

# MapReduce pour les graphes

Nicolas Dugué  
nicolas.dugue@univ-orleans.fr

M2 MIAGE  
Systèmes d'information répartis

# Plan

- 1 Introduction Spark
- 2 Spark avec des graphes
- 3 Composante Connexe
- 4 PageRank

# Introduction

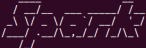
## Spark

- MapReduce mais plus encore : join, flatMap, groupByKey, ...
- In-memory
- Distribué mais aussi en multi-coeurs
- Plus efficace qu'Hadoop
- Utiliser HDFS ou non

# Introduction

## Spark - Shell Scala

```
nicolas@nicolas-Latitude-E5520:~/Cours/M2/NO5QL/spark-1.1.0-bin-hadoop2.4$ ./bin/spark-shell --master local[2]
Spark assembly has been built with Hlve, including Datanucleus jars on classpath
Java HotSpot(TM) Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
14/11/23 17:41:22 INFO SecurityManager: Changing view acls to: nicolas,
14/11/23 17:41:22 INFO SecurityManager: Changing modify acls to: nicolas,
14/11/23 17:41:22 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(nicolas, ); user
s with modify permissions: Set(nicolas, )
14/11/23 17:41:22 INFO HttpServer: Starting HTTP Server
14/11/23 17:41:22 INFO Utils: Successfully started service 'HTTP class server' on port 41753.
Welcome to

 version 1.1.0

Using Scala version 2.10.4 (Java HotSpot(TM) Server VM, Java 1.8.0_20)
```

## Spark - Shell Scala

- Pas de configuration
- Pas de filesystem distribué
- utilisation en multi-coeurs

# Plan

1 Introduction Spark

2 Spark avec des graphes

3 Composante Connexe

4 PageRank

# Un graphe "deuxComposantes"

## La liste d'arcs

1;2

2;3

3;4

6;7

8;7

8;6

# Un graphe "deuxComposantes"

## La liste d'adjacence

1, [2]  
2, [1, 3]  
3, [2, 4]  
4, [2, 3]  
6, [7, 8]  
7, [6, 8]  
8, [6, 7]

# Un graphe "deuxComposantes"

```
scala> val graphFile = sc.textFile("deuxComposantes")
```

```
graphFile: org.apache.spark.rdd.RDD[String]
```



# Un graphe "deuxComposantes"

```
scala> graphFile.collect()
```

```
Array[String] = Array(1;2, 2;3, 3;4, 6;7, 8;7, 8;6)
```

# Un graphe "deuxComposantes"

```
scala> val graphRdd=graphFile.flatMap(x => Seq((x.split(";")(0).toInt,  
x.split(";")(1).toInt),(x.split(";")(1).toInt, x.split(";")(0).toInt) ))
```

```
graphRdd: org.apache.spark.rdd.RDD[(Int, Int)]
```

# Un graphe "deuxComposantes"

```
scala> graphRdd.collect()
```

```
Array[(Int, Int)] = Array((1,2), (2,1), (2,3), (3,2), (3,4), (4,3), (6,7), (7,6),  
(8,7), (7,8), (8,6), (6,8))
```

# Un graphe "deuxComposantes"

```
scala> var graphAdj=graphRdd.groupByKey()
```

```
org.apache.spark.rdd.RDD[(Int, Iterable[Int])]
```

# Un graphe "deuxComposantes"

```
scala> graphAdj.collect()
```

```
Array[(Int, Iterable[Int])] = Array((4,CompactBuffer(3)),  
(6,CompactBuffer(7, 8)), (8,CompactBuffer(7, 6)), (2,CompactBuffer(1,  
3)), (1,CompactBuffer(2)), (3,CompactBuffer(2, 4)),  
(7,CompactBuffer(6, 8)))
```

# Plan

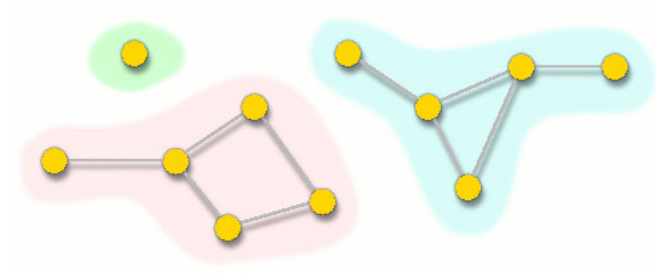
1 Introduction Spark

2 Spark avec des graphes

**3 Composante Connexe**

4 PageRank

# Objectif



# Un graphe simple

1;2  
2;3  
4;5

Chaque sommet est dans sa composante



## Liste d'adjacence + composante

1, [2], 1  
2, [1,3], 2  
3, [2], 3  
4, [5], 4  
5, [4], 5

## Map

Pour chaque noeud  $n$  dans une composante  $c$ , pour tout  $v$  voisin de ce noeud, on produit  $(v, \min(v, n, c))$

Map(1, [2], 1)

(2,1)

## Liste d'adjacence + composante

1, [2], 1  
2, [1,3], 2  
3, [2], 3  
4, [5], 4  
5, [4], 5

## Map

Pour chaque noeud  $n$  dans une composante  $c$ , pour tout  $v$  voisin de ce noeud, on produit  $(v, \min(v, n, c))$

## Map(2, [1,3], 2)

(1,1)  
(3,2)

## Liste d'adjacence + composante

1, [2], 1  
2, [1,3], 2  
3, [2], 3  
4, [5], 4  
5, [4], 5

## Map

Pour chaque noeud  $n$  dans une composante  $c$ , pour tout  $v$  voisin de ce noeud, on produit  $(v, \min(v, n, c))$

Map(3, [2], 3)

(2,2)

## Liste d'adjacence + composante

1, [2], 1  
2, [1,3], 2  
3, [2], 3  
4, [5], 4  
5, [4], 5

## Map

Pour chaque noeud  $n$  dans une composante  $c$ , pour tout  $v$  voisin de ce noeud, on produit  $(v, \min(v, n, c))$

Map(4, [5], 4)

(5,4)

## Liste d'adjacence + composante

1, [2], 1  
2, [1,3], 2  
3, [2], 3  
4, [5], 4  
5, [4], 5

## Map

Pour chaque noeud  $n$  dans une composante  $c$ , pour tout  $v$  voisin de ce noeud, on produit  $(v, \min(v, n, c))$

Map(5, [4], 5)

(4,4)

## Liste d'adjacence + composante

1, [2], 1  
2, [1,3], 2  
3, [2], 3  
4, [5], 4  
5, [4], 5

## Après le map

(2,1)  
(1,1)  
(3,2)  
(2,2)  
(4,4)  
(5,4)

## Liste d'adjacence + composante

1, [2], 1  
2, [1,3], 2  
3, [2], 3  
4, [5], 4  
5, [4], 5

## Avant le Reduce

(2,[1,2])  
(1,1)  
(3,2)  
(4,4)  
(5,4)

## Liste d'adjacence + composante

1, [2], 1  
2, [1,3], 2  
3, [2], 3  
4, [5], 4  
5, [4], 5

## Reduce

Parmi la liste des composantes obtenues, on choisit l'entier minimum

## Après le Reduce

(2,[1,2]) -> (2,1)  
(1,1) -> (1,1))  
(3,2) -> (3,2)  
(4,4) -> (4,4)  
(5,4)-> (5,4)



**Liste d'adjacence + composante : avant**

1, [2], 1  
2, [1,3], 2  
3, [2], 3  
4, [5], 4  
5, [4], 5

**Liste d'adjacence + composante : après**

1, [2], 1  
2, [1,3], 1  
3, [2], 2  
4, [5], 4  
5, [4], 4

## Liste d'adjacence + composante : **après**

1, [2], 1  
2, [1,3], 1  
3, [2], 2  
4, [5], 4  
5, [4], 4

## Map(2, [1,3], 1)

(1,1)  
(3,1) → 3 dans la composante 1 après reduce

# Plan

1 Introduction Spark

2 Spark avec des graphes

3 Composante Connexe

4 PageRank

# Intuition



# Un graphe simple

A;B

A;C

PR initialisé à 1 pour tous les noeuds

$$\text{PR}(\text{Node}) = 0.15 + 0.85 * (\text{PR}(\text{voisin}_1)/\text{degré}(\text{voisin}_1) + \dots + \text{Pr}(\text{voisin}_n)/\text{degré}(\text{voisin}_n))$$

# Un graphe simple - Première itération

A, [B,C], 1

B, [A], 1

C, [A], 1

$$\text{PR}(\text{Node}) = 0.15 + 0.85 * (\text{PR}(\text{voisin}_1)/\text{degré}(\text{voisin}_1) + \dots + \text{Pr}(\text{voisin}_n)/\text{degré}(\text{voisin}_n))$$

$$\text{PR}(A) = 0.15 + 0.85 * (\text{PR}(B)/\text{degré}(B) + \text{Pr}(C)/\text{degré}(C)) = 0.15 + 0.85 * (1 + 1) = 1.85$$

$$\text{PR}(B) = 0.15 + 0.85 * (\text{PR}(A)/\text{degré}(A)) = 0.15 + 0.85 * (1 / 2) = 0.575 = \text{PR}(C)$$

## Un graphe simple - Seconde itération

A, [B,C], 1.85

B, [A], 0.575

C, [A], 0.575

$$\text{PR}(\text{Node}) = 0.15 + 0.85 * (\text{PR}(\text{voisin}_1)/\text{degré}(\text{voisin}_1) + \dots + \text{Pr}(\text{voisin}_n)/\text{degré}(\text{voisin}_n))$$

$$\text{PR}(A) = 0.15 + 0.85 * (\text{PR}(B)/\text{degré}(B) + \text{Pr}(C)/\text{degré}(C)) = 0.15 + 0.85 * (0.575 + 0.575) = 1.1275$$

$$\text{PR}(B) = 0.15 + 0.85 * (\text{PR}(A)/\text{degré}(A)) = 0.15 + 0.85 * (1.85/2) = 0.93625 = \text{PR}(C)$$

# En MapReduce

A, [B,C], 1

B, [A], 1

C, [A], 1

Map(A, [B,C], 1)

(B, 1/2)

(C, 1/2)

Map(B, [A], 1)

(A, 1)

Map(C, [A], 1)

(A, 1)



# En MapReduce

A, [B,C], 1  
B, [A], 1  
C, [A], 1

Après le map

(B, 1/2)  
(C, 1/2)  
(A, [1, 1])

# Reduce (by key)

A;B

A;C

Après le map

$$(B, 1/2) \Rightarrow 0.15 + 0.85 * 1/2$$

$$(C, 1/2) \Rightarrow 0.15 + 0.85 * 1/2$$

$$(A, [1, 1]) \Rightarrow 0.15 + 0.85 * (1 + 1)$$