Word Count Program Using Map and Reduce

Tasks

Name:	Class:	Roll Number:
Ex. No: 8	Date:	

Aim:

To Count the number of words using JAVA for demonstrating the use of Map and Reduce tasks.

Procedure:

1. Analyse the input file content
2. Develop the code
    a. Writing a map function
    b. Writing a reduce function
    c. Writing the Driver class
3. Compiling the source
4. Building the JAR file
5. Starting the DFS
6. Creating Input path in HDFS and moving the data into Input path
7. Executing the program

Program: WordCount.java

```
import java.io.IOException;
import
java.util.StringTokenizer;
import
org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import
org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import
org.apache.hadoop.mapreduce.Mapper;
import
org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{
```

```
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
  StringTokenizer itr = new
  StringTokenizer(value.toString()); while
  (itr.hasMoreTokens()) {
    word.set(itr.nextToke
    n());
    context.write(word,
    one);
  }
}
```

```
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException,
      InterruptedException { int sum = 0;
      for (IntWritable val :
        values) { sum +=
        val.get();
      }
      result.set(sum);
      context.write(key,
      result);
    }
  }

  public static void main(String[] args) throws
    Exception { Configuration conf = new
    Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

save the program as **WordCount.java**

Step 1: Compile the java program

For compilation we need this hadoop-core-1.2.1.jar file to compile the mapreduce program.
https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-core/1.2.1
Assuming both jar and java files in same directory run the following command to compile

```
root@a4cseh160:/#javac -classpath hadoop-core-1.2.1.jar WordCount.java
```

Step 2: Create a jar file

Syntax:
```
jar cf jarfilename.jar MainClassName*.class
```

Output:
root@a4cseh160:/#jar cf wc.jar WordCount*.class

Step 3: Make directory in hadoop file system

Syntax:
hdfs dfs -mkdir directoryname

Output:
root@a4cseh160:/# hdfs dfs -mkdir /user

]Step 4: Copy the input file into hdfs

Syntax:
hdfs dfs -put sourcefile destpath

Output:
root@a4cseh160:/#hdfs dfs -put /input.txt /user

Step 5: **To a run a program**

Syntax:
hadoop jar jarfilename main_class_name inputfile outputpath

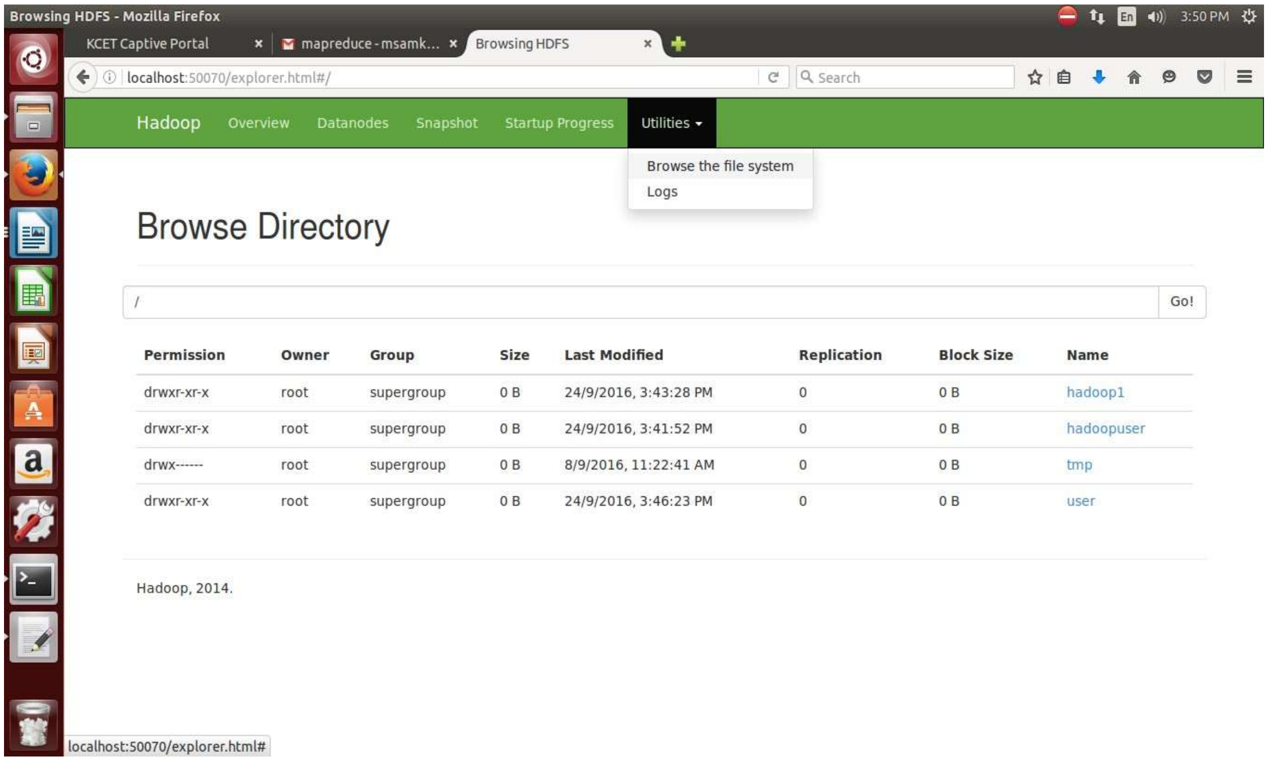
Output:
root@a4cseh160:/#hadoop jar wc.jar WordCount /user/input.txt /user/out

Input File: (input.txt)

Cloud and Grid Lab. Cloud and Grid Lab. Cloud Lab.

Output:
```
18
3       Cloud
3       Lab.
2       Grid
2       and
```

Step 6: Check the output in the Web UI at **http://localhost:50070.**
In the Utilities tab select browse file system and select the correct user.



Browse Directory

/

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	root	supergroup	0 B	24/9/2016, 3:43:28 PM	0	0 B	hadoop1
drwxr-xr-x	root	supergroup	0 B	24/9/2016, 3:41:52 PM	0	0 B	hadoopuser
drwx------	root	supergroup	0 B	8/9/2016, 11:22:41 AM	0	0 B	tmp
drwxr-xr-x	root	supergroup	0 B	24/9/2016, 3:46:23 PM	0	0 B	user

Hadoop, 2014.

The output is available inside the output folder named **user**.

Step 7: To Delete a output folder

Syntax:

```
hdfs dfs -rm -R outputpath
```

Output:

```
root@a4cseh160:/#hdfs dfs -rm -R /user/out.txt
```