CSE6242 / CX4242: Data and Visual Analytics | Georgia Tech | Fall 2015

Homework 3: Hadoop, Spark, and Pig
Due: November 6, 2015, 11:55PM EST

Prepared by Meera Kamath, Gopi Krishnan, Siddharth Raja, Ramakrishnan Kannan, Akanksha, Polo Chau

Submission Instructions:
It is important that you read the following instructions carefully and also those about the deliverables at the end of each question or you may lose points.

- ❏ Submit a **single zipped file**, called "HW3-{YOUR_LAST_NAME}-{YOUR_FIRST_NAME}.zip", containing all the deliverables including source code/scripts, data files, and readme. Example: 'HW3-Doe-John.zip' if your name is John Doe. Only .zip is allowed (no .rar, etc.)
- ❏ You may collaborate with other students on this assignment, but **each student must write up their own answers in their own words**, and must write down the collaborators' names on T-Square's submission page. Any suspected plagiarism and academic misconduct will be reported and directly handled by the Office of Student Integrity (OSI).
- ❏ If you use any "*slip days*", you must write down the number of days used in the T-square submission page. For example, "Slip days used: 1"
- ❏ At the end of this assignment, we have provided a folder structure that describes how we expect the files to be contained in that single zipped file. Please make sure you submit your files in the format specified. 5 points will be deducted for not following this strictly.
- ❏ Wherever you are asked to write down an explanation for the task you perform, please stay within the word limit or you may lose points.
- ❏ In your final zip file, please do not include any intermediate files you may have generated to work on the task, unless your script is absolutely dependent on it to get the final result (which it ideally should not be).

Applying for AWS Educate Account (do this now!) & Set Up Alerts

It is EXTREMELY IMPORTANT that you apply for an "AWS Educate" account RIGHT AWAY to get $100 free credits, and verify that the credit has been properly applied on your account, so that you can work on Task 3. Creating the account can take days and HW3's computation can take hours to run, so if you do not do this now, you may jeopardize your HW3 progress.

- Go to https://aws.amazon.com/education/awseducate/
- Click the **Apply Now** button on the right
- Click the **Apply for AWS Educate for Students** button
- Choose **student**, and click **Next**
- Fill out the application

- If you do not have an AWS account ID, you will need to sign up to get one, as the form suggests.
- To find your AWS account ID, log into the console and then going to this link https://console.aws.amazon.com/billing/home?#/account
- Your AWS Account ID is right at the top of this screen

Also EXTREMELY IMPORTANT is that you set up a billing alarm on AWS to notify you when your credits are running low. Here's how: http://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/free-tier-alarms.html

Shut down *everything* when you're done with the instances, or you may get surprising bills with hundreds or even thousands of dollars on your credit card. Highest record from previous classes goes above $2000!! The good news is you can call AWS to explain the situation and they should be able to waive the charges.

Setting up Development Environment for Task 1 and Task 2

Installing CDH

Download a preconfigured virtual machine (VM) image from Cloudera (CDH). **Please use 5.3.x version of CDH or higher.** You can choose any VM platform, but we recommend VirtualBox because it is free. Instructions to setup and configure the VM can be found here.

Read these important tips to improve the VM's speed and reduce memory consumption.

Once you launch the VM, you will have a GUI environment with cloudera user, which has administrator (sudo) privilege. The account details are:

username: cloudera
password: cloudera

The virtual image comes pre-installed Hadoop and Spark. You will use them for this assignment. Tip: You may want to setup port forwarding to obtain SSH access of guest operating system.

Loading Data into HDFS

Now, let's load our dataset into the HDFS (Hadoop Distributed File System), which is an abstract file system that stores files on clusters. Your Hadoop or Spark code will directly access files on HDFS. Paths on the HDFS look similar to those on the UNIX system, but you can't explore them directly using standard UNIX commands. Instead, you need to use ***hadoop fs*** commands. For example

```
hadoop fs -ls /
```

Download the following two graph files: graph1.tsv[1] (4MB) and graph2.tsv[2] (870MB). Use the following commands to setup a directory on the HDFS to store the two graph datasets. Please do not change the directory structure below (/user/cse6242/) since we will grade your homework using the scripts which assume the following directory structure.

```
sudo su hdfs
hadoop fs -mkdir /user/cse6242/
hadoop fs -chown cloudera /user/cse6242/
exit
su cloudera
hadoop fs -put graph1.tsv /user/cse6242/graph1.tsv
hadoop fs -put graph2.tsv /user/cse6242/graph2.tsv
```

Now both files- graph1.tsv and graph2.tsv are on the HDFS at **/user/cse6242/graph1.tsv** and **/user/cse6242/graph2.tsv**

Setting up Development Environments

We found that compiling and running Hadoop/Scala code can be quite complicated. So, we have prepared a skeleton code, compilation scripts, and execution scripts for you. Please download this.

The zip file has preset directory structures. As you will zip all the necessary files with the same directory structure in the end, you may not want to modify the structure. (See the end of this document for details). In the directories of both *Task1* and *Task2*, you will find ***pom.xml***, ***run1.sh, run2.sh*** and ***src*** directory.

- ***src*** directory contains a main Java/Scala file that you will primarily work on. We have provided some code to help you get started. Feel free to edit it and add your files in the directory, but the main class should be Task1 and Task2 accordingly. Your code will be evaluated using the provided ***run1.sh*** and ***run2.sh*** file (details below).

- ***pom.xml*** contains necessary dependencies and compile configurations for each task. To compile, you can simply call Maven in the corresponding directory (Task1 or Task2 where pom.xml exists) by this command:

  ```
  mvn package
  ```

[1] This graph is originally from the Enron email network data set. There are 321 thousand edges and 77 thousand nodes.
[2] This graph is from the Portuguese Wikipedia link data set. There are 53 million edges and 1 million nodes.

It will generate a single JAR file in the target directory (i.e. target/task2-1.0.jar). Again, we have provided you some necessary configurations to simplify your work for this homework, but you can edit them as long as our run script works and the code can be compiled using mvn package command.

- ***run1.sh***, ***run2.sh*** are the script files that run your code over graph1.tsv (run1.sh) or graph2.tsv (run2.sh) and download the output to a local directory. The output files are named based on its task number and graph number (e.g. task1output1.tsv). You can use these run scripts to test your code. Note that these scripts will be used in grading. Here are what the scripts do. (You can open them in a text editor to see what is going on.)
  1) run your JAR on Hadoop/Scala specifying the input file on HDFS (the first argument) and output directory on HDFS (the second argument)
  2) merge outputs from output directory and download to local file system.
  3) remove the output directory on HDFS.

[35pt] Task 1: Analyzing a Large Graph with Hadoop/Java

Please first go over the Hadoop word count tutorial to get familiar with Hadoop.

Goal

Your task is to write a MapReduce program to calculate the sum of the weights of all **incoming** edges for each node in the graph.

You should have already loaded two graph files into HDFS. Each file stores a list of edges as tab-separated-values. Each line represents a single edge consisting of three columns: (source node ID, target node ID, edge weight), each of which is separated by a tab (\t). Node IDs are positive integers, and weights are also positive integers. Edges are sorted in a random order.

```
src   tgt   weight
51    117   1
51    194   1
51    299   3
151   230   51
151   194   79
130   51    10
```

Your code should accept two arguments upon running. The first argument (args[0]) will be a path for the input graph file on HDFS, and the second argument (args[1]) will be a path for output directory. The default output mechanism of Hadoop will create multiple files on the output directory such as part-00000, part-00001, which will be merged and downloaded to a local

directory by the supplied run script. Please use the run scripts for your convenience.

The format of the output should be as follows. Each line represents a node ID and the sum of its incoming edges' weights. The ID and the sum must be separated by a tab(\t), and lines don't need be sorted. The following example result is computed based on the toy graph above. Please exclude the nodes with no incoming edges (i.e. the sum equals zero).

```
51    10
117   1
194   80
230   51
299   3
```

Deliverables

1. [15pt] **Your Maven project directory including Task1.java.** Please see detailed submission guide at the end of this document. **You should implement your own map/reduce procedure and should not import external graph processing library.**

2. [5pt] **task1output1.tsv**: the output file of processing graph1 by run1.sh.

3. [5pt] **task1output2.tsv**: the output file of processing graph2 by run2.sh.

4. [10pt] **description.txt**: a short description (in no more than 50 words) about your map/reduce procedure.

[30pt] Task 2: Analyzing a Large Graph with Spark/Scala

Please go over this Spark word count tutorial.

Goal

You will implement the same task (calculating the sum of the incoming edge weights for the nodes in the graph) using Spark with the Scala language.

Your Scala program should handle the same two arguments for input and output and should generate the same formatted output file on the supplied output directory (tab-separated-file). Please note that the default Spark *saveastextfile* method uses a different saving format from Hadoop, so you need to format the result beforehand (Tip: use *map* and *mkString*). Again, the result doesn't need to be sorted.

Deliverables

1. [10pt] **Your Maven project directory including Task2.scala**. Please see the detailed submission guide at the end of this document. **You should implement your own map/reduce procedure and should not import external graph processing library.**

2. [5pt] **task2output1.tsv**: the output file of processing graph1 by run1.sh.

3. [5pt] **task2output2.tsv**: the output file of processing graph2 by run2.sh.

4. [10pt] **description.txt**: a short description (in no more than 50 words) about your map/reduce procedure and compare your impressions of using Hadoop/Java vs. Spark/Scala.

[35pt] Task 3: Analyzing Large Amount of Data with Pig on AWS

You will try out PIG (http://pig.apache.org) for processing n-gram data on Amazon Web Service (AWS). This is a fairly simple task, and in practice you may be able to tackle this task using commodity computers (e.g., consumer-grade laptops or desktops). However, we would like you to to use exercise to learn to solve it using distributed computing, on Amazon EC2 to gain experience (very helpful for your future career in research or industry), so you will be prepared to tackle more complex problems.

The services you will primarily be using are Amazon S3 storage, Amazon Elastic Cloud Computing (EC2) virtual servers in the cloud, and Amazon Elastic MapReduce (EMR) managed Hadoop framework.

This task will ideally use up only a very small fraction of your $100 credit. AWS allows you to use up to 20 instances in total (that means 1 master instance and up to 19 core instances) without filling out a "limit request form". **For this assignment, you should not exceed this quota of 20 instances.** You can learn about these instance types, their specs, and pricing at
https://aws.amazon.com/ec2/instance-types/
https://aws.amazon.com/ec2/pricing/
(In the future, for larger jobs, you may want to use AWS's pricing calculator:
http://calculator.s3.amazonaws.com/index.html)

Please read the AWS Setup Guidelines provided to set up your AWS account. In this task, you will use subsets of the Google books *n*-grams dataset (full dataset is here), on which you will perform some analysis. An '*n*-gram' is a phrase with *n* words; the full n-gram dataset lists n-grams present in the books on books.google.com along with some statistics.

You will perform your analysis on two datasets, extracted from the Google books bigrams (2-grams), that we have prepared for you: a large one (**s3://ngrams-english-mega**) and a

smaller one (**s3://bigrams-small**)

The files in these two S3 buckets are stored in a tab(\t) separated format. Each line in a file has the following format:

```
n-gram TAB year TAB occurrences TAB books NEWLINE
```

An example for 2-grams (or bigram) would be:

```
I am       1936  342   90
I am       1945  211   10
very cool  1923  500   10
very cool  1980  3210  1000
very cool  2012  9994  3020
```

This tells us that, in 1936, the bigram 'I am' appeared 342 times in 90 different books. In 1945, 'I am' appeared 211 times in 10 different books. And so on.

Goal

For each unique bigram, compute its average number of appearances per book. For the above example, the results will be:

```
I am       (342 + 211) / (90 + 10) = 5.53
very cool  (500 + 3210 + 9994) / (10 + 1000 + 3020) = 3.40049628
```

Output the 10 bigrams having the highest **average number of appearances per book** along with their corresponding averages, in **tab-separated format**, sorted in descending order. If multiple bigrams have the same average, **order them alphabetically**. For the example above, the output will be:

```
I am       5.53
very cool  3.40049628
```

You will solve this problem by writing a PIG script on Amazon EC2 and save the output.

You can use the interactive PIG shell provided by EMR to perform this task from the command line (grunt). In this case, you can copy the commands you used for this task into a single file to have the PIG script and the output from the command line into a separate file. Please see this for how to use PIG shell. Also, you can upload the script and create a task on your cluster.

To load the data from the s3://bigrams-small bucket into a PIG table, you can use the following command:

grunt> bigrams = LOAD ‘s3://bigrams-small/*’ AS (bigram:chararray, year:int, count:int, books:int);
(**HINT**: You might want to change the data type for **year**, **count** or **books**)

While working with the interactive shell (or otherwise), you should first test on a small subset of the data instead of the whole data (the whole data is over hundreds of GB). Once you believe your PIG commands are working as desired, you can use them on the complete data and ...wait... since it will take some time.

Deliverables:

- pig.txt: The PIG script for the task (using the larger data set).
- output-small.txt: Output (tab-separated) for the smaller data set.
- output-big.txt: Output (tab-separated) for the larger data set.

Submission Guideline

Submit the deliverables as a single **zip file named HW3-*Lastname-Firstname.zip***. Please specify the name(s) of any students you have collaborated with on this assignment, using the text box on the T-Square submission page.

The directory structure of the zip file should be exactly as below (the unzipped file should look like this):

```
HW3-Smith-John/        (this should reflect your name, not skeleton)
    Task1/
        src/main/java/edu/gatech/cse6242/Task1.java
        description.txt
        pom.xml
        run1.sh
        run2.sh
        task1output1.tsv
        task1output2.tsv
        (do not attach target directory)

    Task2/
        src/main/scala/edu/gatech/cse6242/Task2.scala
        description.txt
        pom.xml
        run1.sh
        run2.sh
        task2output1.tsv
```

```
        task2output2.tsv
        (do not attach target directory)

    Task3/
        pig.txt
        output-small.txt
        output-big.txt
```

Please adhere to the naming convention specified above.