Big Data Analytics
Lab 02 - Wikipedia

Damien Rochat, Dorian Magnin, and Nelson Rocha
Master of Science in Engineering
University of Applied Sciences and Arts Western Switzerland

April 4, 2019

1 Results

Here is the number of articles where each programming language has been found. Note, the comparison is made by lowercase.

Rank	**Language**	**Number of articles**
1	JavaScript	1’721
2	C#	707
3	Java	618
4	CSS	400
5	C++	335
6	Python	315
7	MATLAB	307
8	PHP	302
9	Perl	167
10	Ruby	125
11	Haskell	56
12	Objective-C	47
13	Scala	44
14	Clojure	26
15	Groovy	26

Table 1: Wikipedia lab ranking

Rank	**Language**
1	JavaScript
2	Java
3	Python
4	PHP
5	C#
6	C++
7	CSS
8	Ruby
9	C
10	Objective-C
11	Swift
12	Scala
13	Shell
14	Go
15	R
16	TypeScript
17	PowerShell
18	Perl
19	Haskell
20	Lua

Table 2: RedMonk top-20 ranking (June 2018)

Except for JavaScript, large winner in both cases, the Wikipedia ranking doesn’t match RedMonk one. However, the top-5 of the two rankings is almost the same, in different order.

2 Performances

Iteration	**Time**
Naive ranking	54’571 ms
Ranking using inverted index	26’538 ms
Ranking using reduceByKey	23’593 ms

Table 3: Execution times

These are our obtained execution times.
Let’s take the *Naive ranking* time as a base, and see how we manage to speed it up.

At the second step we managed to get a speedup of a roughly 50%.
This is due to the inverted index use. By providing us with the information of which article contains which language, it spares us the work of parsing all the RDD.
So the results are far quicker to retrieve.

Finally, we achieved a bit more speedup by using the ‘reduceByKey‘ instruction.
By reducing before the shuffling, we greatly reduce the amount of data that will be sent through the network.
As we are working on our own machines, the beneficial effects of this change are not very important.
But if we were using a remote Spark cluster, the improvements would have been far more visible.

Finally, we wondered if there was nothing we could do to improve our execution times in this specific configuration, which is by having all running in a single machine?

We manage to use all the logical cores seen by the underlying JVM, by declaring our context with

```
val conf: SparkConf = new SparkConf()
    .setAppName("wikipediaArticle")
    .setMaster(s\"local[\${Runtime.getRuntime.availableProcessors()}]\")
```

instead of

```
val conf: SparkConf = new SparkConf()
    .setAppName("wikipediaArticle").setMaster(s"local[1]")
```

And we got the following execution times:

Iteration	**Time**
Naive ranking	45’991 ms
Ranking using inverted index	25’603 ms
Ranking using reduceByKey	22’242 ms

Table 4: Best execution times obtained