siheung-si, Korea

Abstract: Recently, various new technologies such as Ai, IoT, cloud and big data are being developed in line with the 4th industrial revolution. As the amount of various sensor data based on IoT is increased, many techniques are required to collect and analyze the data. Therefore, we want to present the analysis results through processing of big data. In IoT, sensor data can be various kinds and quantities such as ultrasonic waves, infrared rays, cameras and vibrations. This type of informal data is difficult to obtain the desired analytical results when applied to a general analysis program. In this study, we implemented a system that processes informal data by collecting, storing, processing and analyzing data. We used Raspberry Pi in IoT and generated web server log data. The generated web server log data is collected in real time using flume, a collection solution of big data. Storage is stored in the HDFS of the hadoop solution and the unwanted properties are refined through processing solutions Hive and Pig. At the end of the final refine process, we analyzed the files with R programming and Spark.

Key words: Big data, IoT, sensor data, informal data, analysis, Korea

INTRODUCTION

In recent years, revolutionary changes in information and communication technologies such as computers have been taking place due to the development of various new technologies such as Ai, IoT, cloud and big data in the 4th industrial revolution. Sensor data is generated using IoT in various fields. Sensor data processing such as temperature, ultrasound, infrared, camera and vibration must be analyzed quickly and accurately in each field. Because sensor data is mostly informal data using big data enables qualitative analysis rather than general analysis. Big data is recognized as a trend in data collection and analysis solutions used in today's diverse fields and is applied to various areas that require formal and informal data analysis. Unlike traditional analysis solutions, large amounts of data are stored in a distributed manner, so, storage space efficiency and analysis speed are different (Dong-Beom *et al*., 2017; Jeong-joon, 2010).

In this study, web server is installed among various sensor data generated by Raspberry Pi to generate log data. The generated log data is stored in real time in HDFS of hadoop storage space using flume, a real-time collection solution of big data. Distributed stored logs on web server are refined using Hive and Pig processing solutions. First, the logs stored in HDFS typically use Hive-supplied 'RegexSerde' regular expressions to receive data in tabular form and store it in HDFS with a new delimiter specified. Second, secondly, it uses Pig, loads the re-stored files into memory, refines unnecessary attributes and stores them in HDFS. The final refined file is stored in CSV format, separating the attributes based on comma (,) in order to conveniently code in R. We analyzed the frequency of access through R programming with the final refined CSV file. Also, the contents of the most frequently accessed pages were analyzed using Spark (Kim, 2014; Dong-Gon Kim, 2013).

Literature review

Big data collection solution: What is important in processing based on big data is the collect of data. Data is collected using the big data solution, stored in hadoop and processed and analyzed. Big data collection solutions include 'crawling' using Python, 'Sqoop' for collecting relational databases and 'Flume' for collecting logs. This study refers to the flume solution for log collection. Flume is a solution that efficiently collects large amounts of log data and collects large amounts of log data distributed to a large number of servers into repositories. Flume consists of three layers: source, channel and sink. Figure 1 shows an example of a structure using flume (Han *et al*., 2014).

Corresponding Author: Jeong Joon Kim, Department of Computer Science and Engineering, Korea Polytechnic University, Gyeonggi-do, 15073 Siheung-si, Korea

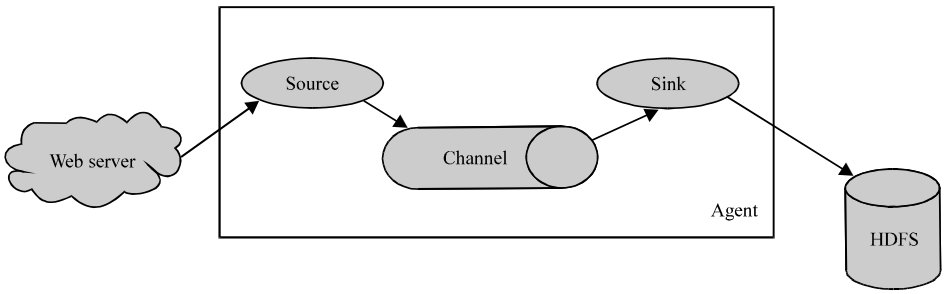

Fig. 1: Structure using flume

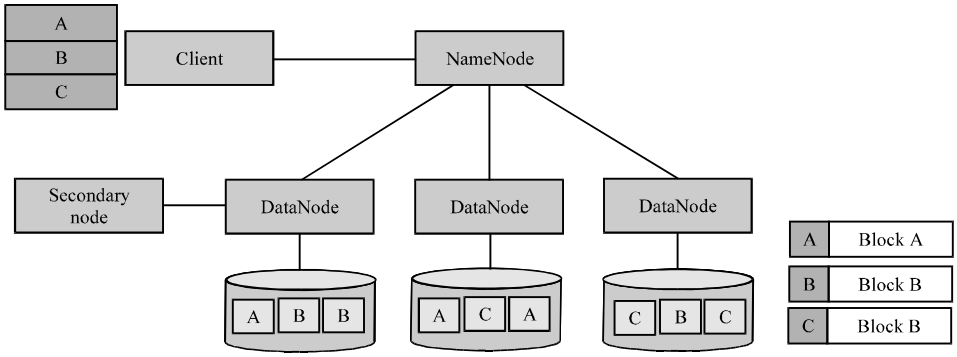

Fig. 2: Distributed storage configuration using hadoop

Big data storage solution: Hadoop stores data in HDFS, a distributed system that supports distributed storage and processing and processes the data using MapReduce. HDFS consists of NameNode and DataNode. NameNode manages meta data to maintain the file system and monitors the data nodes. DataNode is a file stored in HDFS and the block size is distributed based on the basic 64 MB. Among them, SecondaryNode acts as NameNode when NameNode is down. Figure 2 shows the distributed storage structure of hadoop (Lee and Lee, 2015).

MATERIALS AND METHODS

Big data processing solution: MapReduce is primarily used to process big data processing solutions. Pig is one of the sub-projects that make up hadoop. It is based on MapReduce-specific development tool level and consists of a high-level scripting language. Hive provides data summarization, query and analysis capabilities such as relational databases. Pig's main hierarchy is composed of compilers and creates data conversion procedures for MapReduce programs for massively parallel processing. Hive provides a SQL-like language in a relational database called HiveQL and supports all the features of MapReduce. Figure 3 shows the processing structure using Hive and Pig (Kim and Bahn, 2013).

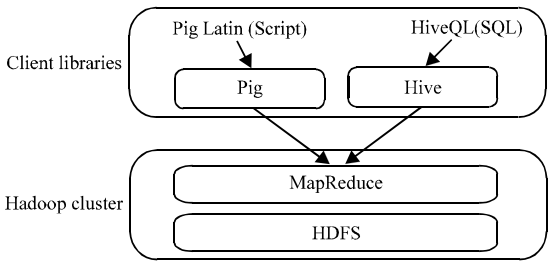

Fig. 3: Data processing configuration through Pig and Hive

Big data analysis solution: Big data analysis solution include R programming and Spark which are mainly used in statistics. R is a language that provides functions such as statistics, visualization and data mining with processed and refined data. This is the language used in the statistics industry and has attracted attention as a large-scale data analysis that needs to process large amounts of data in recent years. Spark is also a high performance cluster computing system for general use. Spark is an optimized engine that provides high-level APIs written in Java, Scala and Python and supports common execution graphs. It also supports rich advanced tools such as spark SQL for SQL and structured data processing, MLlib for machine learning, graph X for graph

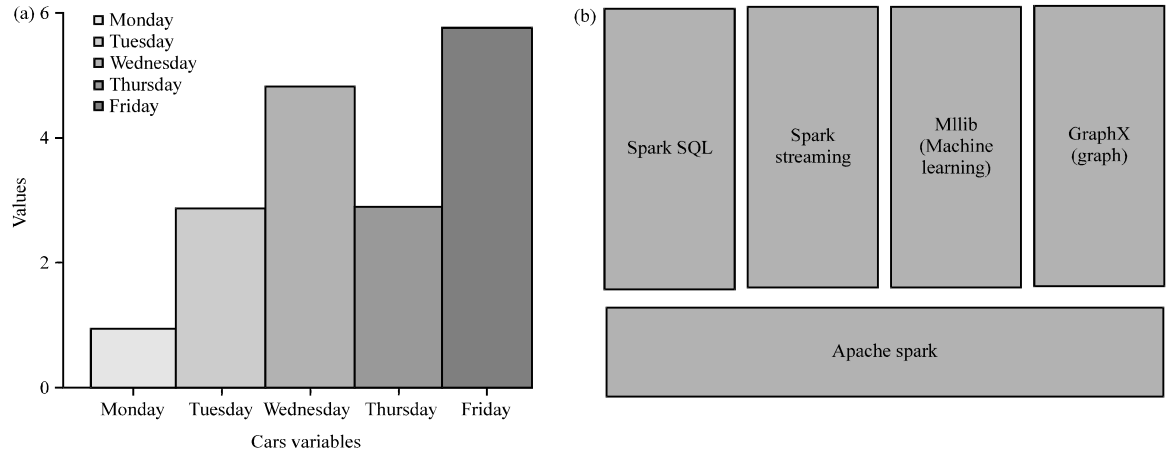

Fig. 4: R and spark visualization and structure: a) Car variables and b) Spark visualization

processing and spark steaming for real time processing. Figure 4 shows an example of the results to be analyzed and the processing structure of Spark (Kim, 2014).

RESULTS AND DISCUSSION

Big data-based log collection and analysis process using IoT

System configuration diagram: The process of collecting and analyzing logs based on big data using IoT proposed in this document is to create a random home page by installing a web server on RaspBerry Pi. Logs accumulated through the generated homepage are collected in real time through "Log real-time collect manager". The collected logs are distributed and stored through the "Data store manager". Distributed logs are refined through "Data process manager". Once the final processing is complete, the analysis results are visualized through the "Analysis manager". Figure 5 shows the overall configuration.

RaspBerry Pi: Among the big data processes proposed in this study, data is generated using RaspBerry Pi among IoT related devices. The type used was Pi3 Version and I installed Apache web server by installing Ubuntu. After the installation was completed, a homepage was created and a log was generated. In The overall configuration means using more RaspBerry Pi but omitted in Fig. 5.

Log real-time collect manager: Logs generated by RaspBerry Pi's web server are transmitted through the "RaspBerry Pi Agent Module" of "Log Real-Time Collect Manager". The transmitted data log is again completed through the secondary transfer of "NameNode Agent Module". This process is performed in real time and the transmitted logs are transferred to "Data Store Manager".

Data store manager: "Data store manager" is a manager that distributes automatically stored when logs are transmitted in real time. This structure is composed "Fully-distributed operation" and it is distributed and stored through four nodes of the actual computer.

Data process manager: "Data process manager" is a data processing manager stored in HDFS. The primary processing is a "Table import module" that uses Hive to input a file in the form of a table. The second process is a "Process Module" that uses the Pig to refine the properties. Finally, it consists of a "file conversion module" that converts the file to CSV format for analysis.

Analysis manager: "Analysis manager" is analyzed through R programming and spark. The analysis consists of "Access per hour module" for checking the frequency of visits per hour during the day and "Access per OS module" for checking the operating system frequency by day. "IP Top N module" analyzes the maximum number of connected IPs. Finally, it consists of a "WordCloud module" that checks the frequency of pages accessed using spark.

System operation process

System log collection process: The real-time log collection process used flume among the big data collection solutions:

- Step 1: RaspBerry Pi Agent Module
- Step 2: NameNode AgentModule

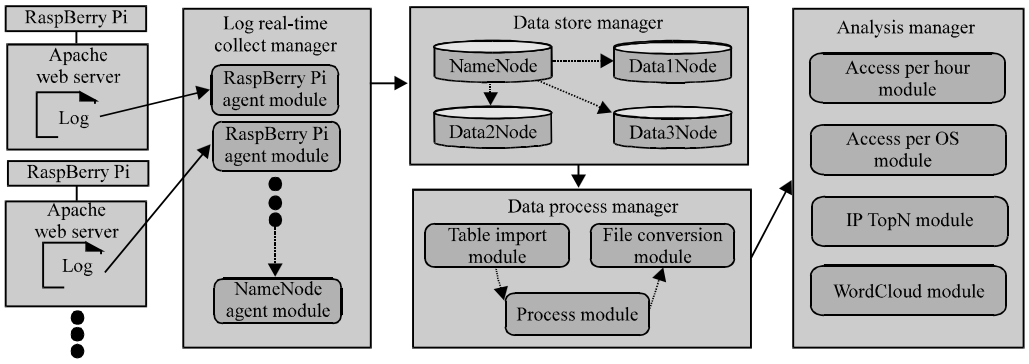

Fig. 5: Expression that determines the initial number of buckets

Step 1; RaspBerry Pi agent module: Install the flume of each RaspBerry Pi and set up as shown in Algorithm 1.

Algorithm 1; Flume setting command of RaspBerry Pi:

1. agent02.sources.execGenSrc.type = exec
   agent02.sources.execGenSrc.command = tail-F/var/log/apache2/access.log
2. agent02.sources.execGenSrc.batchSize = 1000
   agent02.sources.execGenSrc.channels = memoryChannel
3. agent02.sinks.avroSink.type = avro
   agent02.sinks.avroSink.hostname = 192.168.40.11
   agent02.sinks.avroSink.port = 33333
   agent02.sinks.avroSink.batch-size = 1000

The configuration of the stream that transmits RaspBerry Pi's log file in real time using flume is composed of source, channel and sink. Sources is a part that receives events from external sources. It reads '/var/log/apache2/access.log' which is the path where RaspBerry Pi's installed web server logs are accumulated, by reading the Linux 'tail' command. The channel temporarily stores the event and sends it to the sink. It is implemented as a queue in memory as above. The sink part is an external part for outputting events. NameNode sends '192.168.40.11' IP and random port address for '33333' and the number of events that can be received is 1000 (Algorithm 2).

Algorithm 2; Flume setting command of NameNode:

1. agent01.sources.avroGenSrc.type = avro
   agent01.sources.avroGenSrc.bind = 192.168.40.11
   agent01.sources.avroGenSrc.port = 33333
2. agent01.sources.avroGenSrc.channels = memoryChannel
   agent01.sinks.HDFS.type = HDFS
3. agent01.sinks.HDFS.hdfs.path = hdfs://master:5000/web1
   agent01.sinks.HDFS.hdfs.writeFormat = text
   agent01.sinks.HDFS.hdfs.batchSize = 1000
   agent01.sinks.HDFS.hdfs.rollSize = 0
   agent01.sinks.HDFS.hdfs.rollInterval = 600

Step 2; NameNode AgentModule: The events output through the RaspBerry Pi agent module sink are received by the source part in the NameNode flume structure and the IP and port to receive the event are specified. The memory channel used is implemented in the same queue. In the NameNode, the sink part specifies the address of the master node used for distributed storage in HDFS and the path where the file will be stored. Finally, the log file format accumulated in HDFS is set to text, the number of events is limited to 1000, the file size is irrelevant and after 10 min, the file is saved as the next file.

System log store process: Logs collected in real time are distributed and stored in HDFS of the storage process, consisting of four nodes in hadoop 'Fully-Distributed Operation'. Figure 6 shows that the data is distributed and stored.

The path stored through the NameNode Agent Module can confirm that the data has been collected in the path '/web1' under the root. You can see the data collected through flume. It is saved in number format. Figure 7 shows the log data collected with the 'cat' command.

The format of the collected log data is as follows. "Remote host, identity, user, time, request message, status code, size, referer URL, user browser". The remote host is the IP of the connecting client, the user name and the authenticated user name refer to the user name of the connecting host. Time is the requested time and date and is collected in Unix time. Also, the request page is the page address used by the connected client and the status code is the HTTP status code and 200 means that the request is normally completed. The transmitted size is the byte size excluding the header from the requested page, the referenced URL is the previous URL and the requested browser is the version of the browser used by the client.

System log processing process: The process of processing distributed log data is processed through three processes in total:

```
hadoop@hadoop-name:~$ hadoop fs -ls /web1
17/12/08 14:05:31 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Found 13 items
-rw-r--r--   3 hadoop supergroup       7084 2017-11-13 16:28 /web1/FlumeData.151
0557743271
-rw-r--r--   3 hadoop supergroup       6318 2017-11-13 16:41 /web1/FlumeData.151
0558291584
-rw-r--r--   3 hadoop supergroup       1480 2017-11-13 16:51 /web1/FlumeData.151
0558898137
-rw-r--r--   3 hadoop supergroup       7212 2017-11-13 17:14 /web1/FlumeData.151
0560248310
-rw-r--r--   3 hadoop supergroup      52239 2017-11-13 17:46 /web1/FlumeData.151
0562161529
-rw-r--r--   3 hadoop supergroup      23087 2017-11-13 18:09 /web1/FlumeData.151
0563548065
-rw-r--r--   3 hadoop supergroup       8865 2017-11-13 18:24 /web1/FlumeData.151
0564453286
```

Fig. 6: Verifying files stored on HDFS

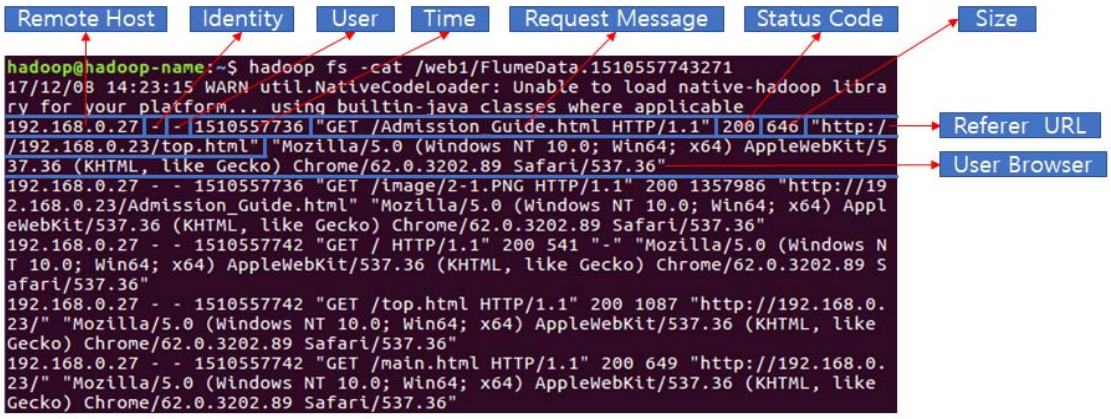

Fig. 7: Check collected log data in HDFS

- Step 1: table import module
- Step 2: process module
- Step 3: file conversion module (Algorithm 3)

Algorithm 3; Insert log data using Hive:

1. hive> SET hive.support.sql11.reserved.keywords = false
2. hive> create table apache_log1 (host STRING, identity STRING, user STRING, time STRING, request STRING, status STRING, size STRING, referer STRING, agent STRING)
   ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
   WITH SERDEPROPERTIES ("input.regex" = "([^]*) ([^ \"]*|\"[^\"]*\") ([^]*) ([^\"]*|\"[^\"]*\") ([^\"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) ([^\"]*|\"[^\"]*\") ([^\"]*|\"[^\"]*\"),"output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s") STORED AS TEXTFILE
3. hive> LOAD DATA INPATH "/web1/*" INTO TABLE apache_log1
4. hive> insert overwrite directory '/web1_output' row format delimited fields terminated by '|' select*from default.apache_log1
5. hadoop fs -ls /web1_output
   -rwxr-xr-x 3 hadoop supergroup 740160 2017-11-17 16:10 /web1_output/000000_0

Step 1: Table import module: To enter data in a tabular format using a hive, you can not specify the user and identity values of the hive as attribute names. Create the hive d apache_log1 table. The most important of these is the 'RegexSerDe' format. For Apache weblog data, the value that separates the attribute is a space of "\t". However, if there is a space between the attribute values, it is difficult to distinguish all the attributes in the second stage of the pig and to perform accurate segmentation. Therefore, if you are using the 'RegexSerDe' format and there are spaces in the long string, set the string to a single property value. Enter the collected log data and save it in the generated apache_log1 table. Save the attributes of the apache_log1 table as pipe "|". HDFS '/web1_output' directory. If you search the file locally, you can see that it is saved as '000000_0' file (Algorithm 4).

Algorithm 4; Process using Pig:

1. grunt> log1 = LOAD './000000_0' USING PigStorage('|')
2. grunt> log1_relation = FOREACH log1 GENERATE $0 as host:chararray, $3 as time:chararray, $4 as request:chararray, $5 as status:chararray, $6 as size:chararray, $7 as referer:chararray, $8 as agent:chararray
3. grunt> store logs1_relation into 'log1_refile' using PigStorage(',')

Step 2: Process module: Input '000000_0' file saved in '/web1_output' into the log1 variable of pig. Use the Pig Foreach syntax to refine (delete) the user and identity attributes and then enter them in the log1_relation variable. HDFS '/log1_refile' Save the file by separating the attributes with commas "," in the directory (Algorithm 5).

Algorithm 5; File CSV conversion process:

```
1. hadoop fs -ls /log1_refile/
   -rw-r--r— 3 hadoop supergroup 111110 2017-11-27 16:20
   /log1_refile/_SUCCESS
   -rw-r--r— 3 hadoop supergroup 740160 2017-11-27 16:20
   /log1_refile/part-m-00000
2. hadoop fs -getmerge /log1_refile/ ./log1_refile.csv
3. ls -al
   -rw-r--r-- 1 hadoop hadoop 740160 Nov 27 16:21 log1_refile.csv
```

Step 3: File conversion module: Check the '/log1_refile' directory in HDFS to see if there are any files created through step 4 of step 2. You can see the file SUCCESS and the generated file 'part-m-00000' Converts the files in the '/log1_refile' directory in HDFS to a file called log1_refile.csv in the local which is the current directory. If you verify the directory locally You can see that the 'part-m-00000' file in HDFS is converted locally to log1_refile.csv.

System log analysis process: Performs four analyzes based on the completed CSV file. In the previous process, only one log was processed. However, in the actual implementation, we performed four RaspBerry Pis and as the entire code became longer, the analysis was coded based on two or one RaspBerry PI units (Algorithm 6).

Algorithm 6; Access per hour module code:

```
1. LOGS = read.csv("log1_refile.csv", header = F)
   LOGS2 = read.csv("log2_refile.csv", header = F)
2. val1 <-LOGS$V2
   val2 <- LOGS2$V2
3. val1_2 = as.POSIXct(val1, origin = "1970-01-01")
   val2_2 = as.POSIXct(val2, origin = "1970-01-01")
4. hours = format (val1_2, "%H")
   hours2 = format (val2_2, "%H")
5. plot (main = "Weekly access time", xlab = "Time", ylab = "Access",
   table (hours), type = "o", col = "blue")
6. lines (table(hours2), type = "o", col = "red")
7. legend ("topright", legend = c ("Web Server1", "Web Server2"),
   pch = c (19,19), cex = 1, col = c ("blue","red"))
```

First; Access per hour module: The header values of the refined log1 and log2_refile.csv are set to false and loaded into the LOGS and LOGS2 variables. The second attribute of the LOGS, LOGS2 variable time value is entered in the val1 and 2 variables. Convert the Unix time values of val1 and 2 to year-month-day and enter them into val1_2 and val2_2, respectively. Only "% H' times of val1_2 and val2_2 are converted to 'hours' and 'hours2' variables. Use the 'Plot' function to visualize the graph for a few hours first. Similarly, 'hours2' is visualized with additional colors. Use the 'Legend' feature to clearly display the legend in the upper right corner of the graph.

Figure 8 is a graph showing the frequency of access by time using the 'plot' function. We checked the frequency of access using two RaspBerry Pi and as a result, we observed a large amount of connection in afternoon time on web server1. On the other hand, the second web server shows a large amount of connection in the morning, indicating that various clients have accessed the web server at various times (Algorithm 7).

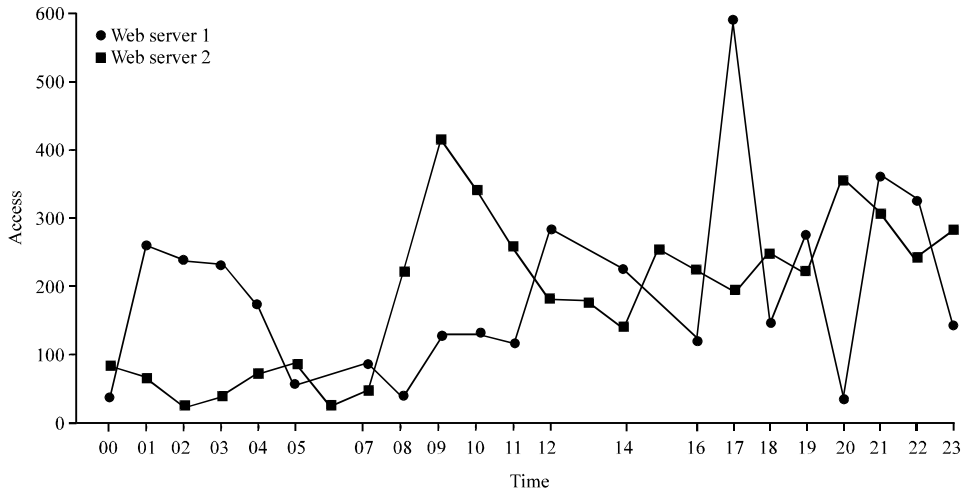

Fig. 8: Access per hour module visualize graph; weekly access time

Algorithm 7; Access per hour OS module code:

```
1. LOGS = read.csv("log1_refile.csv", header = F)
   #LOGS = read.csv("log2_refile.csv", header = F)
2. x<-LOGS
3. blacklist <- c("iPhone")
   blacklist2 <- c("Windows")
4. spam <- x[grep(blacklist,x$V7),]
   spam2 <- x[grep(blacklist2,x$V7),]
5. LOGS$V2<-as.POSIXct(as.numeric(as.character(LOGS$V2)),origin =
   "1970-01-01")
6. LOGS$V2 <- format(LOGS$V2, "%A")
7. d1 <-subset(LOGS,V2 %in% c("monday"))
   d5 <-subset(LOGS,V2 %in% c("friday"))
8. T1<-nrow(d1)
   T5<-nrow(d5)
9. board <- c(T1, T2, T3, T4, T5)
10. names(board) <- c("Monday","Tuesday", "Wendesday", "Thursday",
    "Friday")
11. , ...,
12. barplot(board5, beside = T, xlab = "Days", ylab = "Access",
    col = c("blue","orange","yellow","green"), las = 1, border = "gray",
    main = "OS visit frequency (PIE 2)")
13. legend("topright", legend = c("Total", "iPhone", "Windows"),
    pch = c(15,15,15), cex = 1, col = c("blue","orange","yellow"))
```

Second; Access per hour OS module: Read the CSV file into the LOGS variable. You can annotate each of the two RaspBerry Pis separately. If you use a different Pi, you can set the annotations to reverse. Enter the LOGS original in the x variable. Set the classification of the operating system to use and store it in the blacklist variable. In this case, we set three filtering words: iPhone, Windows and Linux. In the blacklist variable, information of the used browser which is the 7th attribute is Grep the three words specified in filtering and store them in the new variables spam, spam 2, 3. Convert the second attribute of LOGS, Unix time, into year-month-day. Only re-convert the second attribute's day of the week. Enter data for each day of the week in the variables d1-5. Measure the number of values stored in variables d1~d5 and store them in variables T1~T5. Set the value of the number of the board variable to a vector. Name each row in the board vector. Since, it is a process of repeating the conversion only to spam 2 and spam 3 set in some are omitted. Visualize the connection frequency information of OS by graph using barplot function. When the visualization is completed, add a legend to the upper right corner to confirm the graph classification.

Figure 9 is a graph visualizing the frequency of operating system accessing the web server using the barplot function. If you check the frequency of the operating system accessing your web server, you will see that most clients are mostly connected through Windows. There are similar connections on Monday and Tuesday and on Wednesday there are more connections to the web server than any other day. In particular, only client connections using Windows have been confirmed on Friday and access to customers using the iPhone has not been confirmed. Most customers can conclude that they are trying to connect to a web server from a desktop browser instead of a mobile browser (Algorithm 8).

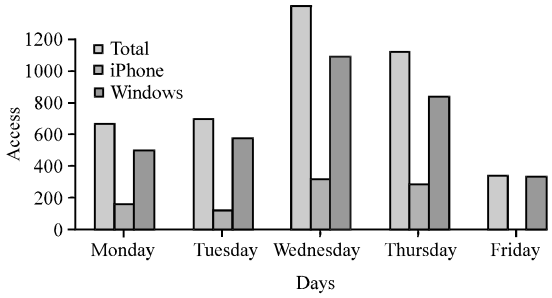

Fig. 9: Access per hour OS module visualize graph; OS visit frequency (PIE 1)

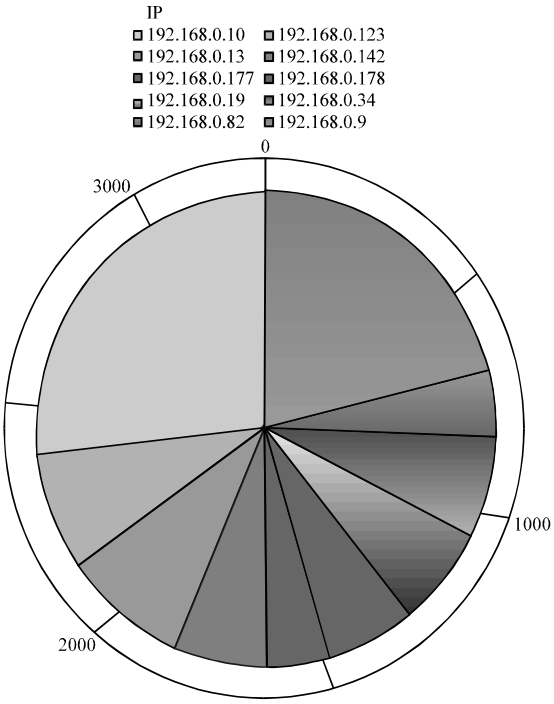

Fig. 10: IP top N module visualize graph

Algorithm 8; IP TopN module:

```
1. log1 <- read.csv("log1_refile.csv",col.names = c("ip","time", "line",
   "hp_status","bytes","URL""OS"))
2. ip_count <- log1 %>% group_by(ip) %>% summarise(N = n())
3. ip_top_N <- ip_count %>% arrange(desc(ip_count$N)) %>% head(10)
4. ggplot(ip_top_N ,aes(x = """,y = N, fill = ip_top_N))+geom_bar(stat =
   "identity",width = 1)+coord_polar(theta = "y")
```

Third; IP top N module: Read the CSV file into the log1 variable. Name each property. Analyze the top 10 IP using only the IP attribute. Calculate the number of each group by grouping the IP attribute in log1 with the ip_count variable. Sort ip_Top_N variable to the number of grouped IP in 2. Draw a circular graph using the ggplot function with the number of sorted ip_top_N.

Figure 10 is a graph visualizing the IP count of the client accessed through the color difference using the

ggplot function. If you check the IP top 10 frequency of connecting web server, 192.168.0.10 client has the most connection and 192.168.0.9 client has the second most connection. From the third, we could see that the majority of clients with similar frequency were approaching.

Fourth; WordCloud module: The last analysis, WordCloud module was used spark and the programming language used Pyspark which is a combination of spark and Python. In order to visualize the result screen, we built Pyspark environment by installing Matlotlib package, Ipython package for using Python in the Linux shell console screen and WordCloud package for analysis Algorithm 9 shows the code for WordCloud module analysis by executing Pyspark.

Algorithm 9; WordCloud module code:

```
1. File = sc.textFile("/log1_refile/part-m-00000")
2. File.first()
   u'192.168.0.27, 1510557736, GET/Admission_Guide.html
   HTTP/1.1, 200, 646, …,
3. split_File = File.map(lambda line: line.split(","))
4. access_page = split_File.map(lambda fields: fields[2])
5. access_page.saveAsTextFile("/word")
6. hadoop fs-ls/word/
   hadoop fs-getmerge/word/* ~/refile/word
7. import matplotlib.pyplot as plt
8. from wordcloud import WordCloud
9. text = open('/home/hadoop/refile/word'). read()
10. WordCloud = WordCloud(background_color = 'white'). generate(text)
11. plt.figure(figsize = (12, 12))
12. plt.imshow(wordcloud, interpolation = "bilinear")
13. plt.axis("off")
14. plt.show()
```

Through step 2 of the process, the final tabulated file saved in HDFS is loaded into the file variable. Print the first value of the file with the first() function, see that, it was input normally. Also, we can confirm that the third attribute is needed to analyze the frequency of accessed pages. Split the variable split_File with the comma ",". In Spark, since, the start of array starts from 0, enter the attribute of in access_page variable. Save the entered variable to the word directory under HDFS root. On Linux locally, use the hadoop 'getmerge' command to split the files in the/word directory in HDFS through to merge them together and store the filename in the/refile path under the local home directory as word. Go back to the Pyspark console screen and import the matplotlib package for visualization of results as alias plt. Import the WordCloud package for analysis. Load the locally stored 'word' file through into the text variable. Run the WordCloud package on the wordcloud variable, the background color is white and the text to load brings up the text variable. Call up the plt specified as an alias and specify the size of the visualization result screen. Outputs wordclouded content through the imshow() function which enhances the clarity of the text. Delete the horizontal and vertical variable values of visualization output. Finally, output the above visualization picture.

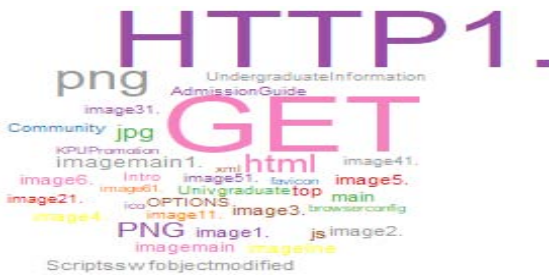

Fig. 11: WordCloud module visualize result

Figure 11 shows the frequency of page accesses using different packages in Pyspark. When you check the visualization results, many words related to HTTP1, GET and png are output. That means you've got a lot of access to the pages associated with the image.

CONCLUSION

In this study, implemented and analyzed with hadoop, a representative solution of big data, flume, a real-time collection solution, Hive, Pig in processing and R and Spark in analysis. In general analysis when log data is accumulated, it is input to the program according to the necessary timing to produce analysis result. Also, the more log data, the more time it takes to analyze and the cost of storing log data will increase. However, if the log data is collected in real time through the processing of big data and distributed using four nodes, it is possible to have more capacity in terms of capacity. In addition, it is possible to refine collected log data differently from general analysis, so that, users can analyze more variously.

Currently, real-time analysis of big data is required in various service fields. This analysis is based on the user's feedback on the service, It will contribute to the selection of improvement direction of various analysis platform. Also, applying machine learning technique to big data will be a leader in data security by enhancing enterprise value through backup of data by predicting server load in the future and applying new attack defense method in terms of security.

RECOMMENDATIONS

In the future, we will implement a fast and efficient analysis system by collecting sensor data from various IoTs. In addition, we will apply machine learning method to log collection and analysis system and aim at realization of accurate service platform by analyzing more various prediction analysis and abnormal behavior detection than statistical analysis.

REFERENCES

Dong-Beom, K., K. Tae-Young, K. Jeong-Joon and P. Jeong-Min, 2017. Sensor data collecting and processing system. Asia Pac. J. Multimedia Serv. Convergent Art Humanities Sociology, 7: 259-269.

Dong-Gon, Kim, 2013. A study on application cases of big data log analysis. Master Thesis, Department of Computer Engineering, Graduate School of Industry and Technology, Chonnam National University, Gwangju, South Korea.

Han, K.H., H.J. Jeong, D.S. Lee, M.H. Chae and C.H. Yoon *et al*., 2014. A study on implementation model for security log analysis system using big data platform. J. Digital Convergence, 12: 351-359.

Jeong-Joon, K., 2010. Efficient processing of aggregate queries in wireless sensor networks. Ph.D Thesis, Computer & Information Communication Engineering, Konkuk University, Seoul, South Korea.

Kim, J. and H. Bahn, 2013. An efficient log data management architecture for big data processing in cloud computing environments. J. Inst. Internet Broadcast. Commun., 13: 1-7.

Kim, J.S., 2014. Big data analysis technologies and practical examples. Korea Contents Assoc. Rev., 12: 14-20.

Lee, S.J. and D.H. Lee, 2015. Real time predictive analytic system design and implementation using Big data-log. J. Korea Inst. Inf. Secur. Cryptology, 25: 1399-1410.