Lab 6 – Kafka vs. Twitter

CC5212-1 – April 28, 2021

We will detect earthquakes with Twitter and Kafka. For this, we will work with Twitter data from September 19th, 2017; specifically we have a 1% sample of Twitter data with a total of 4,905,393 (re)tweets.[1]

On the server, the file `/data/uhadoop/shared/twitter/tweets_20170919.tsv.gz` contains information about tweets, where the columns are as follows: (1) datetime retrieved, (2) datetime written, (3) id of tweet, (4) user id, (5) tweet type, (6) language detected, (7) tweet text, (8) times retweeted. Some of the tweets are retweets, where (2) and (3) refer to the original tweet; hence you may see dates in (2) that are older than the selected date. To peek, use `zcat /data/uhadoop/shared/twitter/tweets_20170919.tsv.gz | more`.

From u-cursos, download the project `mdp-lab06.zip`. Here you will find some example code to get you started with Kafka. A `build.xml` script is provided to help you build a jar (as in the previous labs).

- Have a look at `KafkaExample`, which offers a Hello-World-style example for Kafka. Let's give it a try! Build the project and copy the jar over to the server. Run `java -jar mdp-kafka.jar KafkaExample`. Ah, but it asks for an argument: a *topic* on Kafka. What should we put? Let's create a new topic; note that the following is all one command.

  – `kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic GROUPNAME-example`

  where GROUPNAME is any unique name for your group. Note that here we can set the replication factor and number of partitions. In the lab we will always work with 1 to keep things a bit simpler, but if you set these higher, you can use more machines. By the way, if you want to see the active topics:

  – `kafka-topics.sh --list --zookeeper localhost:2181`

  Ok, now we are ready to run our example again, this time with a topic:

  – `java -jar mdp-kafka.jar KafkaExample GROUPNAME-example`

  Not the most exciting example ever, but hey, it's a start! Okay, let's delete that topic we don't need any more:

  – `kafka-topics.sh --delete --zookeeper localhost:2181 --topic GROUPNAME-example`

- Okay, next we will run a Kafka producer to simulate a stream of tweets from our file. The code is already done. First, create your own topic for tweets called GROUPNAME`-tweets` as before (you can set replication and partitions to 1 again). Now run (and have a look at the source code while you wait):

  – `java -jar mdp-kafka.jar TwitterSimulator /data/uhadoop/shared/twitter/tweets_20170919.tsv.gz GROUPNAME-tweets 1000`

  It will take a minute or two to finish though not much of interest seems to happen. But what is happening is that the code is writing Tweets to your Kafka topic. Unfortunately nothing is subscribed to that topic yet. Note the last argument: `1000`. This is the speed-up. We want to process the tweets of an entire day but don't want to wait an entire day, so 1000 says speed up time by a factor of 1000

[1] Thanks to Hernán Sarmiento for collecting/providing the data!

(one second becomes one millisecond). How long will it take to run a day? Also note that we have a 1% sample of Twitter, so if we select `100`, we are simulating real-time Twitter throughput. Selecting `1000` then means we are 10 times faster than Twitter's real stream.

- So let's do something with those tweets; something earthquake related. Have a look at the code in `PrintEarthquakeTweets`. It reads from a topic (passed as an argument) and prints out any records that have an earthquake-related sub-string in them. Let's try it out:
  - `java -jar mdp-kafka.jar PrintEarthquakeTweets` GROUPNAME`-tweets`

  Anti-climax! The issue is that by default a consumer will read from the current point of the stream and the tweet stream is finished. So we have two options (you should try both).

  1. In a second terminal, run `TwitterSimulator` again while `PrintEarthquakeTweets` is waiting.
  2. Call `java -jar mdp-kafka.jar PrintEarthquakeTweets` GROUPNAME`-tweets replay`. In Kafka we can rewind streams! See how this is done in the source code when `"replay"` is given as an argument.

- Well, some output tweets don't seem so related to earthquakes, and some are about old earthquakes. If we want to detect earthquakes on Twitter, we'll need to detect a sudden *burst* of earthquake tweets at the same time. So your final task is to code your own Kafka producer and consumer to do this!

  Producer: Create a main method class called `EarthquakeFilter` based on `PrintEarthquakeTweets`, but instead of printing to standard out, add another argument that accepts an output topic, and then creates a producer to write to that topic. You can follow the examples in `KafkaExample` and `TwitterSimulator`. When ready, build the code, create a new topic for earthquake tweets in your group, and run the code. (Again it's a bit boring because nothing is there to consume the tweets yet ...)

  Consumer: Create a main method class `BurstDetector`, which uses a consumer to read from an input topic and state when there is a burst of records in that topic. We define a *burst* (x, y) in terms of there being at least x tweets in at most y seconds. We use these bursts to detect events. We define a minor burst with $x = 50$ and $y = 50$ (at least 50 tweets in at most 50 seconds) and a major burst with $x = 50$ and $y = 25$ (at least 50 tweets in at most 25 seconds). Let m_k and t_k denote the message and timestamp (in seconds) of the k^{th} tweet in the stream. We call k the start of a burst (x, y) if $t_{k-1+x} - t_k \leq y$ and $t_{k-2+x} - t_{k-1} > y$ (e.g., for a burst $(x, y) = (50, 25)$, $k = 4$ is the start of a burst if $t_{53} - t_4 \leq 25$ and $t_{52} - t_3 > 25$; note that we check, e.g., t_{53} rather than t_{54} in the first condition as t_4 is included in the burst). Each time the start of a minor/major burst is detected, print to standard out the type of burst, and the message m_k and timestamp t_k of the start of the burst. Run this over the topic to which your producer writes. What bursts did you find? Here are some tips to keep in mind while implementing the consumer:
  - You should only print out messages indicating the start of a burst, where, per the definitions, a new burst cannot start again until the rate of tweets first drops below the threshold.
  - Major and minor bursts are defined independently: the start/end of a major burst does not affect the start/end of a minor burst, for example.
  - Due to the time interval for major bursts being shorter, we can end up having (perhaps unintuitively?) more major bursts than minor bursts.
  - Optional implementation hint: Think about a FIFO queue (hint: `LinkedList` in Java) storing at most x messages, that can compare the difference in time between the oldest and newest elements against y (note that the timestamps will be in milliseconds, so you can use $1000y$).

- SUBMIT to u-cursos your `EarthquakeFilter.java` and `BurstDetector.java` solution. Also submit as `results.txt` the number of minor/major bursts detected, followed by the messages printed for the start of each major/minor burst detected (e.g., in order of timestamp).