Tutorial 2

Wordcount in Hadoop

Step 0: Claim a virtual machine from infomdssLabB

Login to the azure portal (https://portal.azure.com/),

Go to **Dashboard** → **infomdssLabB** → **infomdssLabB** and claim a virtual machine.

Login to your lab B vm (the default password is the same as in tutorial 1) and **change your password**.

```
$ passwd # choose your new password wisely
```

These actions are explained in tutorial 1.

Step 1: Fetch map/reduce Scripts

Within your Azure VM, go to the hadoop folder, create a new directory named `wordcount` and download `wordcount_mapper.py` and `wordcount_reducer.py` from the Infomdss2018 github page.

```
$ cd /usr/local/hadoop
$ mkdir wordcount
$ cd wordcount
$ wget https://raw.githubusercontent.com/Infomdss2018/infomdss/master/wordcount_mapper.py
$ wget https://raw.githubusercontent.com/Infomdss2018/infomdss/master/wordcount_reducer.py
```

Step 2: Check Indentation

Some files have the tendency of changing tabs into spaces and visa versa, with python this is a problem, since the level of indentation is part of the syntax. In order to make sure the scripts mapper and reducer are runned correctly, you have to check if the files are indented correctly.

```
$ more wordcount_reducer.py # check if all code is aligned properly
$ more wordcount_mapper.py  # check if all code is aligned properly
```

Note that improper indentation could lead to unexpected behavior or an error code stating: `IndentationError: expected an indented block`.

Step 3: Make Executable

Make the word count mapper and reducer executable.

```
$ chmod +x wordcount_mapper.py
$ chmod +x wordcount_reducer.py
```

See tutorial 1 for more details about the chmod command.

Step 4: Create Test Files

Create some simple text files from command line to test the mapper and reducer in Hadoop.

```
$ echo "A long time ago in a galaxy far far away" > testfile1
$ echo "Another episode of Star Wars" > testfile2
```

Step 5: Create HDFS Directory

Create a directory on the HDFS file system. Go to your local Hadoop folder and create a directory named `input_wordcount`. Hadoop will use this directory to fetch the input files in dfs format.

```
$ cd /usr/local/hadoop
$ bin/hdfs dfs -mkdir input_wordcount
```

Step 6: Copy Files to the HDFS

Copy the files from local filesystem to the HDFS filesystem. By using the `put`-flag, you can ask Hadoop to convert the two textfiles (`testfile1` and `testfile2`) into a dfs-format and save them in the `input_wordcount` folder.

```
$ bin/hdfs dfs -put ./wordcount/testfile1 input_wordcount/
$ bin/hdfs dfs -put ./wordcount/testfile2 input_wordcount/
```

Step 7: Sanity Check

Check that your files are indeed on HDFS.

```
$ bin/hdfs dfs -ls input_wordcount/
```

The output should look something like:

```
Found 2 items
-rw-r--r-- 1 labuser 41 2018-09-11 21:09 input_wordcount/testfile1
-rw-r--r-- 1 labuser 29 2018-09-11 21:09 input_wordcount/testfile2
```

Step 8: Run Hadoop

Run Hadoop with the wordcount files and directories we've made earlier in this tutorial.

```
$ bin/hadoop jar ./share/hadoop/tools/lib/hadoop-streaming-3.1.1.jar \
    -input input_wordcount \
    -output output_wordcount \
    -mapper ./wordcount/wordcount_mapper.py \
    -reducer ./wordcount/wordcount_reducer.py
```

Note: after you executed this command, a lot of output will appear on the screen which contains the systems feedback on its tasks. This feedback can be very useful to trace errors.

Note2: The backslashes (i.e. "\") at the end of each line in the command above indicate that the command continues on the next line. It is simply one long command. However, you probably need to remove them and any remaining trailing spaces to execute the command as intended.

Note3: If you try to run the hadoop command above and it fails the first time, then you will need to define a new "-output" location, since hadoop will simply fail if you specify an existing directory name here.

Step 9: View Output

View the output directory and look at the output files.

```
$ bin/hdfs dfs -ls output_wordcount/
$ bin/hdfs dfs -cat output_wordcount/part-00000
```

Step 10: Merge Output File

Create one output file for this run as a local file.

```
$ bin/hdfs dfs -getmerge output_wordcount/* wordcount/wordcount_output.txt
```

Question 1 for Honor students: Disable Reduce()

Try running the WordCount example with reduce() disabled.

```
$ bin/hadoop jar ./share/hadoop/tools/lib/hadoop-streaming-3.1.1.jar \
    -input input_wordcount \
    -output output_wordcount_no_reducer \
    -mapper ./wordcount/wordcount_mapper.py \
    -reducer ./wordcount/wordcount_reducer.py \
    -numReduceTasks 0
```

As with all Honors assignments, please submit your console output for the command above as a PDF file at http://bit.ly/infomdss-honors, preceded by a layman's explanation of what happens in all the warning and information messages triggered by the command, and a brief interpretation of the final outcome.

Question 2 for Honor students: Wordcount in R

Hadoop Streaming is really flexible as it supports writing MapReduce jobs in any language. In this tutorial we chose Python for our mapper and reducer scripts. To experience the power of Hadoop Streaming, your final task in this tutorial is to write the scripts for the MapReduce Wordcount example in the R programming language. Put the mapper and reducer scripts into one file and submit as PDF at http://bit.ly/infomdss-honors. Good luck!

Stop your virtual machine at the end of the workshop