Assignment No.2

Title: Design a distributed application using MapReduce (Using Java) which processes a log file of a system. List out the users who have logged for maximum period on the system. Use simple log file from the Internet and process it using a pseudo distribution mode on Hadoop platform.

Objectives: To learn the concept of Mapper and Reducer and implement it for log file processing

Aim: To implement a MapReduce program that will process a log file of a system.

Theory:

Introduction

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.MapReduce is a processing technique and a program model for distributed computing based on java.

The MapReduce algorithm contains two important tasks, namely Map and Reduce.Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce taskis always performed after the map job.

Under the MapReduce model, the data processing primitives are called mappers and reducers.

once we writean application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change

Algorithm

- MapReduce program executes in three stages, namely **map stage**, **shuffle stage**, and **reduce stage**.
    - Input : file or directory
    - Output : Sorted file**<key, value>**

1. **Mapstage :**
    - The map or mapper‟s job is to process the input data.
    - Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS).
    - The input file is passed to the mapper function line by line.
    - The mapper processes the data and creates several small chunks of data.

2. **Shuffle stage:**
   - This phase consumes the output of mapping phase.
   - Its task is to consolidate the relevant records from Mapping phase output
3. **Reduce stage:**
   - This stage is the combination of the Shuffle stage and the Reduce stage.
   - The Reducer‟s job is to process the data that comes from the mapper.
   - After processing, it produces a new set of output, which will be stored in the HDFS.

Inserting Data into HDFS:

- The MapReduce framework operates on **<key, value>**pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of thejob, conceivably of different types.
- The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework.
- Input and Output types of a MapReduce job: **(Input) <k1, v1> -> map -><k2, v2>-> reduce ->**<k3, v3> (Output).

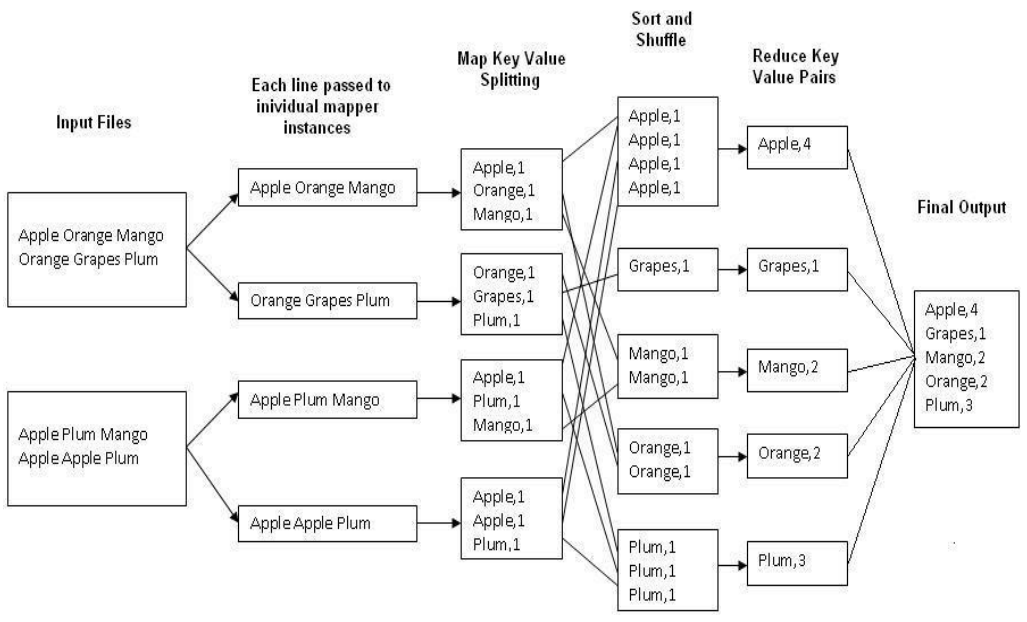

Figure: An Example Program to Understand working of MapReduce Program

#Mapper Class

```
packageSalesCountry;

importjava.io.IOException;

importorg.apache.hadoop.io.IntWritable;
importorg.apache.hadoop.io.LongWritable;
importorg.apache.hadoop.io.Text;
importorg.apache.hadoop.mapred.*;

public class SalesMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text,
IntWritable> {
        private final static IntWritable one = new IntWritable(1);

        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException {

                String valueString = value.toString();
                String[] SingleCountryData = valueString.split("-");
                output.collect(new Text(SingleCountryData[0]), one);
        }
}
```

#Reducer Class

```
packageSalesCountry;
importjava.io.IOException;
importjava.util.*;

importorg.apache.hadoop.io.IntWritable;
importorg.apache.hadoop.io.Text;
importorg.apache.hadoop.mapred.*;

public class SalesCountryReducer extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {

        public void reduce(Text t_key, Iterator<IntWritable> values,
OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException {
                Text key = t_key;
                intfrequencyForCountry = 0;
                while (values.hasNext()) {
                        // replace type of value with the actual type of our value
                        IntWritable value = (IntWritable) values.next();
                        frequencyForCountry += value.get();

                }
                output.collect(key, new IntWritable(frequencyForCountry));
        }
}
```

#Driver Class

```
packageSalesCountry;

importorg.apache.hadoop.fs.Path; import
org.apache.hadoop.io.*;
importorg.apache.hadoop.mapred.*;

public class SalesCountryDriver {
        public static void main(String[] args) {
                JobClientmy_client = new JobClient();
                // Create a configuration object for the job
                JobConfjob_conf = new JobConf(SalesCountryDriver.class);

                // Set a name of the Job
                job_conf.setJobName("SalePerCountry");

                // Specify data type of output key and value
                job_conf.setOutputKeyClass(Text.class);
                job_conf.setOutputValueClass(IntWritable.class);

                // Specify names of Mapper and Reducer Class
                job_conf.setMapperClass(SalesCountry.SalesMapper.class);
                job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);

                // Specify formats of the data type of Input and output
                job_conf.setInputFormat(TextInputFormat.class);
                job_conf.setOutputFormat(TextOutputFormat.class);

                // Set input and output directories using command line arguments,
                //arg[0] = name of input directory on HDFS, and arg[1] =  name of output
directory to be created to store the output file.

                FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
                FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

                my_client.setConf(job_conf);try
                {
                        // Run the job
                        JobClient.runJob(job_conf);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Steps for Compilation & Execution of Program:

```
#sudomkdiranalyzelogsls
 #sudochmod -R 777 analyzelogs/cd
            ls cd
            ..
            pwd
            ls cd
            pwd
#sudochown -R hduseranalyzelogs/cd
            ls
#cd analyzelogs/ls
            cd ..
```

Copy the Files (Mapper.java,Reduce.java,Driver.java to Analyzelogs Folder)

```
#sudocp /home/mde/Desktop/count_logged_users/* -/analyzelogs/
```

Start HADOOP

```
#start-dfs.sh
#start-yarn.sh
#jps
            cd
            cdanalyzelogs
            ls
            pwd
            ls
#ls -ltr #ls
-al
#sudochmod +r *.*
            pwd
```

#export CLASSPATH="$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.9.0.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.9.0.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-2.9.0.jar:~/analyzelogs/SalesCountry/*:$HADOOP_HOME/lib/*"

Compile Java Files

```
# javac -d . SalesMapper.java SalesCountryReducer.java SalesCountryDriver.javals
 #cd SalesCountry/ls
cd ..
  #sudogedit Manifest.txt
  #jar -cfm analyzelogs.jar Manifest.txt SalesCountry/*.classls
cd
jps
#cd analyzelogs/
```

Create Directory on Hadoop

```
#sudomkdir ~/input2000ls
pwd
 #sudocp access_log_short.csv ~/input2000/
# $HADOOP_HOME/bin/hdfsdfs -put ~/input2000 /
 # $HADOOP_HOME/bin/hadoop jar analyzelogs.jar /input2000 /output2000#
$HADOOP_HOME/bin/hdfsdfs -cat /output2000/part-00000
```

```
# stop-all.sh
# jps
```

Conclusion: Thus, we have learnt how to design a distributed application using MapReduce and process a log file of a system.