

Rapport Big Data

Exploration des

technologies MongoDB,

Hadoop et Spark

Demandé par : Dr.Mohamed El Moustapha El Arby

Réalisé par : Dadee zeidane C18059

I. Introduction

Ce rapport présente le travail réalisé dans le cadre de la matière **Big Data**, au cours du semestre, à travers trois travaux pratiques majeurs. Ces TPs permettent de découvrir, d'explorer et de mettre en œuvre des technologies fondamentales pour le traitement des données massives, illustrant concrètement les concepts étudiés en cours.

Le premier TP est consacré à **MongoDB**, une base de données NoSQL orientée document, utilisée pour manipuler des données semi-structurées et effectuer des opérations CRUD. Le deuxième TP porte sur **Hadoop**, un framework distribué permettant de stocker et traiter de grands volumes de données grâce au modèle **MapReduce**.

Le troisième TP est dédié à **Apache Spark**, d'abord sous forme de TD pour s'initier au traitement distribué de graphes sociaux, puis sous forme de TP complet mettant en pratique le calcul d'amis communs sur un graphe social simulé.

Ces activités visent à développer une compréhension concrète et appliquée des outils du Big Data, en reliant théorie et pratique, et en nous préparant à relever les défis du traitement des données à grande échelle.

II. Tp N°1 : Bases de données NoSQL « MongoDB »

Objectif du TP MongoDB :

L'objectif de cette partie du projet est de découvrir les concepts de base de MongoDB à travers l'installation, la configuration et l'utilisation de son serveur et de son client sous Windows. Elle consiste à créer une base de données, manipuler des collections et des documents via des opérations CRUD (Create, Read, Update, Delete), et à exécuter des requêtes pour interagir avec les données de manière pratique.

1. Installation de MongoDB sous Windows :

- Téléchargez la version zip de MongoDB pour Windows (64-bit zip).



- Extraire l'archive zip dans le dossier C:\MongoDB (elle doit contenir un répertoire bin\).

The screenshot shows the Windows File Explorer interface. The address bar indicates the path: Ce PC > Disque local (C:) > mongodb-win32-x86_64-2008plus-2.6.4 > bin. The left sidebar shows various folder icons like Accueil, Galerie, etc. The main area displays a list of files in the 'bin' directory, sorted by name. The files listed are: bsondump.exe, mongo.exe, mongod.exe, mongod.pdb, mongodump.exe, mongoexport.exe, mongoimport.exe, mongooplog.exe, mongoperf.exe, mongorestore.exe, mongos.exe, mongos.pdb, mongostat.exe, and mongotop.exe. The file 'mongodump.exe' is currently selected.

	Nom	Modifié le	Type	Taille
Accueil	bsondump.exe	09/06/2025 22:38	Application	18349 Ko
Galerie	mongo.exe	09/06/2025 22:38	Application	9906 Ko
zeidane : person	mongod.exe	09/06/2025 22:38	Application	18478 Ko
zeidane : person	mongod.pdb	09/06/2025 22:38	Fichier PDB	126043 Ko
Documents	mongodump.exe	09/06/2025 22:38	Application	18405 Ko
Images	mongoexport.exe	09/06/2025 22:38	Application	18356 Ko
Bureau	mongoimport.exe	09/06/2025 22:38	Application	18385 Ko
Téléchargem	mongooplog.exe	09/06/2025 22:38	Application	18347 Ko
Documents	mongoperf.exe	09/06/2025 22:38	Application	18210 Ko
Images	mongorestore.exe	09/06/2025 22:38	Application	18424 Ko
Musique	mongos.exe	09/06/2025 22:38	Application	14915 Ko
Vidéos	mongos.pdb	09/06/2025 22:38	Fichier PDB	101451 Ko
Galerie	mongostat.exe	09/06/2025 22:38	Application	18396 Ko
ilm-classifica	mongotop.exe	09/06/2025 22:38	Application	18351 Ko
add				

- Créez les dossiers C:\data et C:\data\db (c'est là que MongoDB stocke vos données et d'autres choses).

The screenshot shows the Windows File Explorer interface. The address bar indicates the path: Ce PC > Disque local (C:) > data > db. The left sidebar shows various folder icons like Accueil, Galerie, etc. The main area displays a list of files in the 'db' directory. The files listed are: _tmp, journal, info.0, info.ns, local.0, local.ns, and mongod.lock. The file 'info.0' is currently selected.

	Nom	Modifié le	Type	Taille
	_tmp	15/06/2025 14:40	Dossier de fichiers	
	journal	15/06/2025 14:40	Dossier de fichiers	
	info.0	15/06/2025 14:40	Fichier 0	65 536 Ko
	info.ns	15/06/2025 14:40	Fichier NS	16 384 Ko
	local.0	10/06/2025 13:07	Fichier 0	65 536 Ko
	local.ns	10/06/2025 13:07	Fichier NS	16 384 Ko
	mongod.lock	15/06/2025 14:38	Fichier LOCK	1 Ko

2. Serveur mongo :

Dans une ligne de commande: lancez le serveur : bin\mongod.exe

```

C:\mongodb-win32-x86_64-2008plus-2.6.4\bin>mongod.exe --dbpath=C:\data\db
2025-06-15T14:38:28.933+0000 [initandlisten] MongoDB starting : pid=13676 port=27017 dbpath=C:\data\db 64-bit host=DESKT
OP-V172AS4
2025-06-15T14:38:28.938+0000 [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2025-06-15T14:38:28.938+0000 [initandlisten] db version v2.6.4
2025-06-15T14:38:28.938+0000 [initandlisten] git version: 3a830be0eb92d772aa855ebb711ac91d658ee910
2025-06-15T14:38:28.938+0000 [initandlisten] build info: windows sys.getwindowsversion(major=6, minor=1, build=7601, pla
tform=2, service_pack='Service Pack 1') BOOST_LIB_VERSION=1_49
2025-06-15T14:38:28.938+0000 [initandlisten] allocator: system
2025-06-15T14:38:28.938+0000 [initandlisten] options: { storage: { dbPath: "C:\data\db" } }
2025-06-15T14:38:28.946+0000 [initandlisten] journal dir=C:\data\db\journal
2025-06-15T14:38:28.946+0000 [initandlisten] recover : no journal files present, no recovery needed
2025-06-15T14:38:29.014+0000 [initandlisten] waiting for connections on port 27017
2025-06-15T14:39:29.147+0000 [clientcursormon] mem (MB) res:40 virt:4417
2025-06-15T14:39:29.147+0000 [clientcursormon] mapped (incl journal view):160
2025-06-15T14:39:29.147+0000 [clientcursormon] connections:0
2025-06-15T14:39:36.760+0000 [initandlisten] connection accepted from 127.0.0.1:51051 #1 (1 connection now open)
2025-06-15T14:40:20.800+0000 [FileAllocator] allocating new datafile C:\data\db\info.ns, filling with zeroes...
2025-06-15T14:40:20.800+0000 [FileAllocator] creating directory C:\data\db\_tmp
2025-06-15T14:40:20.800+0000 [FileAllocator] done allocating datafile C:\data\db\info.ns, size: 16MB, took 0.001 secs
2025-06-15T14:40:20.825+0000 [FileAllocator] allocating new datafile C:\data\db\info.0, filling with zeroes...
2025-06-15T14:40:20.825+0000 [FileAllocator] done allocating datafile C:\data\db\info.0, size: 64MB, took 0.002 secs
2025-06-15T14:40:20.858+0000 [conn1] build index on: info.produits properties: { v: 1, key: { _id: 1 }, name: "_id", ns
: "info.produits" }
2025-06-15T14:40:20.858+0000 [conn1] added index to empty collection
2025-06-15T14:40:29.285+0000 [clientcursormon] mem (MB) res:41 virt:4578
2025-06-15T14:40:29.285+0000 [clientcursormon] mapped (incl journal view):320
2025-06-15T14:40:29.285+0000 [clientcursormon] connections:1
2025-06-15T14:45:29.891+0000 [clientcursormon] mem (MB) res:41 virt:4576

```

3. Client mongo :

Ce que nous venons de lancer c'est le serveur

Pour interagir avec ce serveur, il nous faut un client. Pour l'instant, nous allons lancer nos requêtes au serveur au travers de la console mongo.

Dans une ligne de commande lancez l'exécutable bin\mongosh

```

C:\mongodb-win32-x86_64-2008plus-2.6.4\bin>mongo.exe
MongoDB shell version: 2.6.4
connecting to: test
> use info
switched to db info

```

Exercice 01

Création d'une BD Mongo et opération CRUD

- Créer une base de données nommée info et vérifiez qu'elle est sélectionnée.

```
> use info
switched to db info
```

- Créer une nouvelle collection nommée produits et y insérer le document suivant :

(nom: Macbook Pro

fabriquant: Apple

prix: 17435,99

options: Intel Core i5

Retina Display

Long life battery)

```
info> db.produits.insertOne({  
...   nom: "Macbook Pro",  
...   fabriquant: "Apple",  
...   prix: 17435.99,  
...   options: ["Intel Core i5", "Retina Display", "Long life battery"]  
... })  
{  
  acknowledged: true,  
  insertedId: ObjectId('6825fa2094fed3f00eab7139')  
}  
info> |
```

c. Rajouter deux autres documents dans produits :

```
(nom: DELL  
fabriquant: Dell Technologies, Inc  
prix: 1143  
options: Intel® Core™ Ultra 9 285H  
SSD  
32 Go)  
(nom: Thinkpad X230  
fabriquant: Lenovo  
prix: 114358,74  
ultrabook: true  
options: Intel Core i5  
SSD  
Long life battery)
```

```

info> db.produits.insertMany([
...   {
...     nom: "DELL",
...     fabriquant: "Dell Technologies, Inc",
...     prix: 1143,
...     options: ["Intel® Core™ Ultra 9 285H", "SSD", "32 Go"]
...   },
...   {
...     nom: "Thinkpad X230",
...     fabriquant: "Lenovo",
...     prix: 114358.74,
...     ultrabook: true,
...     options: ["Intel Core i5", "SSD", "Long life battery"]
...   }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6825fa9c94fed3f00eab713a'),
    '1': ObjectId('6825fa9c94fed3f00eab713b')
  }
}
info> |

```

d. Effectuez les requêtes de lecture suivantes:

- Récupérer tous les produits.

```

info> db.produits.find().pretty()
[
  {
    _id: ObjectId('6825fa2094fed3f00eab7139'),
    nom: 'Macbook Pro',
    fabriquant: 'Apple',
    prix: 17435.99,
    options: [ 'Intel Core i5', 'Retina Display', 'Long life battery' ]
  },
  {
    _id: ObjectId('6825fa9c94fed3f00eab713a'),
    nom: 'DELL',
    fabriquant: 'Dell Technologies, Inc',
    prix: 1143,
    options: [ 'Intel® Core™ Ultra 9 285H', 'SSD', '32 Go' ]
  },
  {
    _id: ObjectId('6825fa9c94fed3f00eab713b'),
    nom: 'Thinkpad X230',
    fabriquant: 'Lenovo',
    prix: 114358.74,
    ultrabook: true,
    options: [ 'Intel Core i5', 'SSD', 'Long life battery' ]
  }
]
info>

```

- Récupérer le premier produit

```
info> db.produits.findOne()
{
  _id: ObjectId('6825fa2094fed3f00eab7139'),
  nom: 'Macbook Pro',
  fabriquant: 'Apple',
  prix: 17435.99,
  options: [ 'Intel Core i5', 'Retina Display', 'Long life battery' ]
}
info> |
```

- Trouver l'id du Thinkpad et faites la requête pour récupérer ce produit avec son id.

```
info> db.produits.findOne({ nom: "Thinkpad X230" })
{
  _id: ObjectId('6825fa9c94fed3f00eab713b'),
  nom: 'Thinkpad X230',
  fabriquant: 'Lenovo',
  prix: 114358.74,
  ultrabook: true,
  options: [ 'Intel Core i5', 'SSD', 'Long life battery' ]
}
info> |
```

```
info> db.produits.findOne({ _id: ObjectId("6825fa9c94fed3f00eab713b") })
{
  _id: ObjectId('6825fa9c94fed3f00eab713b'),
  nom: 'Thinkpad X230',
  fabriquant: 'Lenovo',
  prix: 114358.74,
  ultrabook: true,
  options: [ 'Intel Core i5', 'SSD', 'Long life battery' ]
}
info> |
```

- Récupérer les produits dont le prix est supérieur à 13723 DA

```
info> db.produits.find({ prix: { $gt: 13723 } }).pretty()
[
  {
    _id: ObjectId('6825fa2094fed3f00eab7139'),
    nom: 'Macbook Pro',
    fabriquant: 'Apple',
    prix: 17435.99,
    options: [ 'Intel Core i5', 'Retina Display', 'Long life battery' ]
  },
  {
    _id: ObjectId('6825fa9c94fed3f00eab713b'),
    nom: 'Thinkpad X230',
    fabriquant: 'Lenovo',
    prix: 114358.74,
    ultrabook: true,
    options: [ 'Intel Core i5', 'SSD', 'Long life battery' ]
  }
]
info> |
```

- Récupérer le premier produit ayant le champ ultrabook à true

```
info> db.produits.findOne({ ultrabook: true })
{
  _id: ObjectId('6825fa9c94fed3f00eab713b'),
  nom: 'Thinkpad X230',
  fabriquant: 'Lenovo',
  prix: 114358.74,
  ultrabook: true,
  options: [ 'Intel Core i5', 'SSD', 'Long life battery' ]
}
info> |
```

- Récupérer le premier produit dont le nom contient Macbook

```
info> db.produits.findOne({ nom: { $regex: "Macbook", $options: "i" } })
{
  _id: ObjectId('6825fa2094fed3f00eab7139'),
  nom: 'Macbook Pro',
  fabriquant: 'Apple',
  prix: 17435.99,
  options: [ 'Intel Core i5', 'Retina Display', 'Long life battery' ]
}
info>
```

- Récupérer les produits dont le nom commence par Macbook

```
info> db.produits.find({ nom: { $regex: "^Macbook", $options: "i" } }).pretty()
[
  {
    _id: ObjectId('6825fa2094fed3f00eab7139'),
    nom: 'Macbook Pro',
    fabriquant: 'Apple',
    prix: 17435.99,
    options: [ 'Intel Core i5', 'Retina Display', 'Long life battery' ]
  }
]
info> |
```

- Supprimer le produit dont le fabricant est Lenovo.

```
info> db.produits.deleteOne({ fabriquant: "Lenovo" })
{ acknowledged: true, deletedCount: 1 }
info> |
```

- Supprimer le Lenovo X230 en utilisant uniquement son id.

Insérer de nouveau pour qu'on puisse le supprimer d'une autre façon.

```
info> db.produits.insertOne({
...   nom: "Thinkpad X230",
...   fabriquant: "Lenovo",
...   prix: 114358.74,
...   ultrabook: true,
...   options: ["Intel Core i5", "SSD", "Long life battery"]
... })
{
  acknowledged: true,
  insertedId: ObjectId('6826417394fed3f00eab713c')
}
info> |
```

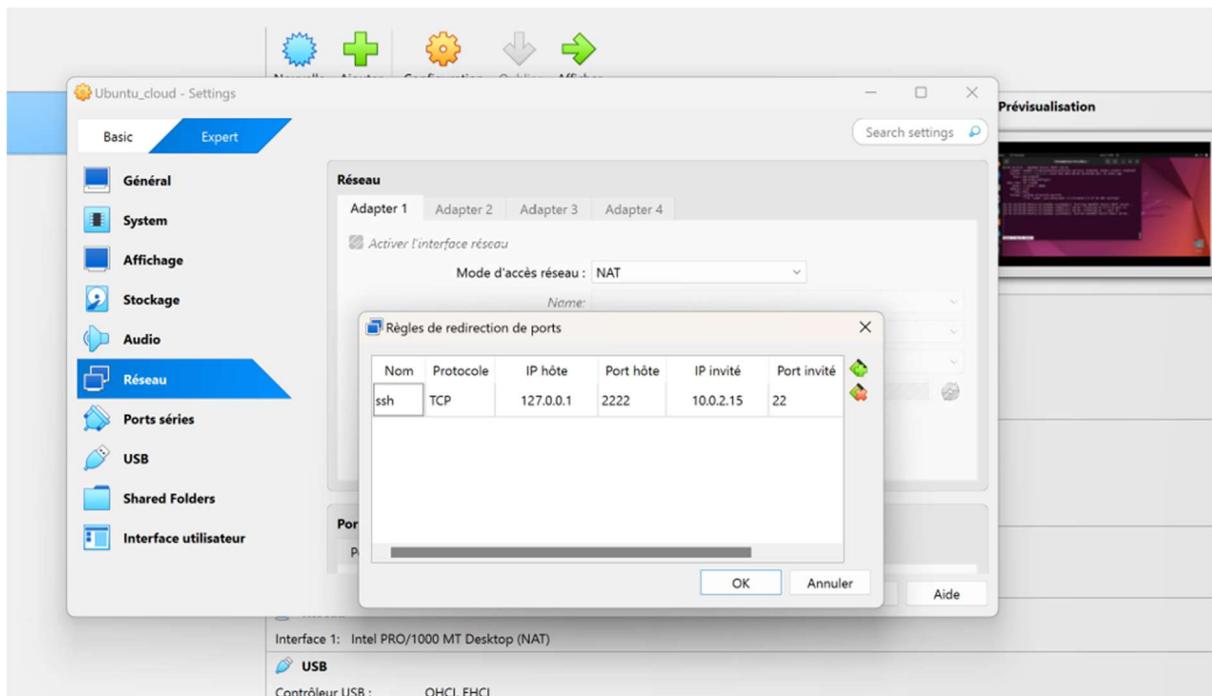
```
info> db.produits.findOne({ nom: "Thinkpad X230" })
{
  _id: ObjectId('6826417394fed3f00eab713c'),
  nom: 'Thinkpad X230',
  fabriquant: 'Lenovo',
  prix: 114358.74,
  ultrabook: true,
  options: [ 'Intel Core i5', 'SSD', 'Long life battery' ]
}
info> db.produits.deleteOne({ _id: ObjectId("6826417394fed3f00eab713c") })
{ acknowledged: true, deletedCount: 1 }
info> |
```

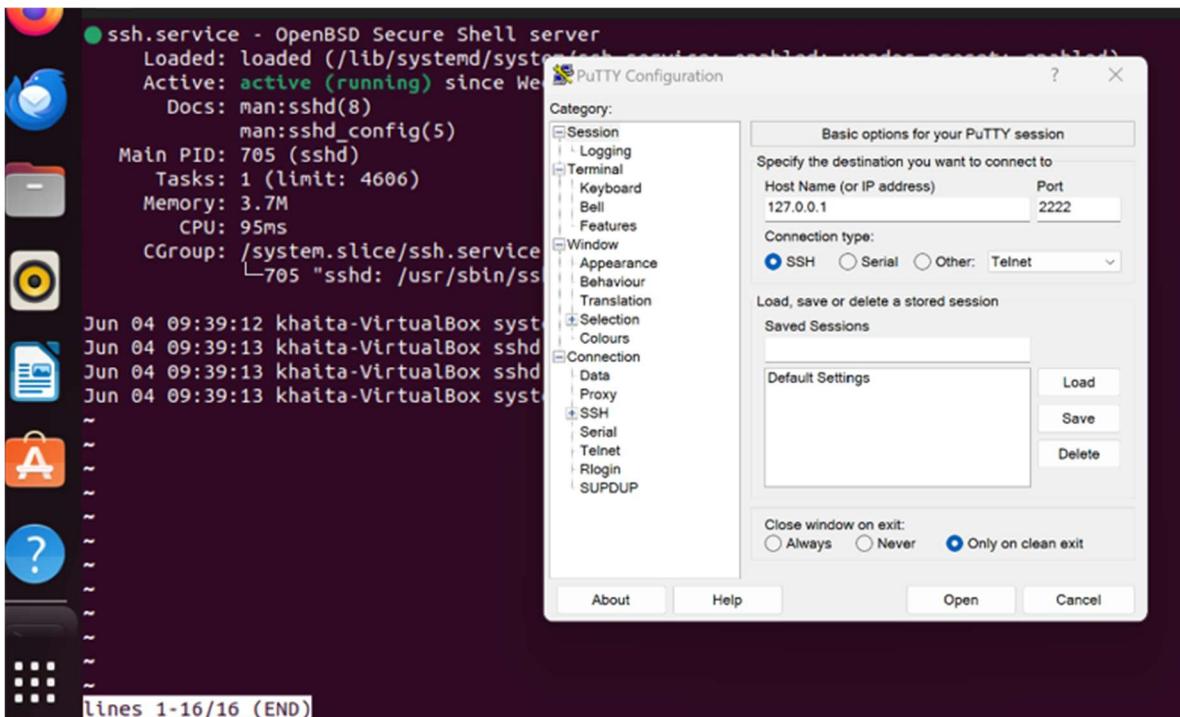
III. Tp N° 2 : Framework HADOOP

Objectif du TP Hadoop (WordCount) :

Ce TP Hadoop a pour objectif de mettre en place un environnement de traitement Big Data en configurant Hadoop en mode pseudo-distribué sur un système Linux. Il s'agit d'installer Hadoop, de préparer le système de fichiers distribué (HDFS), de compiler un programme Java utilisant MapReduce, et de l'exécuter pour compter les occurrences des mots d'un fichier texte. Chaque étape vise à illustrer le fonctionnement des composants principaux d'Hadoop (NameNode, DataNode, YARN) et à montrer comment ils collaborent pour traiter des données de manière parallèle et distribuée. Ce rapport décrit les différentes étapes nécessaires pour atteindre cet objectif.

L'accès à la virtuel machine d'après Putty





```
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-58-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

86 updates can be applied immediately.
75 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Jun  Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/csm or run: sudo pro status

Jun
Jun
Jun  The programs included with the Ubuntu system are free software;
~   the exact distribution terms for each program are described in the
~   individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

1. Creation de nouveau user « hduse »

Un utilisateur nommé 'hduse' a été créé pour isoler l'environnement Hadoop. Les commandes suivantes ont été utilisées :

```
Sudo add user hduse
```

```
Sudo usermod -aG sudo hduse
```

2. Prérequis et installation

Téléchargement et extraction d'Hadoop :

```
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
tar -xvzf hadoop-3.3.6.tar.gz
mv hadoop-3.3.6 hadoop
```

Installation de Java 8 :

```
sudo apt install openjdk-8-jdk -y
```

Définir JAVA_HOME dans hadoop-env.sh :

```
nano hadoop/etc/hadoop/hadoop-env.sh
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

3. Configuration de Hadoop

.bashrc :

```
ajout des variables JAVA_HOME et HADOOP_HOME
```

hadoop-env.sh :

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

core-site.xml :

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

hdfs-site.xml :

```
<configuration>
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
</configuration>
```

mapred-site.xml :

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

yarn-site.xml :

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

4. Préparation des répertoires

Création des répertoires de stockage :

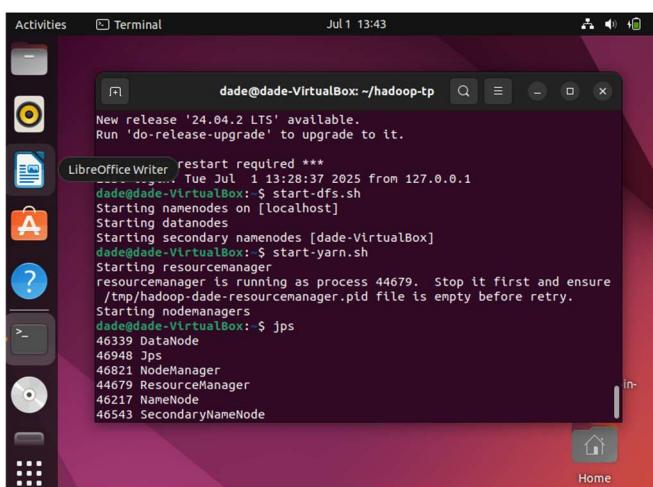
```
~/hadoopdata/hdfs/namenode
~/hadoopdata/hdfs/datanode
```

5. Formatage et démarrage du système Hadoop

Formatage du NameNode : `hdfs namenode -format`

Démarrage des démons : `sbin/start-dfs.sh` et `sbin/start-yarn.sh`

Vérification : `jps`



```
dade@dade-VirtualBox: ~/hadoop-tp$ $ jps
46339 DataNode
46948 Jps
46821 NodeManager
44679 ResourceManager
46217 NameNode
46543 SecondaryNameNode
```

6. Exécution du programme MapReduce

Création des dossiers et ajout du fichier dans HDFS :

```
hdfs dfs -mkdir /user  
hdfs dfs -mkdir /user/hduse  
hdfs dfs -put poeme.txt /user/hduse
```

Compilation :

```
javac -classpath $(hadoop classpath) -d . WCount.java
```

Création du jar :

```
jar -cvf wcount.jar *
```

Exécution du job :

```
hadoop jar wcount.jar WCount /user/hduse/poeme.txt /user/hduse/output
```

Récupération des résultats :

```
hdfs dfs -cat /user/hduse/output/part-r-00000
```



```
Big      2  
Bonjour  2  
Data     2  
Hadoop   1  
puissant 1
```

IV. TD N° 3 : Spark

Objectif du TD Spark :

L'objectif de ce TD est de se familiariser avec les concepts de base de Spark et de la programmation distribuée à travers l'analyse d'un mini-graphe social. Il s'agit de :

- Générer les paires d'amis et leurs listes associées de manière normalisée.
- Appliquer des opérations MapReduce pour calculer les amis communs entre utilisateurs.
- Comprendre et manipuler des RDD dans Spark en Scala.
- Filtrer et formater les résultats pour des paires d'utilisateurs spécifiques.

Ce TD prépare aux notions essentielles pour la réalisation des TPs plus avancés, en consolidant les bases du traitement distribué des graphes sous Spark.

Input :

Nous considérons pour ce TD un mini-graphe dans le fichier ‘soc-LiveJournal1Adj.txt’ :

Utilisateur

Amis

1	2,3,4,5
2	1,3,4
3	1,2,4
4	1,2,3
5	1

Ici, « **Utilisateur** » est un identifiant entier unique correspondant à un utilisateur, et « **Amis** » est une liste d'identifiants uniques séparés par des virgules, correspondant aux amis de cet utilisateur.

À noter que les amitiés sont mutuelles (c'est-à-dire que les arêtes sont non orientées) : si A est ami avec B, alors B est également ami avec A.

Output :

La sortie doit contenir une ligne par paire d'utilisateurs, sous le format suivant :

<User_A><TAB><User_B><TAB><Liste des amis communs>

où <**User_A**> et <**User_B**> sont des identifiants uniques correspondant aux utilisateurs A et B (A et B sont amis).

<**Liste des amis communs**> est une liste d'identifiants uniques séparés par des virgules correspondant à la liste des amis communs des utilisateurs A et B.

1) Explication du rôle de la fonction pairs :

Elle reçoit en entrée un utilisateur avec sa liste d'amis, et génère :

- Toutes les paires possibles formées entre cet utilisateur et chacun de ses amis.
- Des paires normalisées (toujours dans l'ordre (`min, max`) pour éviter les doublons dans le traitement (par exemple, elle crée uniquement (1, 2) et jamais (2, 1)).
- Pour chaque paire générée, elle associe la liste complète des amis de l'utilisateur courant.

```
//generate ((user1, user2), friendList) for pair counts
def pairs(str: Array[String]) = {
    val users = str(1).split(",")
    val user = str(0)
    val n = users.length
    for(i <- 0 until n) yield {
        val pair = if(user < users(i)) {
            (user,users(i))
        } else {
            (users(i),user)
        }
        (pair, users(i).split(","))
    }
}
```

```

} else {
  (users(i), user)
}
(pair, users) }

```

2) On considère le code Spark en Scala (**MutualFriends.scala**) à completer :

```

//Main of our program
---1-- val data = sc.textFile("soc-LiveJournal1Adj.txt")
---2-- val data1= data.map(x => x.split("\t")).filter(li => li.size == 2)
---3-- val pairCounts = data1.flatMap(pairs).reduceByKey((a, b) => a.intersect(b))
---4-- val p1= pairCounts.map { case ((user1, user2), friends) =>
  s"$user1\t$user2\t${friends.mkString(",")}"  }
---5-- p1.saveAsTextFile("output")

```

3) Explication du rôle du code Sparks en Scala suivant :

Rôle global du code :

Sélectionne des paires précises.

- Extrait leurs amis communs sous forme simple.
- Produit un fichier compact avec uniquement ces résultats.

Code avec commentaires :

```

// Initialise une chaîne vide qui servira à accumuler les résultats formatés sous forme texte
var ans=""

// -----1---
// Sélectionne la paire (0,4) dans p1, récupère les amis communs, les transforme en tableau
val p2=p1.map(x=>x.split("\t"))
  .filter(x => (x.size == 3))
  .filter(x=>(x(0)=="0"&&x(1)=="4"))
  .flatMap(x=>x(2).split(","))

```

```
.collect()

// ----2---

// Ajoute à la chaîne ans la ligne formatée des amis communs de la paire (0,4)
ans=ans+"0"+"\t"+"4"+"\t"+p2.mkString(",")+"\n"

// ----3---

// Sélectionne la paire (20,22939), récupère ses amis communs sous forme de tableau
val p3=p1.map(x=>x.split("\t"))

.filter(x => (x.size == 3))

.filter(x=>(x(0)=="20"&&x(1)=="22939"))

.flatMap(x=>x(2).split(","))

.collect()

// ----4---

// Ajoute à la chaîne ans la ligne formatée des amis communs de la paire (20,22939)
ans=ans+"20"+"\t"+"22939"+"\t"+p3.mkString(",")+"\n"

// ----5---

// Transforme la chaîne ans en un RDD Spark pour la sauvegarde
val answer=sc.parallelize(Seq(ans))

// ----6---

// Sauvegarde le résultat sous forme de fichier texte dans le répertoire output1
answer.saveAsTextFile("output1")

// Main of our program

// Charge le fichier contenant les données du graphe social (liste d'adjacence)
val data = sc.textFile("soc-LiveJournal1Adj.txt")
```

```

// Sépare chaque ligne par tabulation et garde les lignes valides avec 2 colonnes (user et liste
d'amis)

val data1 = data.map(x => x.split("\t"))

    .filter(li => (li.size == 2))

// Génère des paires (userA, userB) avec la liste des amis, puis calcule les amis communs
(intersection)

val pairCounts = data1.flatMap(pairs)

    .reduceByKey({ case (param1, param2) => (param1.intersect(param2)) })

// Met en forme le résultat : userA <TAB> userB <TAB> liste des amis communs

val p1 = pairCounts.map({ case ((param1, param2), param3) =>

    param1 + "\t" + param2 + "\t" + param3.mkString(",")

})

// Sauvegarde les résultats globaux dans le dossier output

p1.saveAsTextFile("output")

// ---- Partie filtrage des paires spécifiques ----

// Initialise une chaîne pour accumuler les résultats ciblés

var ans = ""

// Pour la paire (0,4)

val p2 = p1.map(x => x.split("\t"))

    .filter(x => (x.size == 3))

    .filter(x => (x(0) == "0" && x(1) == "4"))

    .flatMap(x => x(2).split(","))

    .collect()

ans = ans + "0" + "\t" + "4" + "\t" + p2.mkString(",") + "\n"

// Pour la paire (20,22939)

```

```
val p3 = p1.map(x => x.split("\t"))
    .filter(x => (x.size == 3))
    .filter(x => (x(0) == "20" && x(1) == "22939"))
    .flatMap(x => x(2).split(","))
    .collect()

ans = ans + "20" + "\t" + "22939" + "\t" + p3.mkString(",") + "\n"
```

// Pour la paire (1,29826)

```
val p4 = p1.map(x => x.split("\t"))
    .filter(x => (x.size == 3))
    .filter(x => (x(0) == "1" && x(1) == "29826"))
    .flatMap(x => x(2).split(","))
    .collect()

ans = ans + "1" + "\t" + "29826" + "\t" + p4.mkString(",") + "\n"
```

// Pour la paire (19272,6222)

```
val p5 = p1.map(x => x.split("\t"))
    .filter(x => (x.size == 3))
    .filter(x => (x(0) == "19272" && x(1) == "6222"))
    .flatMap(x => x(2).split(","))
    .collect()

ans = ans + "6222" + "\t" + "19272" + "\t" + p5.mkString(",") + "\n"
```

// Pour la paire (28041,28056)

```
val p6 = p1.map(x => x.split("\t"))
    .filter(x => (x.size == 3))
    .filter(x => (x(0) == "28041" && x(1) == "28056"))
    .flatMap(x => x(2).split(","))
    .collect()

ans = ans + "28041" + "\t" + "28056" + "\t" + p6.mkString(",") + "\n"
```

// Transforme la chaîne ans en un RDD

```
val answer = sc.parallelize(Seq(ans))

// Sauvegarde les résultats filtrés dans output1
answer.saveAsTextFile("output1")
```

V.TP N° 3 : Spark

Objectif du TP Spark :

Ce rapport présente les étapes réalisées dans le cadre d'un TP Spark qui consiste à calculer les amis communs entre deux utilisateurs (Sidi et Mohamed) dans un graphe social. Le TP se déroule selon deux modes d'exécution :

- Exécution interactive via le shell PySpark
- Exécution d'un fichier Python structuré via spark-submit

Le projet final est hébergé sur GitHub :

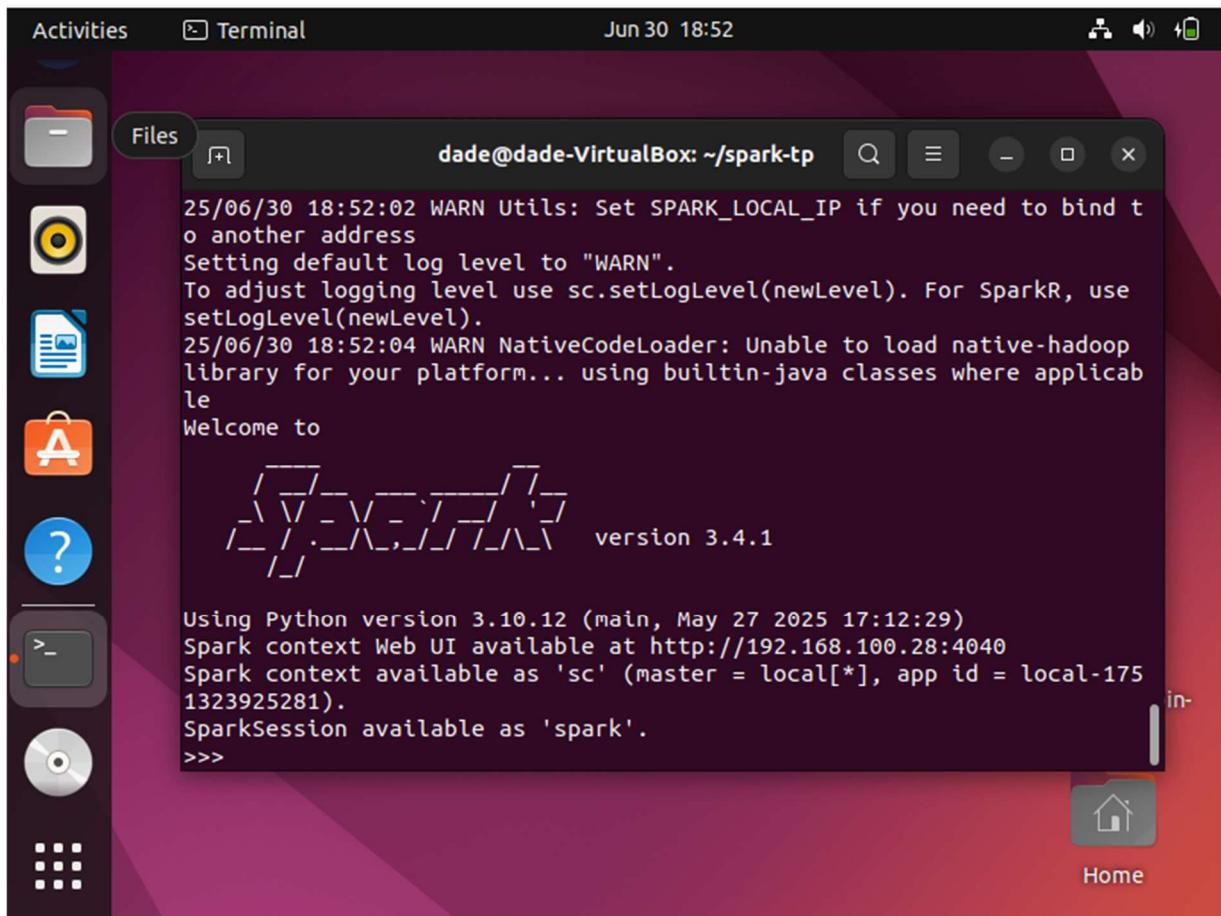


1. Exécution interactive dans Spark

Étape 1 : Démarrage de Spark

Nous avons lancé PySpark avec :

```
pyspark
```

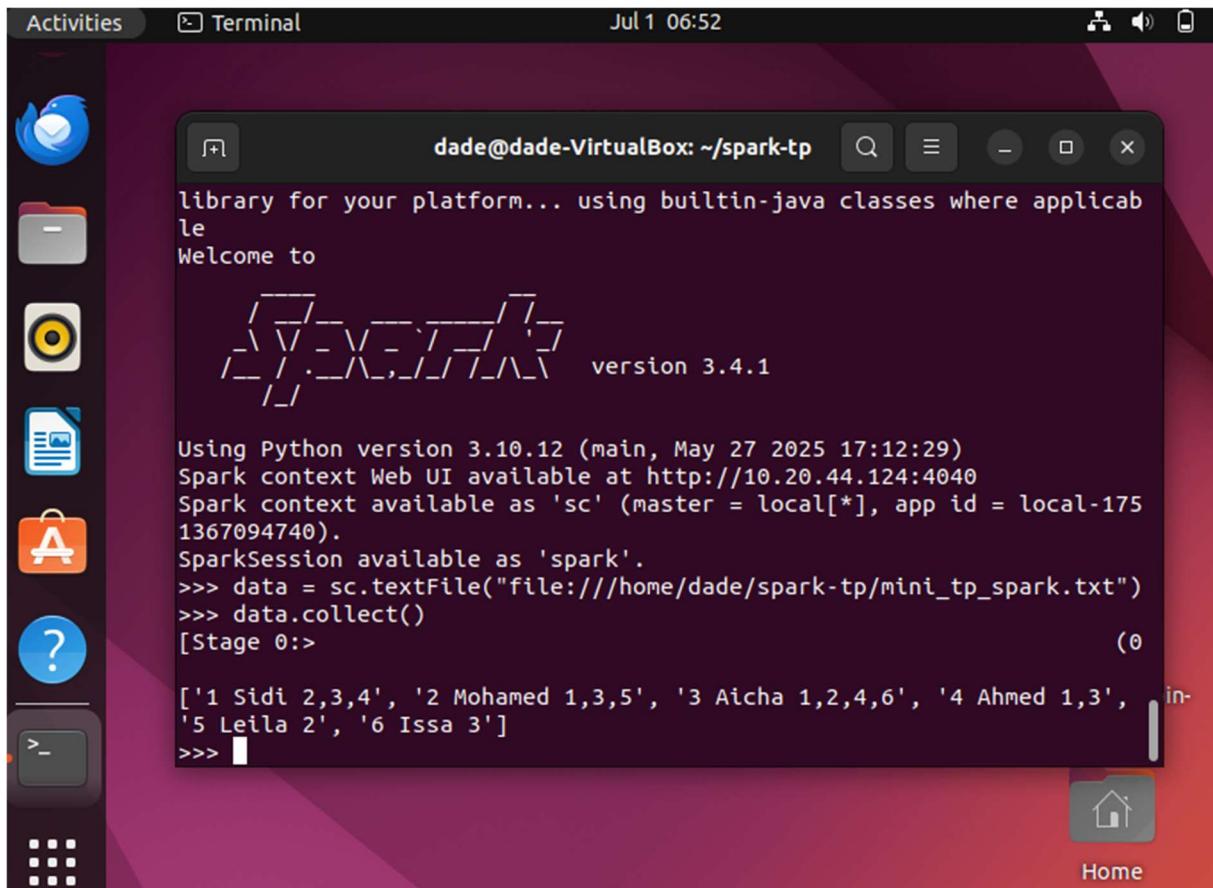


Étape 2 : Chargement des données

Chargement du fichier texte contenant les données :

```
data = sc.textFile("file:///home/dade/spark-tp/mini_tp_spark.txt")
```

`data.collect()`



Étape 3 : Génération des paires d'amis

```
def generate_pairs(line):
    parts = line.split()
    user_id = parts[0]
    friends = parts[2].split(",")
    pairs = []
    for friend in friends:
        pair = tuple(sorted([user_id, friend]))
        pairs.append((pair, set(friends)))
    return pairs

pair_rdd = data.flatMap(generate_pairs)

for pair, friends in pair_rdd.collect():

    print(f"Paire: {pair} -> Amis: {', '.join(friends)}")
```

```
Activities Terminal Jul 1 07:06
dade@dade-VirtualBox: ~/spark-tp
...
...     return pairs
...
>>> pair_rdd = data.flatMap(generate_pairs)
>>> pair_rdd.collect()

[Stage 1::] (0)
[((('1', '2'), {'4', '3', '2'}), ((('1', '3'), {'4', '3', '2'}), ((('1',
'4'), {'4', '3', '2'}), ((('1', '2'), {'1', '5', '3'}), ((('2', '3'), {'1',
'5', '3'}), ((('2', '5'), {'1', '5', '3'}), ((('1', '3'), {'4', '1',
'6', '2'}), ((('2', '3'), {'4', '1', '6', '2'}), ((('3', '4'), {'4', '1',
'6', '2'}), ((('3', '6'), {'4', '1', '6', '2'}), ((('1', '4'), {'1',
'3'}), ((('3', '4'), {'1', '3'}), ((('2', '5'), {'2'}), ((('3', '6'), {'3'}))
...
>>>
>>> common_friends = pair_rdd.reduceByKey(lambda x, y: x.intersection(y))
>>> for pair, friends in common_friends.collect():
...     print(f"Paire: {pair} -> Amis communs: {', '.join(friends) if friends else 'Aucun'}")
...

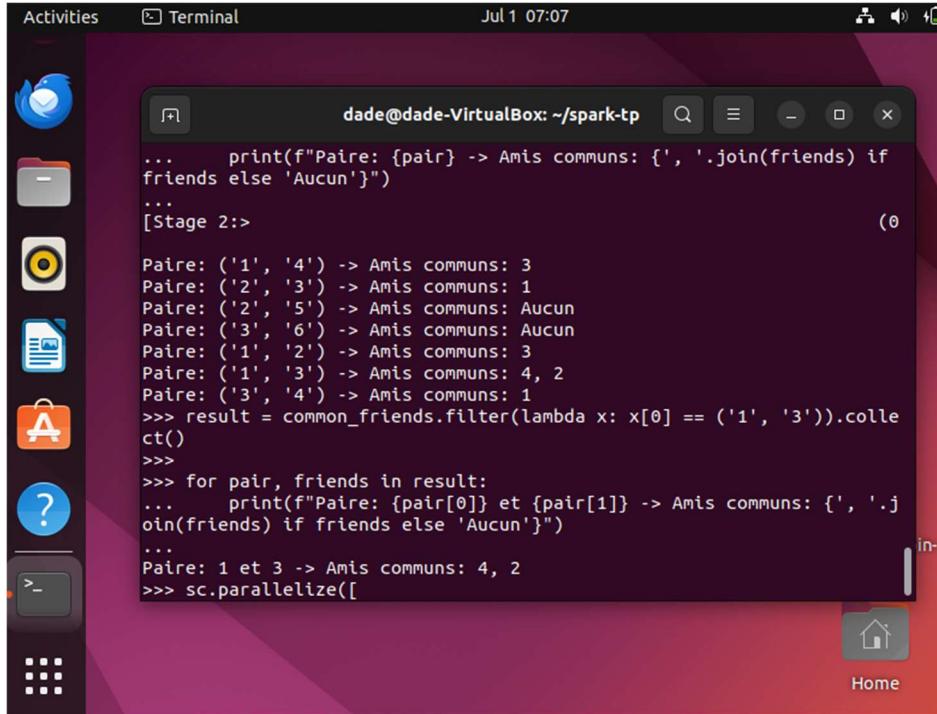
```

Étape 4 : Filtrage des amis communs pour la paire (1, 2)

```
result = common_friends.filter(lambda x: x[0] == ('1', '2')).collect()

for pair, friends in result:

    print(f"Paire: {pair[0]} Sidi {pair[1]} Mohamed -> Amis communs: {', '.join(friends) if friends else 'Aucun'}")
```



A screenshot of an Ubuntu desktop environment. On the left, there's a dock with various icons: Dash, Home, Activities, Terminal, and others. A terminal window titled "dade@dade-VirtualBox: ~/spark-tp" is open, showing Python code running on a Spark cluster. The code prints out pairs of numbers and their common friends. The terminal shows:

```
...     print(f"Paire: {pair} -> Amis communs: {''.join(friends) if friends else 'Aucun'}")
...
[Stage 2:>
Paire: ('1', '4') -> Amis communs: 3
Paire: ('2', '3') -> Amis communs: 1
Paire: ('2', '5') -> Amis communs: Aucun
Paire: ('3', '6') -> Amis communs: Aucun
Paire: ('1', '2') -> Amis communs: 3
Paire: ('1', '3') -> Amis communs: 4, 2
Paire: ('3', '4') -> Amis communs: 1
>>> result = common_friends.filter(lambda x: x[0] == ('1', '3')).collect()
>>>
>>> for pair, friends in result:
...     print(f"Paire: {pair[0]} et {pair[1]} -> Amis communs: {''.join(friends) if friends else 'Aucun'}")
...
Paire: 1 et 3 -> Amis communs: 4, 2
>>> sc.parallelize([
```

2. Exécution d'un fichier Python structuré via spark-submit

Après validation des codes en mode interactif :

- Création du fichier `tp_spark.py` contenant l'ensemble des étapes.

Activities Terminal Jun 30 13:37

dade@dade-VirtualBox: ~/spark-tp

```
GNU nano 6.2          tp_spark.py *
```

```
from pyspark import SparkContext

def generate_pairs(line):
    parts = line.strip().split()
    user_id = parts[0]
    friends = parts[2].split(",")
    pairs = []
    for friend in friends:
        pair = tuple(sorted([user_id, friend]))
        pairs.append((pair, set(friends)))
    return pairs

if __name__ == "__main__":
    sc = SparkContext("local", "TP Spark - Amis Communs")

    data = sc.textFile("mini_tp_spark.txt")
    pair_rdd = data.flatMap(generate_pairs)
    common_friends = pair_rdd.reduceByKey(lambda x, y: x & y)

    result = common_friends.filter(lambda x: x[0] == ('1', '2')).collect()

    for pair, friends in result:
        print(f"Paire: {pair[0]} (Sidi) et {pair[1]} (Mohamed) -> Amis communs")

    sc.stop()
```

Help Write Out Where Is Cut Execute
Exit Read File Replace Paste Justify

- lancer le programme :

The image shows a standard Ubuntu desktop environment with a purple gradient background. A dock on the left contains icons for Dash, Home, Activities, and several application icons. Two terminal windows are open in the center.

Terminal Window 1:

```
dade@dade-VirtualBox:~/spark-tp$ nano tp_spark.py
dade@dade-VirtualBox:~/spark-tp$ ~/Desktop/spark-3.4.1-bin-hadoop3/bin
/spark-submit tp_spark.py
25/06/30 15:45:38 WARN Utils: Your hostname, dade-VirtualBox resolves
to a loopback address: 127.0.1.1; using 192.168.100.28 instead (on int
erface enp0s3)
25/06/30 15:45:38 WARN Utils: Set SPARK_LOCAL_IP if you need to bind t
o another address
25/06/30 15:45:38 INFO SparkContext: Running Spark version 3.4.1
25/06/30 15:45:38 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicab
le
25/06/30 15:45:39 INFO ResourceUtils: =====
=====
25/06/30 15:45:39 INFO ResourceUtils: No custom resources configured f
or spark.driver.
25/06/30 15:45:39 INFO ResourceUtils: =====
=====
25/06/30 15:45:39 INFO SparkContext: Submitted application: TP Spark - | in-
Amis Communs
25/06/30 15:45:39 INFO ResourceProfile: Default ResourceProfile create
```

Terminal Window 2:

```
192.168.100.28:41967 in memory (size: 6.1 KiB, free: 366.3 MiB)
25/06/30 15:45:43 INFO SparkUI: Stopped Spark web UI at http://192.168
.100.28:4040
25/06/30 15:45:43 INFO MapOutputTrackerMasterEndpoint: MapOutputTrack
erMasterEndpoint stopped!
25/06/30 15:45:43 INFO MemoryStore: MemoryStore cleared
25/06/30 15:45:43 INFO BlockManager: BlockManager stopped
25/06/30 15:45:43 INFO BlockManagerMaster: BlockManagerMaster stopped
25/06/30 15:45:43 INFO OutputCommitCoordinator$OutputCommitCoordinator
Endpoint: OutputCommitCoordinator stopped!
25/06/30 15:45:43 INFO SparkContext: Successfully stopped SparkContext
25/06/30 15:45:44 INFO ShutdownHookManager: Shutdown hook called
25/06/30 15:45:44 INFO ShutdownHookManager: Deleting directory /tmp/sp
ark-c9b995b3-f4ad-4cd0-8260-9e86d5d83044
25/06/30 15:45:44 INFO ShutdownHookManager: Deleting directory /tmp/sp
ark-02d78dc3-9a77-44d0-87f8-9ad1bef50ed8/pyspark-89c27d40-ad3b-4c3b-90
45-43e24c0d34c2
25/06/30 15:45:44 INFO ShutdownHookManager: Deleting directory /tmp/sp
ark-02d78dc3-9a77-44d0-87f8-9ad1bef50ed8
dade@dade-VirtualBox:~/spark-tp$ cat output.txt
Paire: 1 Sidi 2 Mohamed -> Amis communs: 3
```

VI. Conclusion

Les travaux réalisés dans ce TP ont permis d'explorer concrètement des technologies majeures du Big Data et de comprendre leurs usages respectifs dans le traitement des données massives.

À travers la manipulation de **MongoDB**, nous avons appris à gérer des données semi-structurées et à appliquer des opérations CRUD dans un environnement NoSQL.

Avec **Hadoop**, nous avons expérimenté le modèle **MapReduce** et les principes du traitement distribué, en apprenant à exécuter des tâches sur des volumes de données simulés.

Enfin, grâce à **Spark**, nous avons découvert une approche plus interactive et rapide du traitement parallèle, en particulier à travers l'analyse de graphes sociaux et le calcul d'amis communs.

Ce TP nous a apporté une richesse d'informations pratiques : il nous a permis de relier théorie et pratique, de comprendre les différences et complémentarités entre ces outils, et de mieux appréhender les défis liés au Big Data.

Les compétences acquises à travers ces exercices nous préparent à manipuler des projets plus complexes et à aborder des cas réels d'analyse de données volumineuses.