# Lab 6 – Kafka vs. Twitter

Procesamiento Masivo de Datos

October 12, 2021

Hoy vamos a detectar terremotos con Twitter y Kafka. Para esto, trabajaremos con los datos de Twitter del 19 de septiembre de 2017; específicamente, tenemos una muestra del 1% de datos de Twitter con un total de 4,905,393 (re)tweets. [1]

En el servidor, el archivo `/data/uhadoop/shared/twitter/tweets_20170919.tsv.gz` contiene información sobre los tweets, donde las columnas son las siguientes: (1) fecha de descarga, (2) fecha del tweet, (3) ID del tweet, (4) ID del autor, (5) tipo de tweet, (6) idioma detectado, (7) texto de tweet y (8) número de veces que ha sido retwiteado. Algunos de los tweets son retweets, donde (2) y (3) se refieren al tweet original; por lo tanto, puede ver las fechas en (2) que son más antiguas que la fecha seleccionada. Para echar un vistazo, use `zcat /data/uhadoop/shared/twitter/tweets_20170919.tsv.gz | more`.

Desde u-cursos, descargue el proyecto gdd-lab10. Aquí encontrará un código de ejemplo para comenzar con Kafka. Se proporciona un script `build.xml` para ayudarlo a construir un jar (como en los laboratorios anteriores).

- Echa un vistazo a `KafkaExample`, que ofrece un ejemplo al estilo "Hello World" para Kafka. ¡Hagamos un intento! Construye el proyecto y copia el jar en el servidor. Ejecute `java -jar gdd-kafka.jar KafkaExample`. Ah, pero pide un argumento: un *topic* en Kafka. ¿Qué debemos poner? Vamos a crear un nuevo tema:

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

  donde `GROUPNAME` es cualquier nombre único para su grupo. Tenga en cuenta que aquí podemos establecer el factor de replicación y el número de particiones. En el laboratorio, siempre trabajaremos con 1 para simplificar un poco las cosas, pero si las configura más alto, puede usar más máquinas. Por cierto, si quieres ver los temas activos:

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

  Ok, ahora estamos listos para volver a ejecutar nuestro ejemplo, esta vez con un tema:

  - `java -jar gdd-kafka.jar KafkaExample GROUPNAME-example`

  No es el ejemplo más emocionante de todos, pero bueno, ¡es un comienzo! Bueno, vamos a eliminar ese tema que no necesitamos más:

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

- Bien, a continuación ejecutaremos un productor Kafka para simular un stream de tweets de nuestro archivo. El código ya está hecho. Primero, cree su propio tema para los tweets llamados `GROUPNAME-tweets` como antes (puede configurar la replicación y las particiones de nuevo en 1). Ahora ejecute (y eche un vistazo al código fuente mientras espera):

[1] ¡Gracias a Hernán Sarmiento por recopilar/proporcionar los datos!

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

  Tardará un minuto o dos en terminar, aunque no parece que pase algo muy interesante. Pero lo que está sucediendo es que el código está escribiendo Tweets para tu tópico Kafka. Lamentablemente todavía no hay nada suscrito a ese tópico. Note el último argumento: `1000`. Esta es la aceleración. Queremos procesar los tweets de un día completo, pero no queremos esperar un día entero, por lo que 1000 dice que el tiempo de aceleración es de 1000 (un segundo se convierte en un milisegundo). ¿Cuánto tiempo tomará correr un día? También tenga en cuenta que tenemos un 1% de muestra de Twitter, por lo que si seleccionamos `100`, estamos simulando el rendimiento de Twitter en tiempo real. Seleccionar `1000` significa que somos 10 veces más rápidos que el stream real de Twitter.

- Entonces hagamos algo con esos tweets; algo relacionado con el terremoto. Eche un vistazo al código en `PrintEarthquakeTweets`. Lee de un tema (pasado como un argumento) e imprime los registros que tienen una subcadena relacionada con el terremoto. [2] Bien, aguantemos la respiración y hagámoslo:

  - `java -jar gdd-kafka.jar PrintEarthquakeTweets GROUPNAME-tweets`

  Anti-climax! El problema es que, de forma predeterminada, un consumidor leerá desde el punto actual del stream y el stream de tweets habrá finalizado. Así que tenemos dos opciones (deberías probar ambas).

  1. En un segundo terminal, ejecute `TwitterSimulator` nuevamente mientras `PrintEarthquakeTweets` está esperando.
  2. Ejecutar `java -jar gdd-kafka.jar PrintEarthquakeTweets GROUPNAME-tweets replay`. En Kafka podemos rebobinar streams! Vea cómo se hace esto en el código fuente cuando `"replay"` se pasa como un argumento.

- Bueno, algunos tweets de salida no parecen estar tan relacionados con los terremotos, y algunos son sobre terremotos antiguos. Si queremos detectar terremotos en Twitter, tendremos que detectar una repentina *ráfaga* de tweets de terremotos al mismo tiempo. ¡Así que tu tarea final es codificar a tu propio productor y consumidor de Kafka para hacer esto!

  **Producer:** Crea una clase con método `main` llamada `EarthquakeFilter` basada en `PrintEarthquakeTweets`, pero en lugar de imprimir en formato estándar, agrega otro argumento que acepte un tema de salida y luego crea un productor para escribir ese tema Puede seguir los ejemplos en `KafkaExample` y `TwitterSimulator`. Cuando esté listo, cree el código, cree un nuevo tema para los tweets de terremotos en su grupo y ejecute el código. (De nuevo, es un poco aburrido porque todavía no hay nada para consumir los tweets ...)

  **Consumer:** Crea una clase con método `main` llamada `BurstDetector`, que usa un consumidor para leer desde un tema de entrada y anuncia cuando hay una ráfaga de registros en ese tema. Podemos definir una ráfaga menor como 50 tweets en 50 segundos (o menos) y una ráfaga mayor como 50 tweets en 25 segundos (o menos). ¿Cómo puedes implementar esto? No queremos leer todo el stream en la memoria. ¿Pero quizás podamos crear una cola FIFO en Java (sugerencia: `LinkedList`) que recuerde la hora y pueda comparar el elemento más antiguo y más nuevo? Imprima en la salida cuando se detecte un evento menor/mayor, junto con el registro más reciente en el stream y la hora en que se detectó; solo debe detectar cada evento una vez, donde puede considerar el evento como terminado cuando cae por debajo del nivel de ráfaga. Ejecute esto sobre el tema en el que su productor escribe. ¿Qué eventos mayores/menores encontraste?

[2] Esta detección de subcadenas podría optimizarse mucho a partir del código actual; ¿Cómo podríamos hacer eso?