Data Analysis Using MapReduce in Hadoop Environment

Muhammad Khairul Rijal Muhammad*, Saiful Adli Ismail, Mohd Nazri Kama, Othman Mohd Yusop, Azri Azmi

Advanced Informatics School (UTM AIS), Universiti Teknologi Malaysia (UTM), Jalan Sultan Yahya Petra, Kuala Lumpur, Malaysia.

Abstract

Data is growing at every moment which makes it almost impossible to process all of it. More than a decade ago, researchers from big company in IT industry such as Google and Yahoo introduce Hadoop and MapReduce as a solution for big volume of data processing. Now, by running on a free and open platform both method had gain popularity among the industry and academicians in the academic world. Thus, this paper will use MapReduce method in Hadoop environment and apply it on a sample dataset as a way to analyse the given data. The purpose of study is to gain understanding of using MapReduce in Hadoop environment.

Keywords: Hadoop, Data Analysis, MapReduce, HDFS

1. INTRODUCTION

Since the computer being made, data has become essential part in life. The growth of data is unstoppable and by 2020, it is said the amount of data will reach 44 zettabytes or equivalent to 44 trillion gigabytes. Data comes in three types: structured, unstructured and semi-structured and it comes from various of source such as documents, computers, sensor and many more. Companies such as Google, Yahoo, Amazon, eBay and so on deal with huge volume of data every day. The big problem is to handle such a huge volume of data for human benefit. Increasingly, data is seeing as an asset. Data that being collected normally has enormous potential and support decision making for people and organization. Traditional way of managing data collection is database management system or called DBMS. DBMS serves as an interface between databases and application or end users. In DBMS, most of the databases are already organized and structured. In real world, data that is still growing and coming from various sources are mostly unstructured and require big investment for IT infrastructure.

* Corresponding author. *E-mail address* mkrijal@gmail.com

In 2002, Doug Cutting started a project call Nutch with a purpose to make a better web search engine. Subsequently, Google released a Google File System and MapReduce paper in 2003. Project Nutch evolved to be Hadoop and MapReduce becomes the programming model of Hadoop.

Now, Hadoop is famous for being an open source software framework with scalable distributed computing that can provide fast and reliable analysis of structured and unstructured data [1]. MapReduce is a programming model that has been ingrained into Hadoop. It is considered to be the most popular computing paradigm for parallel, batch style and analysis of large amount of data [2].

In this paper, section 2 will explain in detail about the relationship between MapReduce and Hadoop. While section 3 will discuss the experiment and result of data analysis based on map reduce programming model using wordcount process. The last section will provide the conclusion of this paper.

2. RELATED WORK

Hadoop, formerly called Apache Hadoop is an open source framework that is based on Java programming. It is a project sponsored by Apache Software Foundation [3]. Hadoop can handle thousand terabytes of data and allow applications to run on systems with thousands of commodities hardware node. One of the advantage of Hadoop is that they are not relying on hardware to deliver high availability but have a library that can detect and handle failures at the application layers which subsequently enable highly-available service to be delivered on top of cluster of computers, each of which may prone to failures [4].

The core component of Hadoop is HDFS and MapReduce [5]. HDFS or Hadoop Distributed File System is a storage and processing system designed for very large amount of distributed unstructured data. HDFS is a reliable place to store data and provide fast and scalable access to information [6]. Furthermore, HDFS is fault tolerance and a cost-efficient storage for big data.

MapReduce is a framework for processing parallelizable problems across large datasets using many computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogenous hardware[7]. MapReduce can take advantage of locality of data, processing it on or near the storage assets in order to reduce the distance over which it must be transmitted [8]. According to Czajkowski et. al, MapReduce can handle multiple process simultaneously even at the scale of petabytes [9]. Both HDFS and MapReduce will be discussed in detail below.

3. HDFS

An HDFS cluster is comprised of a NameNode, which manages the cluster metadata, and DataNodes that store the data. Files and directories are represented on the NameNode by inodes. Inodes record attributes like permissions, modification and access times, or namespace and disk space quotas. The file content is split into large blocks (typically 128 megabytes), and each block of the file is independently replicated at multiple DataNodes. The blocks are stored on the local file system on the DataNodes [10]. Below is the architecture of HDFS.

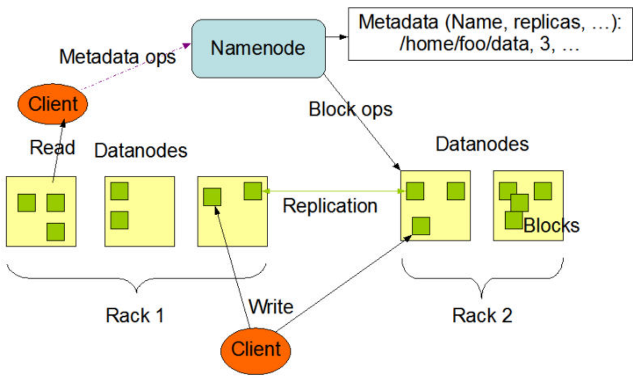


Figure 1. HDFS Architecture [10]

Both components of HDFS are software which run on Linux operating system and cheap commodity hardware [5]. Technically, HDFS can be implemented at low cost. By using Java as their base, HDFS is generally wide accepted among users because of the free and open platform. HDFS is fault tolerant. During the process, the Namenode actively monitors the number of replicas of a block. It does not directly send requests to DataNodes. The Namenode send instruction to DataNode which commands containing replicate blocks to other nodes, remove local block replicas, re-register and send an immediate block report, or shut down the node.

4. MAPREDUCE

MapReduce gained its popularity when Google used it successfully [2]. MapReduce is a programming model and is preferred among industry and academic world because of the fault tolerance, simplicity and scalability. The primary objective of MapReduce is to split input data set into independent chunks that are processed in a completely parallel manner [16]. Initially, a distributed file system (DFS) partitions data in multiple machines and data is represented as (key, value) pairs. On a single master machine, map function is called to map the job which takes a set of data and converts it into another set of data and later the reduce function takes the output from a map as input and combines those data tuples into a smaller set of tuples. A pair of map and reduce functions may be executed once or numerous times as it depends on the characteristics of an application [17]. Figure 2 shows Hadoop MapReduce architecture.

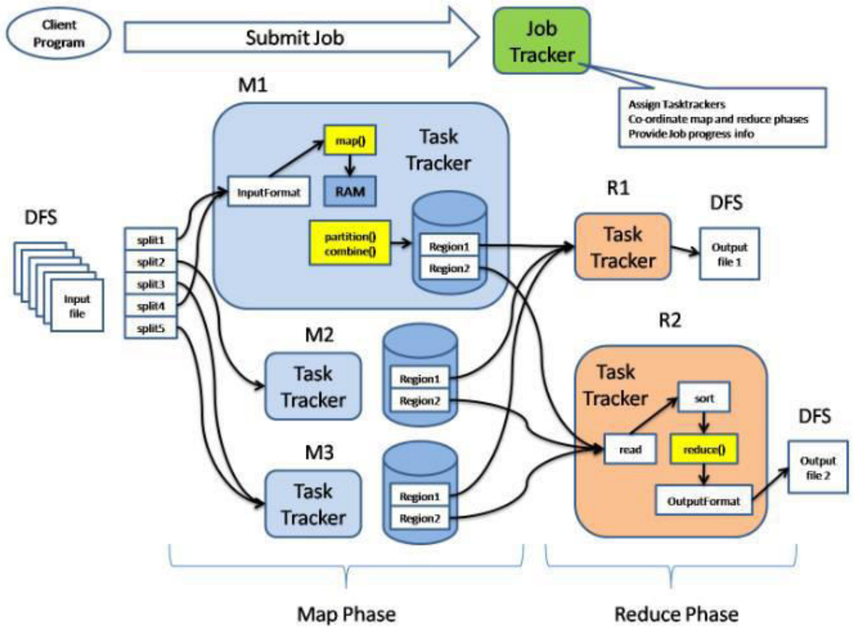


Figure 2. Hadoop MapReduce Architecture[18]

Based on the above figure, there is two main processing stages. The first stage is Map phase and the second stage is Reduce phase. The actual MapReduce process happens in task tracker which is tagged as M1. In between map and reduce stages, there will be an intermediate process. During the intermediate process, the operations such as shuffle and sorting of the mapper output data will be executed. The intermediate data is going to get stored in local file system [18]. Below is the diagram that illustrate the scenario.

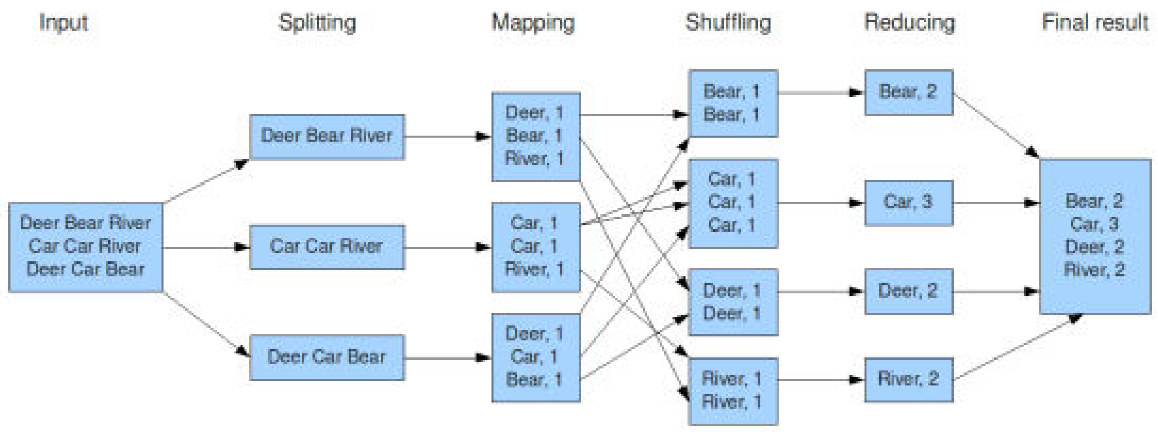


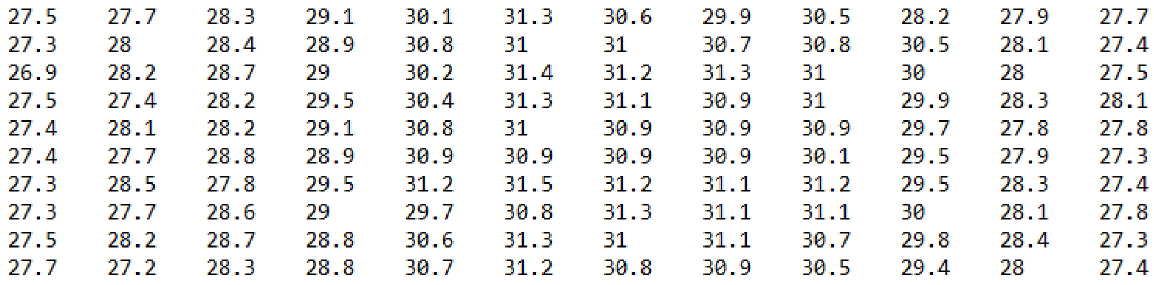
Figure 3. MapReduce Word count Process[19]

The next section will discuss the experiment of wordcount process as MapReduce programming model using Java programming language and Hadoop environment on sample dataset.

5. EXPERIMENT

The experiment required Java IDE Eclipse, Hortonworks Sandbox version 2.1, Jar file that containing library for Hadoop and dataset sample. Java IDE Eclipse is used for writing the source code for wordcount program using Java programming language. Hortonworks Sandbox version 2.1 acts as the Hadoop environment for the experiment. It can be downloaded from Sandbox download site. The sandbox requires computer with at least 8 GB of RAM and can be run in virtual environment such as Virtual Box[20]. A Java library for Hadoop need to be downloaded to make source code functioning. Lastly, a sample dataset containing estimation for Malaysia monthly average temperature from 2041 until 2050. To summarize it, a simple application of MapReduce called wordcount will be developed in Java IDE Eclipse and tested in Hortonworks Sandbox to illustrate the application of MapReduce in Hadoop.

The dataset contains data in numeric value with 12 column representing months and 10 rows representing years. The data is saved in text file format and taken from National Hydraulic Research Institute of Malaysia (NAHRIM) website. The purpose of using this dataset is to analyse the estimation of monthly average temperature in Malaysia from 2041 to 2050. Table below shown the contain of dataset.



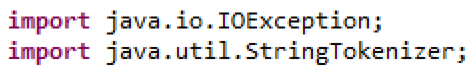
27.5	27.7	28.3	29.1	30.1	31.3	30.6	29.9	30.5	28.2	27.9	27.7
27.3	28	28.4	28.9	30.8	31	31	30.7	30.8	30.5	28.1	27.4
26.9	28.2	28.7	29	30.2	31.4	31.2	31.3	31	30	28	27.5
27.5	27.4	28.2	29.5	30.4	31.3	31.1	30.9	31	29.9	28.3	28.1
27.4	28.1	28.2	29.1	30.8	31	30.9	30.9	30.9	29.7	27.8	27.8
27.4	27.7	28.8	28.9	30.9	30.9	30.9	30.9	30.1	29.5	27.9	27.3
27.3	28.5	27.8	29.5	31.2	31.5	31.2	31.1	31.2	29.5	28.3	27.4
27.3	27.7	28.6	29	29.7	30.8	31.3	31.1	31.1	30	28.1	27.8
27.5	28.2	28.7	28.8	30.6	31.3	31	31.1	30.7	29.8	28.4	27.3
27.7	27.2	28.3	28.8	30.7	31.2	30.8	30.9	30.5	29.4	28	27.4

Figure 4. Malaysian Montly Average Temperature from 2041 to 2050

6. MAP AND REDUCE

Firstly, a new Java project need to be created in Java IDE Eclipse. The project must contain a package and at least one class to run the coding. Inside the class, the first step is to import the library to execute MapReduce. There is two set of import as shown in the list below.

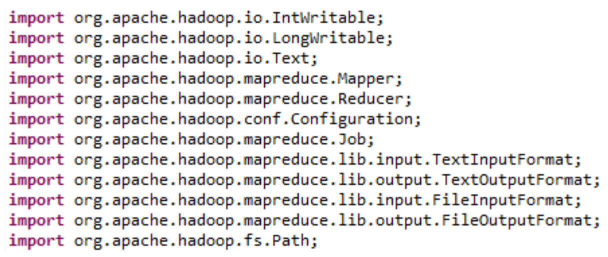
1) *Java Class:* Import Java function such as IOException and StringTokenizer that can be apply on input output data and string input.



```
import java.io.IOException;
import java.util.StringTokenizer;
```

Figure 5. Import Java Classes

2) *Hadoop class:* Import Hadoop library that contains function that enable features for MapReduce to be apply.

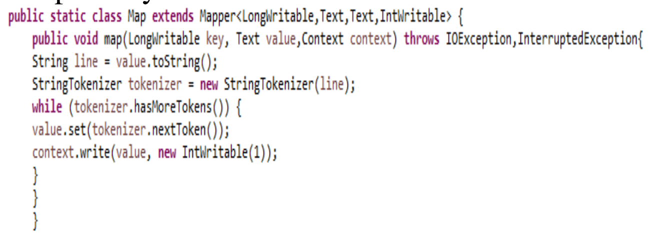


```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.fs.Path;
```

Figure 6. Import Hadoop Classes

The coding section will be divided into three parts: Mapper, Reduce and Configuration. Set of codes are as below.

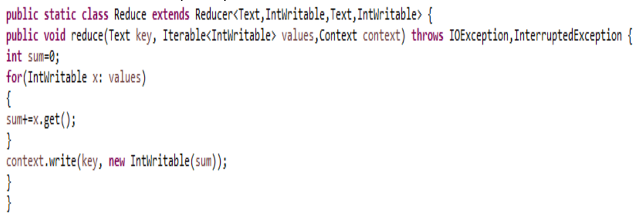
1) *Mapper*: A function that split each line of data in dataset and then map the data inside the line separately.



```
public static class Map extends Mapper<LongWritable,Text,Text,IntWritable> {
    public void map(LongWritable key, Text value,Context context) throws IOException,InterruptedException{
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
    value.set(tokenizer.nextToken());
    context.write(value, new IntWritable(1));
    }
    }
    }
```

Figure 7. Mapper function

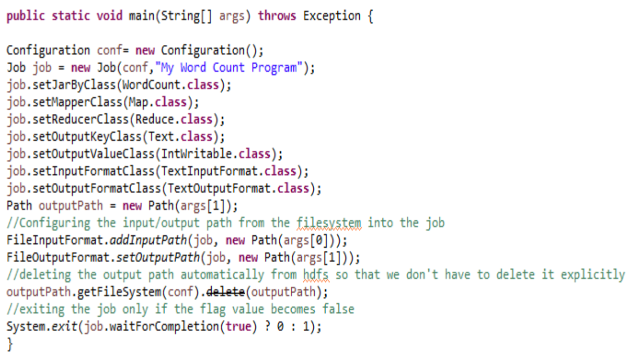
Reduce: A function that takes the result of Mapper. Firstly, by shuffling the map data and then count the occurrence to reduce it.



```
public static class Reduce extends Reducer<Text,IntWritable,Text,IntWritable> {
public void reduce(Text key, Iterable<IntWritable> values,Context context) throws IOException,InterruptedException {
int sum=0;
for(IntWritable x: values)
{
sum+=x.get();
}
context.write(key, new IntWritable(sum));
}
}
```

Figure 8. Reducer function

Configuration: A function that call all classes that have been imported to run.



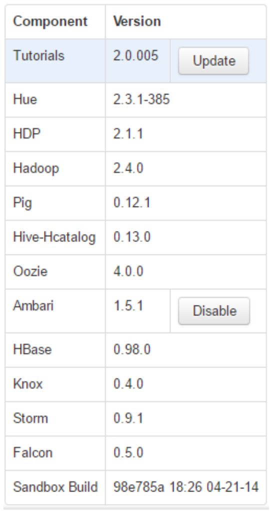
```
public static void main(String[] args) throws Exception {

Configuration conf= new Configuration();
Job job = new Job(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);
//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
//deleting the output path automatically from hdfs so that we don't have to delete it explicitly
outputPath.getFileSystem(conf).~~delete~~(outputPath);
//exiting the job only if the flag value becomes false
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Figure 9. Configuration function

7. HORTONWORKS SANDBOX

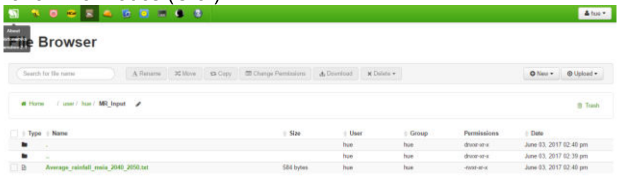
For this experiment, Hortonworks Sandbox version 2.1 will be used as Hadoop environment. The Sandbox version 2.1 uses Linux Centos as their operating system. Figure 10 is the list of component inside Hortonworks Sandbox version 2.1:



Component	Version	
Tutorials	2.0.005	Update
Hue	2.3.1-385	
HDP	2.1.1	
Hadoop	2.4.0	
Pig	0.12.1	
Hive-Hcatalog	0.13.0	
Oozie	4.0.0	
Ambari	1.5.1	Disable
HBase	0.98.0	
Knox	0.4.0	
Storm	0.9.1	
Falcon	0.5.0	
Sandbox Build	98e785a 18:26 04-21-14	

Figure10. Hortonworks Sandbox 2.1 Component

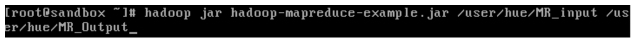
The first step is to create a folder for input file. Type in the url browser http://127.0.0.1:8000 and then login as hue. Go to File Browser and create a new folder. Then, upload the text file containing the sample data as shown in the figure below.



Type	Name	Size	User	Group	Permissions	Date
	.		hue	hue	drwxr-xr-x	June 03, 2017 02:40 pm
	..		hue	hue	drwxr-xr-x	June 03, 2017 02:39 pm
	Average_rainfall_msia_2040_2050.txt	584 bytes	hue	hue	-rwxr-xr-x	June 03, 2017 02:40 pm

Figure 11. File uploaded in the File Browser

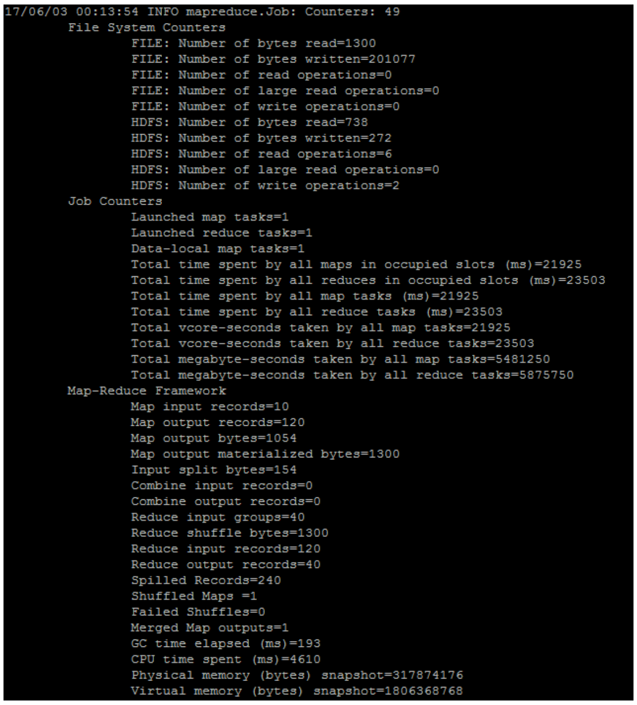
The Java project that has been created need to be saved in Java Archive format (JAR). As the experiment would be run in Hadoop environment, the JAR file need to be uploaded into Hortonworks Sandbox version 2.1. The Java runtime saved for the project must be similar to the one inside Sandbox to avoid compatibility issue. To execute the JAR file, use the terminal and type command as shown below.



```
[root@sandbox ~]# hadoop jar hadoop-mapreduce-example.jar /user/hue/MR_input /us
er/hue/MR_Output_
```

Figure 12 Command to execute MapReduce JAR file

The successful execution will appear as in figure 13.

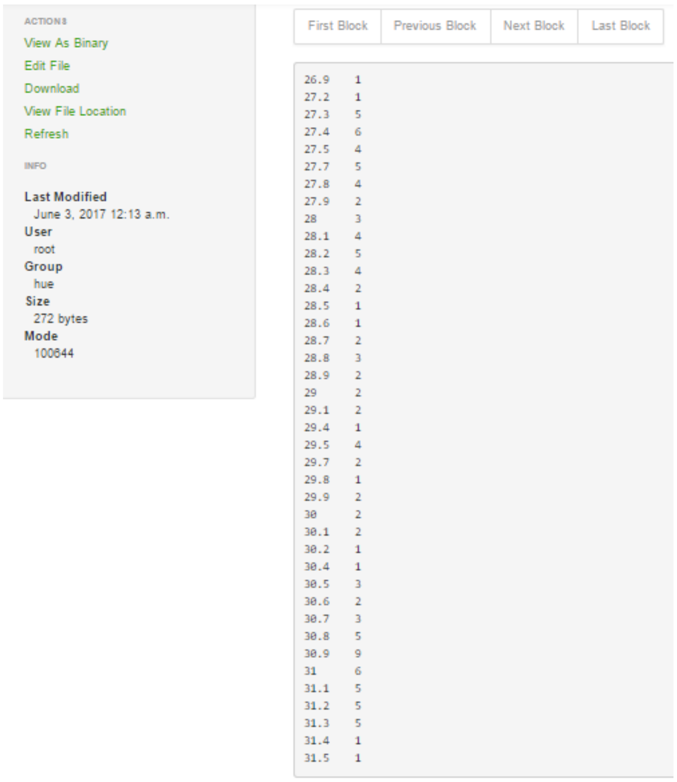


```
17/06/03 00:13:54 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=1300
                FILE: Number of bytes written=201077
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=738
                HDFS: Number of bytes written=272
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=21925
                Total time spent by all reduces in occupied slots (ms)=23503
                Total time spent by all map tasks (ms)=21925
                Total time spent by all reduce tasks (ms)=23503
                Total vcore-seconds taken by all map tasks=21925
                Total vcore-seconds taken by all reduce tasks=23503
                Total megabyte-seconds taken by all map tasks=5481250
                Total megabyte-seconds taken by all reduce tasks=5875750
        Map-Reduce Framework
                Map input records=10
                Map output records=120
                Map output bytes=1054
                Map output materialized bytes=1300
                Input split bytes=154
                Combine input records=0
                Combine output records=0
                Reduce input groups=40
                Reduce shuffle bytes=1300
                Reduce input records=120
                Reduce output records=40
                Spilled Records=240
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=193
                CPU time spent (ms)=4610
                Physical memory (bytes) snapshot=317874176
                Virtual memory (bytes) snapshot=1806368768
```

Figure 13 MapReduce process

8. RESULT

The output of the MapReduce program can be seen inside the output folder that has been created. Open back File Browser in Hue and search the folder called MR_Output. Figure 14 shows the output result.



Last Modified
June 3, 2017 12:13 a.m.
User
root
Group
hue
Size
272 bytes
Mode
100644

```
26.9	1
27.2	1
27.3	5
27.4	6
27.5	4
27.7	5
27.8	4
27.9	2
28	3
28.1	4
28.2	5
28.3	4
28.4	2
28.5	1
28.6	1
28.7	2
28.8	3
28.9	2
29	2
29.1	2
29.4	1
29.5	4
29.7	2
29.8	1
29.9	2
30	2
30.1	2
30.2	1
30.4	1
30.5	3
30.6	2
30.7	3
30.8	5
30.9	9
31	6
31.1	5
31.2	5
31.3	5
31.4	1
31.5	1
```

Figure 14 Output of MapReduce program

Based on the result, all the data have been sort in pairs from the lowest to the highest. At the top of the list is the lowest recorded monthly average temperature which happen only once while at the bottom is the highest recorded monthly average temperature which also happen once. The estimation shows the monthly average temperature will be recorded between 26.9°c and 31.5°c and the most likely to occur is 30.9°c.

9. CONCLUSION

Based on this research it can be concluded that MapReduce simplified the concept of filtering and analyzing data. Traditional method such as DBMS use the query language which performance will suffer when there is a lot of querying need to be done at the same time. The wordcount program did illustrate the concept of MapReduce though it has been over simplified if compare to real world. Nevertheless, the introduction of MapReduce is crucial as it innovates the data processing techniques especially when there is a need for fast response albeit in the large volume of data.

MapReduce run well in Hadoop environment as they are built in the same Java language. Hadoop can speed-up the processing of large amount of data through distributed process and thus enable the fast response. The advantage of fault-tolerance, free and open framework while also offering big data management will tempt industry and academicians in using it for the purpose of advancement.

ACKNOWLEDGEMENT

The authors would like to thanks Advanced Informatics School (AIS), Universiti Teknologi Malaysia Kuala Lumpur for the support in R&D.

REFERENCES

[1] "What is Apache Hadoop?" *Hortonworks*, 17-May-2016. [Online]. Available: https://hortonworks.com/apache/hadoop/. [Accessed: 02-Jun-2017].
[2] S. Maitrey and C. K. Jha, "MapReduce: Simplified Data Analysis of Big Data," *Procedia Comput. Sci.*, vol. 57, pp. 563–571, Jan. 2015.
[3] "What is Hadoop? - Definition from WhatIs.com," *SearchCloudComputing*. [Online]. Available: http://searchcloudcomputing.techtarget.com/definition/Hadoop. [Accessed: 02-Jun-2017].
[4] "Welcome to Apache™ Hadoop®!" [Online]. Available: http://hadoop.apache.org/. [Accessed: 02-Jun-2017].
[5] K. Dwivedi and S. K. Dubey, "Analytical review on Hadoop Distributed file system," in *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*, 2014, pp. 174–181.
[6] F. Chang *et al.*, "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans Comput Syst*, vol. 26, no. 2, p. 4:1–4:26, Jun. 2008.
[7] V. K. Jain, *Big Data and Hadoop*. Khanna Publishing, 2017.
[8] "Define: MapReduce and Text Parsing." [Online]. Available: http://quicktechie.com/cs/data-science-q-a/165-define-mapreduce-and-text-parsing. [Accessed: 02-Jun-2017].
[9] "Sorting Petabytes with MapReduce - The Next Episode," *Research Blog*. .
[10] "Apache Hadoop HDFS," *Hortonworks*, 21-Mar-2016. [Online]. Available: https://hortonworks.com/apache/hdfs/. [Accessed: 02-Jun-2017].
[11] "What is Hadoop MapReduce? Webopedia Definition." [Online]. Available: http://www.webopedia.com/TERM/H/hadoop_mapreduce.html. [Accessed: 02-Jun-2017].
[12] Y. Kim and K. Shim, "Parallel Top-K Similarity Join Algorithms Using MapReduce," in *2012 IEEE 28th International Conference on Data Engineering*, 2012, pp. 510–521.
[13] S. User, "Hadoop Map Reduce Architecture and Example." [Online]. Available: http://a4academics.com/tutorials/83-hadoop/840-map-reduce-architecture. [Accessed: 02-Jun-2017].
[14] "map_reduce_tutorial.pdf." .
[15] "Sandbox Deployment and Install Guide," *Hortonworks*, 24-Apr-2017. [Online]. Available: https://hortonworks.com/hadoop-tutorial/sandbox-deployment-install-guide/. [Accessed: 02-Jun-2017].
[16] "CodHoop: A system for optimizing big data processing (PDF Download Available)," *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/283123951_CodHoop_A_sy stem_for_optimizing_big_data_processing. [Accessed: 21-Jun-2017].

[17] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *Commun ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[18] S. User, “Hadoop Map Reduce Architecture and Example.” [Online]. Available: http://a4academics.com/tutorials/83-hadoop/840-map-reduce-architecture. [Accessed: 02-Jun-2017].

[19] “map_reduce_tutorial.pdf.” .

[20] “Sandbox Deployment and Install Guide,” *Hortonworks*, 24-Apr- 2017. [Online]. Available: https://hortonworks.com/hadoop-tutorial/sandbox-deployment-install-guide/. [Accessed: 02-Jun-2017].