Qualifying Scala project

class Lambda *extends* Architecture *with* Scala

Business appreciated the Scala language. The key moments today are its reliability in the context of highly loaded systems and ability to extend the toolkit to the context of the task.

We offer you to implement a simple application for analytics of real-time streaming data.

Objectives

- You will touch the Lambda Architecture concept
- You will learn hype and trend Scala base-words
- You will taste ML in production
- It will be hard for you but not boring

Deadline

Upload your report into moodle before 3:14 AM, 29 April

- Note: 3:14 of night from Sunday to Monday.

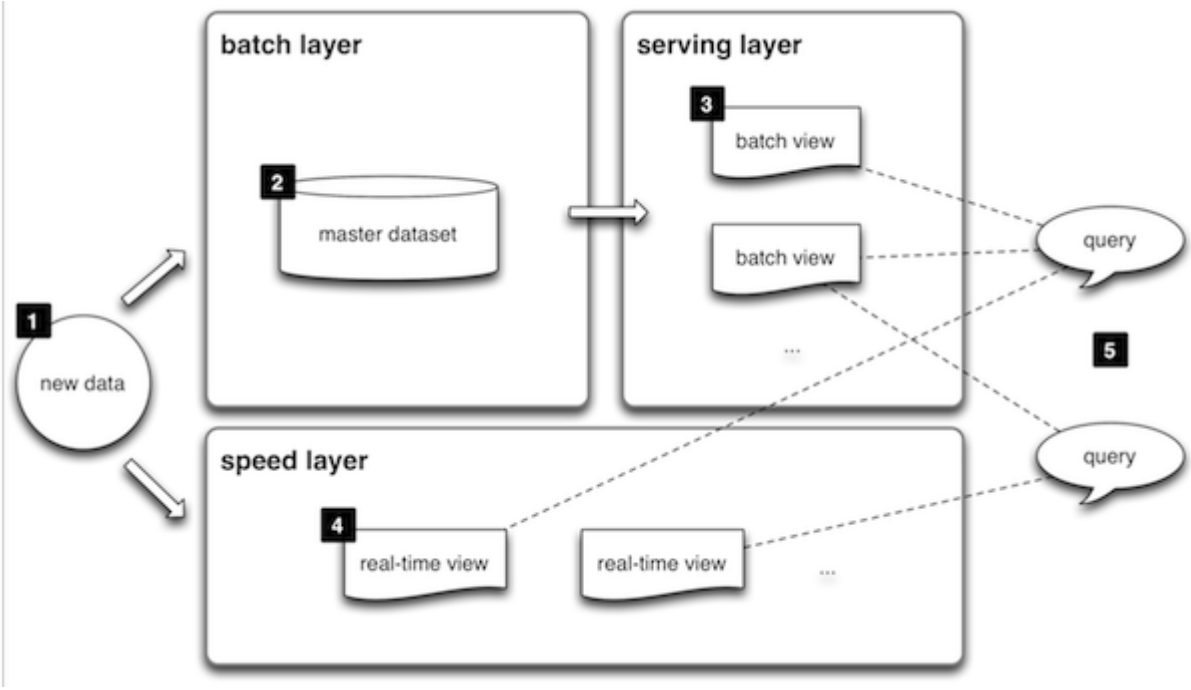
Task overview

We implement an application that will parse twitter messages, process them on the fly, stored unprocessed data in database for the future handling and publish already processed data infographics to Web Interface.

Lambda Architecture

What is the Lambda Architecture? According to Wikipedia: Lambda architecture is a data-processing architecture designed to handle massive quantities of data by taking advantage of both batch- and stream-processing methods. For more information you can use one of these sources * Russian * English

It can be illustrated as follows



Using the Twitter real-time data as *new-data* (look at the picture above) we will try to construct the whole cycle.

Thus, at first we have to register our first Twitter App. And yes, you have to register Twitter Account and verify it.

- Log in your verified twitter account
- Go to twitter applications managment-page and Create New App

Hint: you can use this manual

- After that you have to request twitter application tokens: CONSUMER_KEY, CONSUMER_SECRET, ACCESS_TOKEN, ACCESS_TOKEN_SECRET

These keys will be used on the Data Streaming step.

Data streaming

Now we have to familiarize with one of the powerfull tool for streaming.

(*drumroll*) (*tucket*) *Ladies and gentlemens Apache Spark Streaming!*

But first there will be some history. There are roughly two principal approaches to the task of processing large arrays of information - Data Parallelizm and Task Parallelizm. Apache Spark Streaming works with Task parallelism. Why we have chosen this type of parallelism - this you will consider in your report.

Now let's create our first Spark Streaming Demo.

```
import org.apache.spark.streaming.twitter.TwitterUtils
import org.apache.spark.streaming._
```

```
import org.apache.spark.SparkConf

object TwitterDemo extends App {

System.setProperty("twitter4j.oauth.consumerKey", "CONSUMER_KEY")
System.setProperty("twitter4j.oauth.consumerSecret", "CONSUMER_SECRET")
System.setProperty("twitter4j.oauth.accessToken", "ACCESS_TOKEN")
System.setProperty("twitter4j.oauth.accessTokenSecret", "ACCESS_TOKEN_SECRET")

  val sc = new SparkConf().setAppName("demo").setMaster("local\[2\]") // local
  val ssc = new StreamingContext(sc, Seconds(15))
  val stream = TwitterUtils.createStream(ssc, None)


  case class Tweet(createdAt:Long, text:String)
  val twits = stream.window(Seconds(60)).map(m=>
    Tweet(m.getCreatedAt().getTime()/1000, m.toString)
  )


  twits.foreachRDD(rdd => rdd.collect().foreach(println))

  ssc.start()
  ssc.awaitTermination()
}
```

This short listing is you starting point of the project.

As you can see, we used Twitter Application Access Tokens. Now you have to create Streaming of real-time English text tweets from the USA users. If have managed with that - you already have C mark for the project.

Speed Layer

The Speed Layer is responsible for processing the data entering the system in real time. It is a collection of data warehouses in which they are in the queue, in the stream or in the operating mode.

This is the most interesting part of our project, because here we will use several technologies at the same time.

Because of our messages have to be proccesed as queue we need something what works with queues. And this is Apache Kafka of cause. The idea behind it that we have to accumulate data from Spark Streaming into queue, pass through our 'Basic ML Model' and give data and prediction away upon request to frontend part.

To be true Apache Kafka can be used as data broker (glue) in our system, managing data flows. You can learn more here. But we we confine ourselves to the use of Apache Kafka only for this Speed Layer as a data Producer.

Frontend part is our Consumer. Consumer request classified data from the Data Producer. In our case Apache Kafka is our Data Producer. We will use Apache Akka for that. Akka has its own http server, we don't have to use something else. You have alreasy known how to build simople REST API service with Scalatra, so now it's time to use this knowledge. Our Akka consumer has to request data from the Apache Kafka and demonstrate it on localhost.

More about Apache Akka

Batch Layer

Batch layer is just a database which store new data and give it away upon request. Request in this case is a data request for new ML model or just historical data transfering. Stored Data is immutable and never deleted. For this part we will use Apache Cassandra You have to store each tweet during active session, and be ready giwe away any historical stored data. For instance it would be greate to see script for printing out first N tweets from your Cassandra database. (N - parameter)

Serving Layer

Serving Layer is the the very same 'request of data from the Batch Layer'. In our case it will training of a new 'Specialized ML Model'.

ML model

I haven't told about your. For the minimum acceptable version of your application you have to implement 'Basic ML Model'. Let it Naive Bayes for positive and negative tweets classification. Description of training dataset here. Direct download of Training Dataset by Stanford Link.

'Specialized ML Model' can be of your choice. One of the possible alternative is an attempt to identify geolocation clusters by the text of the message.

Functionality

- Application parse real time Twitter's Tweets (Part C)
  - Logged In your Twitter Account Applications
  - Print out in terminal real-time English text tweets from the USA users.
- Application demonstrate classified tweets on localhost (Part B - Speed Layer)
  - Kafka gets data from Streaming
  - Basic ML model classified tweets (Attention Basic ML Model must be pre-learned by you)
  - Kafka as the Data Producer give classified tweets away upon request of Consumer
  - Akka as the Consumer request classified tweets and demonstrate it on a simple web UI as flow of positive and negative tweets.
- Application demonstrate classified tweets from the database (Part B - Batch & Serving Layers)
  - Application store tweets from Streaming into Cassandra database
  - Specialized ML Model classify tweets from the database upon request (Specialized ML Model must be prelearned)

    - Application print out classified tweets in console
- Application demonstrate both branches of functionalyty (Part A - total functionality)
    - Application demonstrates classification by 'Basic ML Model' in first tab
    - Application retrain Basic Model using stored tweets in real-time and store retrained model as Second Basic ML Model
    - Application demonstrates classification by 'Second Basic ML Model' in second tab
    - (Optional for) Application demonstrates classification by 'Specialized ML Model' in a third tab

Criteria of done

- Code on github
- Project could be built using instructions in your `README.md` in repository
- Make a small report that points out what was done (which functionality is working), not more then 1 page A4. Add link to your github repo into report.
    - If you haven't managed something to work please write why, so we can cover this material during the course.
    - [Optional] If you used some additional tools/materials/guides, please mention them in the report, it could be helpful for other students and try to connect the problem you was solving and material that you was using.

Grading Policy

- Implemented Part C + Brief report of Scala features used in Part C = C mark
- Implemented Part B Speed Layer + Brief report of Scala features used in Part B and Part C = B mark
- Implemented Part B Batch & Serving Layers + Brief report of Scala features used in Part B and C = B mark
- Implemented Part A + Brief report Part B and Part C and Part A = A mark

Hint! On each step you can exchange report by post in Social Media about your Scala Course experience and Scala Course Qualifying Project. It can be Facebook, Vk, Medium, etc.

P.S. you can mention in your post not only instructor, but your TAs too. =)

If you have any questions - write me in Telegram (@cubazis), or you can find my in a cube near 419 aud.