Assignment 2 - Report

Yuzhe Ma & Xin Jin & Hongyi Wang

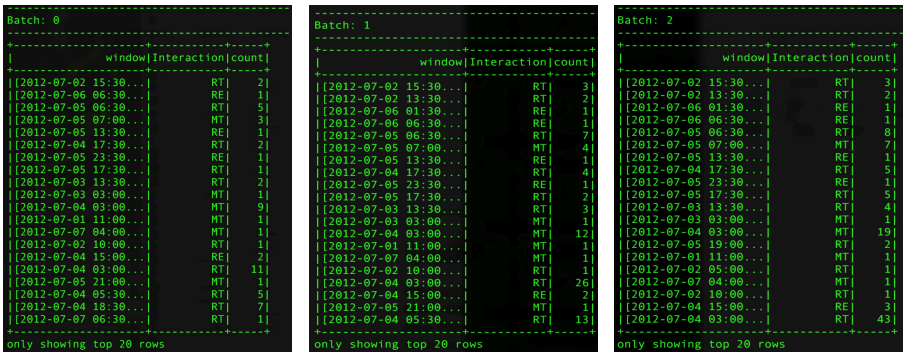
October 30, 2017

1 Part A: Structured Streaming

We use spark streaming programming model to process the twitter dataset, which contains records for the interactions between twitter users. The dataset are formed in rows, each row corresponding to an event between two users: User_A and User_B, followed by the event time and the interaction type. The interaction type can be one of the threes: retweets (RT), mention (MT) and reply (RE). For instance, if the type is MT, then it means User_A mentioned User_B at a particular time as specified in the row.

1.1 Question 1

In question 1, we wrote an application to count the number of events corresponding to each type of interaction, and output the counts for the events that happen in a window of 1 hour. The starting time of the window alters with step size 30 minutes and thus we will see a sliding window in the output. Fig. 1 shows three batches printed to the console.



Batch: 0

window	Interaction	count
[2012-07-02 15:30...]	RT	2
[2012-07-06 06:30...]	RE	1
[2012-07-05 06:30...]	RT	5
[2012-07-05 07:00...]	MT	3
[2012-07-05 13:30...]	RE	1
[2012-07-04 17:30...]	RT	2
[2012-07-05 23:30...]	RE	1
[2012-07-05 17:30...]	RT	1
[2012-07-03 13:30...]	RT	2
[2012-07-03 03:00...]	MT	1
[2012-07-04 03:00...]	MT	9
[2012-07-01 11:00...]	MT	1
[2012-07-07 04:00...]	MT	1
[2012-07-02 10:00...]	RT	1
[2012-07-04 15:00...]	RE	2
[2012-07-04 03:00...]	RT	11
[2012-07-05 21:00...]	MT	1
[2012-07-04 05:30...]	RT	5
[2012-07-04 18:30...]	RT	7
[2012-07-07 06:30...]	RT	1

only showing top 20 rows

Batch: 1

window	Interaction	count
[2012-07-02 15:30...]	RT	3
[2012-07-02 13:30...]	RT	2
[2012-07-06 01:30...]	RE	1
[2012-07-06 06:30...]	RE	1
[2012-07-05 06:30...]	RT	7
[2012-07-05 07:00...]	MT	4
[2012-07-05 13:30...]	RE	1
[2012-07-04 17:30...]	RT	4
[2012-07-05 23:30...]	RE	1
[2012-07-05 17:30...]	RT	2
[2012-07-03 13:30...]	RT	3
[2012-07-03 03:00...]	MT	1
[2012-07-04 03:00...]	MT	12
[2012-07-01 11:00...]	MT	1
[2012-07-07 04:00...]	MT	1
[2012-07-02 10:00...]	RT	1
[2012-07-04 03:00...]	RT	26
[2012-07-04 15:00...]	RE	2
[2012-07-05 21:00...]	MT	1
[2012-07-04 05:30...]	RT	13

only showing top 20 rows

Batch: 2

window	Interaction	count
[2012-07-02 15:30...]	RT	3
[2012-07-02 13:30...]	RT	2
[2012-07-06 01:30...]	RE	1
[2012-07-06 06:30...]	RE	1
[2012-07-05 06:30...]	RT	8
[2012-07-05 07:00...]	MT	7
[2012-07-05 13:30...]	RE	1
[2012-07-04 17:30...]	RT	5
[2012-07-05 23:30...]	RE	1
[2012-07-05 17:30...]	RT	5
[2012-07-03 13:30...]	RT	4
[2012-07-03 03:00...]	MT	1
[2012-07-04 03:00...]	MT	19
[2012-07-05 19:00...]	RT	2
[2012-07-01 11:00...]	MT	1
[2012-07-02 05:00...]	RT	1
[2012-07-07 04:00...]	MT	1
[2012-07-02 10:00...]	RT	1
[2012-07-04 15:00...]	RE	3
[2012-07-04 03:00...]	RT	43

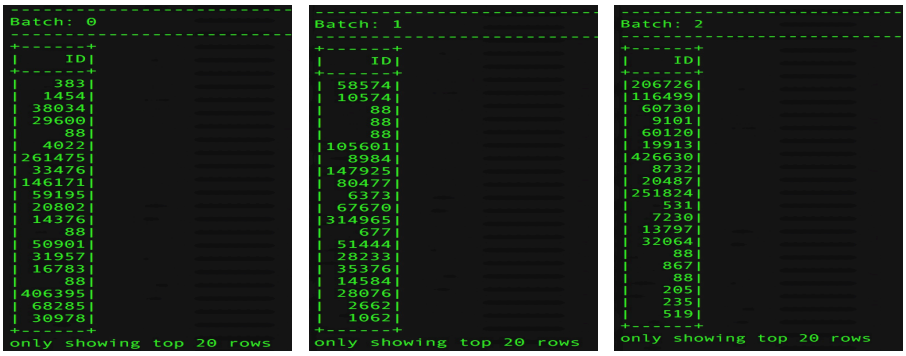
only showing top 20 rows

(a) Batch 0 (b) Batch 1 (c) Batch 2

Figure 1: Counts of the events for three different types of interactions. The first three batches are shown.

1.2 Question 2

In question 2, we wrote an application to emit the twitter IDs of users that have been mentioned by other users every 10 seconds. The output is required to be on HDFS, however, to show our result intuitively, we direct the output to console in this document and keep the output address as is required in the source code. Fig. 2 shows the output printed to the console.

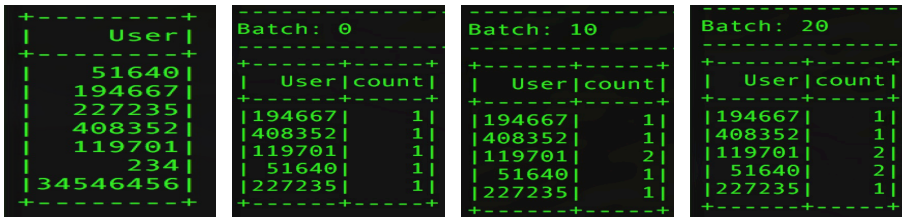


(a) Batch 0 (b) Batch 1 (c) Batch 2

Figure 2: Counts of the events for three different types of interactions. The first three batches are shown.

1.3 Question 3

In question 3, we wrote an application takes as input a list of target twitter user IDs and every 5 seconds, emits the number of tweet actions of a user if it is present in the input list. Fig. 3 shows the output printed to the console.



(a) User list (b) Batch 0 (c) Batch 10 (d) Batch 20

Figure 3: Counts of the events for three different types of interactions. User list, Batch 0, 10 and 20 are shown.

2 Part B - Apache Storm

2.1 Question 1

To answer this question, you need to extend the PrintSampleStream.java topology to collect tweets which are in English and match certain keywords (at least 10 distinct keywords) at your desire. You should collect at least 200'000 tweets which match the keywords and store them either in your local filesystem or HDFS. Your topology should be runnable both in local mode and cluster mode.
Solution:

Source code:

Main function: PartBQ1.java

Spout:

PartBQ1Spout.java

Use Twitter4j library to stream tweets in English language from twitter under the constriction of given keywords.

Bolt:

PartBQ1WriteLocalBolt.java(not used)

Write to local file

Bolt:

use HdfsBolt

Write to hdfs

Compile application:

cd /home/ubuntu/software/apache-storm-1.0.2/examples/storm-starter

mvn clean install -DskipTests=true; mvn package -DskipTests=true

Run application:

cd /home/ubuntu/software/apache-storm-1.0.2/examples/storm-starter

storm jar target/storm-starter-*.jar org.apache.storm.starter.PartBQ1 ¡local/cluster¿ ¡keywords list¿

e.g.:

storm jar target/storm-starter-*.jar org.apache.storm.starter.PartBQ1 cluster halloween love peace Running win sex girls boys man woman father mother

2.2 Question 2

To answer this question, you need to create a new topology which provides the following functionality: every 30 seconds, you should collect all the tweets which are in English, have certain hashtags and have the friendsCount field satisfying the condition mentioned below. For all the collected tweets in a time interval, you need to find the top 50% most common words which are not stop words (http://www.ranks.nl/stopwords). For every time interval, you should store both the tweets and the top common words in separate files either in your local filesystem or HDFS. Your topology should be runnable both in local mode and cluster mode.

Solution:

Main function: PartBQ2.java

Spout:

PartBQ2Spout.java

Use Twitter4j library to stream tweets in English language from twitter under the constriction of hashtags.

SampleFriendCountSpout.java

In every 30 seconds, it will random sample a friendcount from the given friend count lists.

SampleHashTagsSpout.java

In every 30 seconds, it will random sample a list of hashtags based on the given hashtag lists.

Bolt:

TweetFilterBolt.java

Based on the stream received from PartBQ2Spout, SampleFriendCountSpout, SampleHashTagsSpout, use friendcount and hashtags to filter the tweets.

TweetSplitWordBolt.java

Based on the stream received from TweetFilterBolt, split the tweets into words, remove stopwords and remove non-character words. TweetWordRankBolt.java

Based on the stream received from TweetSplitWordBolt, in every 30 seconds, it will sort the words received in current time interval and keep the words with rankings above 50%.

Helper class:

MyWord.java

To store word and its counter.

Compile application:

cd /home/ubuntu/software/apache-storm-1.0.2/examples/storm-starter

mvn clean install -DskipTests=true; mvn package -DskipTests=true

Run application:

cd /home/ubuntu/software/apache-storm-1.0.2/examples/storm-starter

storm jar target/storm-starter-*.jar org.apache.storm.starter.PartBQ2 ¡local/cluster¿ ¡friendcount list¿ ¡hashtags lisg¿

e.g.:

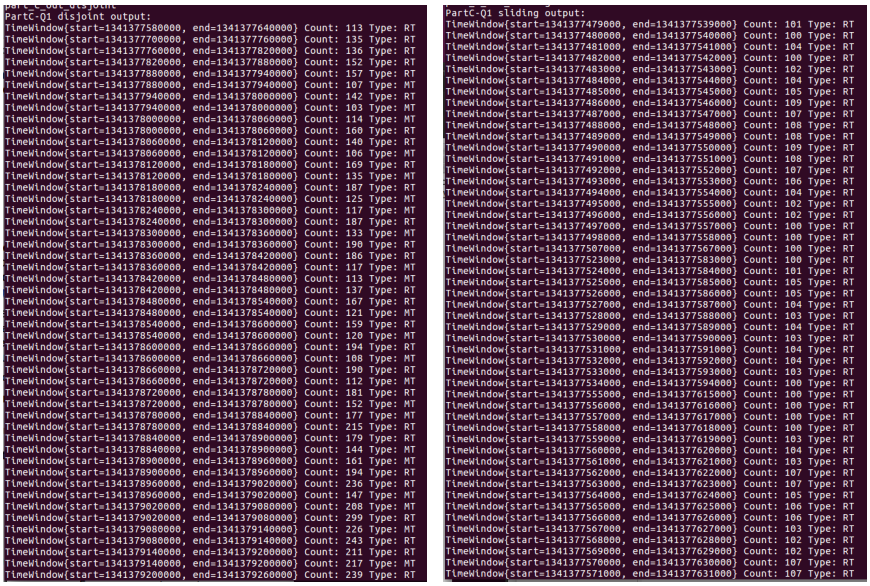
storm jar target/storm-starter-*.jar org.apache.storm.starter.PartBQ2 cluster 5 6 7 8 9 10 20 40 60 80 100 1000 10000 100000 halloween love peace running win sex girls boys man woman father mother design sexy dress fashion free shop style sales today price pumpkin star iphone google airbnb

3 Part C: Flink

3.1 Question 1

In Question 1, we wrote application to find both disjoint 1-min time intervals using *TumblingEventTimeWindows* in *Flink* while application for non-disjoint 1-min time interval was implemented using *SlidingEventTimeWindows*. Since we're using Event time here in Flink, we extracted timestamp provided in sorted

higgs dataset. Noticed that, the timestamp provided in the dataset are in second, we converted them into millisecond for compatibility with Flink (implementation details can be check in our code *PartCQuestion_1_1.java* is the implement of Tumbling Window while *PartCQuestion_1_2.java* implements Sliding Window), part of results can be found in Fig. 4. For full results, please run our applications and the full results will print out to the console.



(a) Tumbling Window (b) Sliding Window

Figure 4: Counts of the events (≥ 100) for tumbling and sliding windows

3.2 Question 2

In Question 2 we wrote application to handle out-of-order data. Different from Question 1, we use allow lateness interface in Flink this time. By tunning the late allowance we get the result in Fig.5. From this results we can tell that, allow lateness provides a good strategy for Flink to handle out-of-order data. More late allowance gives closer results to in order data.

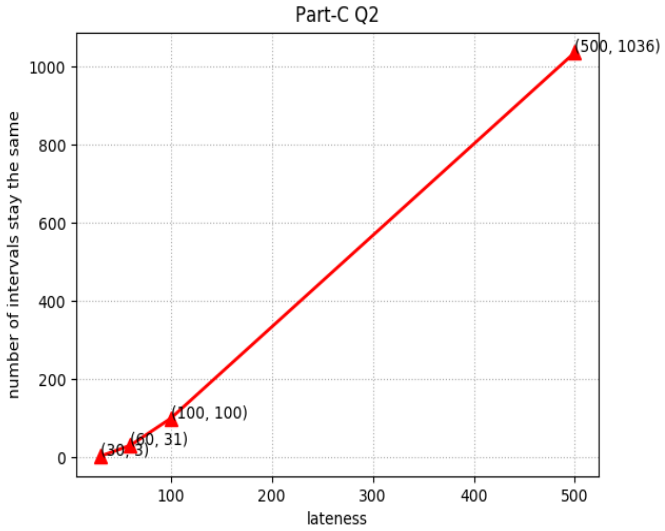


Figure 5: number of intervals that stay the same under various late allowances