Apache Hadoop –

A course for undergraduates

Homework Labs, Lecture 3

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

Lab: Writing a MapReduce Java Program

Projects and Directories Used in this Exercise

Eclipse project: `averagewordlength`

Java files:

`AverageReducer.java` (Reducer)

`LetterMapper.java` (Mapper)

`AvgWordLength.java` (driver)

Test data (HDFS):

`shakespeare`

Exercise directory: `~/workspace/averagewordlength`

In this lab, 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
n    3.0
d    10.0
i    2.0
t    3.5
```

(For the initial solution, your program should be case-sensitive as shown in this example.)

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

The Algorithm

The algorithm for this program is a simple one-pass MapReduce program:

The Mapper

The Mapper receives a line of text for each input value. (Ignore the input key.) 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 in to 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:

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

```
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: Start Eclipse

There is one Eclipse project for each of the labs that use Java. Using Eclipse will speed up your development time.

1. Be sure you have run the course setup script as instructed earlier in the General Notes section. That script sets up the lab workspace and copies in the Eclipse projects you will use for the remainder of the course.

2. Start Eclipse using the icon on your VM desktop. The projects for this course will appear in the Project Explorer on the left.

Step 2: Write the Program in Java

There are stub files for each of the Java classes for this lab: `LetterMapper.java` (the Mapper), `AverageReducer.java` (the Reducer), and `AvgWordLength.java` (the driver).

If you are using Eclipse, open the stub files (located in the `src/stubs` package) in the `averagewordlength` project. If you prefer to work in the shell, the files are in `~/workspace/averagewordlength/src/stubs`.

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

You may wish to refer back to the `wordcount` example (in the `wordcount` project in Eclipse or in `~/workspace/wordcount`) as a starting point for your Java code. Here are a few details to help you begin your Java programming:

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 will write, and the data types of the intermediate and final keys.

4. 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
```

5. 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.

6. 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" lab, or follow the steps below ("Using Eclipse to Compile Your Solution") to use Eclipse.

Step 3: Use Eclipse to Compile Your Solution

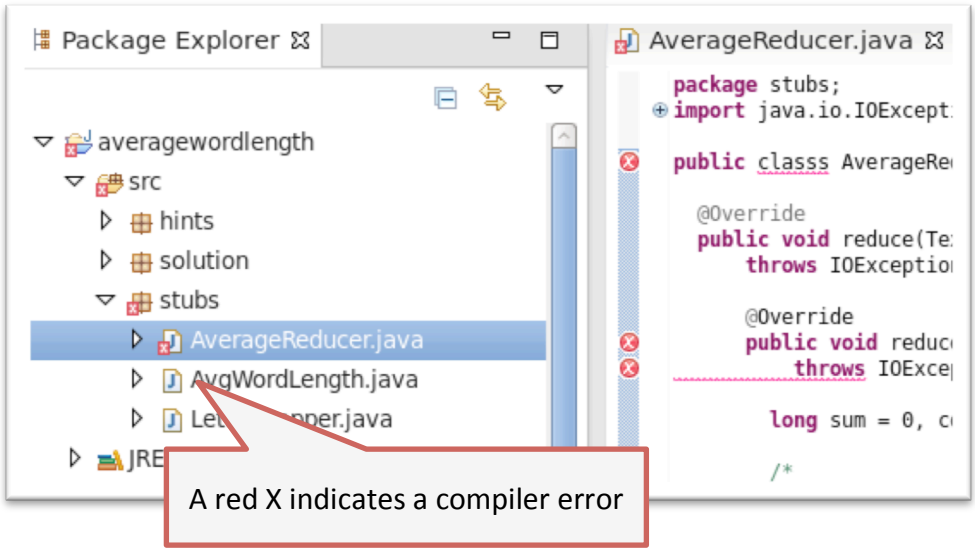
Follow these steps to use Eclipse to complete this lab.

Note: These same steps will be used for all subsequent labs. The instructions will not be repeated each time, so take note of the steps.

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

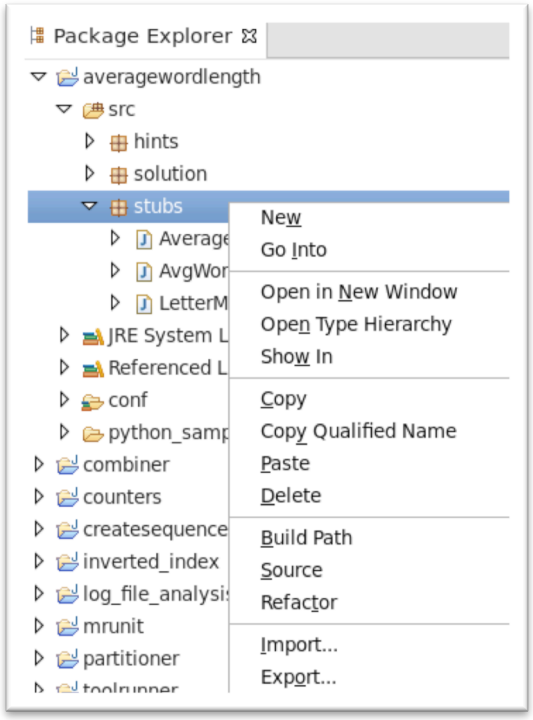
1. Verify that your Java code does not have any compiler errors or warnings.

   The Eclipse software in your VM is pre-configured to compile code automatically without performing any explicit steps. Compile errors and warnings appear as red and yellow icons to the left of the code.

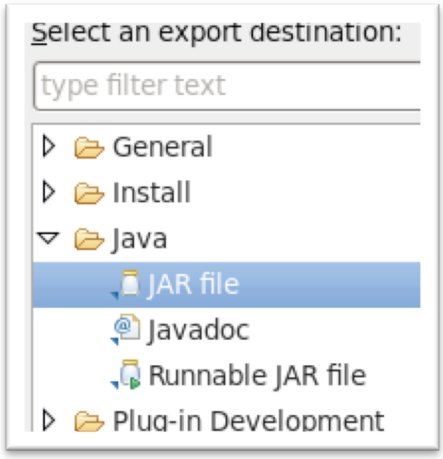


2. In the Package Explorer, open the Eclipse project for the current lab (i.e. `averagewordlength`). Right-click the default package under the `src` entry and select Export.

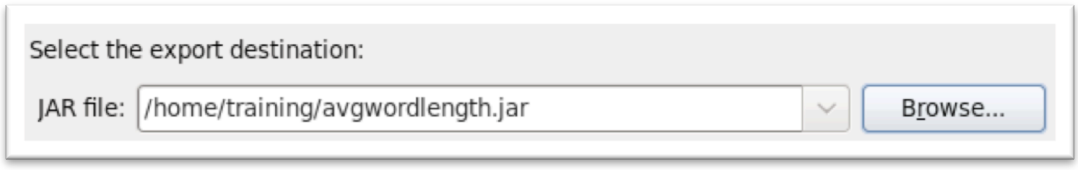
Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.



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, e.g.:



Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

Note: For more information about using Eclipse, see the *Eclipse Reference* in Homework_EclipseRef.docx.

Step 3: Test your program

1. In a 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" lab.

```
$ hadoop jar avgwordlength.jar stubs.AvgWordLength \
  shakespeare wordlengths
```

2. List the results:

```
$ hadoop fs -ls wordlengths
```

   A single reducer output file should be listed.

3. 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.:

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

```
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.

This is the end of the lab.

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

Lab: More Practice With MapReduce Java Programs

Files and Directories Used in this Exercise

Eclipse project: `log_file_analysis`

Java files:

`SumReducer.java` – the Reducer

`LogFileMapper.java` – the Mapper

`ProcessLogs.java` – the driver class

Test data (HDFS):

`weblog` (full version)

`testlog` (test sample set)

Exercise directory: `~/workspace/log_file_analysis`

In this lab, you will analyze a log file from a web server to count the number of hits made from each unique IP address.

Your task is to count the number of hits made from each IP address in the sample (anonymized) web server log file that you uploaded to the `/user/training/weblog` directory in HDFS when you completed the "Using HDFS" lab.

In the `log_file_analysis` directory, you will find stubs for the Mapper and Driver.

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

1. Using the stub files in the `log_file_analysis` project directory, write Mapper and Driver code to count the number of hits made from each IP address in the access log file. Your final result should be a file in HDFS containing each IP address, and the count of log hits from that address. **Note: The Reducer for this lab performs the exact same function as the one in the WordCount program you ran earlier. You can reuse that code or you can write your own if you prefer.**

2. Build your application jar file following the steps in the previous lab.

3. Test your code using the sample log data in the `/user/training/weblog` directory. **Note**: You may wish to test your code against the smaller version of the access log you created in a prior lab (located in the `/user/training/testlog` HDFS directory) before you run your code against the full log which can be quite time consuming.

This is the end of the lab.

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

Lab: Writing a MapReduce Streaming Program

Files and Directories Used in this Exercise

Project directory: `~/workspace/averagewordlength`

Test data (HDFS):

`shakespeare`

In this lab you will repeat the same task as in the previous lab: writing a program to calculate average word lengths for letters. However, you will write this as a streaming program using a scripting language of your choice rather than using Java.

Your virtual machine has Perl, Python, PHP, and Ruby installed, so you can choose any of these—or even shell scripting—to develop a Streaming solution.

For your Hadoop Streaming program you will not use Eclipse. Launch a text editor to write your Mapper script and your Reducer script. Here are some notes about solving the problem in Hadoop Streaming:

1. The Mapper Script

The Mapper will receive lines of text on `stdin`. Find the words in the lines to produce the intermediate output, and emit intermediate (key, value) pairs by writing strings of the form:

```
key <tab> value <newline>
```

These strings should be written to `stdout`.

2. The Reducer Script

For the reducer, multiple values with the same key are sent to your script on `stdin` as successive lines of input. Each line contains a key, a tab, a value, and a newline. All lines with the same key are sent one after another, possibly

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

followed by lines with a different key, until the reducing input is complete. For example, the reduce script may receive the following:

```
t	3
t	4
w	4
w	6
```

For this input, emit the following to `stdout`:

```
t	3.5
w	5.0
```

Observe that the reducer receives a key with each input line, and must "notice" when the key changes on a subsequent line (or when the input is finished) to know when the values for a given key have been exhausted. This is different than the Java version you worked on in the previous lab.

3. Run the streaming program:

```
$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/\
contrib/streaming/hadoop-streaming*.jar \
-input inputDir -output outputDir \
-file pathToMapScript -file pathToReduceScript \
-mapper mapBasename -reducer reduceBasename
```

(Remember, you may need to delete any previous output before running your program by issuing: `hadoop fs -rm -r dataToDelete`.)

4. Review the output in the HDFS directory you specified (`outputDir`).

This is the end of the lab.

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.

Lab: Writing Unit Tests With the MRUnit Framework

Projects Used in this Exercise

Eclipse project: `mrunit`

Java files:

`SumReducer.java` (Reducer from WordCount)

`WordMapper.java` (Mapper from WordCount)

`TestWordCount.java` (Test Driver)

In this Exercise, you will write Unit Tests for the WordCount code.

1. Launch Eclipse (if necessary) and expand the `mrunit` folder.
2. Examine the `TestWordCount.java` file in the `mrunit` project `stubs` package. Notice that three tests have been created, one each for the Mapper, Reducer, and the entire MapReduce flow. Currently, all three tests simply fail.
3. Run the tests by right-clicking on `TestWordCount.java` in the Package Explorer panel and choosing **Run As > JUnit Test**.
4. Observe the failure. Results in the JUnit tab (next to the Package Explorer tab) should indicate that three tests ran with three failures.
5. Now implement the three tests.
6. Run the tests again. Results in the JUnit tab should indicate that three tests ran with no failures.
7. When you are done, close the JUnit tab.

This is the end of the lab.

Copyright © 2010-2014 Cloudera, Inc. All rights reserved.
Not to be reproduced without prior written consent.