
# Big Data Management and Analytics
# Session: Apache Spark Streaming

Lecturer: Petar Jovanovic and Sergi Nadal

November 25th, 2016

In this session we will use Spark Streaming to process live data streams from Twitter. Using SparkSQL we will query the streams and perform further data processing.

All in-class practice will be performed in standalone mode (non-distributed), in order to avoid devoting time to cluster management. For extra scoring, you can implement all the exercises to be executed in cluster mode.

It is highly advisable to complete the tasks listed in section 2 prior to attending the class, in order to get full advantage of the available in-class time.

## 1 Required Tools

- `eclipse` IDE with Java 7 and Maven plugin installed
- The following Java libraries (already included in the provided `pom.xml`):
  - `Spark Core`
  - `Spark Streaming`
  - `Spark Streaming Twitter`
  - `SparkSQL`

## 2 Tasks To Do Before The Session

- Read the slides. It is not necessary to read the examples as they will be discussed in class.
- Create your Twitter App following the guidelines in Appendix A.
- Get familiar with the `Twitter4j` library (`http://twitter4j.org/en/index.html`).

- Checkout the `Spark Streaming` programming guide (`http://spark.apache.org/docs/latest/streaming-programming-guide.html`) and the `SparkSQL` programming guide (`https://spark.apache.org/docs/latest/sql-programming-guide.html`).

# 3 Part A: Examples & Questions (30min)

During the first 30 minutes of session we will go through the 3 proposed examples in the slides. After each one, questions related to that particular topic will be addressed.

# 4 Part B: In-class Practice (2h30min)

## 4.1 Exercise 1 (30 min): Setup environment and familiarization

In this first exercise we will setup the environment to complete the session. First, get the enclosed Java project and import it to your workspace. After this is done, you should be able to see the following packages:

- **(default package)**: with the `main` method and some helper functions to simplify the connection to Twitter.
- **exercises (1, 2, 3)**: with the required classes per exercise to complete.

Do not forget to modify the variable `TWITTER_CONFIG_PATH` in `UPCSchool_Spark.java` with the path of the generated config file as described in apendix A, remember that in Windows the file separator is specified with a double slash. Additionally, remember to correctly set the variable `HADOOP_COMMON_PATH` to its correct location within the resources folder in the project.

Now, run the main method which should display in console the stream of tweets. Once this works, analyze the code provided in `Exercise_1.displayAllTweets`.

The method `displayAllTweets` receives an object `JavaDStream<Status>`, this is a batch of tweets, i.e. `Status`.

A `map` function is applied, this converts each `Status` to a `String` extracting only the user's and text information. Afterwards, it is printed in console.

Note in this session for simplicity we will work in local mode, i.e. in your local eclipse.

## 4.2 Exercise 2 (30 min): Computing Popular Hashtags

Next, let's try something more interesting, say, try printing the 10 most popular hashtags in the last 5 minutes. These next steps explain the set of the DStream operations required to achieve our goal. After every step, you can see the contents of new `DStream` you created by using the `print()` operation

The implementation of exercise 2 can be found in `Exercise_2.get10MostPopularHashtagsInLast5min`, you will see all transformations, but without the required logic (you need to parametrize the methods).

In the nexts steps, you will be guided on how to complete each of them. The code is divided for each of the following subsections.

### 4.2.1 Get the stream of hashtags from the stream of tweets

To get the hashtags from the status string, we need to identify only those words in the message that start with "#".

The `flatMap` operation applies a one-to-many operation to each record in a `DStream` and then flattens the records to create a new `DStream`. In this case, each status string should be split by space to produce a `DStream` where each record is a word. Then we apply the filter function to retain only the hashtags. The resulting hashtags `DStream` is a stream of `RDDs` having only the hashtags. If you want to see the result, add `hashtags.print()` and try running the program. You should see something like this (assuming no other `DStream` has print on it):

```
-------------------------------------------
Time: 1359886521000 ms
-------------------------------------------
#njnbg
#njpw
#?????
#algeria
#Annaba
```

### 4.2.2 Count the hashtags over a 5 minute window

Next, we would like to count these hashtags over a 5 minute moving window. A simple way to do this would be to gather together the last 5 minutes of data and process it in the usual map-reduce way — map each tag to a (tag, 1) key-value pair and then reduce by adding the counts. However, in this case, counting over a sliding window can be done more intelligently. As the window moves, the counts of the new data can be added to the

previous window's counts, and the counts of the old data that falls out of the window can be 'subtracted' from the previous window's counts. Note that `reduceByKeyAndWindow` is a *stateful transformation* as it combines data across multiple batches (generated *RDDs* depends on *RDDs* of previous batches), thus it requires enabling checkpointing (do not worry about this, as it is handled in `UPCSchool_Spark.java`).

There are two functions that should be defined for adding and subtracting the counts. `new Duration(5 * 60 * 1000)` specifies the window size and `new Duration(1 * 1000)` specifies the movement of the window.

Note that only 'invertible' `reduce` operations that have 'inverse' functions (like subtraction is the inverse of addition) can be optimized in this manner. The generated counts *DStream* will have records that are (hashtag, count) tuples. If you print *counts* and run this program, you should see something like this:

```
-------------------------------------------
Time: 1359886694000 ms
-------------------------------------------
(#epic,1)
(#WOWSetanYangTerbaik,1)
(#recharged,1)
(#??????????,1)
(#jaco,1)
(#Blondie,1)
(#TOKIO,1)
(#fili,1)
(#jackiechanisamazing,1)
(#DASH,1)
...
```

### 4.2.3 Find the top 10 hashtags based on their counts

Finally, these counts have to be used to find the popular hashtags. A simple (but not the most efficient) way to do this is to sort the hashtags based on their counts and take the top 10 records. Since this requires sorting by the counts, the count (i.e., the second item in the [hashtag, count] tuple) needs to be made the key. Hence, we need to first use a `map` to flip the tuple and then sort the hashtags. Finally, we need to get the top 10 hashtags and print them.

The `transform` operation allows any arbitrary RDD-to-RDD operation to be applied to each `RDD` of a `DStream` to generate a new `DStream`. The resulting *sortedCounts* `DStream` should be a stream of `RDDs` having sorted

hashtags. The `foreachRDD` operation applies a given function on each `RDD` in a `DStream`, that is, on each batch of data. In this case, `foreachRDD` should be used to get the first 10 hashtags from each `RDD` in *sortedCounts* and print them, every second. If you print the contents of *sortedCounts*, you should see something like this:

```
Top 10 hashtags:
(2,#buzzer)
(1,#LawsonComp)
(1,#wizkidleftEMEcos)
(1,#???????)
(1,#NEVERSHUTMEUP)
(1,#reseteo.)
(1,#casisomoslamismapersona)
(1,#job)
(1,#????_??_?????_??????)
(1,#?????RT(*^^*))
```

Note that this implementation is not scalable, as we are saving the window in memory (which is limited), while we cannot make any assumption on the window's size. In the optional exercises we propose to implement the exercise with an exponential decaying window which solves this problem.

## 4.3 Exercise 3 (1h30min): Sentiment Analysis

Sentiment analysis (SA), sometimes referred as opinion mining, describes the process of using NLP, statistics, or machine learning methods to extract, identify, or otherwise characterize the sentiment content of a text unit.

In this exercise we will perform real-time SA to the stream of tweets. The implementation of the exercise can be found in the method `Exercise_3.sentimentAnalysis`, where some helper functions are provided. In the next steps, you will be guided on how to complete each subtask.

### 4.3.1 Preprocessing

A sentiment analysis process is dependant of the language, therefore first we need to filter the stream in order to retrieve only tweets written in english. The last version of Twitter4j provides a method to detect the language, however the shipped version with Spark does not contain such method. In order to help you, we provide the method `LanguageDetector.isEnglish(String)` which will return `true` or `false` whether the provided text is written in english or not. For further reference on how this is done, you can check the library `Apache Tika`.

Once your stream has been filtered, it is recommended to do further preprocessing by mapping only the pairs (id, text) and making sure you are not working with any `null` text.

### 4.3.2 Applying text functions

It is necessary to put all text to analyze to a common standard form, this can be done by converting the tweet using the following code.

```
1 text.replaceAll("[^a-zA-Z\\s]", "").trim().toLowerCase();
```

After, it is also necessary to perform *stemming*. Stemming techniques can be very complex, in here we will just get rid of stop words (those words that do not provide any value for our purpose). The method `StopWords.getWords()` is provided, which returns a list of stop words.

### 4.3.3 Scoring tweets

Now it's time to get the tweets and decide whether they express positive opinion or not. For both positive and negative, the methods `PositiveWords.getWords()` and `NegativeWords.getWords()` are provided, which return a list with the corresponding words. For the case of positive words, in order to score each word, you should check how many words in the tweet are positive (by checking containment in the positive words list). Finally, the tweet's positive score is calculated by means of $\frac{p}{n}$, where $p$ is the number of positive words and $n$ is the total number of words in the tweet.

For negative words, the process is likewise but using the proper list of negative words.

As we are just interested in relevant tweets, after the tweets have been scored those that have score equal to 0, in both positive and negative variables, should be excluded.

### 4.3.4 Classifying tweets

Once tweets have been scored and two sets (positive and negative) of triples (id, text, score) have been obtained, they should be merged. This can be easily achieved using the method `join`, as the following code shows:

```
1 JavaPairDStream<Tuple2<Long, String>, Tuple2<Float, Float>> joined =
    positiveTweets.join(negativeTweets);
```

The previous code joins the sets `positiveTweets` and `negativeTweets`, using the keys in the first parameter `Tuple2<Long, String>` (the id and text) and adding the two scores in `Tuple2<Float, Float>`.

Afterwards, the two sets of `Tuple2` should be mapped to a single `Tuple4`, representing the structure (id, text, positive score, negative score).

Finally, in order to classify the tweet we just need to map a new attribute describing the sentiment of the tweet:

- **Positive** when $pos\_score > neg\_score$.
- **Neutral** when $pos\_score = neg\_score$.
- **Negative** when $pos\_score < neg\_score$.

This will construct a final structure `Tuple5<Long, String, Float, Float, String>`.

If you print the contents of this final variable, you should see something like this:

```
-------------------------------------------
Time: 1428243800000 ms
-------------------------------------------
(584722988152401921,rt larry baylor called protesters today faith
    miracle temple httptcopqqcrdw,0.15,0.05,positive)
(584723441149935616,morrbecks yeah i bothered good  a waste space
    ,0.0625,0.125,negative)
(584723520833261569, life a book chapters sad happy exciting turn
    page,0.08695652,0.04347826,positive)
```

# 5 Part C: Optional Practice (3h)

## 5.1 Exponentially Decaying Window

As previously said, exercise 2 is not scalable for windows which do not fit into memory. In this optional exercise, it is proposed that you implement the hashtag computation by making use of an exponentially decaying window. Particularly, you should not use any more the transformation `reduceByKeyAndWindow`.

You can read section 4.7, particularly 4.7.3, of `http://infolab.stanford.edu/~ullman/mmds/book.pdf` for the implementation details.

## 5.2 Aggregating sentiment analysis

By the end of exercise 3 we are only able to classify the stream of tweets, but after that they are lost. In this optional exercise, it is proposed that (using SparkSQL) you implement the calculation of cummulative average score per type (positive, negative or neutral). It is recommended that you follow this steps:

- Map the resulting `Tuple5` batch of tweets to the provided class `ScoredTweet`.

- Using the `foreachRDD` method, convert the `RDD` to a `DataFrame` and register it as a temporary table.
- Issue an SQL query in order to extract the average positive and negative score grouped by type.
- By using the method `collect`, you can mix the obtained values with standard variables in your program.
- As average is an additive function you can update it using formula 1.

$$avg_{\text{new}} = \frac{sum_{\text{old}} + SUM(\text{new items})}{count_{\text{old}} + COUNT(\text{new items})} \tag{1}$$

# A Twitter Credentials Setup

Since all of the exercises are based on Twitter's sample tweet stream, it is necessary to configure OAuth authentication with a Twitter account. To do this, you will need to setup a consumer key+secret pair and an access token+secret pair using a Twitter account. Please follow the instructions below to setup these temporary access keys with your Twitter account. These instructions will not require you to provide your Twitter username/password. You will only be required to provide the consumer key and access token pairs that you will generate, which you can easily destroy once you have finished the tutorial. So, your Twitter account will not be compromised in any way.

## A.1 Create Your Application

Go into `https://apps.twitter.com/`. This page lists the set of Twitter-based applications that you own and have already created consumer keys and access tokens for. This list will be empty if you have never created any applications. For this tutorial, create a new temporary application. To do this, click on the "Create a new application" button. The new application page should look the page shown in figure 1. Provide the required fields. The Name of the application must be globally unique, so using your Twitter username as a prefix to the name should ensure that. For example, set it as [your-twitter-handle]-test. For the Description, anything is fine. For the Website, similarly, any website is fine, but ensure that it is a fully-formed URL with the prefix http://. Then, click on the "Yes, I agree" checkbox below the Developer Agreement. Finally, click on the "Create your Twitter application" button.

## A.2 Token Generation

Once you have created the application, you will be presented with a confirmation page similar to the one shown in figure 2. You should be able to see the consumer key and the consumer secret that have been generated. To generate the access token and the access token secret go to the tab "Keys and Access Tokens", and click on the "Create my access token" button at the bottom of the page (lower green arrow in the figure). Note that there will be a small green confirmation at the top of the page saying that the token has been generated.

## A.3 Get OAuth Credentials (Access Token)

Wait for the token to be generated (refresh the screen if required), and in the bottom section you will see that both access token and secret have appeared like in figure 3.

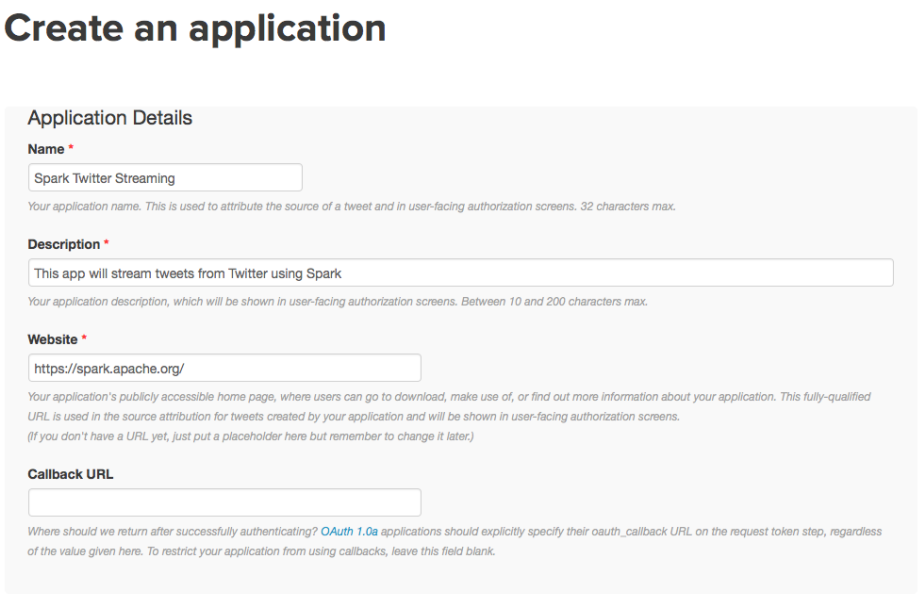


Figure 1: Application creation screen

## A.4 Wrapping Everything

Finally, generate a *.txt* document named `twitter_configuration.txt` using your favorite text editor. The document should have the following structure:

```
consumerKey =
consumerSecret =
accessToken =
accessTokenSecret =
```

Copy the values from the "Keys and Access Tokens" into this appropriate keys in this file. After copying, it should look something like the following:

```
consumerKey = z25xt02zcaadf12 ...
consumerSecret = gqc9uAkjla13 ...
accessToken = 8mitfTqDrgAzasd ...
accessTokenSecret = 479920148 ...
```

Double-check that the right values have been assigned to the right keys. Save the file in a USB flash drive or any cloud-based repository in order to use it during the class.

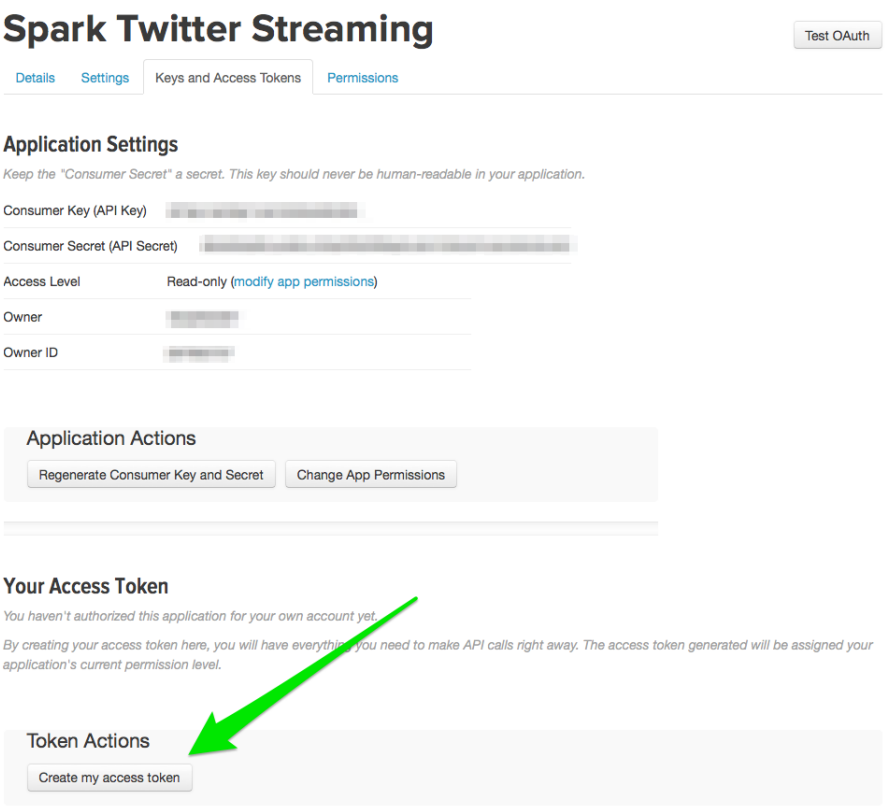


Figure 2: Token generation screen

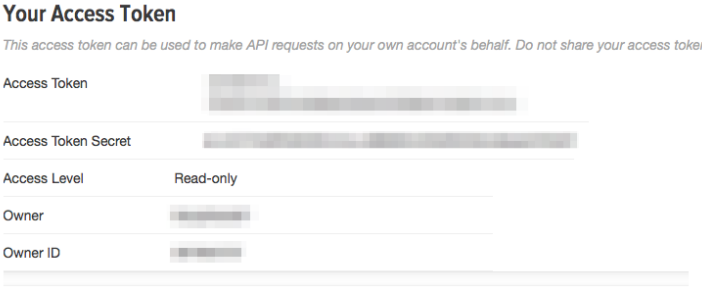


Figure 3: Access Token Screen