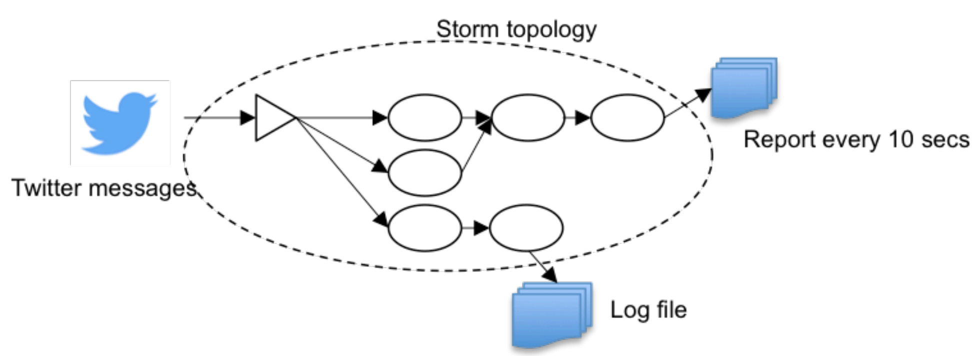
Programming Assignment 2

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

Due: November, 6 Monday 5:00PM
Submission: via Canvas, individual submission
Instructor: Sangmi Lee Pallickara
Web page: http://www.cs.colostate.edu/~cs535/Assignments.html

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
- Applying sentiment analysis over the near real-time streaming data
- 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 and named-entities from the live Twitter data stream and calculate sentiment values 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

[1] https://en.wikipedia.org/wiki/Hashtag

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, ***#vegasstrong, #napafire,*** and ***#colombusday*** were popular tags for the US on October 9, 2017.

Named entities are a real-world objects such as persons, locations, organizations, products, etc. Examples of named entities include, Barak Obama, Colorado, Toyota, or anything that can be named. Named entity recognition (NER) is a process to seek a location and classify named entities in text.

Sentiment analysis refers to the use of Natural Language Processing (NLP), to identify and extract subjective information in source materials. Sentiment analysis is widely applied in reviews and social media for variety of applications such as placing advertisements, marketing, and customer service.

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. Also, associating sentiment information with topics will allow us to use the media more precisely. 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]
- List the top 100 most popular hashtags and the top 100 most popular named-entities with their sentiment analysis every 10 seconds
- Parallelize the analysis of your system

To perform above tasks, you are required to use Apache Storm, Twitter Stream APIs, and Stanford's NLP core 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.**

[2] https://dev.twitter.com

[3] Gurmeet Singh Manku, and Rajeev Motwani, "Approximate Frequency Counts over Data Stream" 2002, VLDB

To identify named-entities and calculate individual sentiment analysis, you should use Stanford's NLP APIs. **Do not use any other APIs or implement it.**

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[4]:

a. Set up a Zookeeper cluster
b. Install dependencies on Nimbus and worker machines
c. Download and extract a Storm release to Nimbus and worker machines
d. Fill in mandatory configurations into `storm.yaml`

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

3.1. Set up your Zookeeper cluster

Storm uses Zookeeper for coordinating the cluster. Zookeeper is not used for message passing, so the load Storm places on Zookeeper is quite low. Single node Zookeeper clusters should be sufficient for most cases.

First, you should download a release of zookeeper:

```
$ wget http://apache.claz.org/zookeeper/zookeeper-3.4.10/zookeeper-
3.4.10.tar.gz
$ tar xvf zookeeper-3.4.10.tar.gz
$ cd zookeeper-3.4.10/conf
```

Now, in your /`conf` directory, create a file "`zoo.cfg`" with the following lines. This configuration does NOT process any environment variables such as `$HOSTNAME` or `$USER`. Please use an absolute path for this configuration.

```
tickTime=2000
dataDir=/s/*your_host_name*/a/tmp/zookeeper_*your_login_name*/data
clientPort=2181
```

To run zookeeper, go to your zookeeper directory and run following command:

```
$ bin/zkServer.sh start
```

To check the status of your zookeeper instance, go to your zookeeper directory and run following command:

[4] https://storm.apache.org/documentation/Setting-up-a-Storm-cluster.html

```
$ bin/zkServer.sh status
```

To stop your zookeeper instance, go to your zookeeper directory and run following command:

```
$ bin/zkServer.sh stop
```

You can also run zookeeper under supervision. See the section 3.3. By default, you will see the log file in `your_zookeeper_directory/bin/zookeeper.out.`

To avoid port conflict, please follow the port/server assignment posted on the assignment web page.

3.2. Set up a Storm cluster

Next, download a Storm release and extract the zip file somewhere on Nimbus and each of the worker machines.

```
$ mkdir storm
$ cd storm
$ wget http://apache.claz.org/storm/apache-storm-1.1.1/apache-storm-1.1.1.tar.gz
$ tar xvf apache-storm-1.1.1.tar.gz
$ cd storm-1.1.1
```

The Storm release contains a file at `conf/storm.yaml` that configures the Storm daemons. You can see the default configuration values https://github.com/apache/storm/blob/v1.1.1/conf/defaults.yaml. `storm.yaml` overrides anything in `defaults.yaml`. The mandatory configuration to get a working cluster includes:

1) `storm.zookeeper.servers`: This is a list of the hosts in the Zookeeper cluster for your Storm cluster. It should look something like:

```
    storm.zookeeper.servers:
      - "your_zookeeper_node.cs.colostate.edu"
```

If the port that your Zookeeper cluster uses is different than the default, you should set `storm.zookeeper.port` as well.

2) `storm.local.dir`: The Nimbus and Supervisor daemons require a directory on the local disk to store small amounts of state (like jars, confs, and things like that). You should create that directory on each machine, give it proper permissions, and then fill in the directory location using this configuration keyword. For example:

```
storm.local.dir: "/tmp/storm-${USER}"
```

3) `nimbus.seeds`: The worker nodes need to know which machine the master is running on, in order to download topology jars and configurations. Specify the master node:

```
nimbus.seeds: ["your_nimbus_node.cs.colostate.edu"]
```

4) `supervisor.slots.ports`: For each worker machine, you can configure how many workers run on that machine with this configuration keyword. Each worker uses a single port for receiving messages, and this setting defines which ports are open for use. If you define five ports here, then Storm will allocate up to five workers to run on this machine. If you define three ports, Storm will only run up to three. By default, this setting is configured to run 4 workers on the ports 6700, 6701, 6702, and 6703. You can specify them with the ports in your port range. For example:

```
supervisor.slots.ports:
    - your_port_number_1
    - your_port_number_2
    - your_port_number_3
    - your_port_number_4
```

To avoid port conflict, please follow the port assignment posted on the assignment web page. To specify the port number for your UI, please see the default configuration file: https://github.com/apache/storm/blob/master/conf/defaults.yaml

3.3. Launch daemons under supervision

Next step is to launch all the Storm daemons. It is critical that you run each of these daemons under supervision. Storm is a fail-fast system, which means the processes will halt whenever an unexpected error is encountered. Storm is designed so that it can safely halt at any point and recover correctly when the process is restarted. This is why Storm keeps no state in-process -- if Nimbus or the Supervisors restart, the running topologies are unaffected. First, go to your storm directory and copy supervisor configuration. Supervisor is already installed in the CS120 cluster.

Go to your storm installation folder and create three configuration files.

```
more /etc/supervisord.conf > zk-supervisord.conf
more /etc/supervisord.conf > nimbus-supervisord.conf
more /etc/supervisord.conf > worker-supervisord.conf
```

Now, add zookeeper commands to your configuration file, `zk-supervisord.conf`.

```
** Add following lines
[program:zookeeper]
command={your_zookeeper_directory}/bin/zkServer.sh start-foreground

**Modify following lines
[unix_http_server]
file=/tmp/{Your-User-ID}-storm/supervisor.sock   ; (the path to the socket
file)

[supervisord]
logfile=/tmp/{Your-User-ID}-storm/supervisord.log  ; (main log file;default
$CWD/supervisord.log)

[supervisorctl]
serverurl=unix:///tmp/{Your-User-ID}-storm/supervisor.sock ; use a unix://
URL  for a unix socket
```

Add nimbus commands to your configuration file, `nimbus-supervisord.conf`.

```
** Add following lines
[program:storm_ui]
command={your_storm_directory}/bin/storm ui

[program:storm_nimbus]
command={your_storm_directory}/bin/storm nimbus


**Modify following lines
[unix_http_server]
file=/tmp/{Your-User-ID}-storm/supervisor.sock   ; (the path to the socket
file)

[supervisord]
logfile=/tmp/{Your-User-ID}-storm/supervisord.log  ; (main log file;default
$CWD/supervisord.log)

[supervisorctl]
serverurl=unix:///tmp/{Your-User-ID}-storm/supervisor.sock ; use a unix://
URL  for a unix socket
```

Add command for the worker nodes to your configuration file, `worker-supervisord.conf`.

```
** Add following lines
[program:storm_supervisor]
command={your_storm_directory}/bin/storm supervisor

**Modify following lines
[unix_http_server]
file=/tmp/{Your-User-ID}-storm/supervisor.sock   ; (the path to the socket
file)
```

```
[supervisord]
logfile=/tmp/{Your-User-ID}-storm/supervisord.log  ; (main log file;default
$CWD/supervisord.log)

[supervisorctl]
serverurl=unix:///tmp/{Your-User-ID}-storm/supervisor.sock ; use a unix://
URL  for a unix socket
```

So now that we have supervisor configuration to manage all the necessary processes to start storm cluster. On your zookeeper node, start your zookeeper cluster. First create directory for your log files and socket.

```
mkdir /tmp/{user_ID}-storm
supervisord -c {your_storm_directory}/zk-supervisord.conf
```

On your nimbus node,

```
mkdir /tmp/{user_ID}-storm
supervisord -c {your_storm_directory}/nimbus-supervisord.conf
```

On your worker nodes, start supervisors.

```
supervisord -c {your_storm_directory}/worker-supervisord.conf
```

The UI can be accessed by navigating your web browser to `http://{nimbus host}:{your-UI-port}`.

3.4. 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 `/storm_directory/examples/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

3.5. 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[5]

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. Using Stanford's NLP Core APIs

Once you get the real-time Twitter streaming data, use *Stanford CoreNLP* to perform nature language process over your Twitter data.

4.1. Downloading and Installation

Stanford CoreNLP 3.8.0 is available at its official website:
http://stanfordnlp.github.io/CoreNLP

Download and unzip the software:

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

```
$ mkdir stanford-nlp
$ cd stanford-nlp
$ wget http://nlp.stanford.edu/software/stanford-corenlp-full-2015-12-09.zip
$ unzip stanford-corenlp-full-2015-12-09.zip
```

This will download a zip file containing:
(1) The CoreNLP code jar
(2) The CoreNLP models jar (required in your `classpath` for most tasks)
(3) The lib files required to run CoreNLP
(4) documentation/source code for the project.

First, modify **`stanford-corenlp-3.6.0-models.jar`** to contain only required functionalities. This will conserve your disk space requirement by up to 75%.

To modify your jar file, unjar your `stanford-corenlp-3.6.0-models.jar` and delete the following directories:

```
## Directories to be deleted
-   edu/stanford/nlp/models/dcoref
-   edu/stanford/nlp/models/gender
-   edu/stanford/nlp/models/hcoref
-   edu/stanford/nlp/models/naturalli
-   edu/stanford/nlp/models/parser
-   edu/stanford/nlp/models/regexner
-   edu/stanford/nlp/models/scoref
-   edu/stanford/nlp/models/supervised_relation_extractor
-   edu/stanford/nlp/models/truecase
-   edu/stanford/nlp/models/ud
-   edu/stanford/nlp/models/upos
```

Finally, jar your directory using the original jar file name, `stanford-corenlp-3.6.0-models.jar.`

The modified jar file should be around 81.5MB (Less than 25% of the original size). Add it to the project classpath in order to run Stanford CoreNLP. Note that you will not need all the functions from the library. To conserve your disk space include only the following jar files in your classpath:

```
## the CoreNLP jar
-   stanford-corenlp-3.6.0.jar
## the libraries required to run CoreNLP
-   ejml-0.23.jar
-   javax.json.jar
-   joda-time.jar
-   jollyday.jar
-   protobuf.jar
-   slf4j-api.jar
-   slf4j-simple.jar
-   xom.jar
## the CoreNLP models jar
-   stanford-corenlp-3.6.0-models.jar
```

Or you can simply use Maven to add this dependency, which is easier and more convenient:

```
<dependency>
    <groupId>edu.stanford.nlp</groupId>
    <artifactId>stanford-corenlp</artifactId>
    <version>3.6.0</version>
</dependency>
```

4.2. Preparation: Language Detection

For the purpose of simplicity, the assignment will only ask you to process tweets in English. You can use the external library to achieve this. For example, language-detector is a feasible tool. The maven dependency is:

```
<dependency>
    <groupId>com.optimaize.languagedetector</groupId>
    <artifactId>language-detector</artifactId>
    <version>0.5</version>
</dependency>
```

Here is the example code for language detection:

```
try {
    List<LanguageProfile> languageProfiles = new
LanguageProfileReader().readAllBuiltIn();
    LanguageDetector languageDetector =
LanguageDetectorBuilder.create(NgramExtractors.standard())
            .withProfiles(languageProfiles).build();
    TextObjectFactory textObjectFactory =
CommonTextObjectFactories.forDetectingOnLargeText();

    String textA = "Hello World";
    TextObject textObject = textObjectFactory.forText(text);
    Optional<LdLocale> lang = languageDetector.detect(textObject);
    if(lang.isPresent()){
        System.out.println(lang.get().toString());
    } else{
        System.out.println("No language matched!");
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

Please visit https://github.com/optimaize/language-detector for more information.

4.3. Identifying Named-Entities

The first thing to do is extracting the named-entities from tweets. Stanford CoreNLP provides APIs to do named-entities recognition. The module is called **Stanford Named Entity Recognizer (NER)**. Visit http://nlp.stanford.edu/software/CRF-NER.shtml for further information.

Here is a simple example to recognize named-entities from a given string:

```
String serializedClassifier =
"edu/stanford/nlp/models/ner/english.muc.7class.distsim.crf.ser.gz";
AbstractSequenceClassifier<CoreLabel> classifier = null;
try {
    classifier = CRFClassifier.getClassifier(serializedClassifier);
} catch (IOException | ClassNotFoundException e) {
    e.printStackTrace();
```

```
}
String text = "Good afternoon Rajat Raina, how are you today?";
System.out.println(classifier.classifyToString(text, "tabbedEntities", false))
```

4.4. Sentiment Analysis with Example

The Second thing to do is sentiment analysis on the tweets. Sentiment analysis is to used to identify and extract subjective information in source materials (Wiki). The CoreNLP also provides modules for sentiment analysis:

http://nlp.stanford.edu/sentiment/index.html

Here is the example code for evaluating sentiment of a given text.

```
Properties properties = new Properties();
properties.setProperty("annotators", "tokenize, ssplit, pos, parse, sentiment");
StanfordCoreNLP pipeline = new StanfordCoreNLP(properties);
String text = "Good afternoon Rajat Raina, how are you today? I am very happy to see
you.";
Annotation document = new Annotation(text);
pipeline.annotate(document);
List<CoreMap> sentences = document.get(SentencesAnnotation.class);
for (CoreMap sentence : sentences) {
    String sentiment = sentence.get(SentimentCoreAnnotations.SentimentClass.class);
    System.out.println(sentiment);
}
```

The output would be:

```
Neutral
Very Positive
```

This suggests that first sentence "`Good afternoon Rajat Raina, how are you today?`" conveys neutral sentiment, while the second one "`I am very happy to see you.`" expresses a very positive sentiment.

5. Lossy Counting Algorithm

5.1. 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.

5.2. 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.

5.3. Example[6]

```
(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


=====================================
b_current = 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 ≤ b_current (=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

======================================
b_current = 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 ≤ b_current (=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
```

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

```
=======================================
b_current = 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 ≤ b_current (=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

=======================================
b_current = 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 ≤ b_current (=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  f_manku    f_actual
   ------------------------
      4      5          5
      3      3          5
```

6. Sentiment Analysis

Once you have retrieved hashtags and named-entities, the sentiment values should be calculated from the **twitter message** (not the hashtag). The NLP Core APIs provides 5 levels of sentiment values:

– –	–	0	+	++

The ++ sign represents the highest positive sentiment value for the given text. The – – sign represents the lowest sentiment values for the text. Value 0 indicates that this text contains neutral information.

You should aggregate the sentiment analysis results across the twitter messages. First, assign values of -2, -1, 0, 1, and 2 to each category.

– –	–	0	+	++
−2	−1	0	1	2

Second, the sentiment analysis results from the multiple twitter messages sharing common hashtag must be averaged out. If there are twitter messages that contain more than one hash tags, the sentiment analysis result of the message will contribute to all of hash tags with the complete value. If there are 3 hash tags and the sentiment analysis result was +, all of the 3 hash tags will get the sentiment value, 1, from this message.

7. Parallelizing Your Topology

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

7.1. 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
- Perform counting, sentiment analysis, and identifying named-entity without redundancy using the lossy counting algorithm
- Provide a report of the top 100 most popular hashtags (sorted in descending order) and 100 most popular named-entities with the sentiment analysis value every 10 seconds.
- Here, the time stamp is the time stamp at the beginning of a 10-second window.
- Generate two log files containing the top 100 hash tags and top 100 named entities with the sentiment analysis results and timestamps. Each log entry should have the following format:

```
Log file 1
<timestamp><hashtag1:Sentiment-Value><hashtag2:Sentiment-Value>
<hashtag3:Sentiment-Value>…

Log file 2
<timestamp><named-ent1:Sentiment-Value><named-ent2:Sentiment-Value><named-
ent3:Sentiment-Value>…
```

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

7.2. 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
- Aggregate values and provide the top 100 most popular named-entities (sorted in descending order) every 10 seconds
- Generate log files that follow the format specified in section 6.1.

8. Grading

Your submissions will be graded based on the demonstration to the instructor in CSB120. The grading will be done on a 15-point scale. This assignment will account for 20% of your final course grade.

- 3 points: Set up the Storm cluster and other software
- 6 points: Implementation of the Non-Parallel topology and analysis
- 6 points: Implementation of the Parallel topology

9. Late Policy

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