Project Report

On

COVID - Analysis from the csv file using MapReduce Programming.

&

"Run Map and Reduce codes" tasks performed on Big Data Platform such as Hadoop

Submitted By:

TUSHAR AGARWAL

Declaration

I hereby declare that the information given in this application is true and correct to the best of my knowledge and belief. In case any information given in this application proves to be false or incorrect, I shall be responsible for the consequences.

Name: - TUSHAR AGARWAL

Date: - 18/03/2021

Abstract

This project "**Analysis from the csv file using MapReduce Programming**" is basically **the task is to count the total number of reported cases for every country/location till April 8th, 2020**. This is project is developed on "**Big Data Platform Hadoop"** in backend and "**MYSQL**" as a database. This project keeps the record of total number of active covid cases country wise.

The product will help the user to work in a highly effective and efficient environment.

We are in the world of technology, where data is moving around all the time. This data is becoming bigdata when it comes in huge volume and data can be structured, unstructured or semi-structured. hadoop is the technology to analysis the big data. The objective of this paper is to analysis the data that is collected from the open source using the Hadoop and mapreduce programming model.

Project: COVID - Analysis from the csv file using MapReduce Programming.

File: MapperClass.java

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class MapperClass extends MapReduceBase implements Mapper
<LongWritable, Text, Text, IntWritable> {

   // initialize the field variable
   private final static IntWritable one = new IntWritable(1);
   private final static int LOCATION = 1;
   private final static int NEW_CASES = 2;
   private final static String CSV_SEPARATOR = ",";


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

      // initiate the variable
      String valueString = value.toString();

      // split the data with CSV_SEPARATOR
      String[] columnData = valueString.split(CSV_SEPARATOR);

      // collect the data with defined column
      output.collect(new Text(columnData[LOCATION]), new
IntWritable(Integer.parseInt(columnData[NEW_CASES])));
   }
}
```

File: ReducerClass.java

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

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class ReducerClass 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 {

      // determine key object and counter variable
      Text key = t_key;
      int counter = 0;

      // as long that the values inside the data being mapped,
      // will counting how many data with the same key
      while (values.hasNext()) {

         // replace type of value with the actual type of our value
         IntWritable value = (IntWritable) values.next();
         counter += value.get();
      }
      output.collect(key, new IntWritable(counter));
   }
}
```

File: MainClass.java

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.*;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

public class MainClass {
```

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

        // create new JobClient
        JobClient my_client = new JobClient();

        // Create a configuration object for the job
        JobConf job_conf = new JobConf(MainClass.class);

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

        // 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(MapperClass.class);
        job_conf.setReducerClass(ReducerClass.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.

        // called the input path for file and defined the output path
        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();
```

```
        }
    }
}
```

How to execute:
Step 1. Creating a directory named as classes/
$ mkdir classes

Step 2. Compiling the source file using the following command.
$ javac -cp hadoop-common-2.2.0.jar:hadoop-mapreduce-client-core-2.7.1.jar:classes:. -d classes/ *.java

Step 3: Creating Jar file for the above created classes which are stored in classes folder.
$ jar -cvf CountMe.jar -C classes/ .
# do not forget to put <space> dot at the end inthe above command.

Output:

added manifest
adding: MainClass.class(in = 1609) (out= 806)(deflated 49%)
adding: MapperClass.class(in = 1911) (out= 754)(deflated 60%)
adding: ReducerClass.class(in = 1561) (out= 628)(deflated 59%)

Step 4: upload the csv file to hadoop distributed file system
$ hadoop fs -put covid.csv .
# do not forget to remove the header line from the covid file provided before uploading to HDFS
# do not forget to put <space> dot in the above command to upload it to the home folder.

Step 5: Running the Hadoop file using hadoop jar command.
$ hadoop jar CountMe.jar MainClass covid.csv output/

Step 6: Checking the output folder has been populated or not and also printing the output on terminal

$ hadoop fs -ls output/
$ hadoop fs -cat output/part-00000

Afghanistan 367
Albania 383
Algeria 1468
Andorra 545
Angola 17
Anguilla 3
Antigua and Barbuda 15
Argentina 1715
Armenia 853
Aruba 74
Australia 5956
Austria 12640
Azerbaijan 717
Bahamas 36
Bahrain 811
Bangladesh 164
Barbados 63

Output is trimmed here.

Project: “Run Map and Reduce codes”
Tasks on Big data platform such as Hadoop

Solutions:

Run Map and Reduce codes

First Hadoop MapReduce Program

	A	B	C	D	E	F	G	H	I	J	K	L
1	Transaction_date	Product	Price	Payment_	Name	City	State	Country	Account_Created	Last_Login	Latitude	Longitude
2	01-02-2009 06:17	Product1	1200	Mastercar	carolina	Basildon	England	United Kir	01-02-2009 06:00	01-02-2009 06:08	51.5	-1.11667
3	01-02-2009 04:53	Product1	1200	Visa	Betina	Parkville	MO	United Sta	01-02-2009 04:42	01-02-2009 07:49	39.195	-94.6819
4	01-02-2009 13:08	Product1	1200	Mastercar	Federica e	Astoria	OR	United Sta	01-01-2009 16:21	01-03-2009 12:32	46.18806	-123.83
5	01-03-2009 14:44	Product1	1200	Visa	Gouya	Echuca	Victoria	Australia	9/25/05 21:13	01-03-2009 14:22	-36.1333	144.75
6	01-04-2009 12:56	Product2	3600	Visa	Gerd W	Cahaba He	AL	United Sta	11/15/08 15:47	01-04-2009 12:45	33.52056	-86.8025
7	01-04-2009 13:19	Product1	1200	Visa	LAURENCE	Mickleton	NJ	United Sta	9/24/08 15:19	01-04-2009 13:04	39.79	-75.2381
8	01-04-2009 20:11	Product1	1200	Mastercar	Fleur	Peoria	IL	United Sta	01-03-2009 09:38	01-04-2009 19:45	40.69361	-89.5889
9	01-02-2009 20:09	Product1	1200	Mastercar	adam	Martin	TN	United Sta	01-02-2009 17:43	01-04-2009 20:01	36.34333	-88.8503
10	01-04-2009 13:17	Product1	1200	Mastercar	Renee Elis	Tel Aviv	Tel Aviv	Israel	01-04-2009 13:03	01-04-2009 22:10	32.06667	34.76667
11	01-04-2009 14:11	Product1	1200	Visa	Aidan	Chatou	Ile-de-Fra	France	06-03-2008 04:22	01-05-2009 01:17	48.88333	2.15
12	01-05-2009 02:42	Product1	1200	Diners	Stacy	New York	NY	United Sta	01-05-2009 02:23	01-05-2009 04:59	40.71417	-74.0064
13	01-05-2009 05:39	Product1	1200	Amex	Heidi	Eindhover	Noord-Bra	Netherlan	01-05-2009 04:55	01-05-2009 08:15	51.45	5.466667
14	01-02-2009 09:16	Product1	1200	Mastercar	Sean	Shavano P	TX	United Sta	01-02-2009 08:32	01-05-2009 09:05	29.42389	-98.4933
15	01-05-2009 10:08	Product1	1200	Visa	Georgia	Eagle	ID	United Sta	11-11-2008 15:53	01-05-2009 10:05	43.69556	-116.353
16	01-02-2009 14:18	Product1	1200	Visa	Richard	Riverside	NJ	United Sta	12-09-2008 12:07	01-05-2009 11:01	40.03222	-74.9578
17	01-04-2009 01:05	Product1	1200	Diners	Leanne	Julianstov	Meath	Ireland	01-04-2009 00:00	01-05-2009 13:36	53.67722	-6.31917

Step 1)

Create a new directory with name **MapReduceTutorial**

```
sudo mkdir MapReduceTutorial
```

```
hduser_@guru99-VirtualBox:~$ sudo mkdir MapReduceTutorial
```

Give permissions

```
sudo chmod -R 777 MapReduceTutorial
```

```
hduser_@guru99-VirtualBox:~$ sudo chmod -R 777 MapReduceTutorial
```

SalesMapper.java

```
package SalesCountry;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.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[7]), one);
	}
}
```

SalesCountryReducer.java

```
package SalesCountry;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.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;
		int frequencyForCountry = 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));
	}
}
```

SalesCountryDriver.java

```
package SalesCountry;

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

public class SalesCountryDriver {
    public static void main(String[] args) {
        JobClient my_client = new JobClient();
        // Create a configuration object for the job
        JobConf job_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();
        }
    }
}
```

heck the file permissions of all these files

```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ ls -al
total 144
drwxrwxrwx 2 root    root      4096 May  5 15:00 .
drwxr-xr-x 6 hduser_ hadoop_   4096 May  5 14:53 ..
-rw-rw-r-- 1 guru99  guru99    1367 May  5 02:28 SalesCountryDriver.java
-rw-rw-r-- 1 guru99  guru99     749 May  5 02:28 SalesCountryReducer.jav
-rw-rw-r-- 1 guru99  guru99  123637 May  5 02:28 SalesJan2009.csv
-rw-rw-r-- 1 guru99  guru99     659 May  5 02:28 SalesMapper.java
```

and if 'read' permissions are missing then grant the same-

```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ sudo chmod +r *.*
```

Step 2)

Export classpath

```
export CLASSPATH="$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.2.0.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.2.0.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-2.2.0.jar:~/MapReduceTutorial/SalesCountry/*:$HADOOP_HOME/lib/*"
```

```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ export CLASSPATH="$HADOOP_HOME/share/hadoop/mapreduce/hadoo
p-mapreduce-client-core-2.2.0.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.2.0
.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-2.2.0.jar:~/MapReduceTutorial/SalesCountry/*:$HADOOP_H
OME/lib/*"
hduser_@guru99-VirtualBox:~/MapReduceTutorial$
```

Step 3)

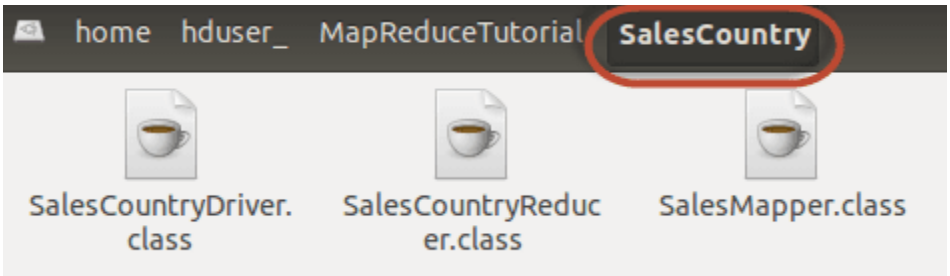
Compile Java files (these files are present in directory **Final-MapReduceHandsOn**). Its class files will be put in the package directory

```
javac -d . SalesMapper.java SalesCountryReducer.java SalesCountryDriver.java
```

```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ javac -d . SalesMapper.java SalesCountryReducer.java SalesC
ountryDriver.java
/home/guru99/Downloads/hadoop/share/hadoop/common/hadoop-common-2.2.0.jar(org/apache/hadoop/fs/Path.class)
: warning: Cannot find annotation method 'value()' in type 'LimitedPrivate': class file for org.apache.had
oop.classification.InterfaceAudience not found
1 warning
hduser_@guru99-VirtualBox:~/MapReduceTutorial$
```

This warning can be safely ignored.

This compilation will create a directory in a current directory named with package name specified in the java source file (i.e. **SalesCountry** in our case) and put all compiled class files in it.



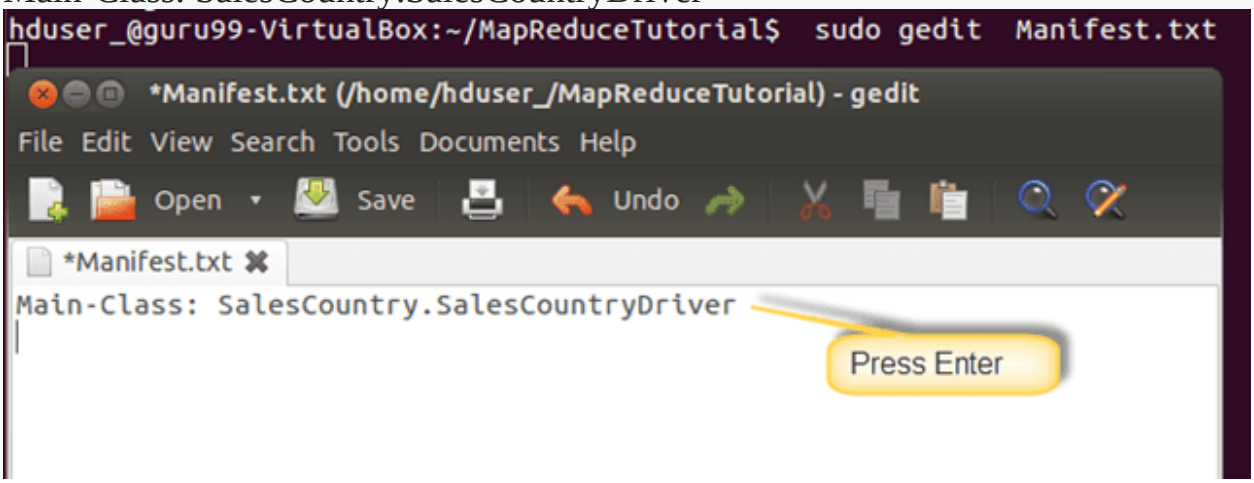
Step 4)

Create a new file **Manifest.txt**

```
sudo gedit Manifest.txt
```

add following lines to it,

```
Main-Class: SalesCountry.SalesCountryDriver
```

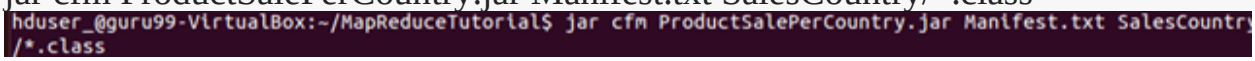


SalesCountry.SalesCountryDriver is the name of main class. Please note that you have to hit enter key at end of this line.

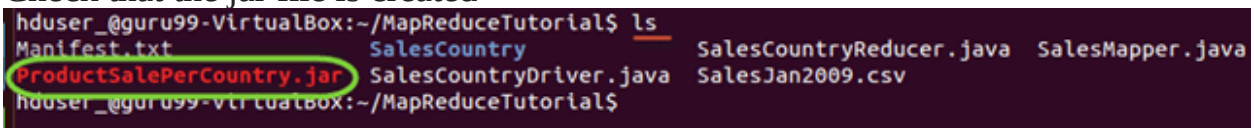
Step 5)

Create a Jar file

```
jar cfm ProductSalePerCountry.jar Manifest.txt SalesCountry/*.class
```



Check that the jar file is created



Step 6)

Start Hadoop

```
$HADOOP_HOME/sbin/start-dfs.sh
$HADOOP_HOME/sbin/start-yarn.sh
```

Step 7)

Copy the File **SalesJan2009.csv** into **~/inputMapReduce**

Now Use below command to copy **~/inputMapReduce** to HDFS.

```
$HADOOP_HOME/bin/hdfs dfs -copyFromLocal ~/inputMapReduce /
```

```
hduser@guru99: ~/MapReduceTutorial
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hdfs dfs -copyFromLocal ~/in
putMapReduce  /
14/05/06 23:33:48 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hduser@guru99:~/MapReduceTutorial$
```

We can safely ignore this warning.

Verify whether a file is actually copied or not.

$HADOOP_HOME/bin/hdfs dfs -ls /inputMapReduce

```
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hdfs dfs -ls /inputMapReduce
14/05/06 23:35:54 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r--   1 hduser supergroup     123637 2014-05-06 23:33 /inputMapReduce/Sal
esJan2009.csv
hduser@guru99:~/MapReduceTutorial$
```

Step 8)

Run MapReduce job

$HADOOP_HOME/bin/hadoop jar ProductSalePerCountry.jar /inputMapReduce /mapreduce_output_sales

```
hduser@guru99: ~/MapReduceTutorial
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hadoop jar ProductSalePerCou
ntry.jar /inputMapReduce /mapreduce_output_sales
```

This will create an output directory named mapreduce_output_sales on HDFS. Contents of this directory will be a file containing product sales per country.

Step 9)

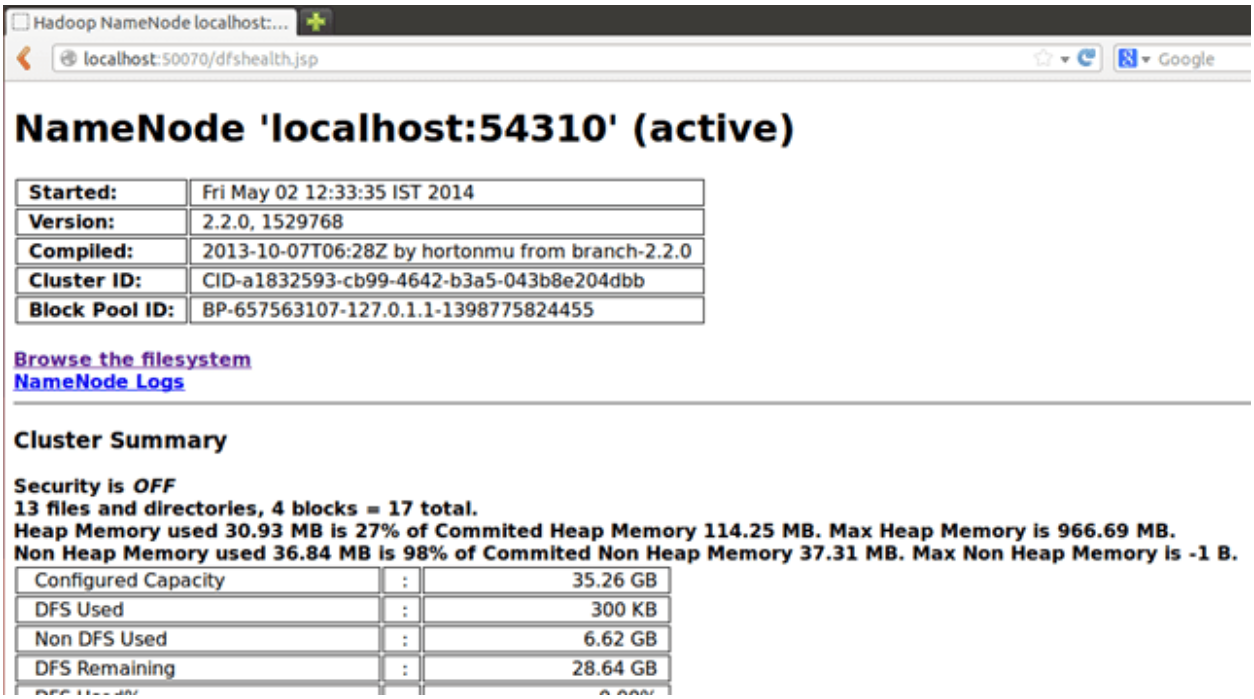
The result can be seen through command interface as,

$HADOOP_HOME/bin/hdfs dfs -cat /mapreduce_output_sales/part-00000

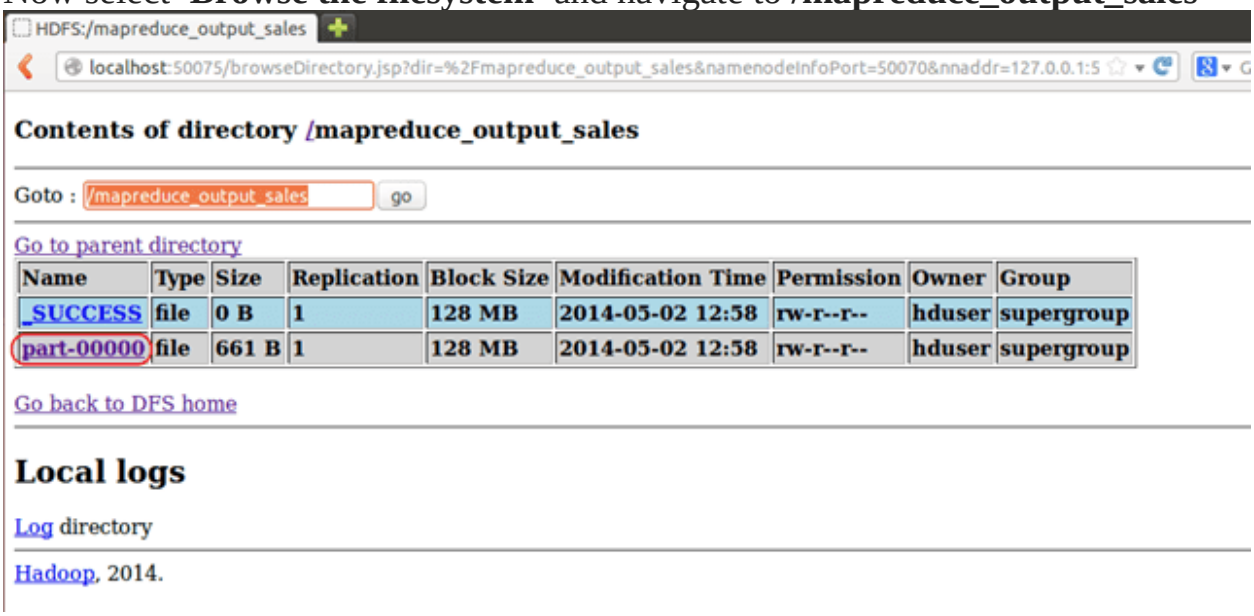
```
hduser@guru99: ~/MapReduceTutorial
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hdfs dfs -cat /mapreduce_out
put_sales/part-00000
14/05/02 13:03:46 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Argentina       1
Australia       38
Austria 7
Bahrain 1
Belgium 8
Bermuda 1
Brazil  5
Bulgaria        1
CO      1
Canada  76
Cayman Isls     1
```

Results can also be seen via a web interface as-

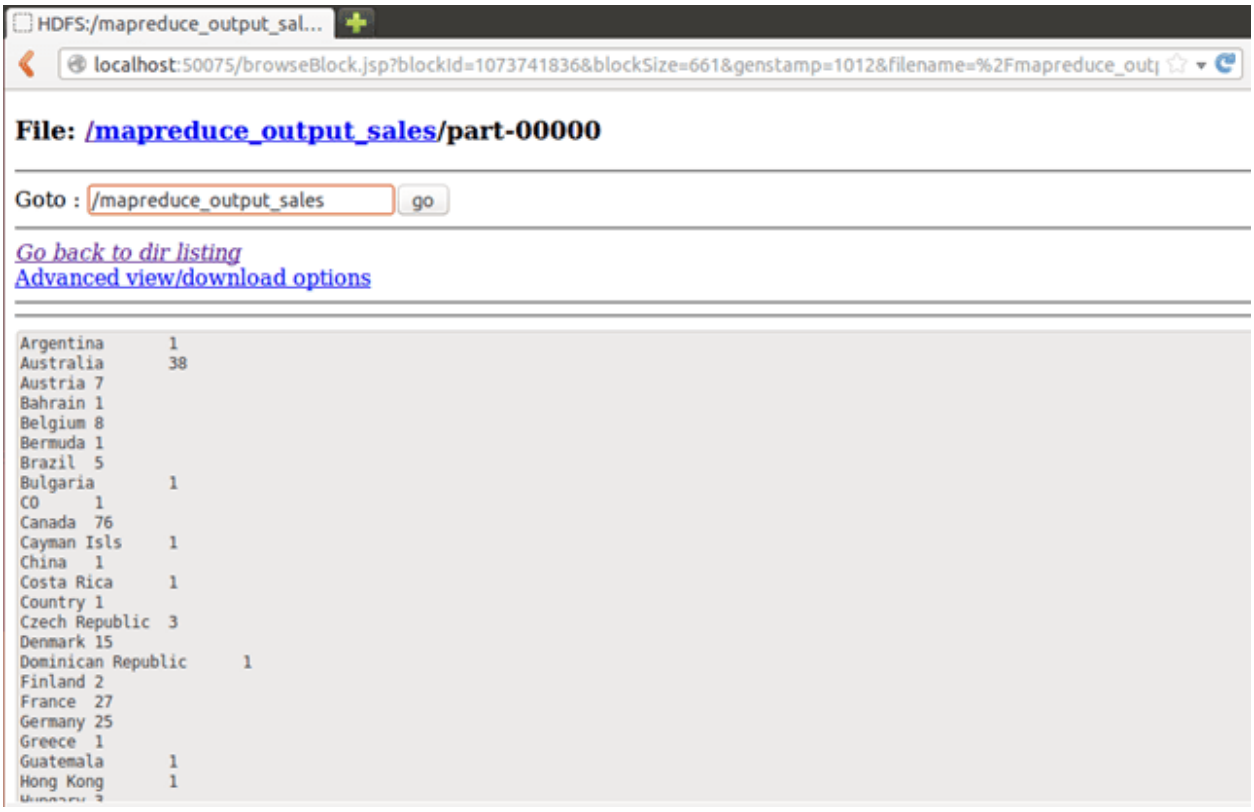
Open r in a web browser.



Now select **'Browse the filesystem'** and navigate to **/mapreduce_output_sales**



Open **part-r-00000**



Explanation of SalesMapper Class

In this section, we will understand the implementation of **SalesMapper** class.

1. We begin by specifying a name of package for our class. **SalesCountry** is a name of our package. Please note that output of compilation, **SalesMapper.class** will go into a directory named by this package name: **SalesCountry**.

Followed by this, we import library packages.

Below snapshot shows an implementation of **SalesMapper** class-

```
*SalesCountryDriver.java x   *SalesMapper.java x
package SalesCountry;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.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[7]), one);
        }
}
```

Package Name

Import Library Packages

Every Map class must extend 'MapReduceBase' class and implement 'Mapper' interface

Every Mapper class must provide definition of 'map' function

Our mapper function maps every input record to '1'

Sample Code Explanation:

1. SalesMapper Class Definition-

public class SalesMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

Every mapper class must be extended from **MapReduceBase** class and it must implement **Mapper** interface.

2. Defining 'map' function-

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

The main part of Mapper class is a **'map()'** method which accepts four arguments. At every call to **'map()'** method, a **key-value** pair (**'key'** and **'value'** in this code) is passed.

'map()' method begins by splitting input text which is received as an argument. It uses the tokenizer to split these lines into words.

```
String valueString = value.toString();
String[] SingleCountryData = valueString.split(",");
```

Here, **','** is used as a delimiter.

After this, a pair is formed using a record at 7th index of array **'SingleCountryData'** and a value **'1'**.

```
output.collect(new Text(SingleCountryData[7]), one);
```

We are choosing record at 7th index because we need **Country** data and it is located at 7th index in array **'SingleCountryData'**.

Please note that our input data is in the below format (where **Country** is at 7th index, with 0 as a starting index)-

Transaction_date,Product,Price,Payment_Type,Name,City,State,**Country**,Account_Created,Last_Login,Latitude,Longitude

An output of mapper is again a **key-value** pair which is outputted using **'collect()'** method of **'OutputCollector'**.

Explanation of SalesCountryReducer Class

In this section, we will understand the implementation of **SalesCountryReducer** class.

1. We begin by specifying a name of the package for our class. **SalesCountry** is a name of out package. Please note that output of compilation, **SalesCountryReducer.class** will go into a directory named by this package name: **SalesCountry**.

Followed by this, we import library packages.

Below snapshot shows an implementation of **SalesCountryReducer** class-

```
*SalesCountryDriver.java ×   *SalesMapper.java ×   *SalesCountryReducer.java ×

package SalesCountry;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class SalesCountryReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, In

        public void reduce(Text t_key, Iterator<IntWritable> values,
                           OutputCollector<Text,IntWritable> output,
                           Reporter reporter) throws IOException {

                Text key = t_key;
                int frequencyForCountry = 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));
        }
}
```

Package Name

Import Library Packages

Every 'Reducer' class must extend 'MapReduceBase' class and implement 'Reducer' interface

Every 'Reducer' class must provide definition of 'reduce' function

Code Explanation:

1. SalesCountryReducer Class Definition-

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

Here, the first two data types, '**Text**' and '**IntWritable**' are data type of input key-value to the reducer.

Output of mapper is in the form of <CountryName1, 1>, <CountryName2, 1>. This output of mapper becomes input to the reducer. So, to align with its data type, **Text** and **IntWritable** are used as data type here.

The last two data types, 'Text' and 'IntWritable' are data type of output generated by reducer in the form of key-value pair.

Every reducer class must be extended from **MapReduceBase** class and it must implement **Reducer** interface.

2. Defining 'reduce' function-

```
public void reduce( Text t_key,
        Iterator<IntWritable> values,
        OutputCollector<Text,IntWritable> output,
        Reporter reporter) throws IOException {
```

An input to the **reduce()** method is a key with a list of multiple values.
For example, in our case, it will be-
<United Arab Emirates, 1>, <United Arab Emirates, 1>, <United Arab Emirates, 1>,<United Arab Emirates, 1>, <United Arab Emirates, 1>, <United Arab Emirates, 1>.
This is given to reducer as **<United Arab Emirates, {1,1,1,1,1,1}>**
So, to accept arguments of this form, first two data types are used, viz., **Text** and **Iterator<IntWritable>**. **Text** is a data type of key and **Iterator<IntWritable>** is a data type for list of values for that key.
The next argument is of type **OutputCollector<Text,IntWritable>** which collects the output of reducer phase.
reduce() method begins by copying key value and initializing frequency count to 0.

```
Text key = t_key;
int frequencyForCountry = 0;
```

Then, using '**while**' loop, we iterate through the list of values associated with the key and calculate the final frequency by summing up all the values.

```
while (values.hasNext()) {
        // replace type of value with the actual type of our value
        IntWritable value = (IntWritable) values.next();
        frequencyForCountry += value.get();

    }
```

Now, we push the result to the output collector in the form of **key** and obtained **frequency count**.
Below code does this-

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

Explanation of SalesCountryDriver Class
In this section, we will understand the implementation of **SalesCountryDriver** class
1. We begin by specifying a name of package for our class. **SalesCountry** is a name of out package. Please note that output of compilation, **SalesCountryDriver.class** will go into directory named by this package name: **SalesCountry**.
Here is a line specifying package name followed by code to import library packages.

```java
package SalesCountry;

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

Package Name

Import Library Packages

2. Define a driver class which will create a new client job, configuration object and advertise Mapper and Reducer classes.

The driver class is responsible for setting our MapReduce job to run in Hadoop. In this class, we specify **job name, data type of input/output and names of mapper and reducer classes**.

```java
package SalesCountry;

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

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

Start of definition of SalesCountryDriver class

Entry point to the application

3. In below code snippet, we set input and output directories which are used to consume input dataset and produce output, respectively.

arg[0] and **arg[1]** are the command-line arguments passed with a command given in MapReduce hands-on, i.e.,
$HADOOP_HOME/bin/hadoop jar ProductSalePerCountry.jar /inputMapReduce /mapreduce_output_sales

```
                // Set input and output directories using command line arguments,
/inputMapReduce ->//arg[0] = name of input directory on HDFS, and
/mapreduce_output->//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);  <-- This code initiates Map-Reduce job
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
```

4. Trigger our job
Below code start execution of MapReduce job-

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