IEMS5730 Spring 2019 Homework 3

Release date: Mar 1, 2019

Due date: 11:59am, Mar 25, 2019

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 [80 marks]: Kafka Integration with Storm

In this question, you are required to integrate Kafka and Storm to implement the WordCount algorithm with processing guarantee (at-least-once mode). The dataset is available at:

http://mobitec.ie.cuhk.edu.hk/ierg4330Spring2019/homework/StormData.txt

The basic idea is to use Kafka as the pipeline to transmit the sentences from the dataset to the Storm Spout. Then you can implement a SplitSentenceBolt class to split every sentence into words and emit the word to the next WordCountBolt class, which in turn computes the frequency of each word. 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) [20 marks] Multi-node Kafka Cluster Setup.
Kafka is a distributed streaming platform used for building scalable, fault-tolerant and real-time data pipelines. In this question, you are required to set up a multi-node Kafka cluster. Kafka uses Zookeeper so you can either reuse your Homework#2 Storm cluster and then follow [7][8][9] to set up the multi-node Kafka cluster, or install the Kafka service under your Hortonworks cluster. In your homework submission, show the key steps and screenshots to verify all the processes are running.

- You are required to run 2 brokers and each broker with 2 partitions[10] in your Kafka cluster.
- After installing the multi-node Kafka cluster, you need to create a Topic and transmit the message "my test message" from the *kafka-console-consumer* to *kafka-console-producer*. Following commands may be useful:

```
$ bin/kafka-topics.sh --create --zookeeper localhost:2181
--replication-factor 2 --partitions 2 --topic my-test-topic
                              //create a topic
$ bin/kafka-console-producer.sh --broker-list localhost:9092
--topic my-test-topic
                              //publish some messages to our topic
$ bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic
my-test-topic --from-beginning
                              //consume some messages of our topic
```

(b) [20 marks] Find the most Frequently used Words: Using Kafka to Guarantee Message/Tuple Processing (At-least-once mode)
Change your Homework#2 program, so that the topology can guarantee to process every message (i.e., at least once model[12][13][14]) with the help of Kafka. The basic idea is to use Kafka as the pipeline to transmit the sentences from the dataset to the Storm Spout. The Storm Spout (KafkaSpout[6] class), worked as the Kafka Consumer, has been implemented

by Apache Storm. It has well-implemented *ack()* and *fail()* functions and is able to support the replay of failed tuples.

- You are required to use KafkaSpout[6] to support the replay of failed tuples.
- You are required to extend BaseRichBolt class to implement your SplitSentenceBolt and WordCountBolt class. In particular, your Bolts should perform proper tuple-anchoring and acknowledgement, i.e., do tuple anchoring via *collector.emit(tuple, new Values(word))* and manually call *collector.ack(tuple)* after the tuple is processed.
- You are required to manually fail the delivery of the word "*the*" for 10 times during a run. 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. Compare the result of Homework#2. Do they generate the same result? Why or why not? Compare the performance of these two models (including running time, num of tuple emitted, number of tuples acked, etc.). Submit the code and the results. Explain your findings.

(c) [20 marks] Scaling the topology.
Refer to Page 41 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 (b) and compare the performance (e.g., running time, num of tuple emitted, number of tuples acked, etc.) under the following 4 different settings. Explain your findings.

	# of KafkaSpout	# of SplitSentenceBolt	# of WordCountBolt
Setting 1	3	3	3
Setting 2	1	3	3
Setting 3	1	1	3
Setting 4	3	10	10

(d) [20 marks] (Bonus for IEMS5730) Find popular hashtags: Using Kafka to Guarantee Message/Tuple Processing (At-least-once model)
Design and implement an algorithm to determine the list of popular hashtags among tweets under the topic of "Trump". You need to extend your codes in Homework#2 Q4 to support the at-least-once tuple-processing model.

- Use the TwitterStream API to continuously ingest live tweets containing the keywords "Trump", "trump" and "TRUMP " in the Kafka Consumer.
- Use Kafka as the pipeline to transmit the tweets from the Kafka Consumer to the KafkaSpout.

- Extend BaseRichBolt class to implement your Bolts. In particular, your Bolts should perform proper tuple-anchoring and acknowledgement.
- Every 10 minutes, you need to identify and report (print out) all the popular hashtags and their corresponding frequencies. For this homework, a hashtag is said to be popular if it appears in more than one percent of all the tweets received since the last report (printout).
- Furthermore, same as in Q1(b), the corresponding bolt should fail the delivery of 10% of the tuples which carry the hashtag "Trump". Record the performance (e.g., running time, num of tuple emitted, number of tuples acked, etc.) of your program.

IMPORTANT Hints:

- You are recommended to use the t2.large instance type for your VMs. Each instance has 2 CPU cores and 8GB memory.
- You are recommended to use a stable version of Storm, e.g., Storm-1.2.x.
- You are allowed to post-process the result dumped from Storm.
- You are **NOT** allowed to use Trident.
- You are recommended to use Maven [11] to compile and build the executable Jar.
- To facilitate debugging, you can find logs under Storm_HOME/logs/directory upon any errors.
- 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.
- Copy your java code to the following directory:
  Storm_HOME/examples/storm-starter/src/jvm/storm/starter/
  and then use Maven to compile.

Q2 [40 marks]: PageRank Algorithm on GraphLab Create

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 TAB.

(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 machine. In your homework submission, show the key steps.

(b) [20 marks] Ref [16] provides one working example of the PageRank algorithm. Read the code, understand how it works and write your own PageRank program or modify the code in [16] (if necessary) to compute the Top 100 nodes for the given dataset. Submit the code and the result.

- To get started, you can first use Python on the machine with Graphlab/Graph-Create installed to try out the examples in [17, 18].
- Do NOT use the existing PageRank toolkit provided by GraphLab/ Graph-Create in Ref [5].

(c) [10 marks] Change the threshold for convergence and maximum iterations in your code and run at least three times with different parameters. Compare your results of your different runs by listing the corresponding top 100 nodes, number of iterations and time-to-convergence/completion. Explain your observations.

Reference:

[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/
[4] Installation Guide for GraphLab Create
https://turi.com/download/install-graphlab-create.html
[5] GraphLab Create PageRank Toolkit
https://turi.com/products/create/docs/graphlab.toolkits.graph_analytics.html#pagerank
[6] KafkaSpout
https://github.com/apache/storm/blob/master/external/storm-kafka-client/src/main/java/org/apache/storm/kafka/spout/KafkaSpout.java
[7] Kafka installation
https://kafka.apache.org/quickstart
[8] twitter4j library
http://twitter4j.org/javadoc/twitter4j/TwitterStream.html#filter-java.lang.String...-
[9] Kafka Storm example
https://github.com/liumingming123/WordCount-real-time
[10] Set the number of partition in Kafka
http://kafka.apache.org/documentation.html#brokerconfigs
[11] Guaranteeing-message-processing
http://storm.apache.org/releases/1.0.6/Guaranteeing-message-processing.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
http://storm.apache.org/releases/current/Fault-tolerance.html
[15] Storm configuration
http://storm.apache.org/documentation/Configuration.html
[16] A sample Pagerank code in Python for GraphLab Create
https://github.com/turi-code/how-to/blob/master/triple_apply_weighted_pagerank.py
[17] Tutorials and How-To code samples for working with GraphLab Create:
https://github.com/turi-code
[18] GraphLab Create User Guide
https://turi.com/learn/userguide/
[19] Storm Kafka Integration
http://storm.apache.org/releases/2.0.0-SNAPSHOT/storm-kafka.html
[20] Hortonworks Storm + Kafka
https://hortonworks.com/blog/storm-kafka-together-real-time-data-refinery/