



Etablissement : **Université de Nouakchott**.....  
.....Faculté des Sciences et Techniques ..

Département : .....**Informatique**.....

Référence : ... TP Big Data – Année Universitaire-**2024/2025**

## Rapport des Travaux Pratiques — Big Data

# Hadoop – MongoDB – Spark/Scala

Réalisé par : Ahmed Yahya SidiAba \_C2085

Encadrant: Dr.El3arbi.....

Date de remise: ...16/06/2025.....



## Table des matières

<i>Table des matières</i> .....	2
<i>Introduction generale</i> .....	4
<i>TP1 – Hadoop MapReduce</i> .....	5
Remarque importante .....	5
◆ Introduction .....	5
Partie 1 : Préparation de la plateforme de développement .....	5
1. installation de hadoop sous windows.....	5
2. lancement des services Hadoop .....	7
⊗ Étapes négligées (et pourquoi) : .....	9
Partie 2 : Exécution du compteur d’occurrences de mots .....	9
3. compilation avec maven.....	9
4. Copie du fichier texte dans HDFS .....	10
5. Vérification de la présence du fichier dans HDFS.....	11
6. execution du job Hadoop (mapreduce) .....	11
☑ 7. résultats du job.....	12
Conclusion .....	12
<i>TP2 – MongoDB (Base NoSQL)</i> .....	12
1. Introduction .....	12
2. Installation de MongoDB .....	13
3. Lancement du serveur et du client.....	13
4. Création de la base de données et insertion de documents .....	14
5. Requêtes de lecture et suppression.....	15
6. Conclusion.....	20
<i>TP-3 Spark : Détection d’amis communs dans un réseau social</i> .....	21

Partie 1 – TP Spark Scala : Amis communs .....	21
3.1 Introduction.....	21
3.2 Préparation de l’environnement.....	21
3.3 Description du fichier source .....	22
3.4 Étapes du traitement Spark .....	22
3.5 Extraction de cas spécifiques .....	24
3.6 Conclusion .....	27
Partie 2 – Détection des amis communs avec noms en Scala Spark .....	27
4.1 Introduction.....	27
4.2 Chargement et parsing .....	27
4.3 Création du mapping ID → Nom.....	28
4.4 Génération des paires triées (min, max).....	28
4.5 Calcul des amis communs.....	28
4.6 Fonction pour afficher amis communs par noms (exemple Mohamed & Sidi).....	28
4.7 Exemple d’utilisation .....	29
4.8 Captures d’écran démonstratives et résultats clés.....	29
4.9 Conclusion .....	31
<i>Conclusion generale.....</i>	<i>31</i>

# Introduction generale

Ce rapport présente une série de Travaux Pratiques réalisés dans le cadre de l'initiation aux technologies du **Big Data**. L'objectif est de se familiariser avec les outils et concepts fondamentaux permettant de manipuler et d'analyser de grandes volumétries de données, au-delà des capacités des systèmes classiques.

Les TP se sont articulés autour de trois technologies majeures :

- ✧ **Hadoop** pour le traitement distribué via le modèle **MapReduce**,
- ✧ **MongoDB**, base de données **NoSQL orientée documents**, adaptée au stockage flexible de données semi-structurées,
- ✧ **Apache Spark** avec **Scala**, pour le traitement distribué rapide en mémoire, et l'analyse de graphes sociaux via les **RDD**.

Chaque partie du rapport met en pratique des concepts clés comme la répartition des tâches, la tolérance aux pannes, l'optimisation des requêtes et l'analyse de relations complexes (ex : amis communs dans un réseau).

# TP1 – Hadoop MapReduce

## Traitement et Analyse de Fichiers Locaux sous Windows

### Remarque importante

L'utilisation directe d'Hadoop sur Windows a été privilégiée en raison des **limitations matérielles** de la machine utilisée, notamment la présence d'un **disque dur HDD lent**. En effet, l'utilisation d'une machine virtuelle GNU/Linux (comme demandé dans le TP)

### ◆ Introduction

Ce TP a pour objectif de mettre en œuvre un programme Hadoop Java pour compter les occurrences de mots dans un fichier texte via HDFS. Dans ce travail, toutes les étapes ont été réalisées sous Windows, sans utiliser de machine virtuelle (VM), ni connexion SSH, ni outil de transfert SCP.

## Partie 1 : Préparation de la plateforme de développement

✚ Étapes réalisées sous Windows

### 1. installation de hadoop sous windows

Pour utiliser Hadoop directement sur Windows, sans passer par une machine virtuelle ou un accès SSH, les étapes suivantes ont été effectuées :

1. Téléchargement et installation de Hadoop (version 3.3.6 compatible avec Java 8).
2. Configuration des variables d'environnement nécessaires :
  - a. JAVA\_HOME pointant vers l'installation de Java 1.8
  - b. HADOOP\_HOME pointant vers le dossier Hadoop
  - c. Ajout du dossier %HADOOP\_HOME%\bin au PATH Windows

### Commandes utilisées pour vérifier la configuration :

```
echo %JAVA_HOME%  
echo %HADOOP_HOME%
```

📷 **Capture 1** : Affichage des variables d'environnement correctement configurées.

```
C:\WINDOWS\system32\cmd.exe

C:\Users\Ahmed yahya>echo %JAVA_HOME%
C:\Program Files\Java\jdk1.8.0_202


C:\Users\Ahmed yahya>echo %HADOOP_HOME%
C:\hadoop-3.3.6\hadoop-3.3.6

C:\Users\Ahmed yahya>
```

## 1.1 vérification de l'installation Hadoop

Pour s'assurer que Hadoop est bien installé et fonctionnel, la commande suivante a été utilisée

### **hadoop version**

 **Capture 2** : Résultat de la commande **hadoop version** affichant la version de Hadoop ainsi que la version de Java utilisée.

```
C:\WINDOWS\system32\cmd.exe

Microsoft Windows [version 10.0.19045.5854]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\Ahmed yahya>hadoop version
Hadoop 3.3.6
Source code repository https://github.com/apache/hadoop.git -r 1be78238728da9266a4f88195058f08fd012bf9c

Compiled by ubuntu on 2023-06-18T08:22Z
Compiled on platform linux-x86_64
Compiled with protoc 3.7.1
From source with checksum 5652179ad55f76cb287d9c633bb53bbd
This command was run using /C:/hadoop-3.3.6/hadoop-3.3.6/share/hadoop/common/hadoop-common-3.3.6.jar

C:\Users\Ahmed yahya>
```

## 2. lancement des services Hadoop

Avant de démarrer Hadoop, il est nécessaire de configurer certains fichiers XML essentiels (situés dans le dossier **etc/hadoop**) :

- ❖ **core-site.xml** : Spécifie l'URI du **NameNode** (hdfs://localhost:9000)
- ❖ **hdfs-site.xml** : Définit les chemins de stockage des données HDFS
- ❖ **mapred-site.xml** : Configure MapReduce (à créer à partir d'un fichier modèle)
- ❖ **yarn-site.xml** : Configure les services YARN (ResourceManager, NodeManager)

Exemple d'extrait du fichier core-site.xml :


```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Ce fichier configure l'URI du **NameNode**, qui est ici défini sur localhost avec le port 9000. Cela permet à Hadoop de localiser le système de fichiers distribué (HDFS) sur la machine locale.

### Commandes utilisées pour démarrer Hadoop :


```
start-dfs.cmd
start-yarn.cmd
jps
```

- ➡ **start-dfs.cmd** : démarre les services HDFS (NameNode, DataNode)
- ➡ **start-yarn.cmd** : démarre les services YARN (ResourceManager, NodeManager)
- ➡ **jps** : liste les processus Java actifs pour vérifier que les services tournent

 **Capture 3** : Résultat de la commande jps montrant les services actifs (NameNode, DataNode, ResourceManager, NodeManager, SecondaryNameNode, etc.)

```
Administrateur : Invite de commandes
C:\hadoop-3.3.6\hadoop-3.3.6\sbin>jps
11048 ResourceManager
1836 NameNode
19276 Jps
3052 DataNode
5868 NodeManager

C:\hadoop-3.3.6\hadoop-3.3.6\sbin>
```

 **Capture 4** : Interface Web Hadoop accessible via <http://localhost:9870> (interface du NameNode)

Namenode information

localhost:9870/dfshealth.html#tab-overview

Gmail YouTube Maps شوار ينتهي لتبدأ الرحا... begin حساب Google ١٣ سبتمبر ٢٠٢٤... (323) (379) Build eComm... كل عام وانتم بخير... (381) Tous les favoris

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities

Overview 'localhost:9000' (✓active)

Started:	Thu Jun 12 18:53:16 +0000 2025
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 08:22:00 +0000 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-da15dbf4-ca6a-4d25-bee0-0279387fa860
Block Pool ID:	BP-237646920-192.168.222.1-1749580112602

Summary



⊗ Étapes négligées (et pourquoi) :

Étape	Description	Raison de la négligence
<b>VirtualBox</b>	Importer et exécuter la VM Debian	Machine lente (HDD) → VM inutilisable efficacement
<b>SSH avec PuTTY</b>	Connexion au terminal de la VM	Pas de VM utilisée, donc inutile
<b>WinSCP</b>	Copier les fichiers dans la VM	Fichiers gérés localement sous Windows


## Partie 2 : Exécution du compteur d'occurrences de mots

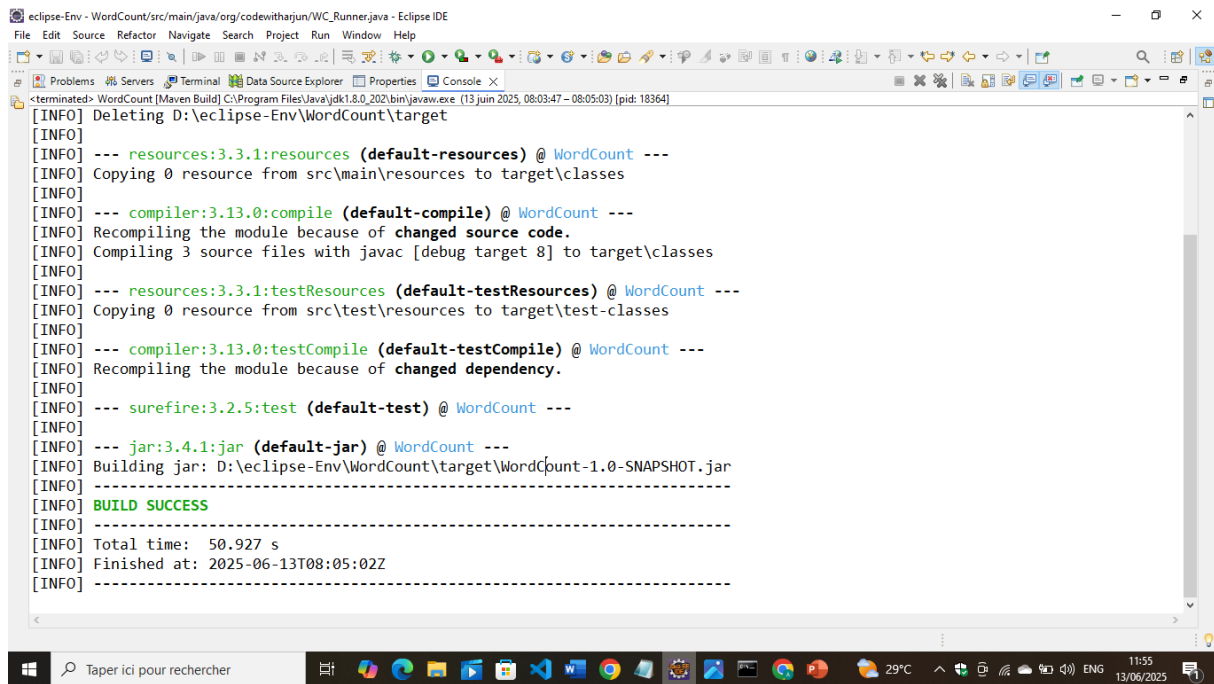
### 3. compilation avec maven

Nous avons créé un projet Maven nommé **WordCount** dans Eclipse, contenant les classes WC\_Runner, WC\_Reducer et WC\_Mapper.

**Commande exécutée dans Eclipse (Terminal Maven) :**

**Mavan run buils → Goals → clean package → run**

 **Capture 4** : Compilation réussie – le fichier JAR WordCount-1.0.0-SNAPSHOT.jar est bien généré dans le dossier target/.



```
eclipse-Env - WordCount/src/main/java/org/codewitharjun/WC_Runner.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help


[INFO] Deleting D:\eclipse-Env\WordCount\target
[INFO] --- resources:3.3.1:resources (default-resources) @ WordCount ---
[INFO] Copying 0 resource from src/main/resources to target/classes
[INFO] --- compiler:3.13.0:compile (default-compile) @ WordCount ---
[INFO] Recompiling the module because of changed source code.
[INFO] Compiling 3 source files with javac [debug target 8] to target/classes
[INFO] --- resources:3.3.1:testResources (default-testResources) @ WordCount ---
[INFO] Copying 0 resource from src/test/resources to target/test-classes
[INFO] --- compiler:3.13.0:testCompile (default-testCompile) @ WordCount ---
[INFO] Recompiling the module because of changed dependency.
[INFO] --- surefire:3.2.5:test (default-test) @ WordCount ---
[INFO] --- jar:3.4.1:jar (default-jar) @ WordCount ---
[INFO] Building jar: D:\eclipse-Env\WordCount\target\WordCount-1.0-SNAPSHOT.jar
[INFO] BUILD SUCCESS
[INFO] Total time: 50.927 s
[INFO] Finished at: 2025-06-13T08:05:02Z
```

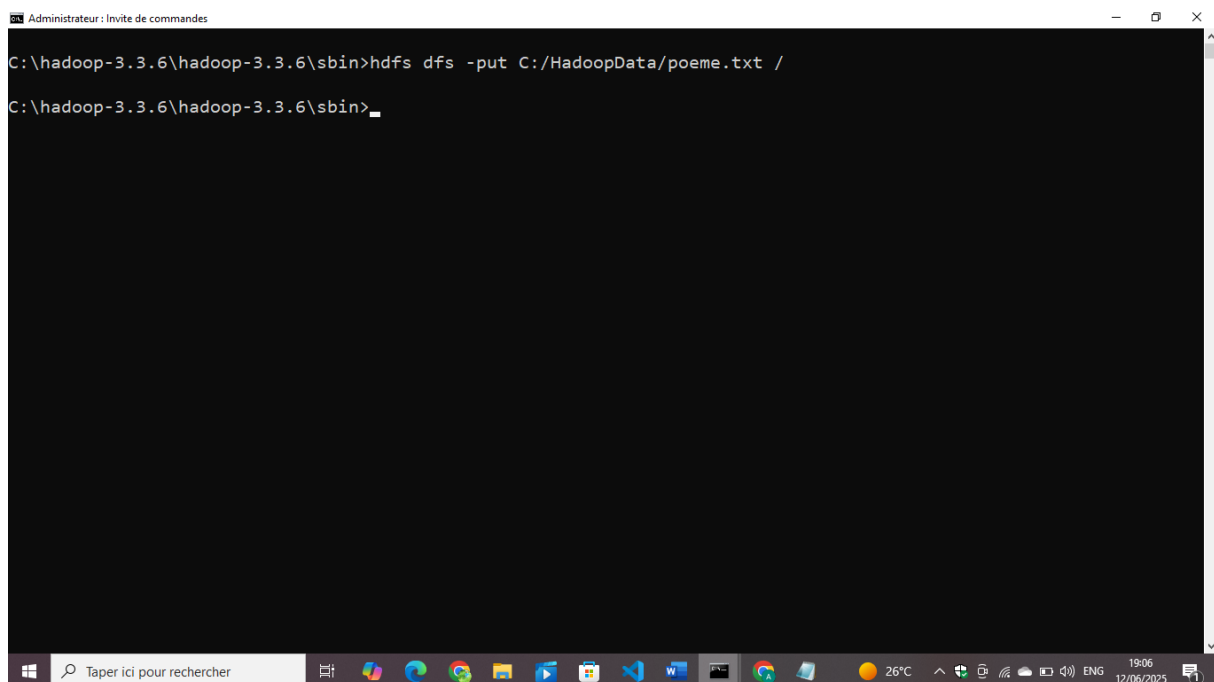
## 4. Copie du fichier texte dans HDFS

Le fichier poeme.txt est copié dans le système de fichiers HDFS.

**Commande :**

**hdfs dfs -put poeme.txt /**

 **Capture 5 :** Copie réussie du fichier poeme.txt dans HDFS.




```
Administrateur : Invite de commandes

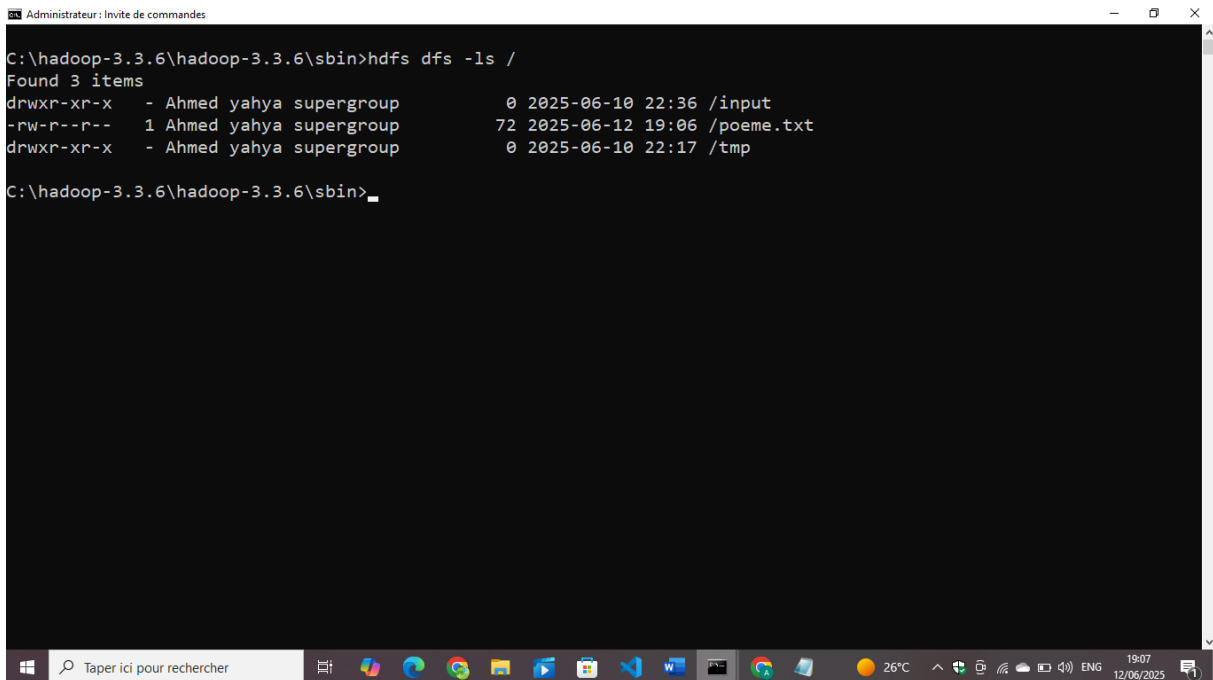
C:\hadoop-3.3.6\hadoop-3.3.6\sbin>hdfs dfs -put C:/HadoopData/poeme.txt /
C:\hadoop-3.3.6\hadoop-3.3.6\sbin>
```

## 5. Vérification de la présence du fichier dans HDFS

**Commande :**

`hdfs dfs -ls /`

 **Capture 6** : Le fichier poeme.txt est visible dans le répertoire racine de HDFS.




```
C:\hadoop-3.3.6\hadoop-3.3.6\sbin>hdfs dfs -ls /
Found 3 items
drwxr-xr-x - Ahmed yahya supergroup          0 2025-06-10 22:36 /input
-rw-r--r-- 1 Ahmed yahya supergroup          72 2025-06-12 19:06 /poeme.txt
drwxr-xr-x - Ahmed yahya supergroup          0 2025-06-10 22:17 /tmp
C:\hadoop-3.3.6\hadoop-3.3.6\sbin>
```

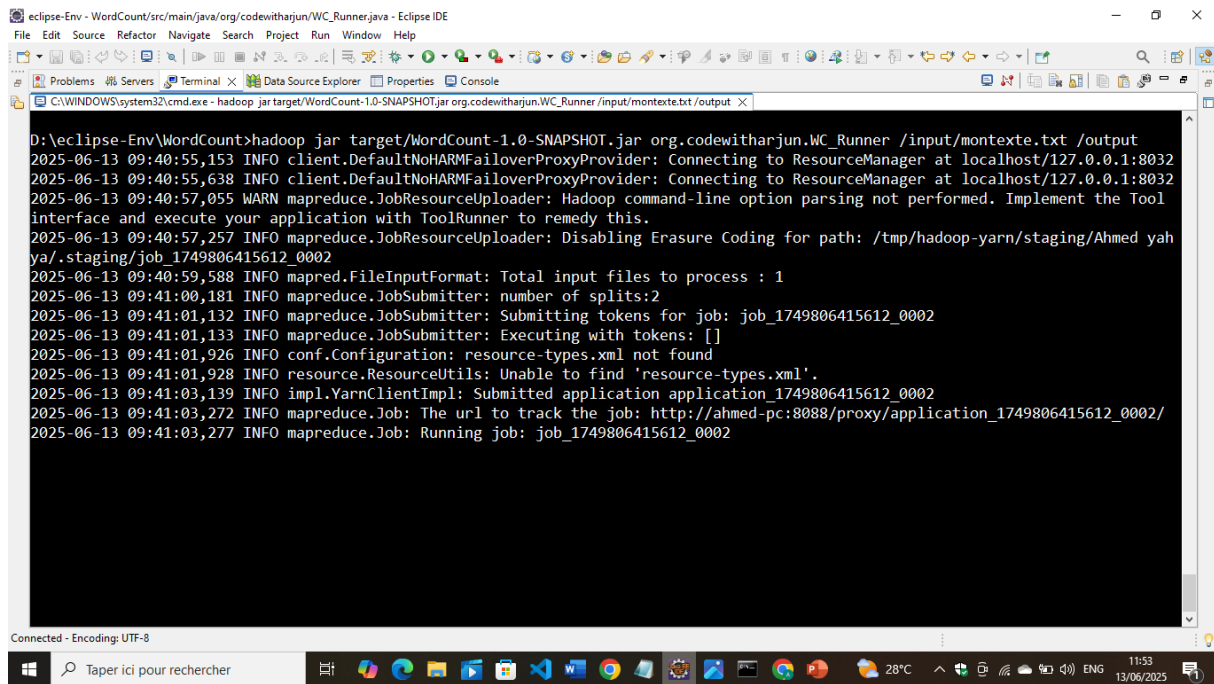
## 6. execution du job Hadoop (mapreduce)

Nous avons exécuté notre programme Hadoop **WCount** en utilisant le fichier JAR généré.

**Commande :**

`hadoop jar target/WordCount-1.0-SNAPSHOT.jar org.codewitharjun.WC_Runner  
/input/montexte.txt /output`

 **Capture 7** : Le job est exécuté avec succès et passe à l'état *Running* sur YARN.



```
D:\eclipse-Env\WordCount>hadoop jar target/WordCount-1.0-SNAPSHOT.jar org.codewitharjun.WC_Runner /input/montexte.txt /output
2025-06-13 09:40:55,153 INFO client.DefaultNoHARMAFailoverProxyProvider: Connecting to ResourceManager at localhost/127.0.0.1:8032
2025-06-13 09:40:55,638 INFO client.DefaultNoHARMAFailoverProxyProvider: Connecting to ResourceManager at localhost/127.0.0.1:8032
2025-06-13 09:40:57,055 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool
interface and execute your application with ToolRunner to remedy this.
2025-06-13 09:40:57,257 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/Ahmed yah
ya/.staging/job_1749806415612_0002
2025-06-13 09:40:59,588 INFO mapred.FileInputFormat: Total input files to process : 1
2025-06-13 09:41:00,181 INFO mapreduce.JobSubmitter: number of splits:2
2025-06-13 09:41:01,132 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1749806415612_0002
2025-06-13 09:41:01,133 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-06-13 09:41:01,926 INFO conf.Configuration: resource-types.xml not found
2025-06-13 09:41:01,928 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2025-06-13 09:41:03,139 INFO impl.YarnClientImpl: Submitted application application_1749806415612_0002
2025-06-13 09:41:03,272 INFO mapreduce.Job: The url to track the job: http://ahmed-pc:8088/proxy/application_1749806415612_0002/
2025-06-13 09:41:03,277 INFO mapreduce.Job: Running job: job_1749806415612_0002
```

## 7. résultats du job

### Commandes :

**hdfs dfs -ls /results**

**hdfs dfs -cat /results/\***

## Conclusion

Le TP a été effectué entièrement sur Windows pour des raisons de performance. Le job Hadoop a été compilé, lancé et vérifié avec succès. La plateforme est fonctionnelle sans passer par une VM grâce à une configuration adaptée aux contraintes matérielles.

# TP2 – MongoDB (Base NoSQL)

## 1. Introduction

Ce rapport présente un travail pratique sur la base de données NoSQL MongoDB (version 2.6.4). L'objectif de ce TP est d'apprendre à manipuler une base MongoDB à travers l'invite de commande. Les tâches réalisées incluent la création d'une base de données, l'insertion de documents, l'exécution de requêtes de lecture, et la suppression de documents. Ce TP constitue

une première prise en main essentielle des concepts fondamentaux de MongoDB, y compris le stockage de données sous forme de documents JSON.

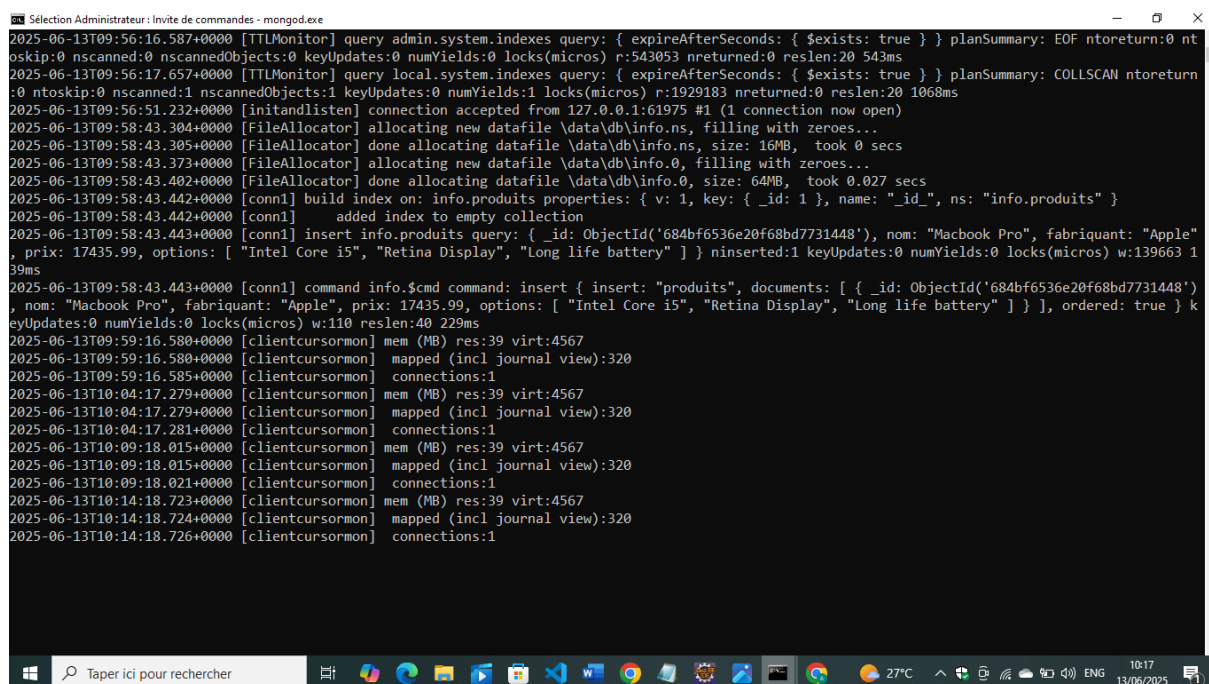
## 2. Installation de MongoDB

Le logiciel MongoDB a été téléchargé sous forme de fichier ZIP (version 2.6.4) depuis le lien fourni. Les étapes suivantes ont été suivies :


- Extraction de l'archive dans le répertoire C:\MongoDB
- Création des dossiers C:\data et C:\data\db pour stocker les données.

## 3. Lancement du serveur et du client

- Démarrage du serveur avec `mongod.exe`



```
2025-06-13T09:56:16.587+0000 [TTLMonitor] query admin.system.indexes query: { $exists: true } } planSummary: EOF nreturned:0 nt
oskip:0 nscanned:0 nscannedObjects:0 keyUpdates:0 numYields:0 locks(micros) r:543053 nreturned:0 reslen:20 543ms
2025-06-13T09:56:17.657+0000 [TTLMonitor] query local.system.indexes query: { $exists: true } } planSummary: COLLSCAN nreturned
:0 ntoskip:0 nscanned:1 nscannedObjects:1 keyUpdates:0 numYields:1 locks(micros) r:1929183 nreturned:0 reslen:20 1068ms
2025-06-13T09:56:51.232+0000 [initandlisten] connection accepted from 127.0.0.1:61975 #1 (1 connection now open)
2025-06-13T09:58:43.304+0000 [FileAllocator] allocating new datafile \data\db\info.ns, filling with zeroes...
2025-06-13T09:58:43.305+0000 [FileAllocator] done allocating datafile \data\db\info.ns, size: 16MB, took 0 secs
2025-06-13T09:58:43.373+0000 [FileAllocator] allocating new datafile \data\db\info.0, filling with zeroes...
2025-06-13T09:58:43.402+0000 [FileAllocator] done allocating datafile \data\db\info.0, size: 64MB, took 0.027 secs
2025-06-13T09:58:43.442+0000 [conn1] build index on: info.produits properties: { v: 1, key: { _id: 1 }, name: "_id_", ns: "info.produits" }
2025-06-13T09:58:43.442+0000 [conn1] added index to empty collection
2025-06-13T09:58:43.443+0000 [conn1] insert info.produits query: { _id: ObjectId('684bf6536e20f68bd7731448'), nom: "Macbook Pro", fabricant: "Apple"
, prix: 17435.99, options: [ "Intel Core i5", "Retina Display", "Long life battery" ] } ninserted:1 keyUpdates:0 numYields:0 locks(micros) w:139663 1
39ms
2025-06-13T09:58:43.443+0000 [conn1] command info.$cmd command: insert { insert: "produits", documents: [ { _id: ObjectId('684bf6536e20f68bd7731448')
, nom: "Macbook Pro", fabricant: "Apple", prix: 17435.99, options: [ "Intel Core i5", "Retina Display", "Long life battery" ] } ], ordered: true } k
eyUpdates:0 numYields:0 locks(micros) w:110 reslen:40 229ms
2025-06-13T09:59:16.580+0000 [clientcursorsmon] mem (MB) res:39 virt:4567
2025-06-13T09:59:16.580+0000 [clientcursorsmon] mapped (incl journal view):320
2025-06-13T09:59:16.585+0000 [clientcursorsmon] connections:1
2025-06-13T10:04:17.279+0000 [clientcursorsmon] mem (MB) res:39 virt:4567
2025-06-13T10:04:17.279+0000 [clientcursorsmon] mapped (incl journal view):320
2025-06-13T10:04:17.281+0000 [clientcursorsmon] connections:1
2025-06-13T10:09:18.015+0000 [clientcursorsmon] mem (MB) res:39 virt:4567
2025-06-13T10:09:18.015+0000 [clientcursorsmon] mapped (incl journal view):320
2025-06-13T10:09:18.021+0000 [clientcursorsmon] connections:1
2025-06-13T10:14:18.723+0000 [clientcursorsmon] mem (MB) res:39 virt:4567
2025-06-13T10:14:18.724+0000 [clientcursorsmon] mapped (incl journal view):320
2025-06-13T10:14:18.726+0000 [clientcursorsmon] connections:1
```


 Capture 1 : Terminal mongod (serveur MongoDB lancé avec succès)

- Démarrage du client avec `mongo.exe`

```
Administrateur : Invite de commandes - mongo.exe
Microsoft Windows [version 10.0.19045.5854]
(c) Microsoft Corporation. Tous droits réservés.

C:\WINDOWS\system32>cd C:\MongoDB\bin

C:\MongoDB\bin>mongo.exe
MongoDB shell version: 2.6.4
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-user
>
> use info
switched to db info
> db
info
> _
```

 Capture 2 : Terminal mongo (accès à la base de données via shell MongoDB)

#### 4. Création de la base de données et insertion de documents

Commande pour sélectionner ou créer la base : ``use info``

➡ Insertion du document Macbook Pro :

```
db.produits.insert({
  nom: "Macbook Pro",
  fabricant: "Apple",
  prix: 17435.99,
  options: ["Intel Core i5", "Retina Display", "Long life battery"]
})
```


➡ Insertion des documents DELL et Thinkpad X230 :

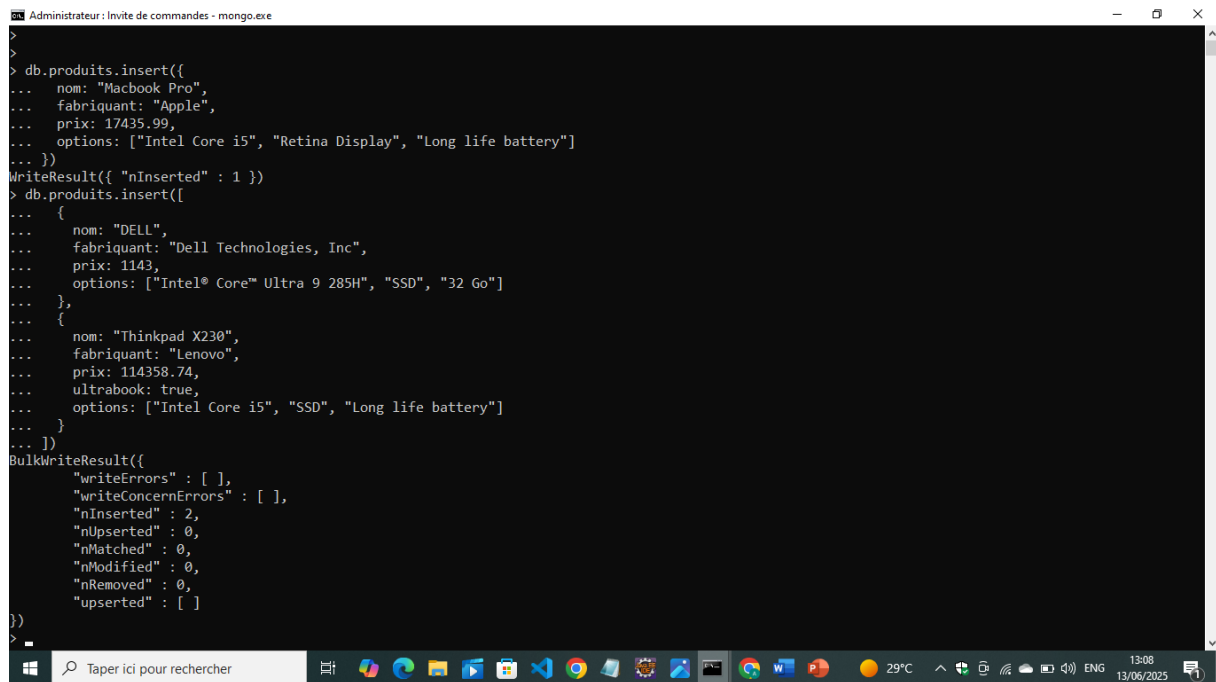
```
db.produits.insert([
  {
    nom: "DELL",
    fabricant: "Dell Technologies, Inc",
    prix: 1143,
```

```

    options: ["Intel® Core™ Ultra 9 285H", "SSD", "32 Go"]
  },
  {
    nom: "Thinkpad X230",
    fabricant: "Lenovo",
    prix: 114358.74,
    ultrabook: true,
    options: ["Intel Core i5", "SSD", "Long life battery"]
  }
]
)

```

 Capture 3 : Insertion des documents dans la collection produits



```

Administrateur : Invite de commandes - mongo.exe
>
>
> db.produits.insert({
...   nom: "Macbook Pro",
...   fabricant: "Apple",
...   prix: 17435.99,
...   options: ["Intel Core i5", "Retina Display", "Long life battery"]
... })
WriteResult({ "nInserted" : 1 })
> db.produits.insert([
...   {
...     nom: "DELL",
...     fabricant: "Dell Technologies, Inc",
...     prix: 1143,
...     options: ["Intel® Core™ Ultra 9 285H", "SSD", "32 Go"]
...   },
...   {
...     nom: "Thinkpad X230",
...     fabricant: "Lenovo",
...     prix: 114358.74,
...     ultrabook: true,
...     options: ["Intel Core i5", "SSD", "Long life battery"]
...   }
... ])
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 2,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
>

```

## 5. Requêtes de lecture et suppression

➤ Tous les produits :

```
db.produits.find()
```

```
Administrateur : Invite de commandes - mongo.exe
> db.produits.find()
{ "_id" : ObjectId("684bf6536e20f68bd7731448"), "nom" : "Macbook Pro", "fabriquant" : "Apple", "prix" : 17435.99, "options" : [ "Retina Display", "Long life battery" ] }
{ "_id" : ObjectId("684bf69e6e20f68bd7731449"), "nom" : "DELL", "fabriquant" : "Dell Technologies, Inc", "prix" : 17435.99, "options" : [ "Intel Core i5", "SSD", "32 Go" ] }
{ "_id" : ObjectId("684bf69e6e20f68bd773144a"), "nom" : "Thinkpad X230", "fabriquant" : "Lenovo", "prix" : 17435.99, "options" : [ "Intel Core i5", "SSD", "Long life battery" ] }
{ "_id" : ObjectId("684c22946e20f68bd773144b"), "nom" : "Macbook Pro", "fabriquant" : "Apple", "prix" : 17435.99, "options" : [ "Retina Display", "Long life battery" ] }
{ "_id" : ObjectId("684c22a56e20f68bd773144c"), "nom" : "DELL", "fabriquant" : "Dell Technologies, Inc", "prix" : 17435.99, "options" : [ "Intel Core i5", "SSD", "32 Go" ] }
{ "_id" : ObjectId("684c22a56e20f68bd773144d"), "nom" : "Thinkpad X230", "fabriquant" : "Lenovo", "prix" : 17435.99, "options" : [ "Intel Core i5", "SSD", "Long life battery" ] }
>
```

➤ Premier produit :

**db.produits.findOne()**

```
C:\WINDOWS\system32\cmd.exe - mongo.exe
> db.produits.findOne()
{
  "_id" : ObjectId("684bf6536e20f68bd7731448"),
  "nom" : "Macbook Pro",
  "fabriquant" : "Apple",
  "prix" : 17435.99,
  "options" : [
    "Intel Core i5",
    "Retina Display",
    "Long life battery"
  ]
}
```

➤ Produit par ID :

**db.produits.find({ \_id: ObjectId("666a1b2c3d4e5f6789012345") })**



```
C:\WINDOWS\system32\cmd.exe - mongo.exe
> db.produits.find({ _id: ObjectId("666a1b2c3d4e5f6789012345") })
>
>
>
>
>
>
>
```

➤ **Prix > 13723 :**

**db.produits.find({ prix: { \$gt: 13723 } })**

```
C:\WINDOWS\system32\cmd.exe - mongo.exe
> db.produits.find({ prix: { $gt: 13723 } })
{ "_id" : ObjectId("684bf6536e20f68bd7731448"), "nom" : "Macbook Pro", "fabrique" : "Apple", "options" : [ "Intel Core i5", "Retina Display", "Long life battery" ] }
{ "_id" : ObjectId("684bf69e6e20f68bd773144a"), "nom" : "Thinkpad X230", "fabrique" : "Lenovo", "prix" : 8.74, "ultrabook" : true, "options" : [ "Intel Core i5", "SSD", "Long life battery" ] }
{ "_id" : ObjectId("684c22946e20f68bd773144b"), "nom" : "Macbook Pro", "fabrique" : "Apple", "options" : [ "Intel Core i5", "Retina Display", "Long life battery" ] }
{ "_id" : ObjectId("684c22a56e20f68bd773144d"), "nom" : "Thinkpad X230", "fabrique" : "Lenovo", "prix" : 8.74, "ultrabook" : true, "options" : [ "Intel Core i5", "SSD", "Long life battery" ] }
>
```

➤ **ultrabook = true :**

**db.produits.findOne({ ultrabook: true })**

```
C:\WINDOWS\system32\cmd.exe - mongo.exe
> db.produits.findOne({ ultrabook: true })
{
  "_id" : ObjectId("684bf69e6e20f68bd773144a"),
  "nom" : "Thinkpad X230",
  "fabriquant" : "Lenovo",
  "prix" : 114358.74,
  "ultrabook" : true,
  "options" : [
    "Intel Core i5",
    "SSD",
    "Long life battery"
  ]
}
```

- Nom contient Macbook :

**db.produits.find({ nom: /Macbook/ })**

```
C:\WINDOWS\system32\cmd.exe - mongo.exe
> db.produits.find({ nom: /Macbook/ })
{ "_id" : ObjectId("684bf6536e20f68bd7731448"), "nom" : "Macbook Pro", "fabriquant" : "Apple", "prix" : 114358.74, "options" : [ "Intel Core i5", "Retina Display", "Long life battery" ] }
{ "_id" : ObjectId("684c22946e20f68bd773144b"), "nom" : "Macbook Pro", "fabriquant" : "Apple", "prix" : 114358.74, "options" : [ "Intel Core i5", "Retina Display", "Long life battery" ] }
```

- Nom commence par Macbook :

**db.produits.find({ nom: /^Macbook/ })**

```
C:\WINDOWS\system32\cmd.exe - mongo.exe
> db.produits.find({ nom: /^Macbook/ })
{ "_id" : ObjectId("684bf6536e20f68bd7731448"), "nom" : "Macbook Pro", "fabriquant" : "Apple", "prix" :
, "options" : [ "Intel Core i5", "Retina Display", "Long life battery" ] }
{ "_id" : ObjectId("684c22946e20f68bd773144b"), "nom" : "Macbook Pro", "fabriquant" : "Apple", "prix" :
, "options" : [ "Intel Core i5", "Retina Display", "Long life battery" ] }
>
```

➤ **Suppression Lenovo :**

**db.produits.remove({ fabriquant: "Lenovo" })**

```
C:\WINDOWS\system32\cmd.exe - mongo.exe
>
> db.produits.remove({ fabriquant: "Lenovo" })
WriteResult({ "nRemoved" : 2 })
>
```

📷 Capture 11

➤ **Suppression par ID :**

```
db.produits.remove({ _id: ObjectId("666123456abcdef123456789") })
```

C:\WINDOWS\system32\cmd.exe - mongo.exe

```
> db.produits.remove({ _id: ObjectId("666123456abcdef123456789") })  
WriteResult({ "nRemoved" : 0 })  
>
```

## 6. Conclusion

Ce premier TP m'a permis de me familiariser avec MongoDB, notamment avec sa structure orientée documents, la simplicité de ses commandes CRUD, et l'utilisation du shell Mongo. Grâce à ce TP, j'ai appris à manipuler une base NoSQL en mode console, à créer des collections dynamiquement, à rechercher des données avec filtres ou expressions régulières, et à effectuer des suppressions ciblées. Cette expérience pose les bases pour les TP futurs orientés vers l'interrogation avancée, l'indexation ou encore l'intégration avec des applications web.

# TP-3 Spark : Détection d'amis communs dans un réseau social

## Partie 1 – TP Spark Scala : Amis communs

### 3.1 Introduction

Cette partie présente l'utilisation d'**Apache Spark** avec **Scala** pour effectuer une analyse de graphe appliquée à un réseau social. L'objectif est de déterminer les amis communs entre utilisateurs à partir d'une liste d'adjacence. Le traitement est réalisé selon le modèle **MapReduce** à l'aide des transformations sur RDDs.

### 3.2 Préparation de l'environnement

✅ Prérequis :

- ❖ Java 8
- ❖ Apache Spark 3.x
- ❖ Scala 2.13
- ❖ Fichier `soc-LiveJournal1Adj.txt`

Vérification des installations :

```
java -version  
  
spark-shell
```

Capture 1 – Lancement réussi de Spark Shell

```
C:\Windows\System32\cmd.exe - spark-shell

C:\SparkTP\MutualFriends>spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to

  ____  __
 / ___/  / /
/ /   /  / /
/ /___/  / /
\____/___/ /
       /_/

version 3.4.3

Using Scala version 2.13.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_202)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context Web UI available at http://ahmed-pc:4040
Spark context available as 'sc' (master = local[*], app id = local-1751197156427).
Spark session available as 'spark'.

scala> _
```

### 3.3 Description du fichier source

**Nom du fichier :** soc-LiveJournal1Adj.txt

**Format attendu :**

<UserID>\t<FriendID1,FriendID2,...>

Exemple :

0 1, 2, 3

1 0, 2, 4

### 3.4 Étapes du traitement Spark

Étape 1 : Chargement du fichier

```
val data = sc.textFile("file:///C:/SparkTP/MutualFriends/soc-  
LiveJournal1Adj.txt")
```

#### Étape 2 : Prétraitement et découpage

```
val data1 = data.map(x => x.split("\t")).filter(_.size == 2)
```

#### Étape 3 : Création des paires d'amis

```
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)  
  }  
}
```

#### Étape 4 : Intersection des listes d'amis

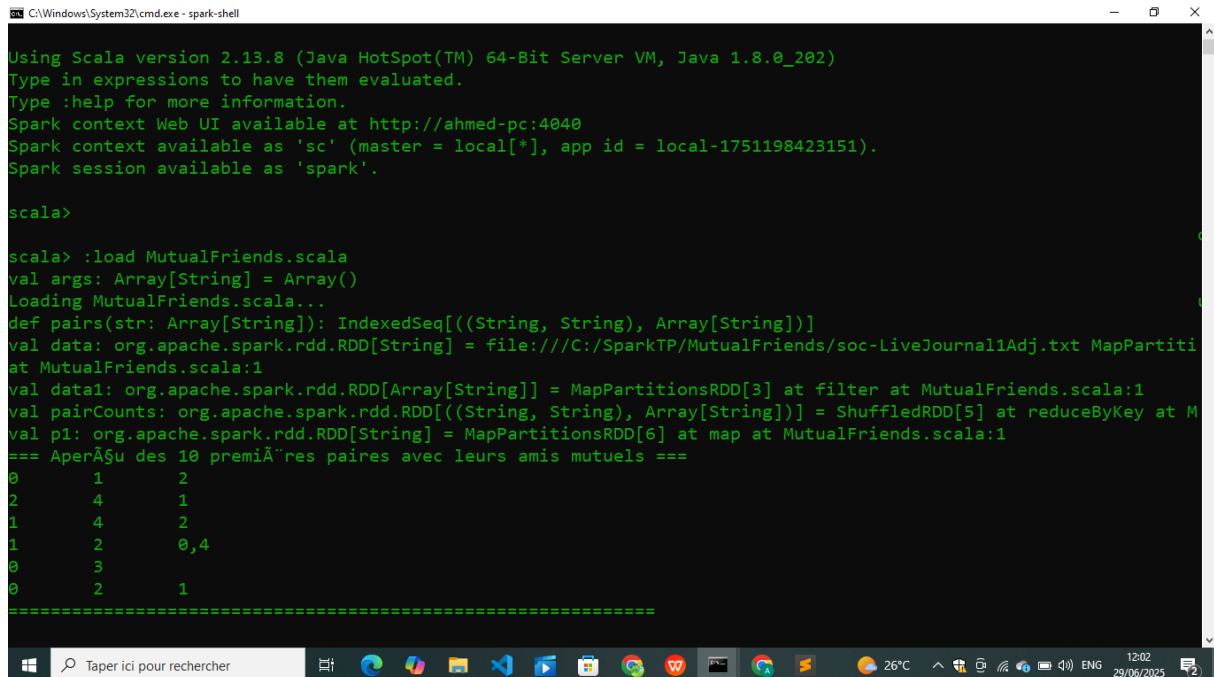
```
val pairCounts = data1.flatMap(pairs).reduceByKey((a, b) =>  
a.intersect(b))
```

#### Étape 5 : Formatage et enregistrement

```
val p1 = pairCounts.map { case ((a, b), mutuals) =>  
s"$a\t$b\t${mutuals.mkString(",")}" }
```

```
p1.saveAsTextFile("output")
```

## Capture 2 – Aperçu des 10 premières lignes affichées dans le terminal



```
C:\Windows\System32\cmd.exe - spark-shell

Using Scala version 2.13.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_202)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context Web UI available at http://ahmed-pc:4040
Spark context available as 'sc' (master = local[*], app id = local-1751198423151).
Spark session available as 'spark'.

scala>

scala> :load MutualFriends.scala
val args: Array[String] = Array()
Loading MutualFriends.scala...
def pairs(str: Array[String]): IndexedSeq[((String, String), Array[String])]
val data: org.apache.spark.rdd.RDD[String] = file:///C:/SparkTP/MutualFriends/soc-LiveJournal1Adj.txt MapPartiti
at MutualFriends.scala:1
val data1: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[3] at filter at MutualFriends.scala:1
val pairCounts: org.apache.spark.rdd.RDD[((String, String), Array[String])] = ShuffledRDD[5] at reduceByKey at M
val p1: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[6] at map at MutualFriends.scala:1
=== Aperçu des 10 premières paires avec leurs amis mutuels ===
0      1      2
2      4      1
1      4      2
1      2      0,4
0      3
0      2      1
=====
```

## 3.5 Extraction de cas spécifiques

### Fonction d'extraction

```
def afficherAmisCommun(userA: String, userB: String): String = {
  val amis = p1
    .map(_ .split("\t"))
    .filter(x => x.length == 3 && ((x(0) == userA && x(1) == userB) ||
(x(0) == userB && x(1) == userA)))
    .flatMap(x => x(2).split(",").filter(_.nonEmpty))
    .collect()

  if (amis.isEmpty)
    s"$userA\t$userB\tAucun ami mutuel"
  else
    s"$userA\t$userB\t${amis.mkString(", ")}"
}
```

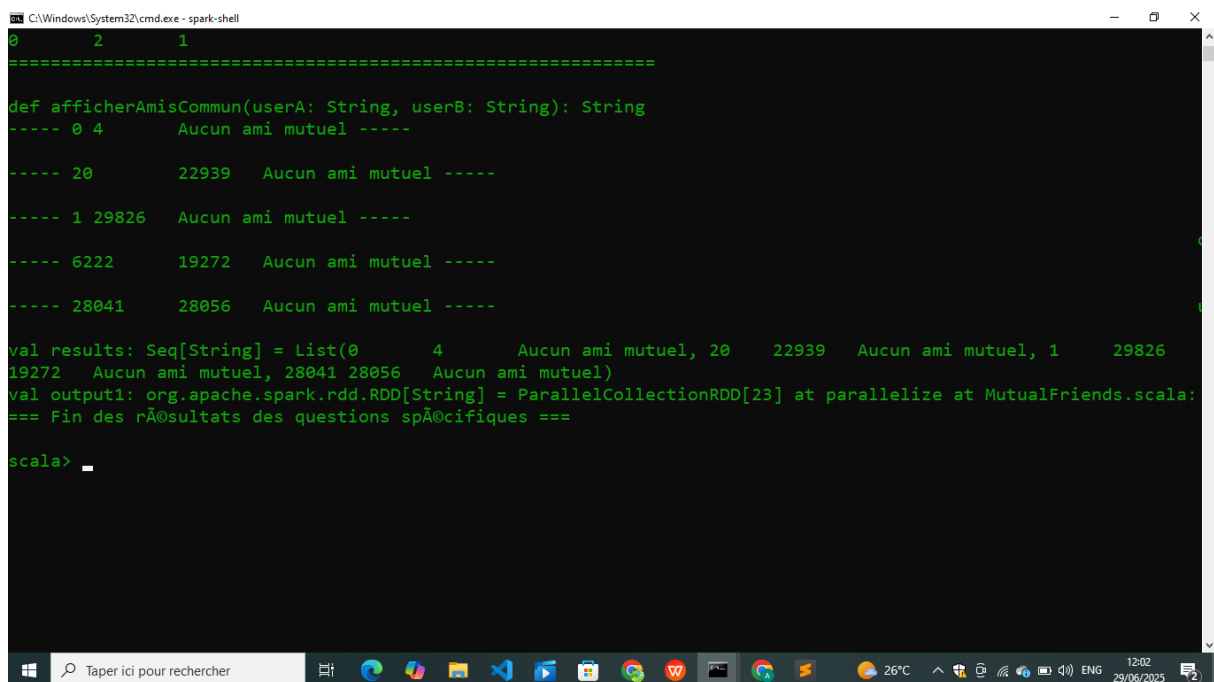


```
}
```

### Appels à la fonction

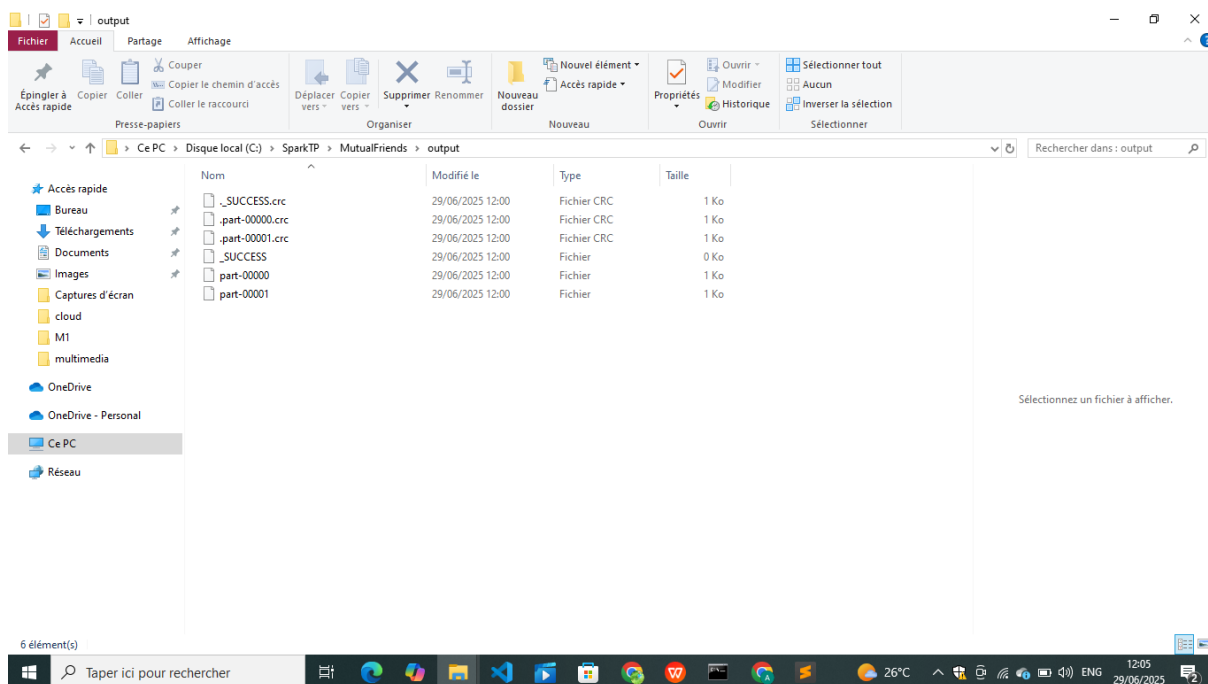
```
val results = Seq(  
    afficherAmisCommun("0", "4"),  
    afficherAmisCommun("20", "22939"),  
    afficherAmisCommun("1", "29826"),  
    afficherAmisCommun("6222", "19272"),  
    afficherAmisCommun("28041", "28056")  
)  
  
val answer = sc.parallelize(results)  
answer.saveAsTextFile("output1")
```

### Capture 3 – Résultats affichés pour les 5 paires demandées

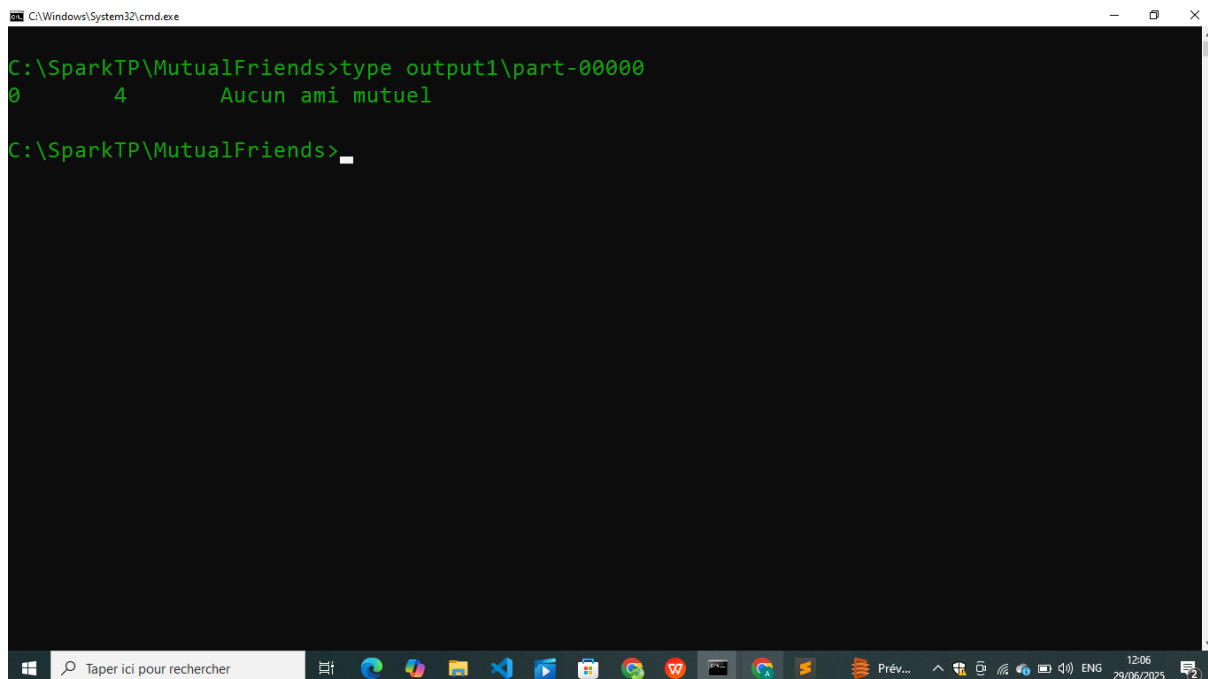


```
C:\Windows\System32\cmd.exe - spark-shell  
0      2      1  
=====  
def afficherAmisCommun(userA: String, userB: String): String  
---- 0 4      Aucun ami mutuel ----  
---- 20      22939  Aucun ami mutuel ----  
---- 1 29826  Aucun ami mutuel ----  
---- 6222      19272  Aucun ami mutuel ----  
---- 28041      28056  Aucun ami mutuel ----  
  
val results: Seq[String] = List(0      4      Aucun ami mutuel, 20      22939  Aucun ami mutuel, 1      29826  
19272  Aucun ami mutuel, 28041 28056  Aucun ami mutuel)  
val output1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[23] at parallelize at MutualFriends.scala:  
=== Fin des résultats des questions spécifiques ===  
  
scala> _
```

## Capture 4 – Dossier output/ généré



## Capture 5 – Contenu du fichier output1/part-00000



## 3.6 Conclusion

Ce TP Spark a permis de traiter un graphe social sous forme de liste d'adjacence. Les transformations Spark sur RDD (flatMap, reduceByKey, map, collect, saveAsTextFile) ont permis d'obtenir :

- ✧ L'ensemble des amis communs pour toutes les paires possibles
- ✧ Les amis communs pour des couples spécifiques d'utilisateurs
- ✧ Une organisation des résultats dans des répertoires distincts pour validation

L'utilisation de Spark s'est révélée efficace même en environnement local.

## Partie 2 – Détection des amis communs avec noms en Scala Spark

### 4.1 Introduction

Dans cette suite, nous développons un programme **Scala Spark** (via spark-shell) pour traiter un fichier texte contenant :

```
<user_id> <Nom> <friend_id1>,<friend_id2>,...
```

L'objectif est d'appliquer le même concept que précédemment mais en intégrant la gestion des noms utilisateurs, la génération de paires normalisées (triées), et la recherche précise des amis communs entre deux utilisateurs spécifiés par leur nom (ici Mohamed et Sidi).

### 4.2 Chargement et parsing

```

val data = sc.textFile("file:///C:/SparkTP/MutualFriends/friends_common.txt")

// Supprimer éventuelles lignes de titre/commentaires
val clean = data.filter(line => !line.startsWith("#"))

// Parser chaque ligne en (id, nom, liste_amis)
val parsed = clean.map { line =>
  val parts = line.trim.split("\\s+")
  val id = parts(0)
  val name = parts(1)
  val friends = if (parts.length > 2) parts(2).split(",").toList else List()
  (id, name, friends)
}

```

### 4.3 Création du mapping ID → Nom

```

val idToName = parsed.map(x => (x._1, x._2)).collect().toMap

```

### 4.4 Génération des paires triées (min, max)

```

val pairs = parsed.flatMap { case (id, name, friends) =>
  friends.map { fid =>
    val pair = if (id < fid) (id, fid) else (fid, id)
    (pair, friends)
  }
}

```

### 4.5 Calcul des amis communs

```

val mutualFriends = pairs.reduceByKey((list1, list2) => list1.intersect(list2))

```

### 4.6 Fonction pour afficher amis communs par noms (exemple Mohamed & Sidi)

```

def afficherAmisCommuns(nomA: String, nomB: String): Unit = {
  // Chercher les IDs des noms donnés
  val ids = idToName.filter { case (id, name) => name == nomA || name == nomB }.keys.toList
  if (ids.length != 2) {
    println(s"Erreur : noms '$nomA' ou '$nomB' introuvables")
    return
  }
  val pair = if (ids(0) < ids(1)) (ids(0), ids(1)) else (ids(1), ids(0))

  val result = mutualFriends.lookup(pair)
  if (result.nonEmpty) {
    val amisNoms = result(0).filter(idToName.contains).map(idToName)
  }
}

```

```

    println(s"${pair._1}<$nomA> ${pair._2}<$nomB> : ${amisNoms.mkString(", ")}")
  } else {
    println(s"${pair._1}<$nomA> ${pair._2}<$nomB> : Aucun ami commun")
  }
}

```

## 4.7 Exemple d'utilisation

```

afficherAmisCommuns("Sidi", "Mohamed")
afficherAmisCommuns("Aicha", "Ahmed")
afficherAmisCommuns("Mohamed", "Leila")

```

## 4.8 Captures d'écran démonstratives et résultats clés

- Capture : Chargement, traitement et affichage des amis communs dans spark-shell

```
C:\Windows\System32\cmd.exe - spark-shell

scala>

scala> :load C:/SparkTP/MutualFriends/MutualFriendsNames.scala
val args: Array[String] = Array()
Loading C:\SparkTP\MutualFriends\MutualFriendsNames.scala...
val data: org.apache.spark.rdd.RDD[String] = file:///C:/SparkTP/MutualFriends/friends_common.txt MapPartitionsRDD[1]
/SparkTP/MutualFriends/MutualFriendsNames.scala:1
val clean: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at C:/SparkTP/MutualFriends/MutualFr
val parsed: org.apache.spark.rdd.RDD[(String, String, List[String])] = MapPartitionsRDD[3] at map at C:/SparkTP/
iendsNames.scala:1
val idToName: scala.collection.immutable.Map[String,String] = HashMap(4 -> Ahmed, 5 -> Leila, 2 -> Mohamed, 3 ->
> Sidi)
val pairs: org.apache.spark.rdd.RDD[((String, String), List[String])] = MapPartitionsRDD[5] at flatMap at C:/Spa
ualFriendsNames.scala:1
val mutualFriends: org.apache.spark.rdd.RDD[((String, String), List[String])] = ShuffledRDD[6] at reduceByKey at
nds/MutualFriendsNames.scala:1
def afficherAmisCommuns(nomA: String, nomB: String): Unit
Sidi <--> Mohamed : Aicha
Aicha <--> Ahmed : Sidi
Mohamed <--> Leila :

scala>
```

## Explication

Cette capture montre le chargement complet du script Scala dans Spark Shell, la création des structures RDD pour les données utilisateurs et leurs amis, ainsi que la définition de la fonction **afficherAmisCommuns** qui affiche la liste des amis communs entre deux utilisateurs donnés par leur nom.

Les exemples d'appels à cette fonction illustrent :

- ❖ Une paire avec amis communs (Sidi et Mohamed ont Aicha en commun),
- ❖ Une autre paire avec un ami commun différent (Aicha et Ahmed),
- ❖ Enfin, une paire sans ami commun (Mohamed et Leila).

Cette démonstration valide la bonne exécution et la pertinence du programme dans Spark.

## 4.9 Conclusion

Cette extension du TP a permis d'enrichir l'analyse des amis communs en intégrant la gestion des noms d'utilisateurs et en permettant une recherche ciblée entre deux personnes précises.

Grâce à Spark en Scala, nous avons pu :

- ✧ Traiter un fichier complexe associant ID, noms et listes d'amis,
- ✧ Générer des paires normalisées pour éviter les doublons,
- ✧ Calculer efficacement les amis communs par paires,
- ✧ Mettre en place une fonction simple pour afficher les amis communs entre deux utilisateurs identifiés par leur nom.

Cette approche démontre la flexibilité de Spark pour manipuler des graphes sociaux complexes et illustre comment exploiter pleinement les capacités distribuées de Spark tout en gardant une interface utilisateur conviviale.

## Conclusion generale

Les Travaux Pratiques ont permis de manipuler concrètement plusieurs outils essentiels de l'écosystème Big Data.

- ✚ Avec **Hadoop**, nous avons appris à segmenter et traiter des données massives à l'aide du paradigme **MapReduce**.
- ✚ Grâce à **MongoDB**, nous avons découvert un modèle de base de données souple, sans schéma fixe, adapté aux structures de données modernes.

✚ Enfin, **Apache Spark** associé à **Scala** nous a permis de traiter rapidement de grands ensembles de données en mémoire, notamment pour l'analyse de graphes comme les réseaux sociaux.

Ces TP nous ont permis de mieux comprendre les enjeux et les solutions techniques du Big Data, ainsi que d'appliquer les bases de la programmation distribuée dans un environnement réel. Ils constituent une base solide pour aborder des projets plus avancés d'analyse de données à grande échelle.