IEMS 5709 Fall 2016 Homework 2

Release date: Oct 24, 2016

Due date: 23:59, Nov 14, 2016

The solution will be posted right after the deadline, so no late homework will be accepted!

Every Student MUST include the following statement, together with his/her signature in the submitted homework.

I declare that the assignment submitted on Elearning system is original except for source material explicitly acknowledged, and that the same or related material has not been previously submitted for another course. I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the website http://www.cuhk.edu.hk/policy/academichonesty/.

Signed (Student__________________________) Date:_______________________________

Name___________________________________ SID_________________________________

Submission notice:

- Submit your homework via the elearning system

General homework policies:

A student may discuss the problems with others. However, the work a student turns in must be created COMPLETELY by oneself ALONE. A student may not share ANY written work or pictures, nor may one copy answers from any source other than one's own brain.

Each student **MUST LIST** on the homework paper the **name of every person he/she has discussed or worked with**. If the answer includes content from any other source, the student **MUST STATE THE SOURCE**. Failure to do so is cheating and will result in sanctions. Copying answers from someone else is cheating even if one lists their name(s) on the homework.

If there is information you need to solve a problem but the information is not stated in the problem, try to find the data somewhere. If you cannot find it, state what data you need, make a reasonable estimate of its value, and justify any assumptions you make. You will be graded not only on whether your answer is correct, but also on whether you have done an intelligent analysis.

Q1 [30 marks]: PageRank Algorithm on GraphLab

In this question, you are required to run the PageRank algorithm on the given dataset, and then output the top 100 nodes ranked by their PageRank scores. The dataset is a directed web graph from the Google programming contest, consisting of 875,713 nodes and 5,105,039 edges. See Ref [1] for more details about the data statistics. Note that the data has been pre-processed to exclude all the dead-end nodes and it is available in Ref [2]. Each line of the data is a [source ID] [target ID] pair separated by whitespace.

(a) [10 marks] GraphLab Create [3] is an extensible machine learning framework that enables developers and data scientists to easily build and deploy intelligent applications and services at scale. Follow the procedure in Ref [4] to install GraphLab Create on one single virtual machine in AWS. In your homework submission, show the commands you used.

(b) [20 marks] On one machine, run the PageRank toolkit, provided by GraphLab Create in Ref [5]. Submit the code and the top 100 nodes.

Q2 [30 marks + 20 bonus marks]: Word Counting on a Storm Cluster

In this question, you are required to implement the word-counting algorithm under the Storm platform in our IE DIC cluster. The dataset is available at:
https://github.com/YangRonghai/100ebooks/blob/master/StormData?raw=true

The basic idea is to first define a FileReaderSpout class to read an input file and emit a sentence every time. Then you can implement a SplitSentenceBolt class to split every sentence into words and emit the word to the next WordCountBolt class , which needs to compute the frequency of each word. The word count result should be dumped to a persistent storage. You are allowed to refer to (or even borrow codes) from publicly available Storm examples/source-codes AS LONG AS you clearly list and acknowledge your sources in your submission.

(a) [15 marks] Find Frequent Word: At most once
Write a word counting program without any processing guarantee, i.e. under the at-most-once model. Your program should read the input file, count the frequency of words and dump the final result to the persistent storage (e.g., Linux file system or HDFS). Storm utilizes Maven [11] to compile and build the executable Jar. You can reuse the same configuration file (i.e., pom.xml) under the *Storm_HOME/examples/storm-starter/* directory.

Find out the Top- 10 most frequently used words in the dataset and their corresponding frequency. Submit your code and the final top 10 words and their corresponding frequency.

Hints:

- You are allowed to post-process the result dumped from Storm.
- DO NOT emit the entire file in one nextTuple to avoid failure caused by network congestion. You can emit one line for each *nextTuple()* call.
- In case of buffer overflow, you can configure *topology.max.spout.pending* [15] to specify the maximum number of unacknowledged tuples that can exist in your topology (from that spout) at a given time. Otherwise, there would be lots of failures given the relatively limited resource in your small cluster.
- Note that when you read a file in the open method, each worker would go to its local path to find the file. With this method, you need to copy the input file to every worker node. A better (yet more complicated) option to solve this issue is to use the shared HDFS. Both ways are acceptable.
- Copy your java code to the following directory: *Storm_HOME/examples/storm-starter/src/jvm/storm/starter/*
  and then use Maven to compile.

(b) [20 bonus marks] Guaranteed Message/Tuple Processing: the At-Least-Once Model.
Refer to [12,13,14] to change the program in part (b), so that the topology can guarantee to process every message (i.e., at least once model). You are required to manually fail the delivery of "*the*" for 10 times. In other words, you can use *collector.fail(tuple)* to simulate the error in the first bolt when the word "*the*" appears for the first 10 times. Under the at- least-once model, find out the Top- 10 most frequently used words. With these artificial errors, re-run the program in part (b) and output the Top- 10 most frequently used words. Do they generate the same result? Why or why not? Compare the performance of these two models (including running time, number of tuples emitted, number of tuples Acked). Submit the code and the results. Explain your findings.

Hints:

- You can use the Java built-in function, e.g., *System.currentTimeMillis(),* to get the running time.

(c) [15 marks] Scaling the topology.
Refer to Page 49 of the lecture slides on Storm to change the parallelism degree (i.e., # of Spout threads, # of Bolt threads) of your program. Re-run the program in part (a) and compare the performance (e.g., running time, number of tuples emitted, number of tuples Acked, number of tuples Failed) under the following 4 different settings. Explain your findings.

	# of FileReaderSpout	# of SplitSentenceBolt	# of WordCountBolt
Setting 1	3	3	3

Setting 2	1	3	3
Setting 3	1	1	3
Setting 4	3	10	10

Q3 [30 marks]: Basic Pig operations

You are required to perform some simple analysis using Pig (already installed in IE DIC Cluster) on the n-grams dataset of Google books. An 'n-gram' is a phrase with n words. The dataset lists all n-grams present in books from books.google.com along with some statistics.

In this question, you only use the Google books bigram (1-grams). Please go to Reference [16] and [17] to download the two datasets. Each line in these two files has the following format (TAB separated):

bigram year match_count volume_count

An example for 1-grams would be:

```
circumvallate	1978	335	91
circumvallate	1979	261	95
```

This means that in 1978(1979), the word "circumvallate" occurred 335(261) times overall, from 91(95) distinct books.

What you need to do is the following:

(a) **[10 marks]** Upload these two files to HDFS and **join** them into one table.

(b) **[10 marks]** For each unique bigram, compute its average number of occurrences per year. In the above example, the result is:

circumvallate (335 + 261) / 2 = 298

Notes: The denominator is the number of years in which that word has appeared Assume the data set contains all the 1-grams in the last 100 years, and the above records are the only records for word 'circumvallate'. Then the average value is

(335 + 261) / 2 = 298,

instead of

(335 + 261) / 100 = 5.96

(c) **[10 marks]** Output the **20** bigrams with the highest average number of occurrences per year along with their corresponding average values sorted in the descending order.

If multiple bigrams have the same average value, write down any one you like (that is, break ties as you wish).

You need to write a Pig script to perform this task and save the output into HDFS.

Submit your output together with Pig script.

Hints:

- This problem is very similar to the word counting example shown in the lecture notes of Pig. You can use the code there and just make some minor changes to perform this task.

[1] SNAP Google Web Data
https://snap.stanford.edu/data/web-Google.html
[2] Processed Google Web Graph Data
https://snap.stanford.edu/data/web-Google.txt.gz
[3] Graphlab Create
https://turi.com/products/create/
[4] Installation Guide for GraphLab Create
https://dato.com/download/install-graphlab-create.html
[5] GraphLab Create PageRank Toolkit
https://dato.com/products/create/docs/graphlab.toolkits.graph_analytics.html#pagerank
[6] Dato Distributed Introduction
https://dato.com/learn/userguide/deployment/pipeline-introduction.html
[7] Dato Distributed Setup
https://dato.com/learn/userguide/deployment/pipeline-hadoop-setup.html
[8] Distributed Machine Learning in Dato Distributed Cluster (Follow the Hadoop section)
https://dato.com/learn/userguide/deployment/pipeline-dml.html
[9] How to install a distributed apache Storm cluster
http://knowm.org/how-to-install-a-distributed-apache-storm-cluster/
[10] Setting up a Storm Cluster
http://storm.apache.org/documentation/Setting-up-a-Storm-cluster.html
[11] Storm Maven
http://storm.apache.org/documentation/Maven.html
[12] Guaranteeing-message-processing:
https://storm.apache.org/documentation/Guaranteeing-message-processing.html
[13] At least once example:
https://www.safaribooksonline.com/library/view/getting-started-with/9781449324025/ch04.html
[14] Storm fault tolerance in practice
https://www.anchormen.nl/apache-storms-fault-tolerance-in-practice/
[15] Storm configuration

http://storm.apache.org/documentation/Configuration.html
[16] Google Books 1 :
http://storage.googleapis.com/books/ngrams/books/googlebooks-eng-all-1gram-20120701-a.gz
[17] Google Books 2:
http://storage.googleapis.com/books/ngrams/books/googlebooks-eng-all-1gram-20120701-b.gz