Lab 2 : Wikipedia

Big Data Analysis

Quentin Vaucher, André Neto da Silva, Sylvain Renaud

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

April 4, 2019

Contents

1 Raw results

The list of retrieved languages is the same independently from the method which is used. This ascertainment seems logic since each attempt tries to compute the same result changing only the way the computations are done.

The following table presents the result obtained, regardless of the attempt.

Table 1: List of languages ranked by number of articles

Rank	Language	# articles
1	JavaScript	1704
2	C#	731
3	Java	700
4	CSS	430
5	Python	409
6	C++	384
7	PHP	333
8	MATLAB	296
9	Perl	176
10	Ruby	161
11	Haskell	65
12	Objective-C	62
13	Scala	53
14	Clojure	29
15	Groovy	29

The analysis becomes more interesting when it targets the time of computations. Here are the comparison of the three different attempts:

Attempt name	Time [s]
Naive	81121
Inverted index	9463
Reduce by key	6359

2 Interpretations

Let's try to understand these results one by one, beginning by the naive implementation.

2.1 Naive

For each language, the code go through the entire RDD and count the number of articles containing its name. Therefore the computations are the followings:

$$N * M$$

where:

- N is the number of languages
- M is the size of the RDD

2.2 Inverted index

For each language, the code has to calculate the size of its inverted index. Therefore, the computations are the followings:

$$\sum_{i=0}^{N} size\,of\,inverted\,index\,i$$

where:

- N is the number of Languages

Compared to the naive implementation, we don't have to go through the entire RDD, and don't ever bother to check which article contains which language because this information is already known.

2.3 Reduce by key

Same idea than the inverted index, except that we do all the computations in “once”. The performance are slightly better than the inverted index.

It seems that the `reduceByKey` method is more optimized by *Spark* than computing an inverted index by hand, hence the lower computation time…