UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
School of Professional & Executive Development

Big Data Management and Analytics
Session: Lambda Architecture

Lecturers: Petar Jovanovic and Sergi Nadal

April 27th, 2016

1 Required Tools

- SSH and SCP client
- `eclipse` IDE with JDK 7 and Maven plugin installed

2 Tasks To Do Before The Session

Before coming to this Lambda Architecture session, you should have checked the session slides and write down all doubts you might have about them.

3 Part A: Examples & Questions (30min)

The first 30 minutes of the session will be spent on answering all the possible questions you might have come up with during the week by preparing the tasks before the session (see above).

4 Part B: In-class Practice (2h 30min)

4.1 Exercise 0 (20 min): Setup of the environment

In this session we are going to set up a Lambda Architecture. One of the main aspects of such architectures is the division in three layers (see slides), being the speed layer the most new one. The problem of speed layer is that streams might happen faster than our computational resources are capable of performing and therefore some streams might be lost. To solve this, in this practical session, we are going to work with Apache Kafka which is nothing else than a distributed queue system and which hence allows queuing streams that can be then processed a bit later.

Consequently, you need to follow the following steps to set up Kafka. We will install it in *slave1* so the next commands **must be executed in such node**.

1. Download and untar it:

   ```
   wget http://ftp.cixug.es/apache/kafka/0.8.2.0/kafka_2.10-0.8.2.0.tgz

   tar xf kafka_2.10-0.8.2.0.tgz
   ```

2. Edit file *kafka_2.10-0.8.2.0/config/zookeeper.properties* and change the *dataDir* property such that:

   ```
   dataDir=/home/bdma**/lambda/zookeeper
   ```

3. Upload the *lambda.sh* script to this node and run it.

   ```
   chmod +x lambda.sh

   ./lambda.sh
   ```

4. Create a topic named *twitter* in Kafka. Topics are essentially "namespaces" under which messages will be published.

   ```
   kafka_2.10-0.8.2.0/bin/kafka-topics.sh --create --zookeeper localhost:2181
          --replication-factor 1 --partitions 1 --topic twitter
   ```

The next change **must be done in the master node**. It basically consists on updating your Spark installation with the Kafka streaming JARs and replicate it to the rest of nodes. Thus, run (it is all a single URL):

```
wget http://central.maven.org/maven2/org/apache/spark/spark-streaming-kafka_2.10/
       1.6.1/spark-streaming-kafka_2.10-1.6.1.jar

mv spark-streaming-kafka_2.10-1.6.1.jar spark-1.6.1/lib/.

scp -r spark-1.6.1 bdma**@slave1:.
scp -r spark-1.6.1 bdma**@slave2:.
```

Finally you should start Hadoop, Spark and HBase and we are ready to go.

Note: Every time you start up the cluster you will need to run the *lambda.sh* script and recreate the topic twitter.

4.2 Exercise 1 (20min): Streaming from Kafka

If you look at the code in the class *LambdaListener* you will see that when a tweet is received from the streaming, it is transformed into a JSON and this JSON is what is finally published in the queue. This is so in order to simplify the transmission of tweets between the different components of our architecture as it requires serialization processes.

Also, for the sake of simplicity, only the ID, the text and the date the tweet was created are recollected though in typical scenarios we would gather as much information as possible.

In this first exercise then what you are expected to do is to complete the code in *Exercise1.java*. Up to here, we are still not doing anything with the Lambda Architecture but in order to check you understood all the previous configurations and explanations, you are supposed to implement a Spark Streaming job that reads tweets from Kafka and simply prints its content in the screen.

To run this exercise:

```
spark-1.6.1/bin/spark-submit labo7.jar -exercise1
```

4.3 Exercise 2 (1h): Speed layer

This is the exercise where you will spend most of the time. The idea of the exercise is to set up a Lambda Architecture to figure out the time intervals at which a given hash tag has been occurring. For instance the hash tag *#MTVStars* might occur in the time "Fri Nov 27 131304 CET 2015" or "Fri Nov 27 153247 CET 2015" (the intervals themselves will be computed at the serving layer). The rationale of applying a Lambda Architecture here is that this way we can have the interval up to the last moment so we could detect in real-time how trending topics are being formed.

Thus, what we are going to do is to build such an architecture in which tweets will be firstly saved in Kafka. Afterwards, a Spark Streaming job (right as you just did in the previous exercise) will save these whole JSON tweets in a HDFS plain file in a folder named "lambda" (the batch layer; to be seen in next exercises) and it will also save some information in HBase (the serving layer; to be seen in next exercises). More precisely, in HBase we want to have one table also called "lambda" containing one family "tweets" and where each row will represent one hash tag, and in its family "tweets" there will be as many qualifiers as tweets containing such hash tag. Finally the value stored in each cell is the time at which that tweet happened. You can use tweet ID to create the qualifier name.

Last but not least thing to mention as introduction for this exercise is that HDFS only allows having one writer for the same file at once, which consequently contradicts the fact that many Spark workers will have to write in the same file. To overcome this limitation, in this practical session you have to use the *WriterClient* class that will solve this limitation transparently to you; you only have to call its method *write* with an array of bytes to be written.

Your tasks for this exercise then are:

1. Create the "lambda" folder in HDFS.

2. Create the "lambda" table in HBase.

3. Complete the code in *Exercise2.java*.

To run this exercise:

```
spark-1.6.1/bin/spark-submit labo7.jar -exercise2
```

4.4 Exercise 3 (10min): Batch layer

Similarly as in previous exercise, we need a Spark job to move data from the batch layer to the serving layer. One of the advantages of using a Lambda Architectures is that it provides real-time analysis capabilities while all the data is also saved in the batch layer. Consequently, this batch layer data can be used for future use cases.

According to this, your task in this exercise is to complete the code in *Exercise3.java*.

To run this exercise:

```
spark-1.6.1/bin/spark-submit labo7.jar -exercise3
```

4.5 Exercise 4 (40min): Serving layer

Finally, we query the serving layer in order to retrieve the information we want. In this case what we want is to obtain the time intervals within which a given hash tag has been occurring. For instance the hash tag *#MTVStars* might occur in the second 10, 12 and 18, so what we want to return is 2 and 6.

Your task in this exercise is to complete the code in *Exercise4.java*. Note that in this case you don't really need to implement a Spark job but just a simple script that connects to HBase, queries it and returns the requested information.

To run this exercise:

```
spark-1.6.1/bin/spark-submit labo7.jar -exercise4 \#MTVStars
```

Deliverable: Upload a single zip file to the campus containing all the Java classes you have modified in the exercises. Name such zip file as "name_surname.zip"

5 Part C: Optional Practice (3h)

In section 4 we set up a Lambda Architecture with just a Kafka as new tool but many other technologies can be plugged in here to make it more sophisticated. Our proposal for this section, and as this is the last Hadoop/Spark practical work, is to let you decide on what technologies you want to explore more in depth and include them in the architecture. Some ideas we give are:

- Use Flume to flush all the stream data also in the batch layer.

- Use high-level querying tools like Hive on top of the serving layer.
- Play with the different HDFS file formats.
- ...

Deliverable: Write and upload a small document (1-2 pages) introducing the extension you have done, the steps you have followed, the results and your conclusions. Name such file as "name_surname.doc"