CPSC 4330 Big Data Analytics

In-Class Exercise 3 – Writing a MapReduce Java Program

In this exercise, you will write a MapReduce job that reads any text input and computes the average length of all words that start with each character.

For any text input, the job should report the average length of words that begin with ‘a’, ‘b’ and so forth. For example, for input:

```
No   now   is   definitely   not   the   time
```

The output would be:

```
N      2.0

d      10.0

i       2.0

n       3.0

t        3.5
```

Your program should be case-sensitive as shown in this example.

The Algorithm

The algorithm for this program is a simple MapReduce program:

The Mapper

The Mapper receives a line of text for each input value. (Ignore the input key: byte-offset within the file) For each word in the line, emit the first letter of the word as a key, and the length of the word as a value. For example, for input value:

```
No  now  is  definitely  not  the  time
```

Your Mapper should emit:

```
N   2

n    3

i    2

d    10

n     3

t     3

t     4
```

The Reducer

Thanks to the shuffle and sort phase built into MapReduce, the Reducer receives the keys in sorted order, and all the values for one key are grouped together. So, for the Mapper output above, the Reducer receives this:

```
N   (2)

d    (10)

i     (2)

n     (3, 3)

t     (3, 4)
```

The Reducer output should be:

```
N      2.0

d      10.0

i       2.0

n       3.0

t        3.5
```

Step 1: Install JDK and Eclipse

1. Java 8 runs on the VM in docker. So on your computer, download and install Java 8 JDK from https://www.oracle.com/java/technologies/javase-jre8-downloads.html. If you have multiple java versions on your computer, set the default JDK version to Java 8.

2. Download and install Eclipse. Pick one that supports Java 8. For example, I downloaded 2020-03 version from https://www.eclipse.org/downloads/packages/release/2020-03/r .

Step 2: Start Eclipse

1. Create a new Java Project named “averagewordlength”. For JRE, choose the version of Java that you installed. Then new a package named “stubs” under the project.
2. Download the zip file “ex3.zip” from Canvas and unzip it. There are three files after unzipping: AverageReducer.java, LetterMapper.java, AvgWordLength.java. Copy them to the eclipse workspace on your computer for the project “averagewordlength” under the “stubs” folder (e.g. /Users/linli/eclipse-workspace/averagewordlength/src/stubs). Now, in Eclipse, move mouse over the project “averagewordlength”, right click the mouse and hit “refresh”. The three java files should appear under the package “stubs”.
3. Download “mapreducelib.zip” from Canvas and unzip it on your computer.
4. Configure the Build Path of the project “averagewordlength”. Right click on the project → choose “Build Path” → ”Configure Build Path”. Then go to the tab “Libraries” → “Add External JARs”. Import all JARs that are downloaded from Canvas (in “mapreducelib.zip”) in the previous step.

Step 3: Write the Program in Java

Complete the code in AverageReducer.java, LetterMapper.java, AvgWordLength.java. Here are a few details to help you begin your Java programming.

1. Define the Mapper
   Note these simple string operations in Java:

```
str.substring(0, 1)      // String:  first letter of str

str.length()             // int:   length of str
```

2. Define the Reducer

   In a single invocation the *reduce()* method receives a string containing one letter (the key) along with an iterable collection of integers (the values), and should emit a single key-value pair: the letter and the average of the integers.

3. Define the driver
   This class should configure and submit your basic job. Among the basic steps here, configure the job with the Mapper class and the Reducer class you write, and the data types of the intermediate and final keys and values.

4. Compile your classes and assemble the jar file
   To compile and jar, you may either use the command line *javac* command as you did earlier in the "Running a MapReduce Job" exercise, or follow the steps below (Step 4: "Using Eclipse to Compile Your Program") to use Eclipse.

Step 4: Use Eclipse to Compile Your Program

Follow these steps to use Eclipse to compile your program.

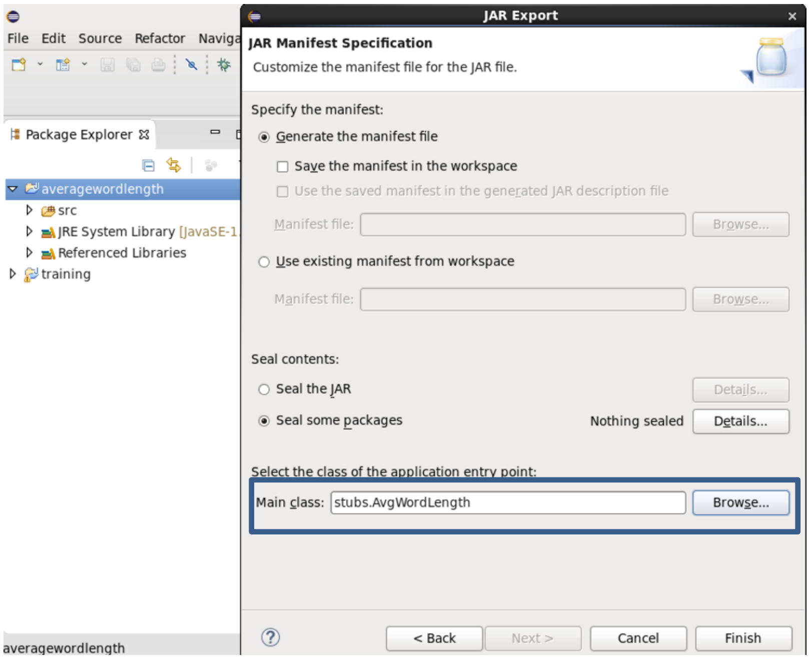
1. Verify that your Java code does not have any compiler errors or warnings.

2. In the Package Explorer, right click on the "averagewordlength" project and select "export".

3. Select Java > JAR file from the Export dialog box, then click Next.

4. Specify a location for the JAR file. You can place your JAR files wherever you like and name the jar file whatever you like, e.g. avgwordlength.jar.

5. Make sure that you have provided the Main class (stubs.AvgWordLength), as shown below.



JAR Export

JAR Manifest Specification

Customize the manifest file for the JAR file.

Specify the manifest:

● Generate the manifest file

□ Save the manifest in the workspace

□ Use the saved manifest in the generated JAR description file

Manifest file:

○ Use existing manifest from workspace

Manifest file:

Seal contents:

○ Seal the JAR

● Seal some packages — Nothing sealed

Select the class of the application entry point:

Main class: stubs.AvgWordLength

Step 5: Test your program

1. Copy the jar file from the previous step (e.g. avgwordlength.jar) to Docker.

2. In a docker terminal window, change to the directory where you placed your JAR file. Run the *hadoop jar* command as you did previously in the “Running a MapReduce Job” exercise.

```
# hadoop  jar  avgwordlength.jar  /exercise/shakespeare   /wordlengths
```

3. List the results:

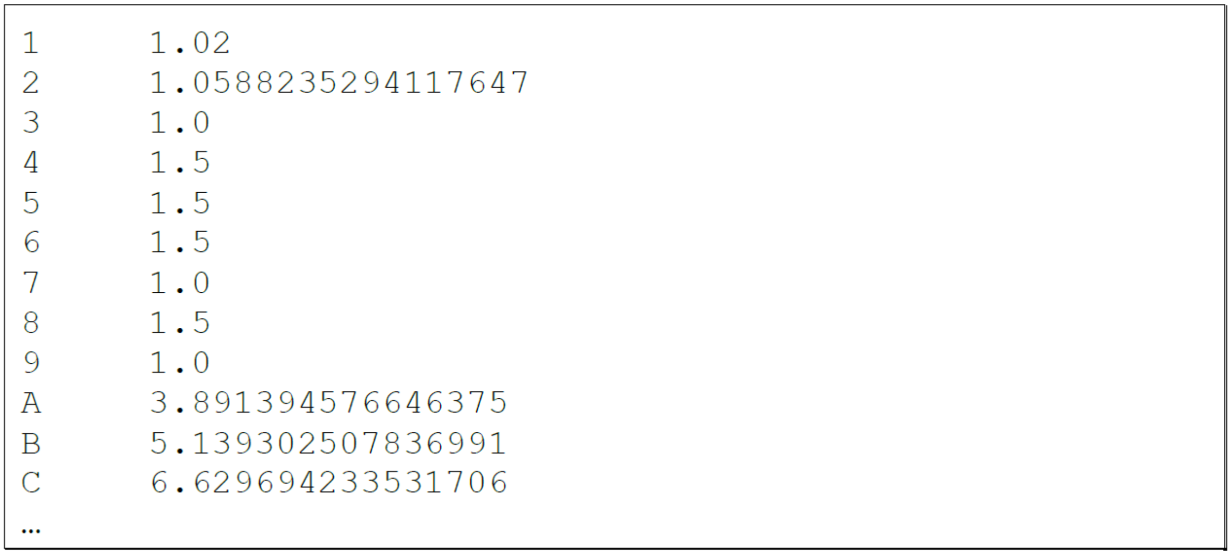
```
# hadoop  fs  -ls   /wordlengths
```

A single reducer output file should be listed.

4. Review the results:

```
# hadoop  fs   -cat  /wordlengths/*
```

The file should list all the numbers and letters in the data set, and the average length of the words starting with them, e.g:



```
1      1.02
2      1.0588235294117647
3      1.0
4      1.5
5      1.5
6      1.5
7      1.0
8      1.5
9      1.0
A      3.891394576646375
B      5.139302507836991
C      6.629694233531706
...
```

This example uses the entire Shakespeare dataset for your input; you can also try it with just one of the files in the dataset, or with your own test data.