Programming Assignment 2: Hadoop MapReduce

Sayed Hadi Hashemi

Last update: September 2, 2015

1 Overview

Welcome to MapReduce Machine Practice. It is highly recommended that you practice the **Tutorial: Introduction to Hadoop MapReduce** before beginning this assignment.

2 Requirements

All these assignments are designed to work on and will be graded based on the **Hortonworks Sandbox 2.3** virtual machine. You need to have a working HortonWorks Sandbox machine either locally or on the Amazon Web Services.

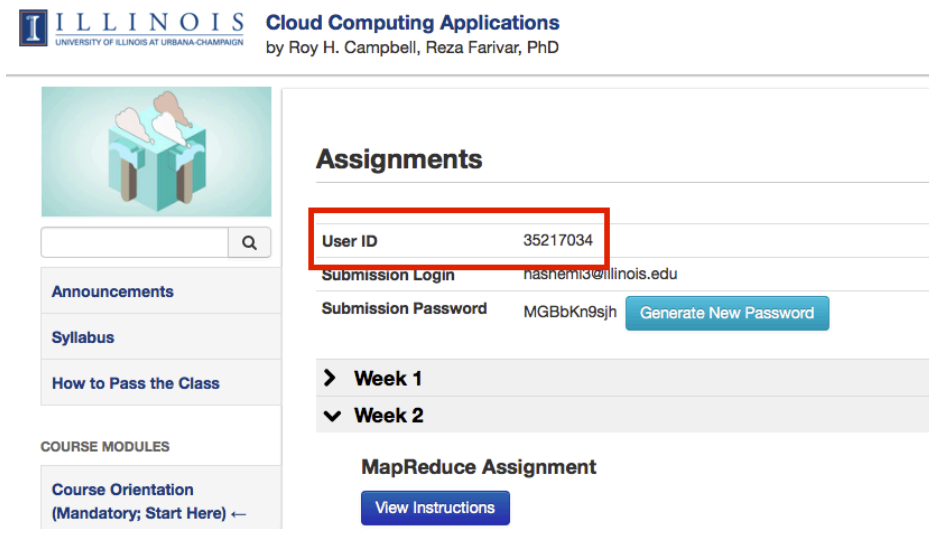
All assignments are also designed based on **JDK 7** (included in the virtual machine).

Please refer to "Tutorial: Run HortonWorks Sandbox 2.3 Locally" or "Tutorial: Run HortonWorks Sandbox 2.3 on AWS" for more information.

For a quick review of how to use this virtual machine and MapReduce, take a look at "Tutorial: Introduction to Hadoop MapReduce."

3 Setup

Step 1: Find your **User ID** in the Assignment page. You will need this ID to submit your solution.



Cloud Computing Applications
by Roy H. Campbell, Reza Farivar, PhD

Assignments

User ID	35217034
Submission Login	hashemi3@illinois.edu
Submission Password	MGBbKn9sjh

MapReduce Assignment

Step 2: Start the virtual machine, and then connect to it through the SSH.

Step 3: After successfully logging in, you should see a prompt similar to the following:

```
[root@sandbox ~]#
```

Step 4: Download the assignments files:

```
# git clone https://github.com/xldrx/cloudapp-mp2.git
```

Step 5: Change the current folder to:

```
# cd cloudapp-mp2/
```

Step 6: Initialize the assignment environment using the following command. This command will initialize the HDFS at /mp2/:

```
# bash start.sh
```

You will be asked for your **Coursera user ID.** Make sure the user ID is an exact match with the information on the assignment page on Coursera; otherwise, your submission might not be graded. If you want to update the entered **Coursera user ID, just** run the command again.

Step 7: Finish the exercises by editing the provided template files. All you need to do is complete the parts marked with **TODO**.

- Each exercise has a Java code template. All you must do is edit this file.
- Each exercise should be implemented in one file only. Multiple file implementation is not allowed.
- Only standard JDK 7 and Apache Hadoop 2.7.1 libraries may be used. In other words, **the code should be complied and run on a vanilla HortonWorks Sandbox VM**.

More information about these exercises is provided in the next section.

Step 8: After you are done with the assignments, make a submission using the following command (it is not possible to make the submission through the Coursera website):

```
# bash submit.sh
```

This command will restore the changes to the HDFS, the run some tests on your codes. This process can takes up to **10 minutes**. After that, you will be asked for your **Submission Login** and **Submission Password.** Make sure these are an exact match with the information on the assignment page on Coursera; otherwise, your submission might not be graded. Remember **Submission Login** is different than your **Coursera ID**, also **Submission Password** is not the same as your **Coursera Password.**

Step 9: You may check the score of your submission on the Coursera Programming Assignments web page. Please note that it takes a while (sometimes a few hours) for your submission to be graded. If you are not completely satisfied with your grade, you have a chance to make more submissions until the deadline.

4 Sorting

When you are to select top N items in a list, sorting is implicitly needed. Use the following steps to sort:

1. Sort the list ASCENDING based on **Firstly** count then **Secondly** on the value. If the value is string, sort lexicographically.
2. Select the bottom N items in the sorted list as Top items.

There is an implementation of this logic in the the third example of the Hadoop MapReduce Tutorial.

For example, to select top 5 items in the list {"A": 100, "B": 99, "C":98, "D": 97, "E": 96, "F": 96, "G":90}, first sort the items ASCENDING:

"G":90
"E": 96
"F": 96
"D": 97
"C":98
"B": 99
"A": 100

Then, the bottom 5 items are **A, B, C, D, F**.

Another example, to select 5 top items in the list {"43": 100, "12": 99, "44":98, "12": 97, "1": 96, "100": 96, "99":90}

"99":90
<u>**"1"**</u>: 96
<u>**"100"**</u>: 96
"12": 97
"44":98
"12": 99
"43": 100

Then, the bottom 5 items are <u>**43, 12, 44, 12, 100**</u>.

Exercise A: **Title Count**

In this exercise, you are going to implement a counter for words in Wikipedia titles. To make the implementation easier, we have provided a boilerplate for this exercise in the following file:

```
TitleCount.java
```

All you need to do is make the necessary changes in the file. However, do not change the parts marked as **"Don't Change".**

Your application takes a huge list of Wikipedia titles (one in each line) as an input and first tokenizes them using provided delimiters, after that make the tokens lowercased, then removes common words from the provided list, and finally, saves the count for them in the output.

The following is the sample command we will use to run the application:

```
# hadoop jar TitleCount.jar TitleCount -D
stopwords=/mp2/misc/stopwords.txt -D delimiters=/mp2/misc/delimiters.txt
/mp2/titles /mp2/A-output
```

Note that the actual values might be different. The order of output is not important.

Here is **a part** of a sample output of this application if "**0**" is used for user ID:

```
list  1948
de    1684
new   1077
school      1065
county      1020
state 940
john  936
disambiguation    893
station     800
route 771
...
```

The order of lines is not important. Remember, in order to submit the application **your own Coursera user ID** should be used as the user ID. Changing the user ID may change the output. To change the user ID, run the following command:

```
# bash submit.sh
```

Exercise B: **Top Titles**

In this exercise you are going to implement an application to find the top words used in Wikipedia titles. To make the implementation easier, we have provided a boilerplate for this exercise in the following file:

```
TopTitles.java
```

All you need to do is make the necessary changes in the file. However, do not change the parts marked as **"Don't Change".**

Your application takes a huge list of Wikipedia titles (one in each line) as an input and first tokenizes them using provided delimiters, after that make the tokens lowercased, then removes common words from the provided list, selects top **n** words next, and finally, saves the count for them in the output. **Use the method in section 4 Sorting to select top words.**

The following is the sample command we will use to run the application:

```
# hadoop jar TopTitles.jar TopTitles -D
stopwords=/mp2/misc/stopwords.txt -D delimiters=/mp2/misc/delimiters.txt
-D N=5 /mp2/titles /mp2/B-output
```

Note that the actual values might be different for each student.

Here is the output of this application if "**0**" is used for user ID (N=5):

```
county      1020
school      1065
new   1077
de    1684
list  1948
```

The order of lines is not important. **TextArrayWriter** and **Pair** classes are included in the template to optionally help you with your implementation.

Remember, in order to submit the application **your own Coursera user ID** should be used as the user ID. Changing the user ID may change the output. To change the user ID, run the following command:

```
# bash submit.sh
```

Exercise C: **Top Title Statistics**

In this exercise, you are going to implement an application to find some statistics about the top words used in Wikipedia titles. To make the implementation easier, we have provided a boilerplate for this exercise in the following file:

```
TopTitleStatistics.java
```

All you need to do is make the necessary changes in the file. However, do not change the parts marked as **"Don't Change".**

Your application takes a huge list of Wikipedia titles (one in each line) as an input and first tokenizes them using provided delimiters, after that make the tokens lowercased, then removes common words from the provided list, and next, selects top **n** words. **Use the method in section 4 Sorting to select top words.**

Finally, the application saves the following statistics about the top words in the output: "Mean" count, "Sum" of all counts, "Minimum" and "Maximum" of counts, and "Variance" of the counts. All values should be **floored** to be an integer. For the sake of simplicity, simply use Integer in all calculations.

The following is the sample command we will use to run the application:

```
# hadoop jar TopTitleStatistics.jar TopTitleStatistics -D
stopwords=/mp2/misc/stopwords.txt -D delimiters=/mp2/misc/delimiters.txt
-D N=5 /mp2/titles /mp2/C-output
```

Note that the actual values might be different.

Here is the output of this application if "**0**" is used for user ID (**N**=5):

```
Mean  1358
Sum   6794
Min   1020
Max   1948
Var   146686
```

The order of lines is not important. **TextArrayWriter** and **Pair** classes are included in the template to optionally help you with your implementation.

Remember, in order to submit the application **your own Coursera user ID** should be used as the user ID. Changing the user ID may change the output. To change the user ID, run the following command:

```
# bash submit.sh
```

Exercise D: **Orphan Pages**

In this exercise, you are going to implement an application to find orphan pages in Wikipedia. To make the implementation easier, we have provided a boilerplate for this exercise in the following file:

```
OrphanPages.java
```

All you need to do is make the necessary changes in the file. However, do not change the parts marked as **"Don't Change".**

Your application takes a huge list of Wikipedia links (not Wikipedia titles any more) as an input. All pages are represented by their ID numbers. Each line starts with a page ID, which is followed by a list of all the pages that the ID has a link to. The following is a sample line in the input:

```
2: 3 747213 1664968 1691047 4095634 5535664
```

In this sample, page 2 has links to page 3, 747213, and so on. Note that links are not necessarily two-way. The application should save the IDs of orphan pages in the output. Orphan pages are pages to which no other pages link.

The following is the sample command we will use to run the application:

```
# hadoop jar OrphanPages.jar OrphanPages /mp2/links /mp2/D-output
```

Note that the actual values might be different.

Here is **a part** of a sample output of this application if "**0**" is used for user ID:

```
2
14
24
51
68
83
103
107
149
158
191
...
```

The order of lines is not important. All values are integers.
Remember, in order to submit the application **your own Coursera user ID** should be used as the user ID. Changing the user ID may change the output. To change the user ID, run the following command:

```
# bash submit.sh
```

Exercise E: **Top Popular Links**

In this exercise, you are going to implement an application to find the most popular pages in Wikipedia. To make the implementation easier, we have provided a boilerplate for this exercise in the following file:

```
TopPopularLinks.java
```

All you need to do is make the necessary changes in the file. However, do not change the parts marked as **"Don't Change".**

Your application takes a huge list of Wikipedia links as an input. All pages are represented by their ID numbers. Each line starts with a page ID which is followed by a list of all the pages that the ID has a link to. The following is a sample line in the input:

```
2: 3 747213 1664968 1691047 4095634 5535664
```

In this sample, page 2 has a links to page 3, 747213, and so on. Note that links are not necessarily two-way. The application should save the IDs of top **n** popular pages as well as the number of links to them in the output. A page is popular if more pages are linked to it. **Use the method in section 4 Sorting to select top links.**

The following is the sample command we will use to run the application:

```
# hadoop jar TopPopularLinks.jar TopPopularLinks -D N=5 /mp2/links
/mp2/D-output
```

Note that the actual values might be different.

Here is the output of this application if "**0**" is used for user ID (**N**=5):

```
1921890     721
481424      729
84707 1060
5302153     1532
88822 1676
```

The order of lines is not important. All values are integers. **IntArrayWriter** and **Pair** classes are included in the template to optionally help you with your implementation.

Remember, in order to submit the application **your own Coursera user ID** should be used as the user ID. Changing the user ID may change the output. To change the user ID, run the following command:

```
# bash submit.sh
```

Exercise F: **Popularity League**

In this exercise, you are going to implement an application to find the most popular pages in Wikipedia. To make the implementation easier, we have provided a boilerplate for this exercise in the following file:

```
PopularityLeague.java
```

All you need to do is make the necessary changes in the file. However, do not change the parts marked as **"Don't Change".**

Your application takes a huge list of Wikipedia links as an input. All pages are represented by their ID numbers. Each line starts with a page ID, which is followed by a list of all the pages that the ID has a link to. The following is a sample line in the input:

```
2: 3 747213 1664968 1691047 4095634 5535664
```

In this sample, page 2 has a links to page 3, 747213, and so on. Note that links are not necessarily bidirected.

The **popularity** of a page is determined by the number of pages in the whole Wikipedia graph that link to that specific page. (Same number as **Exercise E**)

The application also takes a list of page IDs as an input (also called a **league list**). The goal of the application is to calculate the rank of pages in the league using their popularity.

The **rank** of the page is the numberof pages in the **league** with strictly less (not equal) popularity than the original page.

The following is the sample command we will use to run the application:

```
# hadoop jar PopularityLeague.jar PopularityLeague -D
league=/mp2/misc/league.txt  /mp2/links /mp2/F-output
```

Note that the actual values might be different. The maximum number of items in the league list is **16**.

The output is the list of items in the league with their ranks. The output doesn't need to be sorted. Here is the output of this application if "**0**" is used for user ID. **League**={5300058,3294332,3078798,1804986,2370447,81615,3}):

```
5300058    2
3294332    4
3078798    4
2370447    1
1804986    6
81615 3
3     0
```

The order of lines is not important. All values are integers. **IntArrayWriter** class is included in the template to optionally help you with your implementation.

Remember, in order to submit the application **<u>your own Coursera user ID</u>** should be used as the user ID. Changing the user ID may change the output. To change the user ID, run the following command:

```
# bash submit.sh
```