Lab 3 - Spark Streaming & Dynamo DB

Jesus A. Omaña Iglesias & Pablo Castagnaro

Large Scale Distributed Systems - 2024 Edition

0.1 Changelog

2024/02/28 10:00 Initial version

Objectives of this lab:

- to make a stateless transformation with Spark Streaming
- to make a stateful transformation by using windows
- to make a fully-stateful transformation by using state objects
- to use DynamoDB

1 Introduction

In this Lab we'll explore a few, simple, real-time applications, using Mastodon as source. For all exercises, the input will be a (Java) `DStream` of `SimplifiedTweetWithHashtags`, similar to the ones used in previous labs, which will be our *unit of processing*.

In the provided skeleton, you'll find a starting point for each exercise, plus:

- a class named `MastodonStreamingExample`, which implements a working example of reading toots from Mastodon and displaying them on the screen
- a class named `MastodonWindowExample`, which implements a working example of counting toots from Mastodon using a window

There's some additional code in package `com.github.tukaaa`: it's needed for the lab to work properly: feel free to explore it, but do not change it.

The Lab is divided into 4 exercises:

- Running a stateless application that joins a stream with a static RDD
- Running a stateful transformation that uses windows
- Running a fully-stateful transformation that holds a state
- Writing and reading from DynamoDB

2 Running example application locally

Run the `MastodonStreamingExample` locally, to check that everything works correctly.

- The application expects a configuration file `app.conf`; a valid, usable example is provided in the skeleton folder.

Default configuration file

```
mastodon-server = mastodon.social
sleep-interval-ms = 200
```

- Since you'll be running applications locally, you might want to separate the verbose Spark output with the real output. You can use the provided `log4j.properties` file from the repo with the following spark-submit command option:

  `--conf spark.driver.extraJavaOptions=-Dlog4j.configuration=file:///path/to/properties`

- if you're having troubles running the application locally, with errors similar to *"cannot find method methodName()"*, it might be due to jar conflicts between spark and the dependencies of your application. Find your Spark installation, move to the jar directory (in downloaded spark, the `jars` directory; in brew spark, the `libexec/jars` directory, etc.) and remove the following files: `gson-2.2.4.jar` (or equivalent versions), `okhttp-3.12.12.jar` (or equivalent versions), `okio-1.14.0.jar` (or equivalent versions)

- notice how a stream is created: `new MastodonDStream(sc, appConfig).asJStream();` make sure you keep doing it in the same way for all exercises: `asJStream()` wraps a `DStream` into a `JavaDStream`, which you need to apply Spark transformations

- To recap, the command to launch the script should look like this:

  `spark-submit --conf <conf_option> --class <class_path> <jar_file>`

3 Stateless: joining a static RDD with a real time stream (4 points)

In this section, you'll use a join operation to join a static RDD with a real-time stream. With an interval of 20 seconds, we want to display the number of tweets for each language. Since this is a *stateless transformation*, you'll only work on the current micro-batch.

An additional file, named `map.tsv`, is provided in the resources folder of the skeleton project. It's a tab-separated file containing the mapping between a language code (for instance `en`) and the corresponding full language name (for instance `English`). The file contains additional information that you'll discard.

You should:

- complete the method `buildLanguageMap` in the util class `LanguageMapUtils`. This method transforms the language mapping file, after being loaded as an `RDD<String>`, to an `RDD<String, String>`. It's in a separate method for convenience, since you'll use it more than once in this Lab.

- complete the Spark application `MastodonStateless`

The output should be similar to the one below.

Expected output excerpt (values are arbitrary)

```
-------------------------------------------
Time: 1551746660000 ms
```

```
-------------------------------------------
(English,257)
(Japanese,89)
(Thai,35)
(Arabic,34)
(Korean,26)
(Tagalog,14)
(Turkish,4)
(Italian,3)
(Estonian,2)
(Persian,1)
```

3.1 Remarks

– You should sort the output by number of appearances in descending order.

4 Spark Stateful transformations with windows (4 points)

In this section, you'll use windows to keep track of values between consecutive micro batches. The size of the batches should be the same as in the previous exercise (20 seconds). From the window you'll display only the *15 most frequent languages* of the last minute. You will print out the result for both the batch and the window, as from the code snippet.

Complete the Spark application named `MastodonWindows`. Be sure that results are sorted by the number of times each entry appeared. The output should be similar to the one below:

Expected output excerpt (values are arbitrary, and reduced to three per window)

```
-------------------------------------------
Time: 1551747120000 ms        <-- MICRO BATCH
-------------------------------------------
(147,English).
(58,Japanese)
(54,Spanish; Castilian)
...

-------------------------------------------
Time: 1551747120000 ms       <-- WINDOW
-------------------------------------------
(147,English)
(58,Japanese)
(54,Spanish; Castilian)
...

-------------------------------------------
Time: 1551747140000 ms        <-- MICRO BATCH
-------------------------------------------
(266,English)
(105,Japanese)
(90,French)
...

-------------------------------------------
Time: 1551747140000 ms        <-- WINDOW
-------------------------------------------
```

```
(413,English)
(163,Japanese)
(90,French)
...
```

5 Spark Stateful transformations with state variables (2 points)

In this section, you will answer the following question:

What are the 20 users for a given language with most toots while the application is running?

You will specify the language as a parameter to the application. Complete the Spark application named `MastodonWithState`, that answers the question above. The output should be similar to the one below.

Expected output excerpt (values are arbitrary)

```
-------------------------------------------
Time: 1551740880000 ms
-------------------------------------------
(24,Kirin_Company)
(18,asahibeer_jp)
(15,shimamura_gr)
(6,JessVel32187965)
(6,SolidMcMc)
(5,sSShOlu5jvTa4Ae)
(5,lefterisp7)
(5,mhstra25)
(5,YolumAk17)
(5,6Trakyal)
[...]
```

6 DynamoDB (2 points)

In this section, we'll create an application that writes data to dynamo DB, then reads it. *The application writing to DynamoDB will be a Spark application, while the application reading from DynamoDB will be a traditional java one.* They will both use the same code (the *repository* implementation) to access Dynamo.

6.1 Writing to Dynamo DB (1 point)

In this exercise, we'll create a Spark streaming application that collects hashtags from Tweets while the application is running and stores the result in a DynamoDB table named `LsdsTwitterHashtags`. To read-/write from storage, we'll use an interface called `IHashtagRepository`, using a well-known design pattern named Repository, which provides high-level methods to read/write from storage.

Interface for a Repository

```
public interface IHashtagRepository {

  /**
   * Read top 10 hashtags
   * @param lang
   * @return
   */
```

```
  List<HashTagCount> readTop10(String lang);

  /**
   * Write on storage an element of type T
   * @param h
   */
  void write(SimplifiedTweetWithHashtags h);
}
```

The name of the concrete implementation must be `DynamoHashTagRepository`.
For each hashtag (remember: a tweet might have more than one!), the table must store:

- the *hashtag*
- the *language*
- an *incremental counter*, that will be continuously updated each time an hashtag appears
- *the list of tweet ids* where the hashtag appears.

You need to properly choose the keys and the DB operation for the application to work correctly. Then you'll run the application for at least 5 minutes to have enough data for the next exercise.

6.1.1 Remarks

- You can manually create the table in the AWS console, as shown in the slides

6.2 Reading from DynamoDB (1 point)

For this exercise, you will access the data stored during the execution of the previous exercise. For this, you should implement a class `MastodonHashtagsReader` that, given a language specified by the user, and using the Repository object you created previously, retrieves from DynamoDB the top 10 hashtags in the given language.

6.2.1 Remarks

- This is a *normal Java class*, not a Spark streaming application!
- The expected return value for the top10 is a `List<HashTagCount>`, where `HashTagCount` is a simple Data Class holding some basic information for a hashtag. You'll need to sort on this and to do it properly you'll need to write a Java Comparator (a class that describes ordering for items of a given class)

7 Final remarks. Important!

- Your code should compile. Non-compiling code won't be evaluated
- Your code should be a valid Maven project
- Your code should be able to create a jar containing all your non-test classes (`mvn clean package` should work)
- The implemented classes should be runnable, that is, they should work with `spark-submit`
- Finally, do not forget to submit a copy of your final code in Moodle (one submission per group).

7.1 Deadlines

– deadline is March 15th at 23:59