CS144: Web Applications

Project Part 5: Spark and Map Reduce
Due date: Wednesday, March 15, 2017, 11:00 PM

Important Notes

- **Submission deadline:** Programming work is submitted electronically and must be submitted by Wednesday at 11:00 PM. However, we recognize that there might be last minute difficulty during submission process, so as long as you started your sumission process before 11:00PM, you have until 11:55PM to completely upload your submission. After 11:55PM, you will have to use grace period as follows.
- **Late Policy:** Programming work submitted after the deadline but less than 24 hours late (i.e., by Thursday 11:00 PM) will be accepted but penalized 25%, and programming work submitted more than 24 hours but less than 48 hours late (i.e., by Friday 11:00 PM) will be penalized 50%. No programming work will be accepted more than 48 hours late. Since emergencies do arise, each student is allowed a *total* of four unpenalized late days (four periods up to 24 hours each) for programming work, but no single assignment may be more than two days late.
- **Honor Code reminder:** For more detailed discussion of the Honor Code as it pertains to CS144, please see the Assigned Work page under Honor Code. In summary: You must indicate on all of your submitted work *any assistance* (human or otherwise) that you received. Any assistance received that is not given proper citation will be considered a violation of the Honor Code. In any event, you are responsible for understanding and being able to explain on your own all material that you submit.

Overview

In this project, you will learn how to use the popular Apache Spark engine to perform a (potentially heavy) computational task on a large number of machines using the Map-Reduce framework. In particular, you will identify the most "popular" users on the Twitter network, measured by the number of their followers. We will provide a new Virtual Machine image with the Spark engine preinstalled and a snapshot of the follower-following graph of Twitter. Your job is to write a (simple) code on Spark that returns the IDs of the users with the high follower counts

Files needed:

- twitter.edges: Twitter follower-following graph data
- spark.ova: Virtual machine image with Spark

Learning Basics

Twitter Graph File and Our Task

Download the twitter.edges file that contains a snapshot of the follower-following graph structure of Twitter. Each line of the file represents the "following" edges from a particular Twitter user in the format below:

```
user1: user2,user3,...,userk
```

The above line indicates that user1 is "following" user2 through userk. Note that each user in the file is represented as a unique random integer. For example, the first line of the file:

```
138134695: 62902439,15224867
```

indicates that the user "138134695" is following two other users, 62902439 and 15224867.

Given this file, it is relatively straightfoward to find the user who follows the largest number of users. We simply need to identify the line with the largest of user IDs behind colon. Unfortunately, our task is more complex. **We need to identify the users who are followed by a large number of other users (more precisely, 1,000 other users).** While our dataset is reasonably small -- it is only 21 MB in size -- you can imagine that this dataset can potentially be huge, so we want to implement this task using the Apache Spark Engine, so that we can perform this task in parallel on as many available machines as possible.

Apache Spark

Writing a program that runs on a large number of machines in parallel can be a daunting task. To make this task easier, a number of distributed software infrastructures have been developed. In particular, as we learned in class, Map-Reduce framework asks the programmer to provide just the core computational logic of the given task as a set of Map and Reduce functions. Given this core functions, Map-Reduce framework takes care of the rest, including data distribution, parallel execution, process monitoring, and result shuffling and collection. Apache Spark is a popular open source software that provides Map-Reduce style programming environment on a cluster with a large number of distributed machines.

We have created a Virtual Machine image with Apache Spark and made it available as spark.ova. Please download the Virtual Machine image, import it to your VirtualBox using our earlier instruction, and make sure that you can power up and log in to the machine. As before, the default username is "cs144" with password "password". Once you are inside our new virtual machine, you can run a Spark interactive shell by executing the `spark-shell` command:

```
cs144@cs144:~$ spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/03/07 18:16:29 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/03/07 18:16:30 WARN Utils: Your hostname, cs144 resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
```

```
17/03/07 18:16:30 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/03/07 18:16:37 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording th
17/03/07 18:16:37 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
17/03/07 18:16:38 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1488939390852).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_121)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Inside the Spark shell, you can execute any scala command using Spark API. You can exit from the Spark interactive shell by pressing [Control+D] key

Now that we have a virtual machine with Apache Spark, it is time to learn how to use it, by going over one of widely available Apache Spark tutorials on the Internet. For example, the official Quick Start Tutorial provides a ten-minute introduction to essential basics. The file wordCount.scala also constains the example code that we went over in the class:

```
val lines = sc.textFile("input.txt")
val words = lines.flatMap(line => line.split(" "))
val word1s = words.map(word => (word, 1))
val wordCounts = word1s.reduceByKey((a,b) => a+b)
wordCounts.saveAsTextFile("output")
System.exit(0)
```

Again, the above code computes the frequency of each word in the text file input.txt and generates (word, frequency) pairs as the output. When `saveAsTextFile("output")` is called, the program creates a new subdirectory named `output`, where `part-xxxxx` file(s) are generated that contain the (word, frequency) output pairs in `wordCounts`. Note the last line in the `wordCount.scala` script: `System.exit(0)`. Calling the system exit function ensures that once the script finishes, the interactive shell is aborted as well. You can execute this file using the Spark shell using the following command:

```
cs144@cs144:~/wordCount$ spark-shell -i wordCount.scala

     ...

cs144@cs144:~/wordCount$ ls -l output/
total 8
-rw-r--r-- 1 cs144 cs144 4187 Mar  7 19:19 part-00000
-rw-r--r-- 1 cs144 cs144    0 Mar  7 19:19 _SUCCESS
cs144@cs144:~/wordCount$ head output/part-00000
(country,2)
(House,2)
(it,2)
(outside,1)
(nothing,1)
(travel,,1)
(order,7)
(Trump's,2)
(national,1)
(federal,1)
```

Note that the provided `wordCount.scala` code is written to be executed in an interactive shell only. If we want to properly run our program on multiple machines in a Spark cluster, we need to "wrap" this code within an object with a "main" function. We also need to create proper "Spark Configuration" and "Spark Context" in which our program will run. For example, wordCountFull.scala shows an extended version of our wordCount program that can be prooperly compiled into an executable "package", say `word-count-project.jar` using a tool chain such as `sbt`. Once packaged, it can be submitted to a Spark cluster for parallel execution using a command like the following:

```
spark-submit --class edu.ucla.cs144.WordCount --master spark://23.195.26.187:7077 --deploy-mode cluster word-count-project.jar
```

Since the goal of this project is to introduce you to the main programming paradigm/API of Spark, not the nitty gritty details of the Spark packaging tool chain and job submission interface, in this project we will assume that your code will be executed **through a Spark interactive shell** using the command "`spark-shell -i`".

While Spark supports multiple languages, Scala is the most popular (and syntactically clean) language to program on Spark, so **we use Scala for this project.** Basic Scala (needed to complete this project) is easy to learn and your code will be much clearner than when you use other languages. There exist many quick online tutorials on Scala, such as this one. Fortunately, this project can be completed without using any advanced Scala construct (like class inheritance), except that you may want to use "anonymous functions", so that you can pass a function as a parameter to a "Map-Reduce" function easily.

Writing Your Code

Now that you got the basics, it is time to write code. Your code must read the text file twitter.edges located in the current directory, parse it to obtain the Twitter follower-following graph, perform necessary computation, and return the list of all (userid, follower_count) pairs for the users with more than 1000 followers. The output from your code should contain many lines of (userid, follower_count) pairs like:

```
(40981798,8569)
(43003845,7621)
...
```

The first two lines of the above output, for example, indicate that the users 40981798 and 43003845 have 8569 and 7621 followers, respectively. The output (userid, follower_count) pairs should be saved as a (set of) text file(s) in the "output" directory using the Spark `saveAsTextFile()` function. **The output (userid, follower_count) pairs may appear in any order and need not be sorted.**

In writing your code, you may find the list of Spark transformation functions helpful. Also, if you need a "hint" on parsing the provided twitter.edges file, you may find this question and answer at StackOverflow helpful (local mirror is available here).

Before we finish, we reiterate the essential requirements of your code.

Code Requirements

1. Your code should read the twitter graph from the file "twitter.edges" located in the current directory.
2. The output from your code should be the list of (userid, follower_counts) pairs for all the users whose follower count is larger than 1000. The output does not have to be sorted.
3. Your code should save the output in the "output" subdirectory within the current working directory using `saveAsTextFile()` of Spark RDD.
4. Your code should compute the final results using Map-Reduce-style programming by applying a series of Spark transformation functions to the input dataset.

Testing Your Code

Before submitting your code, throughly test your code so that it computes the correct results. To help you ensure that your code produces the correct output, here are a few sample results from our dataset:

User ID	Follower Count
40981798	8569
3359851	3905
88323281	2315
18742444	1585
9451052	1184
302847930	1182
12925072	1002

In total, there are 177 users with more than 1000 followers.

What to Submit

For this project, you need to submit **a single zip file** named `project5.zip` that has the following packaging structure.

```
project5.zip
 |
 +- team.txt
 |
 +- topUsers.scala
 |
 +- README.txt
```

Each file or directory is as following:

1. **`team.txt`**: A plain-text file (no word or PDF, please) that contains the UID(s) of every member of your team. If you work alone, just write your UID (e.g. 904200000). If you work with a partner, write both UIDs separated by a comma (e.g. 904200000, 904200001). **DO NOT put any other content, like your names, in this file!**
2. **`topUsers.scala`**: this is the main Scala code that you wrote to compute the top Twitter users. This code should be executable simply by typing "`spark-shell -i topUsers.scala`". Please **DO NOT** submit any input or output files for your code. Just submit your main Scala script.
3. **`README.txt`** includes any comments you find worth noting, regarding your code structure, etc.

The three files should be **contained directly** under the **project5.zip** (without any enclosing folders).

Testing Your Zip File

To ensure the correct packaging of your submission, we have made a grading script p5_test for Project 5, which can be executed like:

```
cs144@cs144:~$ ./p5_test project5.zip
```

(Add the appropriate path to the project5.zip if needed. You may need to use "chmod +x p5_test" if there is a permission error.)

You **MUST** test your submission using the script to minimize the chance of an unexpected error during grading. When evertything runs properly, you will see an output similar to the following from the script:

```
Executing your Spark code.....
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/03/07 20:19:47 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/03/07 20:19:47 WARN Utils: Your hostname, cs144 resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
17/03/07 20:19:47 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/03/07 20:19:55 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording th
17/03/07 20:19:55 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
17/03/07 20:19:56 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
```

```
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1488946788877).
Spark session available as 'spark'.
Loading topUsers.scala...

...

(20,1010)
(99,1010)
(10,1010)

SUCCESS! We finished testing your zip file integrity.
```

Once your work is properly packaged as a zip file, submit your zip file via our submission page at CCLE.

You may submit as many times as you like, however only the latest submission will be saved, and those are what we will use for grading your work and determining late penalties.

Grading Criteria

Overall grading breakdown is as below

- Submitted code runs without any error and produces output (30%)
- Submitted code computes correct answer on the provided dataset (30%)
- Submitted code computes correct answer on different datasets with the same format (40%)