Big Data Frameworks

Stream Processing and Messaging using Apache SPARK

Authors

Guillaume Fradet
Félix Larrouy

Project is hosted on github: https://github.com/felixlarrouy/tweets-streaming

November 13, 2018

Summary

1 Server side

Here is the code of our **server.py** file.

```
import tweepy
import socket
import json
import sys
from tweepy import OAuthHandler
from tweepy import Stream
from tweepy.streaming import StreamListener

consumer_key = '8bwa6ory4xsBzOIJ2gAO2ukK2'
consumer_secret = 'cEDvbXAa9DkLPWxutnTZpF2gptgruRRj3KhGMfkYdMqkqCfdqj'
access_token = '1059729535053295617-SgThBTq7GRhA8bvqVxwatOBku3COBA'
access_token_secret = '3F6qTH4Avtqus6HowNKZHE1epGtzPPtMAk7IRhFE20HyN'

host = "localhost"      # Get local machine name
port = 5555             # Reserve a port for streaming.

class TweetsListener(StreamListener):
    def __init__(self, csocket):
        self.client_socket = csocket

    def on_status(self, status):
        print(status.text)

    def on_data(self, data):
        try:
            tweet = json.loads(data)
            print(tweet['text'])
            self.client_socket.send(tweet['text'].encode('utf-8'))
            # return True
        except BaseException as e:
            print("Error on_data: %s" % str(e))
            return False

    def on_error(self, status):
        print(status)
        return False

def sendData(c_socket):
    auth = OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_token_secret)
    twitter_stream = Stream(auth, TweetsListener(c_socket))
    twitter_stream.filter(track=[sys.argv[1]])
```

```
# s: socket object
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((host, port)) # Bind to the port
    print("Listening on port: %s" % str(port))

    s.listen(1) # Now wait for 1 client connection.
    conn, addr = s.accept() # Establish connection with client.
    with conn:
        print('Received request from:', addr)
        while True:
            sendData(conn)
        print('Connection lost with:', addr)
```

We used the **Tweepy** library, which is a Twitter client for Python. First, we need to store our Twitter developper account credentials. Then, we created a **TweetsListener** class which will be handling the streaming of tweets. The most important method here is *on_data*, which receives the data from Tweeter and send it to a socket object. We get each tweet in a JSON format and only send the *'text'* value to the socket.

Next we connect to tweeter streaming with the function *sendData*. We set the authorizations with our credentials and collect every tweet containing *sys.argv[1]* (which is a *string*).

Finally, we will create a socket object and start the streaming process. We bind the socket to a reserved localhost port, wait for a client connection, connect to the client, and send data to the client.

We are now ready to receive the tweets from the client side.

2 Client side

```
import findspark
findspark.init()

import sys
from pyspark import SparkConf, SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import Row, SQLContext

from operator import add
import matplotlib.pyplot as plt

conf = SparkConf()
conf.setAppName("trending_hashtags")

batch_duration = 10
window_duration = 20
```

```python
sc = SparkContext(conf=conf)
ssc = StreamingContext(sc, batchDuration)

host = "localhost"
port = 5555
socket_stream = ssc.socketTextStream(host, port)

lines = socket_stream.window(window_duration)

hashtags = lines.flatMap(lambda line: line.split())\
                .filter(lambda word: word.lower().startswith("#"))

pairs = hashtags.map(lambda hashtag: (hashtag, 1))
hashtags_counts = pairs.reduceByKey(add)


def get_sql_context_instance(spark_context):
    if ('sqlContextSingletonInstance' not in globals()):
        globals()['sqlContextSingletonInstance'] = SQLContext(
            spark_context)
    return globals()['sqlContextSingletonInstance']


def rdd_operation(time, rdd):
    try:
        # Get spark sql singleton context from the current context
        sql_context = get_sql_context_instance(rdd.context)
        # convert the RDD to Row RDD
        row_rdd = rdd.map(lambda w: Row(hashtag=w[0], hashtag_count=w[1]))
        # create a DF from the Row RDD
        hashtags_df = sql_context.createDataFrame(row_rdd)
        # Register the dataframe as table
        hashtags_df.registerTempTable("hashtags")
        # get the top 10 hashtags from the table using SQL and store
        # in pandas dataframe
        hashtag_counts_df = sql_context.sql(
            "select hashtag, hashtag_count \
            from hashtags \
            order by hashtag_count \
            desc limit 10").toPandas()

        # plot trending hashtags
        hashtag_counts_df.plot.barh(
            x='hashtag', y='hashtag_count',
            title=str(time), legend=False)
        plt.show()

    except:
        e = sys.exc_info()
        print("Error: {}".format(e))


hashtags_counts.foreachRDD(rdd_operation)

# start streaming
ssc.start()
ssc.awaitTermination()

# close the connection
ssc.stop()
```

We create a *socketTextStream*, where we expect a Twitter streaming connection on the same port as the one specified in the server side. Next we create a *DStream* where we specify the window duration, which has to be a multiple of the batch duration.

Next we do some transformations on the *DStream* to count the words where there is a '#' as the first character.

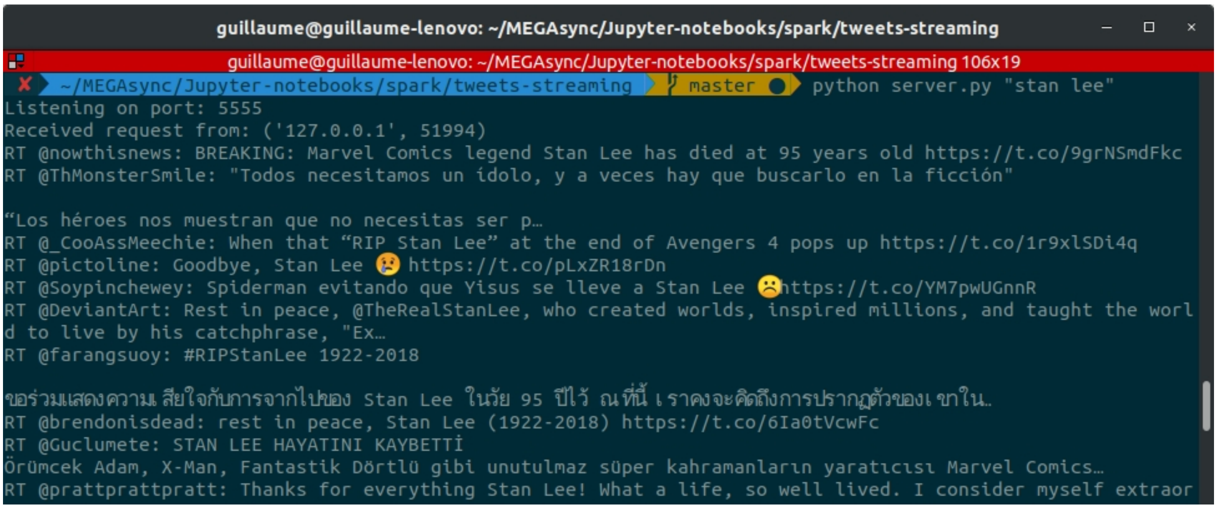
Then for each RDD of the *DStream*, we we store the RDD as a temporary SQL table thanks to the *SQLContext* instance of our *DStream*. We can then query the temporary table which as been created to get the most popular tweets in the window, store the results in a dataframe and plot the hastags.

We are now ready to start the streaming process. Top hashtags will be displayed every *window_duration* seconds.

3 Usage

The first thing we need to do is to launch our server.
To do so, we run the python file *server.py* with a word (or chain of words) as argument. This argument is a condition on the tweets that we will stream, such that all tweets content must include our word(s).

Example: if the argument is "Messi", we will get all the tweets containing "Messi".



guillaume@guillaume-lenovo: ~/MEGAsync/Jupyter-notebooks/spark/tweets-streaming
guillaume@guillaume-lenovo: ~/MEGAsync/Jupyter-notebooks/spark/tweets-streaming 106x19

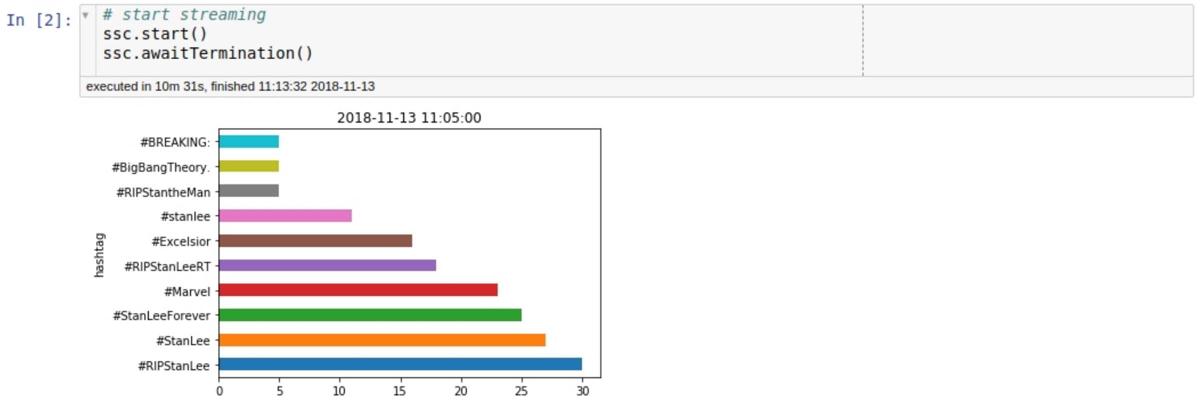
```
✘ ~/MEGAsync/Jupyter-notebooks/spark/tweets-streaming  master ● python server.py "stan lee"
Listening on port: 5555
Received request from: ('127.0.0.1', 51994)
RT @nowthisnews: BREAKING: Marvel Comics legend Stan Lee has died at 95 years old https://t.co/9grNSmdFkc
RT @ThMonsterSmile: "Todos necesitamos un ídolo, y a veces hay que buscarlo en la ficción"

“Los héroes nos muestran que no necesitas ser p…
RT @_CooAssMeechie: When that “RIP Stan Lee” at the end of Avengers 4 pops up https://t.co/1r9xlSDi4q
RT @pictoline: Goodbye, Stan Lee 😢 https://t.co/pLxZR18rDn
RT @Soypinchewey: Spiderman evitando que Yisus se lleve a Stan Lee ☹https://t.co/YM7pwUGnnR
RT @DeviantArt: Rest in peace, @TheRealStanLee, who created worlds, inspired millions, and taught the worl
d to live by his catchphrase, "Ex…
RT @farangsuoy: #RIPStanLee 1922-2018

ขอร่วมแสดงความ เสียใจกับการจากไปของ Stan Lee ในวัย 95 ปีไว้ ณ ที่นี้ เราคงจะคิดถึงการปรากฏตัวของเขาใน..
RT @brendonisdead: rest in peace, Stan Lee (1922-2018) https://t.co/6Ia0tVcwFc
RT @Guclumete: STAN LEE HAYATINI KAYBETTİ
Örümcek Adam, X-Man, Fantastik Dörtlü gibi unutulmaz süper kahramanların yaratıcısı Marvel Comics…
RT @prattprattpratt: Thanks for everything Stan Lee! What a life, so well lived. I consider myself extraor
```

Figure 1: Launch the server to stream the tweets containing "Stan Lee"

Then we can launch the client code in a Jupyter notebook to perform the spark operations on the tweets.



```
In [2]: # start streaming
        ssc.start()
        ssc.awaitTermination()
```

executed in 10m 31s, finished 11:13:32 2018-11-13

Figure 2: Start the pyspark streaming in a Jupyter notebook

4 Outputs

We streamed tweet containing "stan lee", for window durations of 30 seconds, 1 minute, and 5 minutes. Here are the results we got.

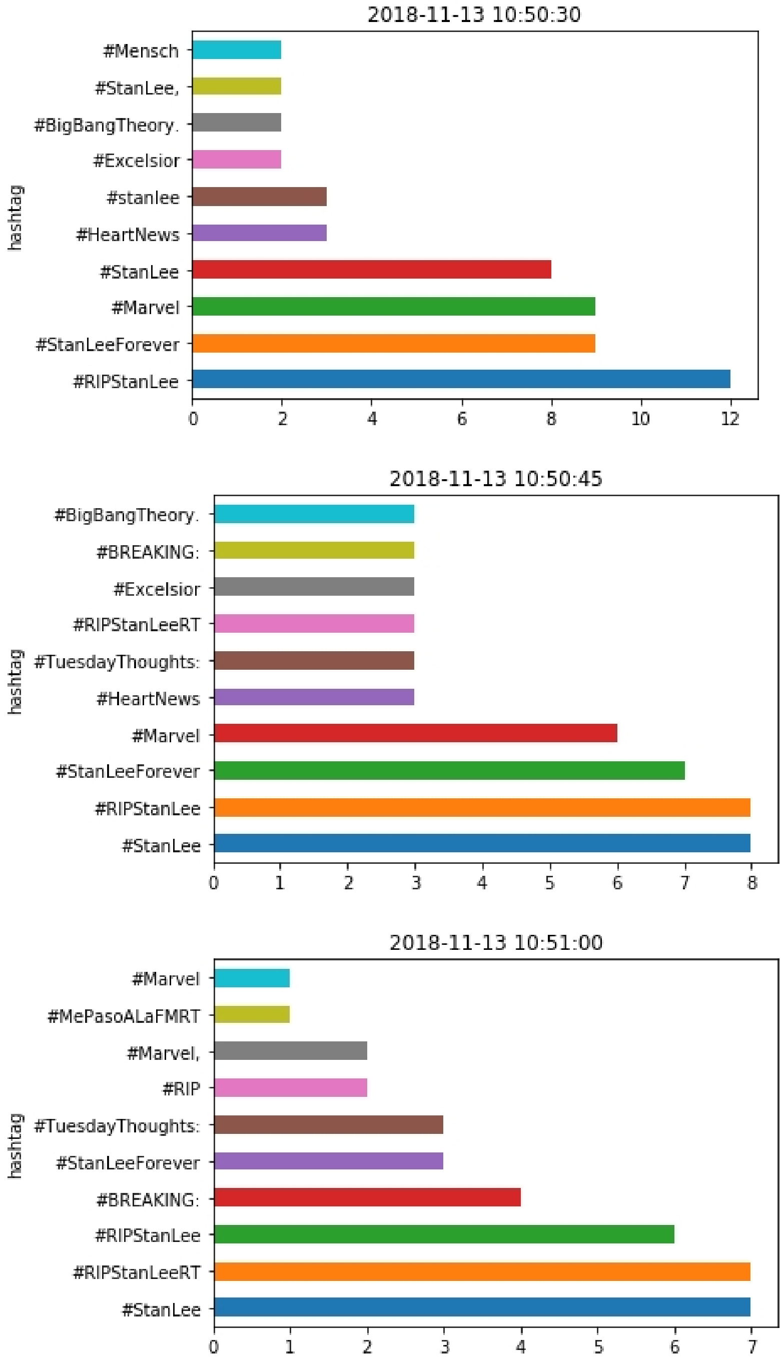


Figure 3: Batch duration of 15 seconds, window duration of 30 seconds

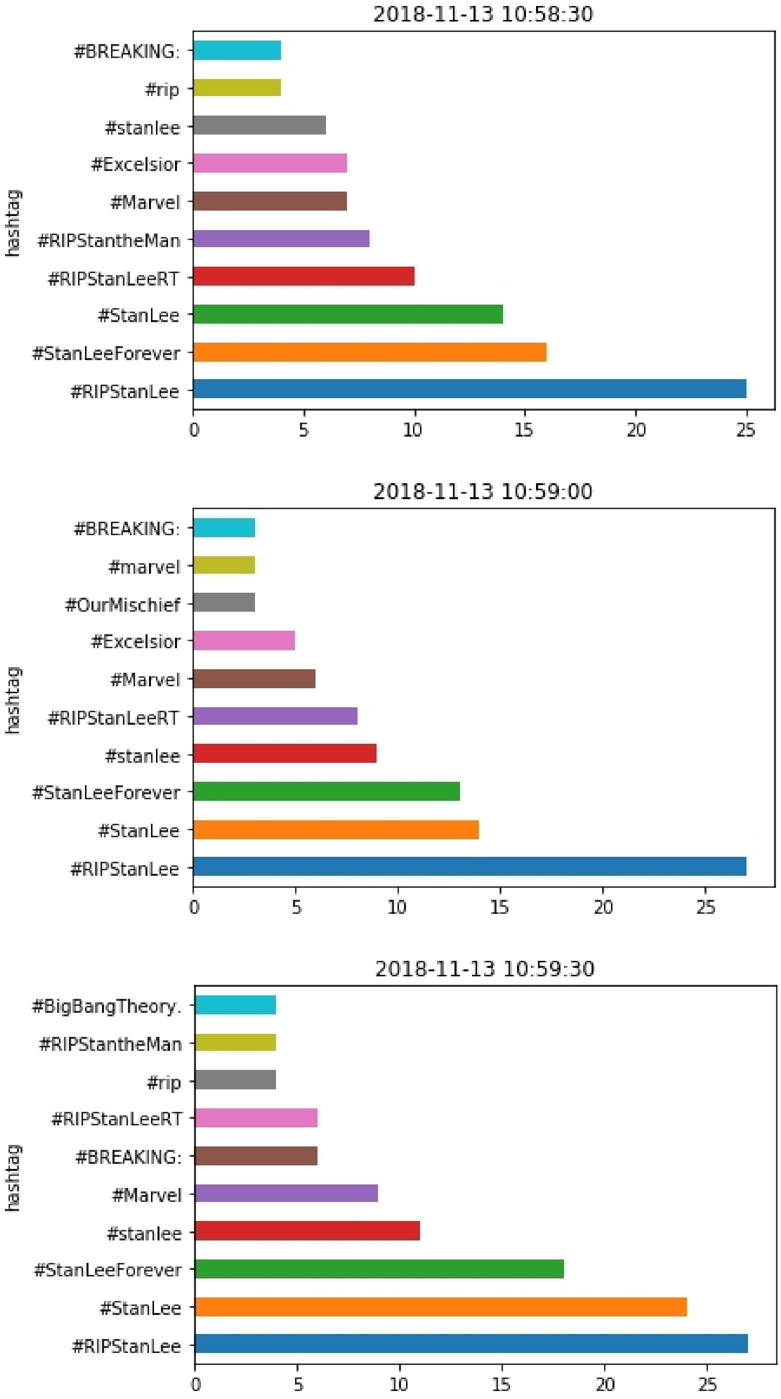
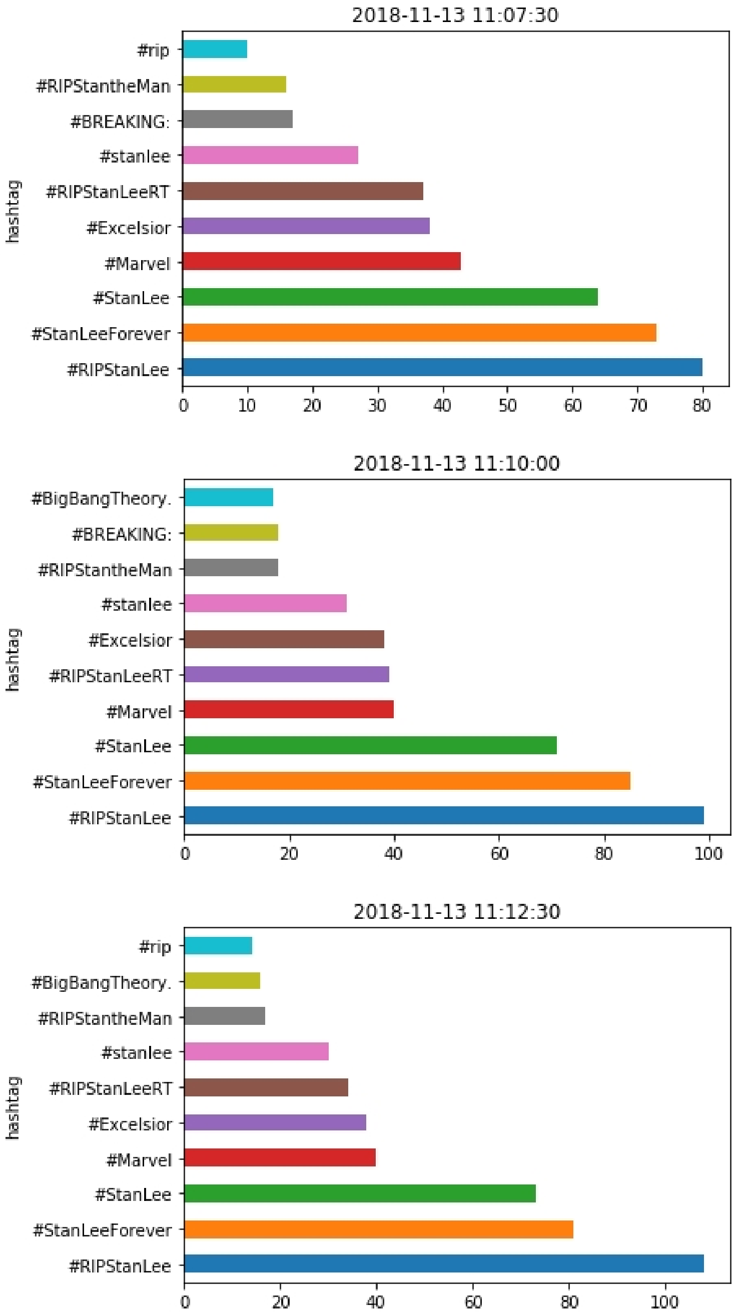


Figure 4: Batch duration of 30 seconds, window duration of 1 minute



2018-11-13 11:07:30

2018-11-13 11:10:00

2018-11-13 11:12:30

Figure 5: Batch duration of 2 minutes 30 seconds, window duration of 5 minutes