"Did they meet?" Challenge

Claudio SCALZO

May 8, 2018

Abstract

Small report about the "Did they meet?" challenge.

1 Definition of meeting

A main part of the problem consisted in deciding and making concrete the definition of *"meeting"*. Of course, it would be wrong to refer to a meeting as the exact collision of two userIDs in the same x, y, f and t!

The decision taken to face the problem was to think and approximate a **range** in which two people can met. This is a simple parameter (degree of freedom) available in the code, and I set it to 3 meters.

The range, however, can't be only spatial, it has to be also temporal: let's say indeed, that the *user1* in a given second *t1* is in the position *pos1*, and the *user2* is, a second after (*t1+1*), in the same spatial range of the user1. They obviously met. However, an algorithm with strict temporal comparisons, would have considered them as never met, because the comparison couldn't find a compatible *t1 == t2*. This would work only if we had the explicit constraint that in the dataset, for **all seconds available**, we would have had the position of **every** user. In this case, the constraint is **not given**.

So, the final solution consists in having **both** the spatial and temporal ranges to determine an hit between two users.

2 Build the code

I worked with *Scala* and *sbt*. To compile the code, the requirements are just to have *Java* and *sbt* installed. The code is equipped with both the *build.sbt* file and the assembly plug-in so a simple command will build the program and take care of all the dependencies. The command to be run (in the *code* directory of the project) is:

```
sbt assembly
```

Then, a fat-jar will be created in the *target* directory. It can be executed with the *java -jar* command, including also the three arguments that it expects.
For example:

```
java -jar claudioscalzotest-assembly-1.0.jar reduced.csv b20e5d99 5e7b40e1
```

In the next page, you can find the explanation of the algorithmic approach and some examples of the execution of the program.

3 The approach to the problem

For the sake of dealing with a big quantity of data (even if not extremely high), I mainly used *Apache Spark* and, specifically, the *Spark SQL* and the *DataFrame* data-structure.
It seemed to me a good approach to face the problem, making fast and reliable the computation of the people "meetings".

The code is easy to understand and can be resumed in the following steps:

- Import of the CSV into a DataFrame
- Filtering of the DataFrame (keep only the two users)
- Simplification of the DataFrame (timestamp expansion, etc.)
- Self-join (with proper rules) to compute the pairs and keep only the ones in the same spatial and temporal ranges

4 Run examples

In the following part you can find an example of the result of the code.
It's important to remark that the visualization is done is a way that prevents some rows with the same date and hour to show up multiple times only because they differ for seconds or fractions of seconds. The **granularity** of the final view has been indeed reduced to *hh:mm* on purpose.

In the first case the program finds some hits:

```
claudio > java -jar claudioscalzochallenge-assembly-1.0.jar reduced.csv b20e5d99 5e7b40e1

################################################
[Program started]

-> I'll find if the user b20e5d99 and the user 5e7b40e1 met each other.
   Wait a second...

-> Loading the dataset...

-> Building the Spark Session...
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
-
> Preparing the dataset for the computation...

-> Computing the hits history...


=> YES, they've met each other!
   Just a moment, I show you when and where...

+---------+---------+------+-----+-----+
|date     |hourOfDay|x     |y    |floor|
+---------+---------+------+-----+-----+
|19-7-2014|19:46    |109.42|66.67|2    |
|19-7-2014|20:59    |105.25|88.25|2    |
|19-7-2014|21:10    |106.0 |88.0 |2    |
|19-7-2014|21:15    |106.33|87.67|2    |
+---------+---------+------+-----+-----+
```

In the following case, instead, not:

```
claudio > java -jar claudioscalzochallenge-assembly-1.0.jar reduced.csv ed3c7f8f 167990a4

################################################
[Program started]

-> I'll find if the user ed3c7f8f and the user 167990a4 met each other.
   Wait a second...

-> Loading the dataset...

-> Building the Spark Session...
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
-
> Preparing the dataset for the computation...

-> Computing the hits history...


=> NO! Never met each other.
################################################
```