Cloud Computing (INGI2145) - Lab Session 6

Muhammad Bilal and Marco Canini

1. Background:

In this lab tutorial, you'll be introduced to Spark - an open-source data analytics cluster computing framework which builds on-top of the Hadoop Distributed File System (HDFS). During the lectures, you've received an introduction on the advantages of using Spark and what it has to offer. Now, we're going to go through practical examples on some of the features offered by this framework.
Our exercises are based on the UC Berkeley AMP Camp tutorials. You can find archived and recent exercises at http://ampcamp.berkeley.edu/.

2. Setup:

General instructions:

- Do a `git pull` from the course repository.
- We will use the course VM to perform the exercise. Go to the INGI2145-vm directory and start your VM using `vagrant up`.
- **Install Scala & SBT** in the VM following these steps:
    1. Download Scala by running in the VM:
       `wget http://downloads.typesafe.com/scala/2.10.6/scala-2.10.6.deb`
    2. Next, run `sudo dpkg –i scala-2.10.6.deb`
    3. Then, install SBT by running:

```
echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 642AC823
sudo apt-get update
sudo apt-get install sbt
```

- **To run Spark applications**, you'll need to perform the following steps:
    a. First, you'll need to package it into a .jar file using a tool called Maven. To do this, just cd to your project directory and type "mvn package" into your console
    b. After bundling your application, you can launch it by running:

```
/usr/local/spark/bin/spark-submit --class <classname> --master
local[<number of worker threads>] target/<package>.jar
```

  - If you're running mvn using the default configuration, the <package>.jar file will be located in the `target` folder
  - Check Master URLs for more details on options for the master command line argument.

3. Quick Start:

Let's first get acquainted with Spark. We will follow the Quick Start tutorial from the official project documentation: https://spark.apache.org/docs/1.4.1/quick-start.html.

Basics

Spark's shell provides a simple way to learn the API, as well as a powerful tool to analyze data interactively. It is available in either Scala (which runs on the Java VM and is thus a good way to use existing Java libraries) or Python. Start it by running the following:

```
/usr/local/spark/bin/spark-shell
```

Spark's primary abstraction is a distributed collection of items called a Resilient Distributed Dataset (RDD). RDDs can be created from Hadoop InputFormats (such as HDFS files) or by transforming other RDDs. Let's make a new RDD from the text of the README file in the Spark source directory:

```
scala> val textFile = sc.textFile("/labs/6_spark/README.md")

textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at textFile
 at <console>:21
```

RDDs have `actions`, which return values, and `transformations`, which return pointers to new RDDs. Let's start with a few actions:

```
scala> textFile.count() // Number of items in this RDD

res0: Long = 22



scala> textFile.first() // First item in this RDD

res1: String = "This project uses maven as the build system. "
```

Now let's use a transformation. We will use the `filter` transformation to return a new RDD with a subset of the items in the file.

```
scala> val linesWithJar = textFile.filter(line => line.contains("jar"))

linesWithJar: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filt
er at <console>:23
```

We can chain together transformations and actions:

```
scala> textFile.filter(line => line.contains("jar")).count()  // How many li
nes contain "jar"?

res2: Long = 7
```

More on RDD Operations

RDD actions and transformations can be used for more complex computations. Let's say we want to find the line with the most words:

```
scala> textFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b
) a else b)

res3: Long = 22
```

This first maps a line to an integer value, creating a new RDD. `reduce` is called on that RDD to find the largest line count. The arguments to `map` and `reduce` are Scala function literals (closures), and can use any language feature or Scala/Java library. For example, we can easily call functions declared elsewhere. We'll use `Math.max()` function to make this code easier to understand:

```
scala> import java.lang.Math

import java.lang.Math

scala> textFile.map(line => line.split(" ").size).reduce((a, b) => Math.max(
a, b))

res4: Int = 22
```

One common data flow pattern is MapReduce, as popularized by Hadoop. Spark can implement MapReduce flows easily:

```
scala> val wordCounts = textFile.flatMap(line => line.split(" ")).map(word =
> (word, 1)).reduceByKey((a, b) => a + b)

wordCounts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[8] at redu
ceByKey at <console>:24
```

Here, we combined the `flatMap`, `map` and `reduceByKey` transformations to compute the per-word counts in the file as an RDD of `(String, Int)` pairs. To collect the word counts in our shell, we can use the `collect` action:

```
scala> wordCounts.collect()

res5: Array[(String, Int)] = Array((package,1), (For,2), (this,2), (Jar,2),
(it,1), (is,4), (The,3), (VM,1), (run,2), ...)
```

Now try to experiment for a few minutes on your own with the transformations and actions documented at https://spark.apache.org/docs/1.4.1/programming-guide.html.

4. Exercises:

I. Machine Learning (K-Means Clustering)

In this exercise, we'll learn how to apply a machine-learning algorithm, specifically K-means clustering, on data using the Spark framework. To apply most machine learning algorithms, we must first preprocess and featurize the data. That is, for each data point, we must generate a vector of numbers describing the salient properties of that data point.

We've already done this step for you. In our case, each data point will consist of a unique Wikipedia article identifier (i.e., a unique combination of Wikipedia project code and page title) and associated traffic statistics. We generated 24-dimensional feature vectors; with each feature vector entry summarizing the page view counts for the corresponding hour of the day.

The K-Means clustering algorithm's objective is to partition your data into *K* clusters. In a nutshell, the algorithm is as follows:

a. Initialize centroids (central points for each cluster) by randomly picking *K* points from the input dataset
b. Discern the closest point to each centroid
c. Assign points sharing the same centroid to a cluster
d. Re-calculate the centroids by averaging the values of points for every cluster
e. Repeat the process until the margin of error between centroids is less than a predefined threshold

In this exercise, we've created a standalone Spark application that takes in the aforementioned featurized file as an input and returns a sample of the resulting clusters.

Task:

Before starting, first familiarize yourself with the code, and how our function operates. This shouldn't require any prior machine-learning background (an explanation of K-Means was given in a previous lecture). Your task is to implement the "ClosestPoint" function in order to **calculate the closest points to a centroid**. You'll also need to **calculate the error between the older and newer centroids** (this is the aggregated squared distances between the centroid vectors).

Before starting to work on these tasks, please refer to the following files:

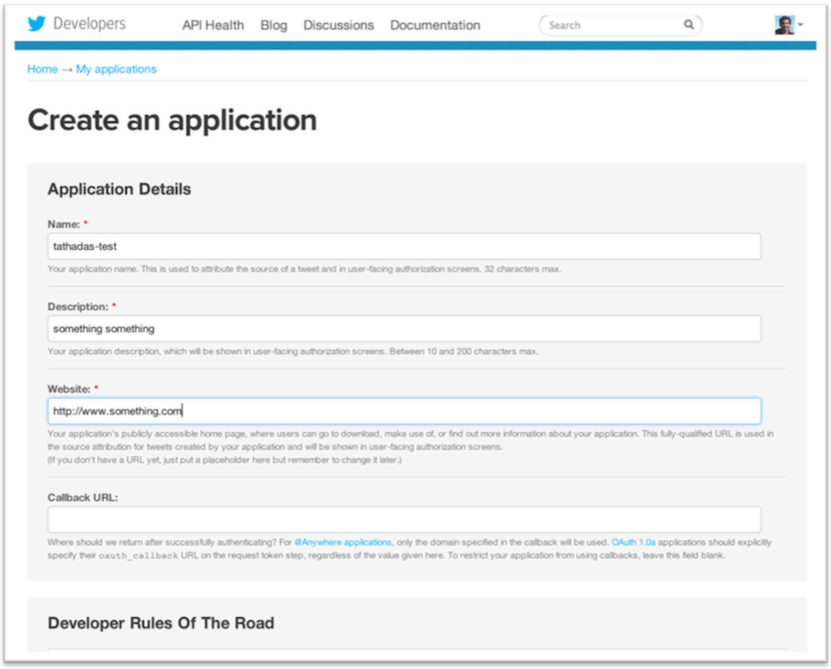
- `KMeans/src/main/java/WikipediaKMeans.java`: This is your main class, which should comprise all the clustering functionality.
- `KMeans/src/main/resources/wiki-stats`: This is the featurized input file that you'll be using as an input to your K-Means clustering algorithm.

II. Stream Processing:

In this exercise, we will walk you through using Spark Streaming to process live data streams. Remember, Spark Streaming is a component of Spark that provides highly scalable, fault-tolerant streaming processing. This exercise is designed as a standalone Java program that will receive and process Twitter's real sample tweet streams.

A – Setup:

1- Before starting, you'll need to setup your twitter tokens. First, open this link: dev.twitter.com/apps. 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 blue "Create a new application" button. You should see a page similar to the one below:



Home → My applications

Create an application

Application Details

Name: *

tathadas-test

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description: *

something something

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website: *

http://www.something.com

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

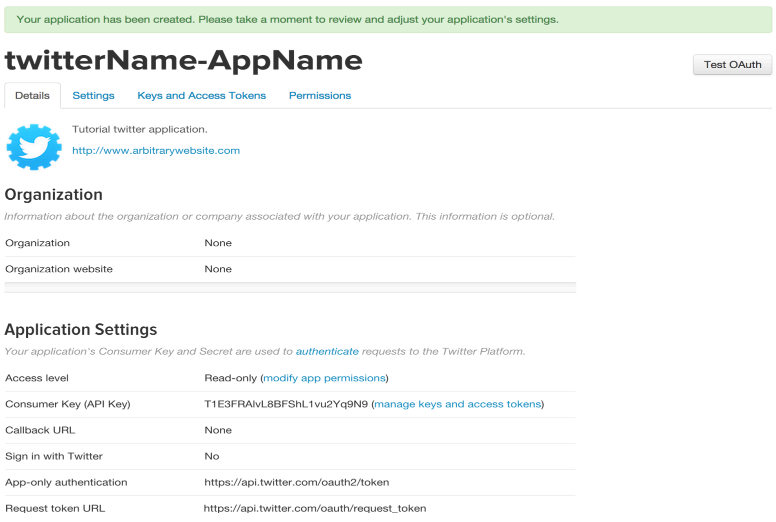
Callback URL:

Where should we return after successfully authenticating? For @Anywhere applications, only the domain specified in the callback will be used. OAuth 1.0a applications should explicitly specify their `oauth_callback` URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Developer Rules Of The Road

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 Rules of the Road**. Finally, click on the blue "Create your Twitter application" button.

2- Once you have created the application, you will be presented with a confirmation page (as shown below). 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, click on the "Keys and Access Tokens" tab. Then click on "Generate my access token" on the bottom of the page.


Your application has been created. Please take a moment to review and adjust your application's settings.

twitterName-AppName

Tutorial twitter application.
http://www.arbitrarywebsite.com

Organization

Information about the organization or company associated with your application. This information is optional.

Organization	None
Organization website	None

Application Settings

Your application's Consumer Key and Secret are used to authenticate requests to the Twitter Platform.

Access level	Read-only (modify app permissions)
Consumer Key (API Key)	T1E3FRAlvL8BFShL1vu2Yq9N9 (manage keys and access tokens)
Callback URL	None
Sign in with Twitter	No
App-only authentication	https://api.twitter.com/oauth2/token
Request token URL	https://api.twitter.com/oauth/request_token

3- To get all of the keys and secrets required for authentication, click on the *"Test OAuth" button* in the top right of the page.

4- Finally, update your twitter configuration file (located at "src/main/resources/twitter.txt") using your favorite text editor. You should see a template of = separated key-value pairs already setup (like the one shown below)

```
consumerKey =

consumerSecret =

accessToken =

accessTokenSecret =
```

Copy the values of your keys into this file and save it.

B- Exercise:

In this exercise, we'll work on processing a stream of tweets and then output the top 10 most frequently found hashtags in a 5-minute window that shifts in 1-second increments.

Task:
Before starting to work on these tasks, please refer to the following files:

- "Twitter/src/main/java/TwitterStreaming.java": This is your main class, which should comprise the entire stream processing functionality.
- "Twitter/src/main/resources/twitter.txt": This is the Twitter OAuth configuration file which should be filled with your access keys

After setting up your keys, refer to the "TwitterStreaming.java" file in your "lab4" directory. Walkthrough the code, and make sure you read the comments and can understand the general structure of the application. You'll be required to implement the following tasks to make the application work:

1. Split the retrieved tweets into words
2. Fetch the hashtags from these words
3. Map hashtags to integer values of 1

Appendix A:

- You'll find the following functions useful for running transform operations on the JavaPairDStream object:
    - **map**(*func*): Return a new DStream by passing each element of the source DStream through a function *func*.
    - **flatMap**(*func*): Similar to map, but each input item can be mapped to 0 or more output items.
    - **mapToPair**(*func*): Return a new JavaPairDStream by passing each element of the source JavaPairDStream through a function *func*.
    - **filter**(*func*): Return a new DStream by selecting only the records of the source DStream on which *func* returns true

- You can consult Apache's spark programming guides https://spark.apache.org/docs/1.4.1/programming-guide.html
  and/or API https://spark.apache.org/docs/1.4.1/api/java/index.html for more details
- If twitter refuses to authenticate and returns a 401 error, this could possibly mean:
    - Your keys are incorrect. In which case, you need to verify the correctness of the keys entered in your twitter.txt configuration file
    - The timestamps for your requests are behind. Verify that your VM's clock isn't lagging behind (use the command `date` to check time and date).
    - For more information on what the different response codes entail, visit: https://dev.twitter.com/overview/api/response-codes
- You might receive a **warning** similar to: "Block input-x-xxxx already exists on this machine; not re-adding it" while running the streaming application. This is quite normal; it occurs because Spark attempts to replicate streaming data without having any actual nodes (because we're running locally).

Appendix B:

Modifying Clock of the VM

If the time zone is set incorrectly:
In the VM open the timezone file

```
sudo vim /etc/timezone
```

and write `Europe/Brussels` in it.
then

```
export TZ=Europe/Brussels
```

Now use the command `date` to check the current time and date to make sure that it is correct.

If the clock is not synchronized:

```
sudo service ntp stop
sudo ntpdate -s time.nist.gov
sudo service ntp start
```