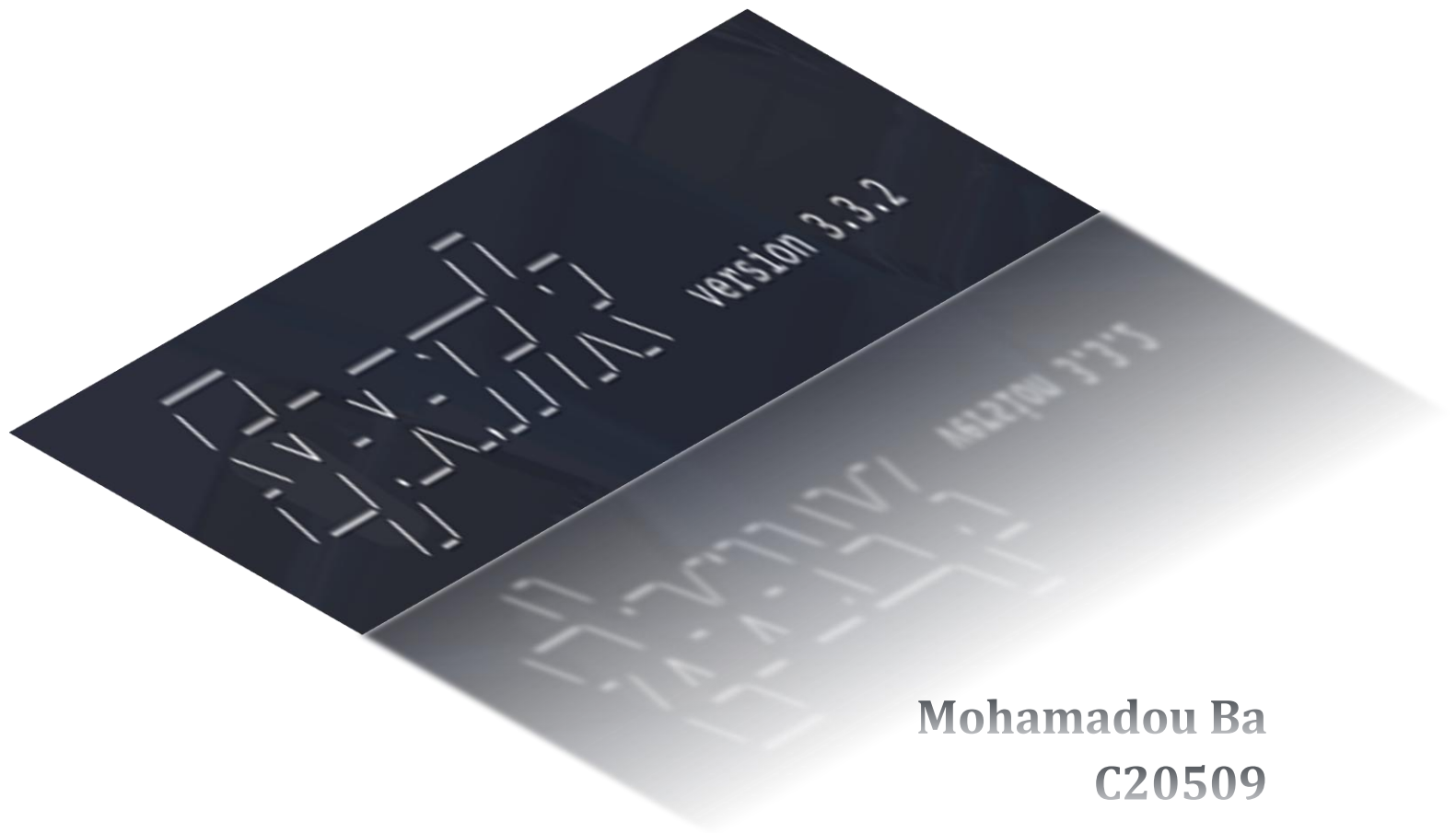


TP – Big Data Technologies

Traitement de Données avec Spark

Projet :

Recherche d'Amis Commun(s) entre deux utilisateurs
dans un réseau social



Mohamadou Ba
C20509

1. Introduction

Le but de ce projet est de développer une application distribuée avec Apache Spark permettant de détecter les amis communs entre deux utilisateurs à partir d'un fichier de données sociales 'soc-LiveJournal1Adj.txt', représentant un graphe d'amitié.

Chaque ligne du fichier indique un utilisateur et la liste de ses amis directs. À partir de ces données, notre objectif est de trouver la liste des amis communs pour une paire d'utilisateurs donnée.

2. Données utilisées

- Fichier : soc-LiveJournal1Adj.txt (environ 850 Mo)
- Format : <User_ID>\t<Friend_ID1>,<Friend_ID2>,...,<Friend_IDn>
- Les relations sont mutuelles : si A est ami avec B, alors B est ami avec A.

3. Méthodologie et outils

Langage : **Scala**

Framework : **Apache Spark**

Éditeur : **Nano**

OS : **Kali Linux (VirtualBox)**

JDK : **Java 11**

Scala version : 2.12.15

Spark version : 3.3.2

Étapes du traitement :

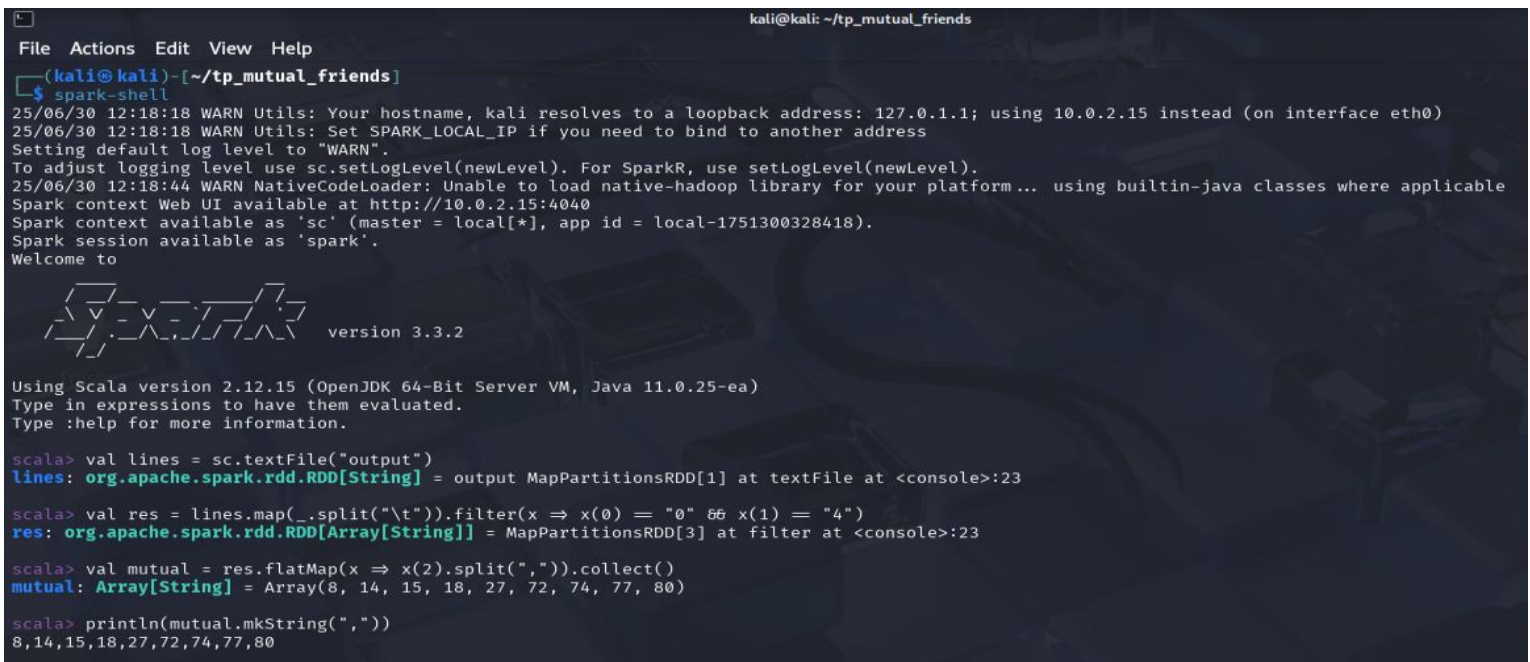
1. Lecture du fichier avec `sc.textFile`
2. Transformation des données en couples (utilisateur, ami)
3. Génération des paires (A, B) triées
4. Intersection des listes d'amis de chaque paire
5. Formatage du résultat

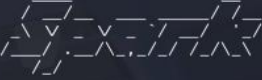
4. Code Scala principal

```
def pairs(str: Array[String]) = {  
  val user = str(0)  
  val friends = str(1).split(",")  
  for (f <- friends) yield {  
    val pair = if (user < f) (user, f) else (f, user)  
    (pair, friends)  
  }  
}  
  
val data = sc.textFile("soc-LiveJournal1Adj.txt")  
val data1 = data.map(_._split("\t")).filter(_._length == 2)  
val pairCounts = data1.flatMap(pairs).reduceByKey((a, b) => a.intersect(b))  
val p1 = pairCounts.map { case ((u1, u2), commons) =>  
  s"$u1\t$u2\t${commons.mkString(",")}" }  
p1.saveAsTextFile("output")
```

5. Résultats obtenus

Voici un exemple de résultat extraits via Spark Shell :



```
kali@kali: ~/tp_mutual_friends  
File Actions Edit View Help  
(kali@kali)~[~/tp_mutual_friends]  
$ spark-shell  
25/06/30 12:18:18 WARN Utils: Your hostname, kali resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface eth0)  
25/06/30 12:18:18 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
25/06/30 12:18:44 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
Spark context Web UI available at http://10.0.2.15:4040  
Spark context available as 'sc' (master = local[*], app id = local-1751300328418).  
Spark session available as 'spark'.  
Welcome to  
 version 3.3.2  
  
Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 11.0.25-ea)  
Type in expressions to have them evaluated.  
Type :help for more information.  
  
scala> val lines = sc.textFile("output")  
lines: org.apache.spark.rdd.RDD[String] = output MapPartitionsRDD[1] at textFile at <console>:23  
  
scala> val res = lines.map(_._split("\t")).filter(x => x(0) == "0" && x(1) == "4")  
res: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[3] at filter at <console>:23  
  
scala> val mutual = res.flatMap(x => x(2).split(",")).collect()  
mutual: Array[String] = Array(8, 14, 15, 18, 27, 72, 74, 77, 80)  
  
scala> println(mutual.mkString(","))  
8,14,15,18,27,72,74,77,80
```

6. Conclusion

Ce projet a permis de manipuler un grand fichier de graphe social avec Apache Spark, de mettre en œuvre une logique de MapReduce pour générer toutes les paires amies, et d'en extraire les amis communs via une intersection.

Il a illustré les avantages de Spark pour :

- Manipuler des données massives
- Réaliser des opérations complexes comme l'intersection
- Travailler de façon distribuée avec un code simple et lisible