Assignment No:11

Introduction

This document serves as a tutorial to setup and run a simple application in Hadoop MapReduce framework. A *job* in Hadoop MapReduce usually splits input data-set into independent chucks which are processed by *map tasks*. Later, the output form maps are sorted and then input to the *reduce tasks*. Usually all the outputs are stored in file systems.

In order to run an application a *job client* will submits the job which can be a JAR file or an executable to a single master in Hadoop called ResourceManager. This master will then distribute tasks, configure nodes, monitor tasks and schedule tasks. Moreover, all the files for correspondence in the framework need to be moved to Hadoop File System (HDFS); the user has to feed input files into the HDFS directory and the output files will also be saved in HDFS directories.

This tutorial will walk-through of these main steps by running an application that will count the number of words in file(s). The application will run it in a Single Node setup.

1   Setup

1.1   Prerequisites:
1. Linux System/ Virtual Machine
2. Java Must be installed in the system.
3. ssh, sshdand rsyncmust be installed. Link

1.2   Install Hadoop
One can download the stable release of Hadoop from one of the Apache Download Mirrors.

1.3   Setting Path Names
After installation please check the variables JAVA_HOMEand HADOOP_CLASSPATH. Often the values returned will be empty. To check these variables type the following command in terminal *(to open a terminal -> {[CtrlAlt + t] or [Ctrl + ⌥ Opt + t]}).*

> echo $JAVA_HOME
> echo $HADOOP_CLASSPAT

```
hadoop@hadoop:~$ echo $JAVA_HOME
hadoop@hadoop:~$ echo $HADOOP_CLASSPATH
hadoop@hadoop:~$
```

If the variables are empty then the commands will return a blank line similar to one above.

In order to pass the correct path names for JAVA_HOMEplease find the appropriate version of javacompiler. For example on typing the following command one gets the following result:

```
hadoop@hadoop:~$ javac -version
javac 1.7.0_95
hadoop@hadoop:~$
```

As the version of Java Compiler is *1.7.0_95*. Thus, corresponding version to the environment variable JAVA_HOMEcan be updates as below.

> export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64

```
hadoop@hadoop:~$ javac -version
javac 1.7.0_95
hadoop@hadoop:~$ export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64
hadoop@hadoop:~$
```

After updating the above variable one can later change the HADOOP_CLASSPATHvariable which is asfollows:

> export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar

```
hadoop@hadoop:~$ javac -version
javac 1.7.0_95
hadoop@hadoop:~$ export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64
hadoop@hadoop:~$ export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
hadoop@hadoop:~$
```

Later one can check if the variables indeed contain the values:

```
hadoop@hadoop:~$ echo JAVA_HOME
JAVA_HOME
hadoop@hadoop:~$ echo HADOOP_CLASSPATH
HADOOP_CLASSPATH
hadoop@hadoop:~$
```
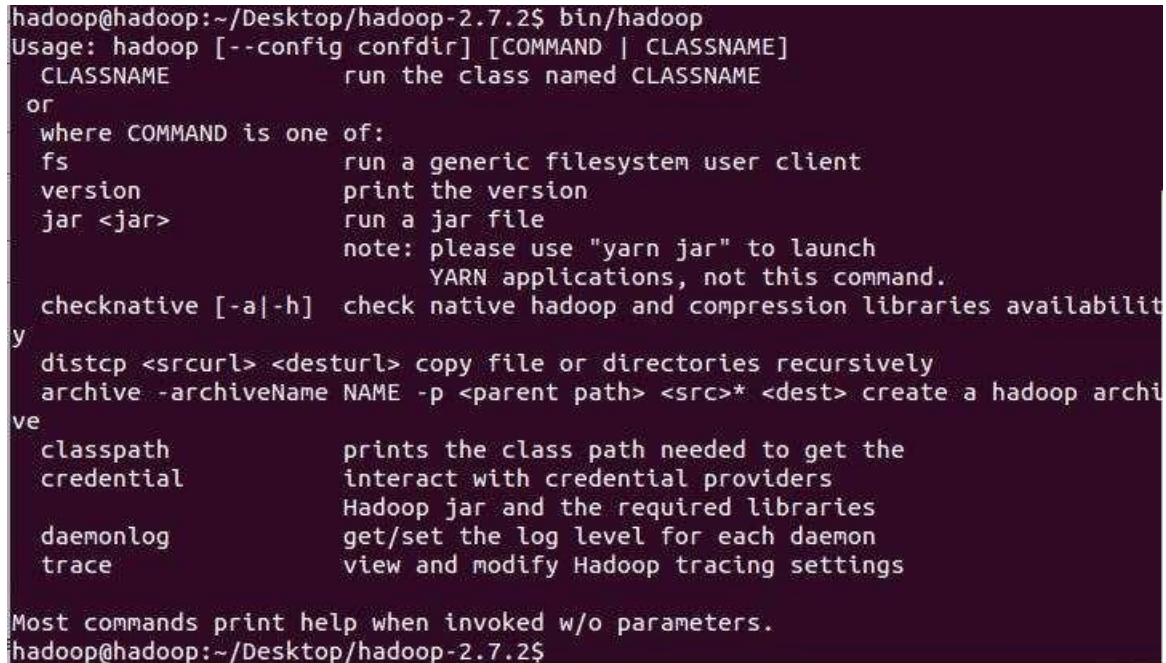
1.4 Checking bin/hadoop

Now for the next step navigate to the folder that contains the source of Hadoop framework, simply type the following:

> cd ~/Desktop/hadoop-2.7.2

Type the following command, after one reaches the folder:

> bin/hadoop

```
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hadoop
Usage: hadoop [--config confdir] [COMMAND | CLASSNAME]
  CLASSNAME            run the class named CLASSNAME
 or
  where COMMAND is one of:
  fs                   run a generic filesystem user client
  version              print the version
  jar <jar>            run a jar file
                       note: please use "yarn jar" to launch
                             YARN applications, not this command.
  checknative [-a|-h]  check native hadoop and compression libraries availabilit
y
  distcp <srcurl> <desturl> copy file or directories recursively
  archive -archiveName NAME -p <parent path> <src>* <dest> create a hadoop archi
ve
  classpath            prints the class path needed to get the
  credential           interact with credential providers
                       Hadoop jar and the required libraries
  daemonlog            get/set the log level for each daemon
  trace                view and modify Hadoop tracing settings

Most commands print help when invoked w/o parameters.
hadoop@hadoop:~/Desktop/hadoop-2.7.2$
```

The above screenshot shows the documentation of the Hadoop script.

1.5   Configurations

Before continuing, some simple configurations need to be performed. Edit the files core-site.xml and hdfs-site.xml, they can be found at ~/Desktop/hadoop-2.7.2/etc/hadoop/

Add the details as mentioned below to the respective files, in order to do that type the following command, this command will open *gedit*which is a word editor

> gedit etc/hadoop/core-site.xml

Add the following details, refer to the screenshot below for further clarifications:

core-site.xml

```
<configuration
      >
  <property>
      <name>fs.defaultFS</name>
      <value>hdfs://localhost:9000</value>
  </property>
</configuration
      >
```

3

```
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ gedit etc/hadoop/core-site.xml

core-site.xml (~/Desktop/hadoop-2.7.2/etc/hadoop) - gedit

    Open    ▾    Save           Undo

core-site.xml  ✕

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

Save the file ($Ctr + s$) and then close it. Repeat the procedure for the hdfs-site.xml file as well. The configuration details are mentioned below for the same.

hdfs-site.xml

```
<configuration
      >
  <property>
      <name>dfs.replication</name>
      <value>1</value>
  </property>
</configuratio>
```

4

Check ssh to localhost

In order to start the daemons one needs to check the sshto localhost:

> ssh localhost

If prompted by the terminal then press yor type Yes. [**Error**]

```
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ ssh localhost
hadoop@localhost's password:
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.19.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

303 packages can be updated.
141 updates are security updates.

Last login: Fri Mar 18 23:08:53 2016 from localhost
hadoop@hadoop:~$
```

If ssh to localhost is not successful after typing yor Yes, then type these commands:

> ssh-keygen -t dsa -P " -f ~/.ssh/id_dsa
> cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
> chmod 0600 ~/.ssh/authorized_keys

1.6   Format the filesystem

The Hadoop File System (HDFS) needs to be formatted before running application for the first time. Type the following command:

> bin/hdfs namenode –format

Press Yor Yeswhenever prompted.

1.7   Run the daemons

The hadoop daemons could be started by typing the command, this will start three nodes viz. *namenode, datanode and secondarynamenode*.

> sbin/start-dfs.sh

If prompted, enter the password. The screenshot below shows the prompts to enter password.
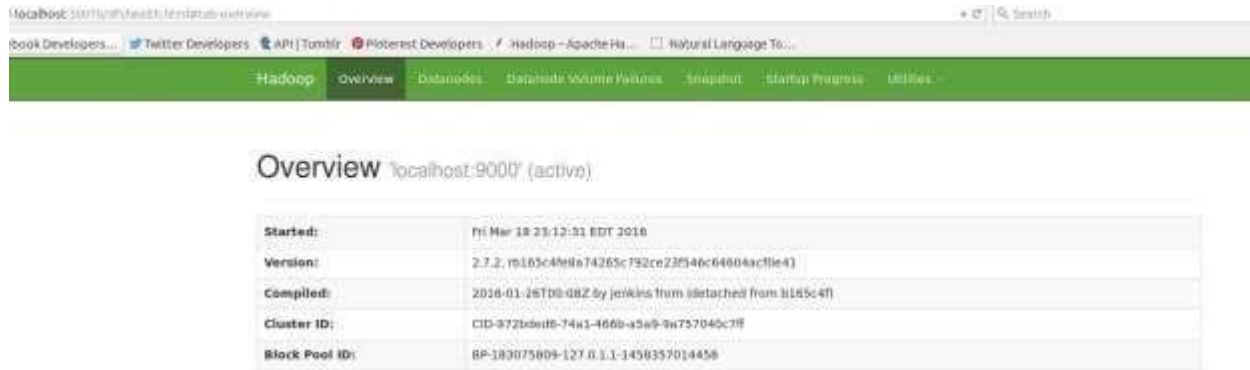
```
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ sbin/start-dfs.sh
Starting namenodes on [localhost]
hadoop@localhost's password:
localhost: starting namenode, logging to /home/hadoop/Desktop/hadoop-2.7.2/logs/
hadoop-hadoop-namenode-hadoop.out
hadoop@localhost's password:
localhost: starting datanode, logging to /home/hadoop/Desktop/hadoop-2.7.2/logs/
hadoop-hadoop-datanode-hadoop.out
Starting secondary namenodes [0.0.0.0]
hadoop@0.0.0.0's password:
0.0.0.0: starting secondarynamenode, logging to /home/hadoop/Desktop/hadoop-2.7.
2/logs/hadoop-hadoop-secondarynamenode-hadoop.out
hadoop@hadoop:~/Desktop/hadoop-2.7.2$
```

Check the web interface for

NameNode. By default it is available

at:

Overview localhost:9000' (active)

Started:	Fri Mar 18 23:12:31 EDT 2016
Version:	2.7.2, rb165c4fe8a74265c792ce23f546c64804acf8e41
Compiled:	2016-01-26T00:08Z by jenkins from (detached from b165c4f)
Cluster ID:	CID-372bdedf6-74a1-466b-a5a9-9a757040c7ff
Block Pool ID:	BP-183075809-127.0.1.1-1458357014458

> sbin/stop-dfs.sh

2 Execution Steps:

2.1 Compiling WordCount.java

In order to continue forward one needs to create a local repository for the application. A repository where the .java files and input files can be stored. One can create a local directory outside directorycontaining hadoop source. Type the following:

> mkdir ../tutorial01

Later the following snippet of code can be pasted to a file called WordCount.java, this file should reside in the newly created directory. To do that one needs to open a word editor (ex. Gedit) opening a new file called WordCount.java, later copy the snippet provided below and then save and close. Typethe following command:

> gedit ../tutorial01/WordCount.java

Copy the following code into the blank space.

```
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
```

6

```java
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.nextToken());
        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);
    }
  }
```
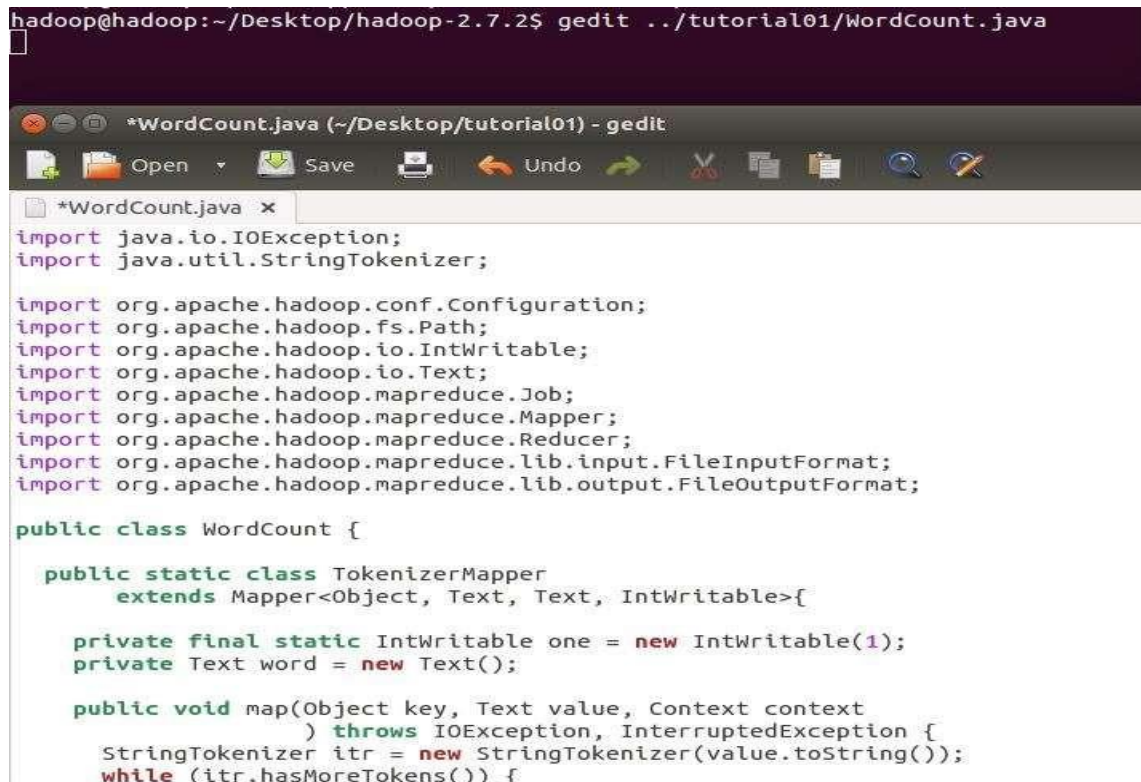
```java
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    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);
 }
}
```


The next step is to compile the code and create JAR file. But before that please copy the JAVA file tothe current directory by typing the following: **[Error]**

```
>  cp ../tutorial01/WordCount.java .
>  bin/hadoop com.sun.tools.javac.Main WordCount.java
>  jar cf wc.jar WordCount*.class
```

```
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hadoop com.sun.tools.javac.Main ../tutorial01/WordCount.java
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ jar cf ../tutorial01/wc.jar ../tutorial01/WordCount*.class
hadoop@hadoop:~/Desktop/hadoop-2.7.2$
```

This operation will create several files. To check, perform a listing, sorted according to files createdlately. Type the following command:

> ls -li –-sort=time

The above commands will display the details similar to the ones in the screenshot below:

```
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ ls -li --sort=time
total 96
328588 -rw-rw-r-- 1 hadoop hadoop  5616 Mar 18 22:25 wc.jar
328587 -rw-rw-r-- 1 hadoop hadoop  2479 Mar 18 22:25 WordCount.class
328566 -rw-rw-r-- 1 hadoop hadoop  1739 Mar 18 22:25 WordCount$IntSumReducer.class
328372 -rw-rw-r-- 1 hadoop hadoop  4524 Mar 18 22:25 WordCount$TokenizerMapper.class
328366 -rw-rw-r-- 1 hadoop hadoop  1027 Mar 18 22:25 WordCount$TokenizerMapper$CountersEnum.class
328373 -rw-rw-r-- 1 hadoop hadoop  4702 Mar 18 22:25 WordCount.java
328365 drwxrwxr-x 2 hadoop hadoop  4096 Mar 18 21:45 output
328353 drwxrwxr-x 2 hadoop hadoop  4096 Mar 18 21:45 input
318846 drwxr-xr-x 2 hadoop hadoop  4096 Jan 25 19:20 bin
```

2.2 Creating Directories in HDFS

Now after the above steps, one needs to create directories for the current application in the Hadoop File Sytems. As the directories are not present, one must create the directories one by one as follows:

[Error]

> bin/hdfs dfs –mkdir /user
> bin/hdfs dfs –mkdir /user/hadoop
> bin/hdfs dfs –mkdir /user/hadoop/wordcount

One can always check the contents within the HDFS directories. For example if one has to determine the directories within /user/hadoop directory, simple type the following command:

> bin/hdfs dfs –ls /user/hadoop

```
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -mkdir /user
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -mkdir /user/hadoop
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -mkdir /user/hadoop/wordcount
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -ls /user
Found 1 items
drwxr-xr-x   - hadoop supergroup          0 2016-03-20 18:22 /user/hadoop
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -ls /user/hadoop
Found 1 items
drwxr-xr-x   - hadoop supergroup          0 2016-03-20 18:22 /user/hadoop/wordco
unt
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ 
```

2.3 Creating Input files

After creating directories in HDFS viz. /user/hadoop/wordcount one can create another directory within wordcount as follows. This directory will contain file(s) which will then be used by the application to perform word counting.

> bin/hdfs dfs –mkdir /user/hadoop/wordcount/input

Now in local file system one will create input files as follows, these files are filled up with texts viz. **"Hello World Bye World"** and **"Hello Hadoop GoodBye Hadoop"** for file01 and file02 respectively.

> echo "Hello World Bye World" > ../tutorial01/file01
> echo "Hello Hadoop Goodbye Hadoop" ../tutorial01/file02

Move these files to the HDFS directory inputusing the following commands:

> bin/hadoop fs –copyFromLocal ../tutorial01/file0*
/user/hadoop/wordcount/input

One could also verify if the files that were copied correctly. Simply type the command below,
it willdisplay the content of the files.
> bin/hdfs dfs –cat /user/hadoop/wordcount/input/file01
> bin/hdfs dfs -cat /user/hadoop/wordcount/input/file02

```
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -mkdir /user/hadoop/wordcount/input
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ echo "Hello World Bye World" > ../tutorial01/file01
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ echo "Hello Hadoop Goodbye Hadoop" > ../tutorial01/file02
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hadoop fs -copyFromLocal ../tutorial01/file0* /user/hadoop/wordcount/input
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -ls /user/hadoop/wordcount/input
Found 2 items
-rw-r--r--   1 hadoop supergroup         22 2016-03-21 15:44 /user/hadoop/wordcount/input/file01
-rw-r--r--   1 hadoop supergroup         28 2016-03-21 15:44 /user/hadoop/wordcount/input/file02
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -cat /user/hadoop/wordcount/input/file01
Hello World Bye World
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -cat /user/hadoop/wordcount/input/file02
Hello Hadoop Goodbye Hadoop
hadoop@hadoop:~/Desktop/hadoop-2.7.2$
```

The above screenshot shows that file01 and file02 have been moved successfully to input directoryresiding in HDFS.
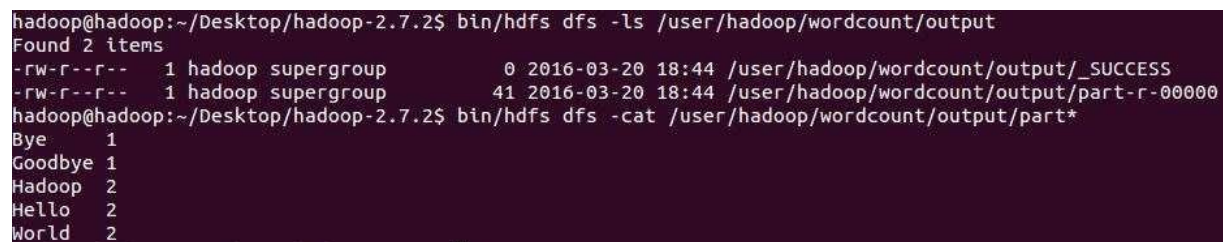
2.4 Execute the JAR

Run the following command to execute the jar: [Error]

> bin/hadoop jar wc.jar WordCount /user/hadoop/wordcount/input
/user/hadoop/wordcount/output

This will create a directory user/hadoop/wordcount/output and two files within it viz. _SUCCESSand part-r-00000

The output of this application i.e. counting of words will be stored in part-r-00000 file.
One can view the contents of the file just like we did above.
> bin/hdfs –cat /user/hadoop/wordcount/output/part*

```
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -ls /user/hadoop/wordcount/output
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2016-03-20 18:44 /user/hadoop/wordcount/output/_SUCCESS
-rw-r--r--   1 hadoop supergroup         41 2016-03-20 18:44 /user/hadoop/wordcount/output/part-r-00000
hadoop@hadoop:~/Desktop/hadoop-2.7.2$ bin/hdfs dfs -cat /user/hadoop/wordcount/output/part*
Bye     1
Goodbye 1
Hadoop  2
Hello   2
World   2
```

Conclusion :-

Hence, In this practical assignment we learn Hadoop Map-Reduce framework on Local-standalone set-up. We successfully implemented JAVA code for a simple Word Count application that counts the number of occurrences of each word in a given input.