CPSC 4330 Big Data Analytics

In-Class Exercise 2 – Running a MapReduce Job

Exercise 2

Download the file stubs.zip from Canvas and unzip it. The folder "stubs" contains the following three files.

WordCount.java: A simple MapReduce driver class.

WordMapper.java: A mapper class for the job.

SumReducer.java: A reducer class for the job.

In this exercise you will compile Java files, create a JAR, and run MapReduce jobs.

In addition to manipulating files in HDFS, the wrapper program *hadoop* is used to launch MapReduce jobs. The code for a job is contained in a compiled JAR file. Hadoop loads the JAR into HDFS and distributes it to the slave nodes, where the individual tasks of the MapReduce job are executed.

One simple example of a MapReduce job is to count the number of occurrences of each word in a file or a set of files. In this exercise you will compile and submit a MapReduce job to count the number of occurrences of every word in the works of Shakespeare.

Compiling and Submitting a MapReduce Job

1. Start the docker container. See exercise 1 on how to start the exited docker container (Note that you want to start the exited container which you started in exercise 1 – it has the files that you uploaded on HDFS in exercise 1, not a brand new and different container which does not have HDFS files from your exercise 1).
2. In the docker terminal window, go to the directory where you store data (e.g. hadoop-data), and create a subdirectory for this exercise.

```
#cd hadoop-data

#mkdir wordcount
```

3. Open a terminal on your computer, copy the stubs folder and its contents to the "wordcount" folder in docker. For example, on Mac (see exercise 1 on how to copy from Windows)

```
docker cp  /Users/linli/Documents/data/stubs/   bda:/hadoop-data/wordcount/
```

4. In the docker terminal window, change to the exercise source directory, and list the contents:

```
# cd /hadoop-data/wordcount

# ls
```

List the files in the stubs package directory:

```
# ls stubs
```

The package contains the following Java files:

WordCount.java: A simple MapReduce driver class.
WordMapper.java: A mapper class for the job.
SumReducer.java: A reducer class for the job.

Examine these files if you wish, but do not change them. Remain in this directory while you execute the following commands.

5. Before compiling, examine the classpath *Hadoop* is configured to use:

```
# hadoop classpath
```

This shows the locations where the Hadoop core API classes are installed.

6. Compile the three Java classes:

```
# javac –classpath  `hadoop classpath`  stubs/*.java
```

Note: in the command above, the quotes around *hadoop classpath* are backquotes. This runs the *hadoop classpath* command and uses its output as part of the *javac* command.

The compiled (.class) files are placed in the stubs directory.

7. Collect your compiled Java files into a JAR file:

```
# jar cvf  wc.jar  stubs/*.class
```

8. Submit a MapReduce job to Hadoop using your JAR file to count the occurrences of each word in Shakespeare:

```
# hadoop  jar  wc.jar   stubs.WordCount   /exercise/shakespeare    /wordcounts
```

This *hadoop jar* command names the JAR file to use (wc.jar), the driver class whose *main* method should be invoked (stubs.WordCount), and the HDFS input and output directories to use for the MapReduce job. Your job reads all the files in your HDFS *shakespeare* directory (which you prepared in exercise 1), and places its output in a new HDFS directory called *wordcounts*.

9. Try running this same command again without any change:

```
# hadoop   jar   wc.jar   stubs.WordCount   /exercise/shakespeare     /wordcounts
```

Your job halts right away with an exception, because Hadoop automatically fails if your job tries to write its output into an existing directory. This is by design; since the result of a MapReduce job may be expensive to reproduce, Hadoop prevents you from accidentally overwriting previously existing files.

10. Review the result of your MapReduce job:

```
# hadoop fs –ls   /wordcounts
```

This lists the output files for your job. (Your job ran with only one Reducer, so there should be one file, named part-r-00000, along with a _SUCCESS file).

11. View the contents of the output for your job:

```
# hadoop fs –cat  /wordcounts/part-r-00000 | more
```

You can page through a few screens to see words and their frequencies in the works of Shakespeare. (The spacebar will scroll the output by one screen; the letter ‘q’ will quit and return to the terminal.)

12. Try running the WordCount job against a single file:

```
# hadoop  jar  wc.jar    stubs.WordCount  /exercise/shakespeare/poems     /pwords
```

When the job completes, inspect the contents of the *pwords* HDFS directory.

13. If you wish, you can clean up the output files produced by your job runs to save your HDFS storage.

```
# hadoop fs -rm -r /wordcounts /pwords
```