

# BIG DATA

## TP - Page Rank

Nicolas LAGAILLARDIE  
Paul BREUGNOT

11 octobre 2018

### Table des matières

<b>1</b>	<b>Objectif</b>	<b>2</b>
1.1	Définitions . . . . .	2
<b>2</b>	<b>Installation de Spark</b>	<b>2</b>
<b>3</b>	<b>Fonctionnement de PageRank</b>	<b>2</b>
<b>4</b>	<b>Implémentation de PageRank</b>	<b>3</b>
<b>5</b>	<b>Résultats et bilan</b>	<b>4</b>

# 1 Objectif

Implémenter une première version de PageRank en Scala sur Spark.

## 1.1 Définitions

**Spark** Framework open source de calcul distribué. Il s'agit d'un ensemble d'outils et de composants logiciels structurés selon une architecture définie.

**Scala** Langage de programmation multi-paradigme conçu pour exprimer les modèles de programmation courants dans une forme concise et élégante. Son nom vient de l'anglais Scalable language. Il peut être vu comme un métalangage.

**PageRank** Cet algorithme produit une distribution de probabilité utilisée pour représenter la probabilité qu'une personne cliquant au hasard sur des liens arrive sur une page particulière.

## 2 Installation de Spark

Sur Ubuntu, il n'y a pas réellement d'installation à réaliser. Il suffit de télécharger une version de Spark sur le site officiel <http://spark.apache.org>, puis de lancer des exemples ou la commande suivant :

```
$ spark-shell -i <file.scala>
```

*spark-shell* permet d'accéder au shell de Spark puis de lancer diverses commandes en langage *Scala*.

## 3 Fonctionnement de PageRank

PageRank tente de résoudre le problème suivant : à partir d'un ensemble de pages liées entre elles (cf Figure 1), on désire connaître la probabilité qu'un utilisateur tombe sur une page au hasard.

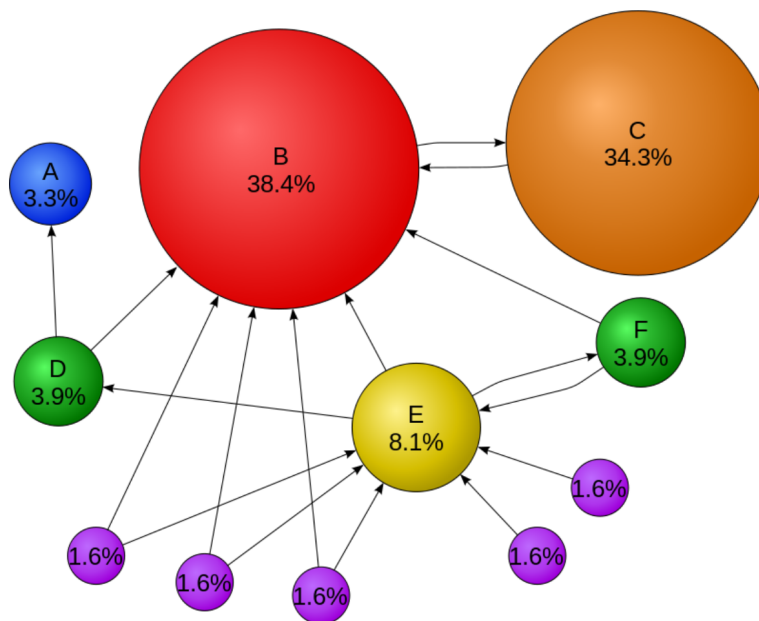


FIG. 1 – Schéma d'un set d'URLs

Et voici le principe de fonctionnement de PageRank.

a.

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

La valeur du PageRank pour une page  $u$  dépend des valeurs du PageRank pour chaque page  $v$  contenue dans l'ensemble  $B_u$  (l'ensemble contenant toutes les pages liées à la page  $u$ ), divisé par le nombre  $L(v)$  de liens de la page  $v$ .

b. L'algorithme itératif est le suivant :

A  $t = 0$ , une distribution de probabilité est initialisée de la sorte :

$$PR(p_i; 0) = \frac{1}{N}$$

c. A chaque étape  $t$ , voici comment est calculée la nouvelle valeur du PageRank pour chaque page :

$$PR(p_i; t+1) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j; t)}{L(p_j)}$$

Pour des raisons de précision et de facilité de code, nous avons décidé de tout multiplier par  $N$ .

## 4 Implémentation de PageRank

Nous avons écrit un script *Scala* qui, à partir des fichiers d'exemple *example\_arcs* et *example\_index*, ainsi qu'un nombre d'itérations, calcule le PageRank de chacune des pages recensées dans le second fichier.

a. Récupération du nombre d'itérations à réaliser

```
val iters = if (args.length > 1) args(2).toInt else 10
```

b. Récupération des arcs entre les différentes URLs

```
val lines = spark.read.textFile(args(0)).rdd
val links = lines.map{ s =>
  val parts = s.split("\\s+")
  (parts(0), parts(1))
}.distinct().groupByKey().cache()
```

c. Assignment des poids initiaux aux différents éléments

```
var ranks = links.mapValues(v => 1.0)
```

d. Récupération des URLs dans l'ordre

```
val linesIndex = spark.read.textFile(args(1)).rdd
val index = linesIndex.map{ s =>
  val parts = s.split("\\s+")
  (parts(0))
}.distinct().cache()
val URLs = index.collect()
```

e. Calcul de chaque nouvelles valeurs du PageRank pour l'ensemble des itérations

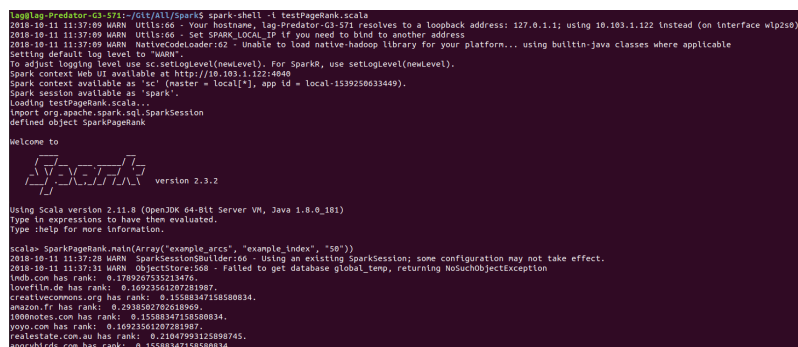
```
for (i <- 1 to iters) {
  val contribs = links.join(ranks).values.flatMap{ case (urls, rank) =>
    val size = urls.size
    urls.map(url => (url, rank / size))
  }
  ranks = contribs.reduceByKey(_ + _).mapValues(0.15 + 0.85 * _)
}
```

f. Affichage des résultats

```
val output = ranks.collect()
output.foreach{
  tup =>
  val element = URLs(tup._1.toInt)
  println(element + s" has rank: ${tup._2}.")
}
```

## 5 Résultats et bilan

Voici une partie des résultats lorsqu'on lance le script avec les fichiers exemples.



```
Laurent@Predator-Q3-S71:~/git/all/spark$ spark-shell -i testPageRank.scala
2018-10-11 11:37:09 WARN Utils:06 - Your hostname, lag-predator-Q3-S71 resolves to a loopback address: 127.0.1.1; using 10.103.1.122 instead (on interface wlp2s0)
2018-10-11 11:37:09 WARN Utils:06 - Set SPARK_LOCAL_IP if you need to bind to another address
2018-10-11 11:37:09 WARN NativeCodeLoader:02 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://10.103.1.122:4040
Spark context available as 'sc' (master = local[*], app id = local-1539250633449).
Spark session available as 'spark'.
Loading testPageRank.scala...
import org.apache.spark.sql.SparkSession
defined object SparkPageRank
Welcome to
Spark version 2.3.2
Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_181)
Type in expressions to have them evaluated.
Type :help for more information.
scala> SparkPageRank.main(Array("example_arcs", "example_index", "50"))
2018-10-11 11:37:28 WARN SparkSessionBuilder:06 - Using an existing SparkSession; some configuration may not take effect.
2018-10-11 11:37:33 WARN ObjectStore:50 - Failed to get database global_temp, returning HadoopObjectException
indb.com has rank: 0.1789267535213476
Overlin.de has rank: 0.16923561207281987
creativescommons.org has rank: 0.15588347158580834
amazon.fr has rank: 0.2938502702618949
100notes.com has rank: 0.15588347158580834
yoyo.com has rank: 0.16923561207281987
reatestate.com.au has rank: 0.21047993120308745
mpr9birds.com has rank: 0.15588347158580834
```

FIG. 2 – Résultats de notre PageRank

Les résultats semblent concluants, en effet, “à la main”, on constate que les pages qui possèdent le plus d’entrées sont le plus susceptible d’être visitées, et vice-versa. PageRank est donc un puissant outil, et nous commençons à comprendre le principe de fonctionnement des moteurs de recherche tels que *Google* ou *Ecosia*.