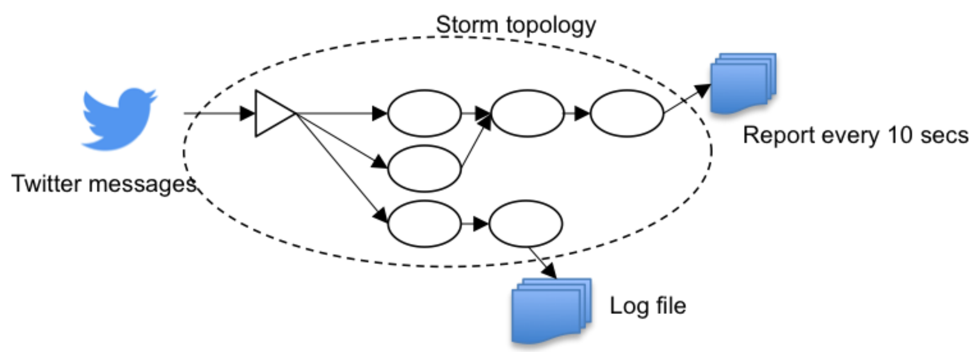
Programming Assignment 2

Detecting the Most Popular Topics from Live Twitter Message Streams using the Lossy Counting Algorithm with Apache Storm

Due: March, 27 Monday 5:00PM
Submission: via Canvas, group submission
Instructor: Sangmi Lee Pallickara

Objectives
The goal of this programming assignment is to enable you to gain experience in:
- Implementing approximate on-line algorithms using a real-time streaming data processing framework
- Understanding and implementing parallelism over a real-time streaming data processing framework



1. Overview

In this assignment, you will design and implement a real-time streaming data analytics system using Apache Storm. The goal of your system is to detect the most frequently occurring hash tags from the live Twitter data stream in real-time.

A hashtag is a type of label or metadata tag used in social networks that makes it easier for users to find messages with a specific theme or content[1]. Users create and use hashtags by placing the hash character (or number sign) # in front of a word or un-spaced phrase. Searching for that hashtag will then present each message that has been tagged with it. For example, `#springbreak` and `#zidane` were popular tags for the US on March 11, 2019.

Finding popular and trendy topics (via hashtags and named entities) in real-time marketing implies that you include something topical in your social media posts to help increase the overall reach. In this assignment, we will target data from live Twitter message provided by Twitter developers[2].

In this assignment, you will:
- Implement the Lossy Counting algorithm[3]

[1] https://en.wikipedia.org/wiki/Hashtag
[2] https://dev.twitter.com
[3] Gurmeet Singh Manku, and Rajeev Motwani, “Approximate Frequency Counts over Data Stream” 2002, VLDB

- List the top 100 most popular hashtags every 10 seconds
- Parallelize the analysis of your system

To perform above tasks, you are required to use Apache Storm, and Twitter Stream APIs.

2. Requirements of Programing Assignment 2

Your submission should include **the source codes** of two storm topologies specified in the section 5. Do not submit any data or result file.

To count the occurrences of hashtags, you should use the online algorithms included in this description. You are not allowed to use any existing lossy counting algorithm implementations.

Demonstration of your software should be on machines in CSB-120. This will include an interview discussing implementation and design details. Your submission should be via Canvas.

3. Install and setup your Storm cluster

You should create your own Storm cluster in CS120 with at least 5 nodes including the Nimbus node for this assignment. Here is the summary of the steps for setting up a Storm cluster:

a. Add paths to .bashrc
b. Run setup_storm_zoo.sh script

Launch daemons under supervision using "storm" script and a supervisor of your choice

3.1. Add paths to .bashrc

Add the following paths to the bottom of your .bashrc file, which is in your home directory.

```
module purge
module load courses/cs535/pa2

export ZOOCFGDIR=$HOME/zookeeperConf
export ZOOPIDFILE=/s/$HOSTNAME/a/tmp/zookeeper_$USER/pid
export ZOO_LOG_DIR=/s/$HOSTNAME/a/tmp/zookeeper_$USER/logs
export ZOO_DATADIR=/s/$HOSTNAME/a/tmp/zookeeper_$USER/data
export PATH=$HOME/.local/bin:/usr/local/python-env/py39/bin:$PATH
export PYTHONPATH="$HOME/.local/lib/python3.9/site-packages:/usr/local/python-env/py39/lib/python3.9/site-packages"
```

Now run source .bashrc (you only have to do this once after making changes to the .bashrc file).

```
$ source .bashrc
```

3.2. Run setup_storm_zoo.sh script

Download the setup_storm_zoo.sh from the PA2 page on canvas and run it in your home directory.

```
$ bash setup_storm_zoo.sh
```

Follow the outputted instructions from the setup_storm_zoo.sh script for starting and stopping your cluster. The outputted instructions also show how to view the storm UI and how to run storm jobs using the example jobs in the stormExamples folder which will be downloaded into your home folder.

3.3. Launch your Storm job

To compile your file and launch your job, using Maven (maven is installed in CS120 cluster) is a good option.
To find examples, go to `~/stormExamples/storm-starters` and follow the `README.markdown` in the same directory. You can start from the section, "## Build and install Storm jars locally".

Once you have created your local Storm jar that also includes your topology, you can also use following command to launch your job.

```
storm jar target/storm-starter-1.1.1.jar
org.apache.storm.starter.RollingTopWords production-topology remote
```

To kill a topology, simply run,

```
storm kill {stormname}
```

You should use the name that you have used when submitting the topology. For more information, please visit;
https://storm.apache.org/documentation/Running-topologies-on-a-production-cluster.html

3.4. Reading Live Twitter messages from your Storm Spout

1. Getting your application keys for OAuth

Create your Twitter account and log in. Next, go to `https://dev.twitter.com/apps` and create your application. On the application page, click "Keys and Access Tokens" tag and create your access token. You should use four keys (Consumer Key (API key), Consumer Secret (API secret), Access Token, and Access Token Secret) from this web page.

2. Downloading Twitter4J[4]

Twitter4J is a Java library for the Twitter API. You can use pure HTTP GET to retrieve messages. However, Twitter4J will provide the simplest access to the Twitter messages from your Storm spout.

3. Using Twitter4J from your Storm Spout

```
TwitterStream twitterStream = new TwitterStreamFactory(
        new ConfigurationBuilder().setJSONStoreEnabled(true).build())
        .getInstance();

twitterStream.addListener(listener);
twitterStream.setOAuthConsumer(consumerKey, consumerSecret);
AccessToken token =
        new AccessToken(accessToken, accessTokenSecret);
twitterStream.setOAuthAccessToken(token);
```

4. Lossy Counting Algorithm

[4] http://twitter4j.org/en/

4.1. Definitions

The incoming stream is conceptually divided into *buckets* of width $w = \left\lceil \frac{1}{\varepsilon} \right\rceil$ transactions each.

Buckets are labeled with *bucket ids*, starting from 1. We denote the current bucket id by $b_{current}$, whose value is $\left\lceil \frac{N}{w} \right\rceil$. For an element e, we denote its true frequency in the stream seen so far by f_e.

Note that ε and w are fixed while N, $b_{current}$ and f_e are running variables whose values change as the stream progresses.

Data structure D is a set of entries of the form (e, f, Δ), where e is an element in the stream, f is an integer representing its estimated frequency, and Δ is a maximum possible error in f.

4.2. Algorithm

Initially D is empty. Whenever an element arrives, first lookup to see whether an entry for that element already exists or not. If the lookup succeeds, update the entry by incrementing its frequency f by one. Otherwise, create a new entry of the form $(e, 1, b_{current} - 1)$. We also prune D by deleting some of its entries at bucket boundaries, i.e., whenever $N \bmod w == 0$. The rule for deletion is: an entry (e, f, Δ) is deleted if $f + \Delta \leq b_{current}$. When a user requests a list of items with threshold s, we output those entries in S where $f \geq (s - \varepsilon)N$.

For an entry (e, f, Δ), f represents the exact frequency count of e ever since this entry was inserted into D. The value of Δ assigned to a new entry is the maximum number of times e could have occurred in the first $b_{current} - 1$ buckets. This value is exactly $b_{current} - 1$. Once an entry is inserted into D, its Δ value remains unchanged.

4.3. Example[5]

```
(The parameter s is not used until the end of the algorithm)
  ε = 0.2

  w = 1/ε= 5      (5 items per "bucket")


 Input: 1 2 4 3 4 3 4 5 4 6 7 3 3 6 1 1 3 2 4 7
        |       | |       | |       | |       |
        +-------+ +-------+ +-------+ +-------+
         bucket 1  bucket 2  bucket 3  bucket 4


=====================================
bcurrent = 1    inserted: 1 2 4 3 4
-------------------------------------
Insert phase:
  D (before removing):(x=1;f=1;Δ=0) (x=2;f=1;Δ=0)(x=4;f=2;Δ=0)(x=3;f=1;Δ=0)

Delete phase: delete elements with f + Δ ≤ bcurrent (=1)
  D (after removing) :(x=4;f=2;Δ=0)

NOTE: elements with frequencies ≤ 1 are deleted
      New elements added has maximum count error of 0

======================================
```

[5] http://www.mathcs.emory.edu/~cheung/Courses/584-StreamDB/Syllabus/07-Heavy/Manku.html

```
bcurrent = 2    inserted: 3 4 5 4 6
-------------------------------------
Insert phase:
  D (before removing):(x=4;f=4;Δ=0) (x=3;f=1;Δ=1)(x=5;f=1;Δ=1)(x=6;f=1;Δ=1)

Delete phase: delete elements with f + Δ ≤ bcurrent (=2)
  D (after removing) :(x=4;f=4;Δ=0)

       NOTE: elements with frequencies ≤ 2 are deleted
      Newly added elements have a maximum count error of 1

=====================================
bcurrent = 3    inserted: 7 3 3 6 1
-------------------------------------
Insert phase:
  D (before removing):(x=4;f=4;Δ=0) (x=7;f=1;Δ=2) (x=3;f=2;Δ=2)(x=6;f=1;Δ=2)
(x=1;f=1;Δ=2)

Delete phase: delete elements with f + Δ ≤ bcurrent (=3)
  D (after removing) :(x=4;f=4;Δ=0) (x=3;f=2;Δ=2)

NOTE: elements with frequencies ≤ 3 are deleted
      Newly added elements have a maximum count error of 2

=====================================
bcurrent = 4    inserted: 1 3 2 4 7
-------------------------------------
Insert phase:
  D (before removing):(x=4;f=5;Δ=0) (x=3;f=3;Δ=2) (x=1;f=1;Δ=3)(x=2;f=1;Δ=3)
(x=7;f=1;Δ=3)

Delete phase: delete elements with f + Δ ≤ bcurrent (=4)
  D (after removing) :(x=4;f=5;Δ=0) (x=3;f=3;Δ=2)

NOTE: elements with frequencies ≤ 4 are deleted
      Newly added elements have a maximum count error of 3


Interpreting the content in D:

    Item  fmanku     factual
   ------------------------
      4     5         5
      3     3         5
```

5. Software Requirements

For this assignment, you should implement 2 topologies: (1) Non-parallel and (2) parallel topologies.

5.1. Non-parallel topology

For a non-parallel topology, your topology should:

- Retrieve data from the Twitter stream API in real-time from a single spout
- Provide a report of the top 100 most popular hashtags (sorted in descending order) every 10 seconds. If your software does not have 100 hashtags at the end of the 10 second interval, your report may contain hash tags those are available at that time.
- Here, the time stamp is the time stamp at the beginning of a **10-second window**.
- Generate a log file containing the top 100 hash tags and timestamps. Each log entry should have the following format:

```
Log file
<timestamp_in_epochtime><hashtagTop1><hashtagTop2><hashtagTop3>…<hashtagTop1
00>
```

Each of the messages should generate no more than 1 line of log entry. If there is no hashtag included in the message, please do not create a log entry. The bolts for counting and logging should be implemented as two different bolts.

5.2. Parallel topology

The parallel topology tracks the counts in parallel. Design and implement your topology including 4 different bolts that implement the lossy counting algorithm. These bolts should run on different nodes.

Your topology should,

- Retrieve data using the Twitter stream API in real-time from a single spout
- Perform counting without redundancy using the lossy counting algorithm on 4 bolts on different nodes
- Aggregate values and provide the top 100 most popular hashtags (sorted in descending order) every 10 seconds
- Generate log files that follow the format specified in section 6.1.

6. Grading

Your submissions will be graded based on the demonstration to the instructor in CSB120 or online. The grading will be done on a 15-point scale. This assignment will account for 15% of your final course grade.

- 3 points: Set up the Storm cluster and other software
- 5 points: Implementation of the Non-Parallel topology and analysis
- 6 points: Implementation of the Parallel topology
- 1 point: Participation (This will be conducted at the end of the semester.)

7. Late Policy

Please check the late policy posted on the course web page.