Large-Scale Web Traffic Log Analyzer using Cloudera Impala on Hadoop Distributed File System

Choopan Rattanapoka[*] and Prasertsak Tiawongsombat

Abstract

Resource planning and data analysis are important for network services in order to increase the service efficiency. Nowadays, Large websites or web servers have a large number of visitors, which mean a large web traffic log need to be stored in the plain text or the relational database. However plain text and relational database are not efficient to handle a large number of data. Moreover, the web traffic log analysis hardware or software that can handle such a big data is also expensive. This research paper proposes the design of a large-scale web traffic log analyzer using PHP language to show the visitors' traffic data analysis in the form of charts. The Hadoop Distributed File System (HDFS) is used in conjunction with other related techniques to gather and store visitors' traffic log. Cloudera Impala is used to query web traffic log stored in HDFS while Apache Thrift is an intermediary connecting Cloudera Impala to PHP web. Upon testing our large-scale web traffic log analyzer on HDFS Cluster of 8 nodes with 50 gigabytes of traffic log, our system can query and analysis web traffic log then display the result in about 4 seconds

Keywords: Hadoop Distributed File System, Cloudera Impala, Big Data, Log Analysis, PHP

Department of Electronics Engineering Technology, College of Industrial Technology, King Mongkut University of Technology North Bangkok.

[*] Corresponding author, E-mail: choopan.r@cit.kmutnb.ac.th Received 30 March 2016, Accepted 16 December 2016

1. Introduction

Resource planning and data analysis are important for network services in order to increase the service efficiency. System administrators need tools that simple and easy to use for monitoring and analyzing the use of the services. Nowadays, the most active network service on the internet is Hypertext Transfer Protocol (HTTP) and its traffic is increasing gradually.

Thus, the web traffic data that needs to be analyzed is large and need a big storage to store all of the traffic data. However, the software and hardware that can store and analyze these big data are too expensive. Although, there are free web traffic analyzer softwares such as AWStats [1], Open Web Analytics (OWA) [2] or Piwik [3] that store traffic data in the plain text format or in the relational database system such as MySQL, they are not efficient when data is too big.

A few years ago, C. Rattanapoka [4], S. Narkhede and T. Baraskar [5] published research papers on Log analyzer using Hadoop. However, both of them used Hadoop Mapreduce paradigm to extract and retrieve data from Hadoop Distributed File System (HDFS). However, the cost of starting MapReduce process is expensive. So, both of their systems cannot be used for analyzing real-time data. Later on, S. Adhikari et al. [6] proposed the system that analyses and generates log data and statistics report by using Hadoop. But, instead of using directly Hadoop Mapreduce paradigm, this system uses Pig [7] which is a tool that abstracts and hides the complexity of Mapreduce paradigm to more human-understandable commands and scripts. However, the operations underneath of Pig are still Mapreduce which take time to start processes and cannot be used for real-time analytics. Moreover, the Pig commands are not intuitive like the structural query language (SQL).

Hence, this research paper presents the design and implementation of a PHP web-based large-scale web traffic analyzer using Hadoop Distributed File System (HDFS) to store large-scale web traffic data and using Cloudera Impala (SQL-like commands) to query and analyze web traffic data from HDFS in real-time.

2. Background

In this section, we review Hadoop Distributed File System, Cloudera Impala, Apache Flume and Apache Thrift which are main 4 components to implement our large-scale web traffic log analyzer.

2.1 Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) [8] is a distributed file system that can be scalable to support thousands of commodity machines. The design of HDFS is fully inspired by Google File System (GFS) [9] which has master/slave architecture. HDFS is designed to work with large data sets requiring tens of petabyte of storage. HDFS operates on top of file systems of the underlying OS. HDFS is written in Java language and is highly fault-tolerant. Each HDFS cluster has one master node, called *namenode*, which manages the metadata information and several nodes, called *datanodes*, which manage storage attached to the nodes that they run on, store the actual data.

Fig. 1. shows the mechanism of HDFS. Clients contact the namenode machine for file metadata and perform actual file I/O directly with the datanodes.

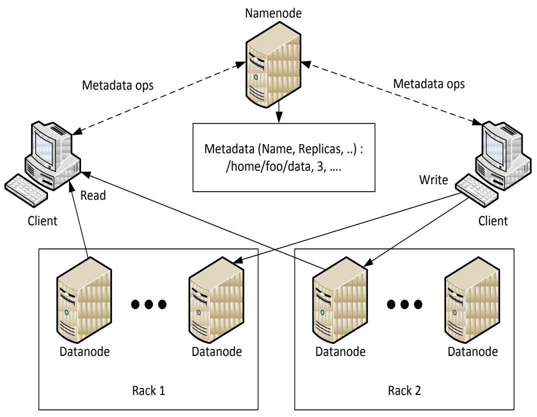


Fig. 1. The Mechanism of HDFS.

2.2 Cloudera Impala

Cloudera Impala [10] is an open source Massively Parallel Processing (MPP) query engine that runs on Apache Hadoop. Impala provides high-performance, low-latency SQL queries on data stored in popular Apache Hadoop file formats. The fast response for queries enables interactive exploration and fine-tuning of analytic queries, rather than long batch jobs traditionally associated with SQL-on-Hadoop techniques. Impala provides access to data in Hadoop without requiring the Java skills required for MapReduce [11] jobs. Impala can access data directly from the HDFS file system. Impala is pioneering the use of the Parquet [12] file format, a columnar storage layout that is optimized for large-scale queries.

2.3 Apache Flume

Apache Flume [13] is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flow. It is robust and fault-tolerant with tunable reliability mechanisms and many failovers and recovery mechanisms. It uses a simple extensible data model that allows for online analytic applications. Fig. 2. shows a Flume agent which receives log data, called Event. Then, the Event flows from Source to Channel and then to Sink.

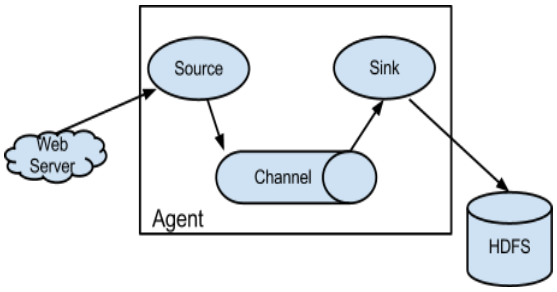


Fig. 2. Apache Flume Architecture.

2.4 Apache Thrift

Apache Thrift [14] is an interface definition language and binary communication protocol that is used to define and create services for numerous languages. It is used as a remote procedure call (RPC) framework that aims to make reliable, performant communication and data serialization as efficient and seamless as possible. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between C#, C++, Cappuccino, Cocoa, Delphi, Erlang, Go, Haskell, Java, Node.js, OCaml, Perl, PHP, Python, Ruby, and Smalltalk. It is originally developed at Facebook. Thrift was open sourced in April 2007 and entered the Apache Incubator in May, 2008. Thrift became an Apache TLP in October, 2010.

Apache Thrift aims to embody the following values: Simplicity, Transparency, Consistency, and Performance.

3. Design and Implementation

In this paper, we propose a design and implementation of a large-scale web traffic log analyzer on Hadoop Distributed File System. Fig. 3. shows how our system works. We have a set of web servers that send web traffic log from Apache web servers (access log and error log) to HDFS via Apache Flume. Then, users can use our web-based log analyzer implemented in PHP language to query and display web traffic log information such as Top Visiting IP, URL error, Top Website Access, and Top Webpage Access. The web-based log analyzer invokes user's requests via PHP Thrift library to contact and order Cloudera Impala to query user's request in SQL style.

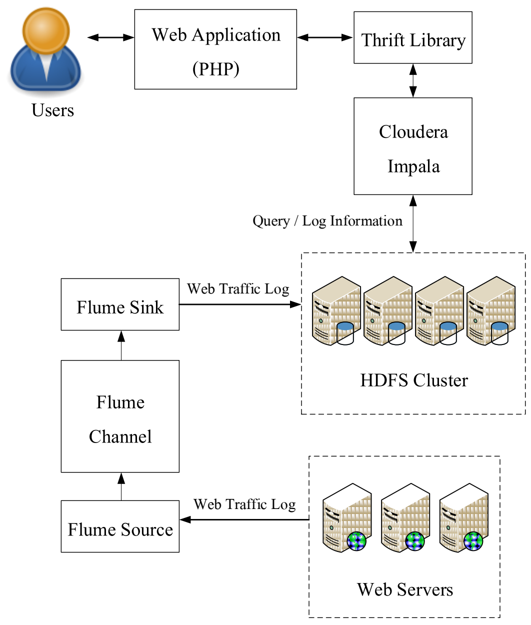


Fig. 3. The Design of Large-Scale Web Traffic Log Analyzer on HDFS.

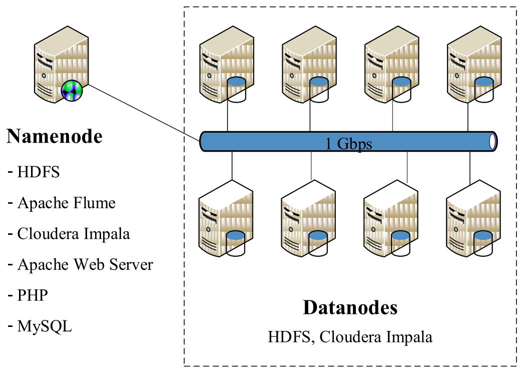


Fig. 4. HDFS Cluster Diagram.

3.1 HDFS Cluster

We installed our testbed system on 9 PCs (nodes) which are one namenode and eight datanodes. All of the nodes connect to 1Gbps Ethernet switch. Because we have limit on the number of nodes, so we also use namenode as the web server for our large-scale web traffic log analyzer.

Hence, in namenode, we have installed 6 programs which are HDFS, Apache Flume, Cloudera Impala, Apache web server, PHP, and MySQL. For datanodes, we have installed only 2 programs which are HDFS and Cloudera Impala as shown in Fig. 4.

3.2 Apache Web Server and Apache Flume

In the proposed web traffic log analyzer, the web traffic log (access log and error log) comes from Apache Web Server and we want to store them in HDFS. The design of Apache Flume in this paper is implemented as shown in Fig. 5.

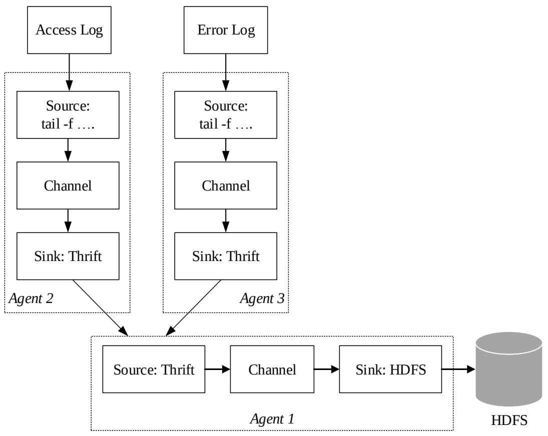


Fig. 5. The Design of Apache Flume.

Each web server has to install Apache Flume and configure Apache Flume's Source with "tail –f" command. For web server access log and error log, by default, locates at /etc/httpd/logs/access_log and /etc/httpd/logs/error_log respectively. Thus, we configure Flume's Source with command "tail –f /etc/httpd/log/access_log" for access log and command "tail –f /etc/httpd/log/error_log" for error log.

Agent2 and Agent3 are agents of Apache Flume that read web server traffic log via Source and transfer them to another Thrift agent (Agent1) via Sink. Then Agent1 read traffic log from multiple Sources (Sink of Agent2 and Agent3) and put that data to Sink which connected to HDFS.

Because we need more information from web server traffic logs. So, we modified both Apache web server's access log and error log format to give us more useful information to analyze.

We modified web server's access log format to *LogFormat "%V`%h`%{%Y-%m-%d %H:%M:%S}t` %r`%D`%{User-agent}i"* where %V is the web server name, %h is the remote hostname, %{%Y-%m-%d %H:%M%S}t is the time request, %r is the first line of request, %D is the time taken to serve the request (in microsecond) and %{User-agent}i is the user-agent of requester. Also, we use the backquote (`) as the delimiter because it is easy to Impala to split a log message to get information of each field.

For error log, we modified to *ErrorLogFormat "%V`%{c}t`%a`%m"* where %V is the web server name, %{c}t is the time request, %a is requester's IP and %m is the request method. We use the backquote as the delimiter just like we did for access log.

3.3 Cloudera Impala Table Creation

Web traffic log in HDFS that receives from web servers are stored in plain text. Normally, we need to write a Java program in MapReduce paradigm to extract and retrieve data. However, the cost of starting MapReduce process is quite expensive and its method used to query information is not intuitive like the structural query language (SQL). The Elasticsearch [15] is another solution to operate (extract and retrieve) on big data. Unfortunately, it also does not have capability of SQL-like command to do more complex querying on data. With Cloudera Impala, we can turn HDFS plain text file format into Impala table and then we can use Impala to query information with SQL syntax. Moreover, The Impala's Parquet file format, a column-oriented binary file format, is good for queries scanning particular columns within a table.

In order to use Impala, we need to map HDFS plain text file to Impala table with plain text file format. In our case, the web server access log and error log are sent from web servers to HDFS via Flume and are saved in */tmp/accesslog* and */tmp/errorlog* respectively.

Thus, we create an Impala table that maps to our web server access log by using following command.

```
CREATE TABLE textaccesslog (
      servername    STRING,
      ip            STRING,
      accessdate    STRING,
      website       STRING
      timeused      STRING,
      user_agent    STRING
)
ROW FORMAT DELIMIED FIELDS TERMINATED BY '`'
STORED AS TEXTFILE
LOCALTION '/tmp/accesslog'
```

From the command above, The Impala table named *textaccesslog* will be created. This Impala table maps to file */tmp/accesslog* in HDFS. The content of this file, in each line, will be split by using backquote as the delimiter and map to each field of this table.

The same applies for web server error log. We have to create another Impala table that maps to error log file stored in HDFS. Because, in our case, the web server error log file format has 4 fields and is stored at */tmp/errorlog* in HDFS. Thus, we create an Impala table named *texterrorlog* which corresponds to the web server error log by using the following command.

```
CREATE TABLE texterrorlog (
      servername  STRING,
      accessdate   STRING,
      clientname  STRING,
      pathURL      STRING
)
ROW FORMAT DELIMIED FIELDS TERMINATED BY '`'
STORED AS TEXTFILE
LOCALTION '/tmp/errorlog'
```

However, the query speed on Impala tables with plain text file format is not good (see the result of the experiment in Section 4). We can solve this problem by using Impala Parquet file format instead of plain text file format. Hence, we need to create another 2 Impala tables *pqaccesslog* and *pqerrorlog* that can store information of access log and error log in Parquet file format, respectively.

The Cloudera Impala command to create the table *pqaccesslog* that store access log using Parquet file format is as follows

```
CREATE TABLE pqaccesslog (
    servername  STRING,    ip            STRING,
    accessdate  STRING,    website       STRING
    timeused    STRING,    user_agent    STRING
)
STORED AS PARQUET;
```

Unfortunately, the Impala table with Parquet file format cannot directly map to our logs in HDFS. Thus, we need to transfer data from our Impala tables with plain text file format, *textaccesslog* and *texterrorlog*, to our new Impala tables with Parquet file format, *pqaccesslog* and *pqerrorlog* respectively. In our paper, we use crontab (task scheduler) to transfer data every day at midnight to keep the number of records in Impala table with plain text file format as low as possible. After the data transfer is complete, we remove data from Impala tables with plain text file format.

The command using for transferring data from *textaccesslog* to *pqaccesslog* as follows.

```
REFRESH  textaccesslog;
INSERT INTO pqaccesslog SELECT * FROM textaccesslog;
```

Thus, we have 4 Impala tables in our database. *textaccesslog*, *texterrorlog* are Impala table with plain text file format that respectively maps today's web server access log and error log in HDFS. While, *pqaccesslog, pqerrorlog* are the Impala table with Parquet file format used to store all of history of web server's access log and error log.

To facilitate our web-based log analyzer, we create 2 views on our database named *accesslog* and *errorlog* that union the information of Impala table with plain text file format and Impala table with Parquet file format. For creating the *accesslog* view, we use the following command.

```
CREATE VIEW accesslog AS
SELECT * FROM textaccesslog UNION ALL
SELECT * FROM pqaccesslog;
```

3.4 PHP and Cloudera Impala via Apache Thrift

We use PHP language to implement our web-based large-scale web traffic log analyzer. Then, our web application cannot directly connect to Impala because its implementation is in Java language. However, Thrift is a middle layer that make the communication between PHP and Cloudera Impala possible.

The PHP phar file containing the Thrift library to connect and send quries to an Impala server can be downloaded at [16].

4. Experiments

In the experiments, we have setup 8 datanodes HDFS cluster with 1Gbps Ethernet connection. Each datanode has the specification as follows: Intel Core 2 Quad Processor CPU, 8 GB of RAM, and 1 TB of Hard disk. Then, we simulate a 50 GB of web traffic log and store it in HDFS via Impala table with plain text file format and Impala table with Parquet file format.

We adjust the number of datanodes that store our 50 GB of web traffic log from 3 to 8 datanodes and then we prepare 3 queries *q1*, *q2* and *q3* to measure the query time usage of Impala tables as follows:

q1: select COUNT(*) from table.

q2: select COUNT(*) from table GROUP BY ip.

q3: select ip, COUNT(*) from table GROUP BY ip ORDER BY COUNT(*).

4.1 Query Time Usage

For the Impala tables with plain text file format, the result in Fig. 6. shows that with 3 datanodes all of 3 queries' time usage are about 37 seconds and query time usage drop dramatically when we add more datanodes to HDFS cluster. Thus, we end up with around 15 seconds query time usage on 8 datanodes HDFS cluster.

For the Impala tables with Parquet file format, the result in Fig. 7. shows that with 3 datanodes the query *q1* takes about 1 second and the query *q2* and *q3* take about 5 seconds. When we add more datanodes to HDFS cluster, the query time of all 3 queries does not effect much because they are already fast. Hence, we end up with 0.9 seconds query time usage for *q1* and about 3.8 seconds query time usage for *q2* and *q3* when using 8 datanodes.

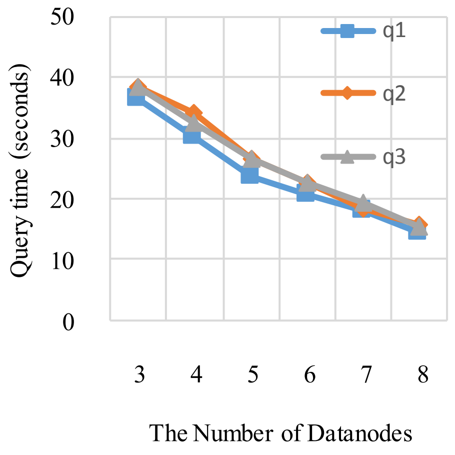


Fig. 6. The Query Time Usage on Impala Tables with Plain Text File Format.

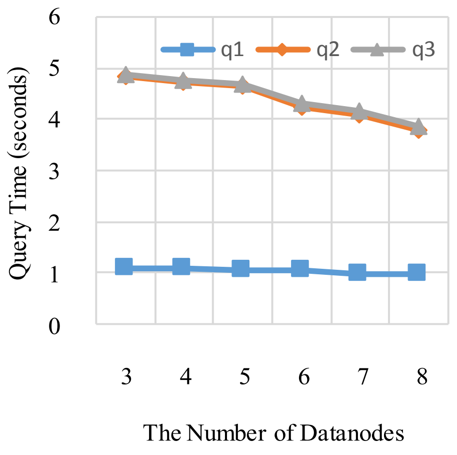


Fig. 7. The Query Time Usage on Impala Tables with Parquet File Format.

4.2 Query Time Speedup

We calculate the query time speedup (*QTS*) when adding more datanodes into the system in Eq. (1) where *Q3D* is the query time usage for 3 datanodes and *QND* is the query time usage for n datanodes.

$$QTS = Q3D / QND \qquad (1)$$

The query time speedup on Impala tables with plain text file format and Impala tables with Parquet file format shown in Fig.8. and Fig.9, respectively. We found that the increasing of datanodes to work with Impala tables with plain text format of 50GB data size has linear speedup. The query can be speeded up almost 2.5 times when we have 8 datanodes comparing to 3 datanodes in the cluster.

However, the query time speedup on Impala tables with Parquet file format does not scale very well because the query time is already quite low. Thus, we can gain speed up about 1.2 times when we use 8 datanodes comparing to 3 datanodes in the cluster.

4.4 Web Traffic Log Analyzer UI

Our large-scale web traffic log analyzer is a web application implemented in PHP language. On the dashboard, it displays the information of Top 10 visiting IP, Top 10 website access, Top 10 URL error and Top 10 webpage access as shown in Fig.10. Also, system administrators can generate reports in the specific range of time as shown in Fig.11.

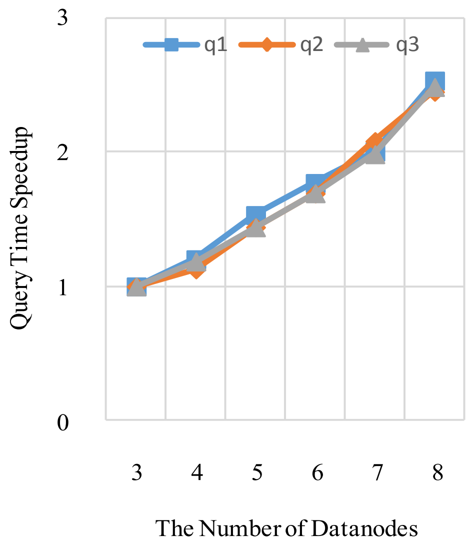


Fig. 8. The Query Time Speedup on Impala Tables with Plain Text File Format.

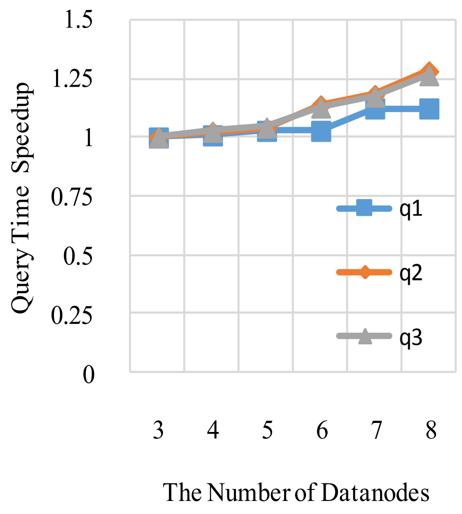


Fig. 9. The Query Speedup on Impala Tables with Paquet File Format.

We also measure the response time of each page. The average response time of each page is around 4 seconds on the 50 GB of log size and 8 datanodes in the HDFS cluster.

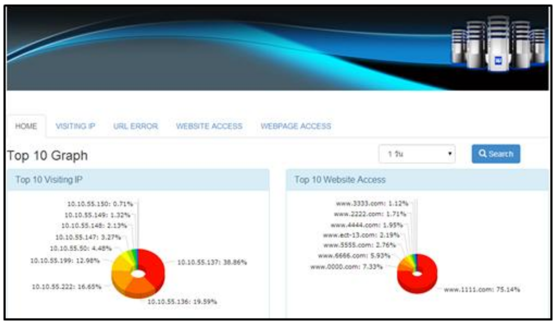
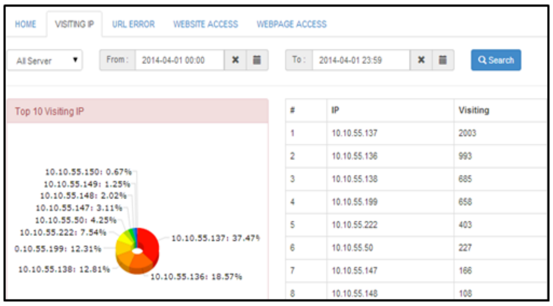


Fig. 10. Web Traffic Log Analyzer Dashboard.



#	IP	Visiting
1	10.10.55.137	2003
2	10.10.55.136	993
3	10.10.55.138	685
4	10.10.55.199	658
5	10.10.55.222	403
6	10.10.55.50	227
7	10.10.55.147	166
8	10.10.55.148	108

Fig. 11. Web Traffic Log Analyzer Detail Page.

5. Conclusion

This paper presents the design and implementation of a large-scale web traffic log analyzer using PHP language together with HDFS to store traffic log, Cloudera Impala to query data from HDFS, Apache Flume to receive apache web server traffic log and send them to store in HDFS, and Apache Thrift which is a communication channel between our web in PHP language and Cloudera Impala API in Java language.

In our experiments, we have queried data from HDFS via Impala tables with plain text file format and Impala tables with Parquet file format. We found that the query time usage on Impala tables with Parquet file format is a lot faster than on Impala tables with plain text file format. With 50 GB of traffic log data stored on 8 datanodes of HDFS cluster, the query time usage for Impala table with plain text file format takes around 15 seconds whereas 0.9 seconds is used for querying data on Impala table with Parquet file format.

Also, the use of hybrid between Impala tables with plain text file format to store the current day traffic log and Impala tables with Parquet file format to store history traffic log, each page of our web traffic log analyzer has only about 4 seconds response time.

6. Reference

[1] AWstats, Available: http://www.awstats.org/, 04 March 2016.

[2] OWA, Available: http://www.openwebanalytics.com/, 04 March 2016.

[3] Piwik, Available: http://www.piwik.org/, 04 March 2016.

[4] C. Rattanapoka, “The Design and Implementation of Computer Traffic Log Searcher System using Hadoop Map/Reduce Framework”, The Journal of Industrial Technology 8(3), 2012. (in Thai).

[5] S. Narkhede and T. Baraskar, “HMR Log Analyzer: Analyze Web Application Logs over Hadoop Mapreduce”, The International Journal of UbiComp (IJU) 4(3), July 2013.

[6] S. Adhikari, D. Saraf, M. Revanwar and N. Ankam, “Analysis of Log Data and Statistics Report Generation using Hadoop”, The International Journal of Innovative Research in Computer and Communication Engineering 2(4), April 2014.

[7] Apache Pig, Available: https://pig.apache.org/

[8] K. Shvachko, H. Kuang, S. Radia and R. Chansler, “The Hadoop Distributed File System”, The 26th IEEE Symposium on Massive Storage Systems and Technologies, 2010.

[9] S. Ghemawat, H. Gobioff and S.T. Leung, “The Google File System”, Proceedings of the 19th ACM Symposium on Operating Systems Principles, 2003.

[10] Cloudera Impala, Available: http://www.cloudera .com/products/apache-hadoop/impala.html, 04 March 2016.

[11] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters”, Communications of the ACM 51, 2008, pp 107-113.

[12] Apache Parquet, Available: http://parquet.apache .org/, 04 March 2016.

[13] Apache Flume, Available: http://flume.apache. org/, 04 March 2016.

[14] Apache Thrift, Available: http://thrift.apache. org/, 04 March 2016.

[15] Elasticsearch, Available: https://www.elastic.co/

[16] PHP Impala Phar, Available: https://github.com/ rmcfrazier/ php_impala_phar, 04 March 2016.